CARNEGIE MELLON UNIVERSITY

Game-Theoretic Power Management in Mobile Ad Hoc Networks

A DISSERTATION
SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY
in
ELECTRICAL AND COMPUTER ENGINEERING

by

John G. Dorsey

Pittsburgh, Pennsylvania
August 2004

Abstract

This thesis studies the interaction between on-demand *ad hoc* routing protocols and wireless network interface power management. When traffic sources choose their routes without regard for the preferences of relays and sinks, nodes can experience highly variable energy consumption. We argue for negotiations that allow nodes to express their valuation for different network configurations. A practical method for negotiation in mobile *ad hoc* networks is detailed and then evaluated in a realistic simulation environment.

Energy consumption in 802.11 radios is dominated by the idle state, in which the interface is waiting to exchange data. This induces a non-intuitive cost structure: if the radio is awake and servicing traffic, the marginal energy costs of servicing *more* traffic are small. If a relay or sink wants to report its costs in a negotiation, it must be able to express this energy complementarity when asked to support overlapping traffic flows. Source routing networks implement a combinatorial reverse auction, which cannot account for overlap. We show that the combinatorial exchange is the right concept to address overlap and other auction deficiencies such as the lack of unaffordable routes.

The use of credit incentives in *ad hoc* networks is complementary to this work. We address an important issue related to the incentive compatibility of payments to our exchange. If agents can improve their expected utility through strategic bidding, then they are incented to perform costly modeling of one another to learn their expected best bidding strategy. Our design uses a Vickrey Clarke Groves mechanism to make the bidding process strategy-proof. We show that an optimal exchange solution that accounts for energy complementarity is *required* to ensure strategy-proofness, and that no previous mechanism is strategy-proof in the power managing environment.

We have developed Exchange Power Management, which improves the energy performance of protocols such as Dynamic Source Routing. Compared with unmodified 802.11 power management, we can reduce the range of node energy consumption by up to a factor of 5 when high message delivery ratios are required. Permitting unaffordable routes, the improvement increases up to a factor of 12. Our design offers better average case energy consumption than 802.11 power management, and is competitive with previous work in cross-layer power management. We also address setup latency under power management, and introduce the fast wakeup technique which can improve route discovery latency by up to a factor of 16.

Our work is the first to take a systems view of mechanism design application to *ad hoc* networks. We have incorporated realistic models of energy consumption in popular radios, and have addressed the practical issues of integration with real protocols such as 802.11 and DSR. The experimental results of this thesis characterize the kinds of energy performance improvements that could be expected from negotiation-based power management. Future work will refine the fault-tolerance and scalability of our distributed protocol, increase the sophistication of our agent valuation functions, and examine application awareness of exchange-based route selection.

Acknowledgments

I owe the successful completion of this thesis to my advisor, Dan Siewiorek. Without his guidance and support, I could not have explored the breadth of research areas that comprise this work. Dan allowed me to investigate topics ranging from practical system implementation to theoretical protocol design. This freedom has made the graduate school experience a truly wonderful time in my life, both as an engineer and as a person. Dan's ability to see the fundamental nature of seemingly new problems will always amaze me, and I am honored to be his "#56."

I was fortunate to receive the guidance of three excellent researchers who served on my thesis committee. David Johnson sparked my interest in systems programming as an undergraduate, when he taught the Operating Systems course at Carnegie Mellon. His work in wireless networking, and our conversations over the years, have inspired and shaped this thesis. Tuomas Sandholm has been an excellent mentor for my education in mechanism design. Without his help, I would surely have proceeded down many blind alleys. Finally, Tom Martin has been a wonderful friend since the days when we shared an office in Hamburg Hall. I have benefitted from the example of his dissertation and his work on power management in mobile devices, but more importantly, from his stories and our conversations.

I was privileged to work with many fine colleagues on the development of the Spot wearable computer. Francine Gemperle is a superior artist, whose many talents will surely take her far. Brian Gollum's encyclopædic knowledge of mobile and wearable systems was invaluable, as is his friendship. Bruce Brodsky, entrepreneur and engineer, was exactly the right man for the job of bringing Spot to life.

Many other friends have helped me during my time in graduate school. Jolin Warren, Harvey Vrsalovic, and Matt Hornyak have all made the office a brighter place to work. David Maltz contributed the extended *ns* simulator as well as his feedback on this research. Neeraj Bansal, Tim Kniveton, and Andrew Willis have been irreplaceable comrades since our days as undergraduates. David Tolliver and Erica Brusselars are brilliant, and ensured that I took the time to have fun.

My most treasured friend throughout this entire process has been Melissa Chan. We met just as I was beginning graduate school, and she has been my sunshine ever since. Without her patience and love, I could not have crossed this finish line. She is my favorite traveling companion, cooking partner, and athlete. I am so proud of her.

Finally, I could never have reached this point without my parents, Greg and Mary Jane, and my sister Heather. They endured all manner of lessons, practices, and performances during my youth, and supported me through every endeavor. Everything I enjoy I can trace back to the experiences they provided. I dedicate this thesis to my family.

Thank you all.
JGD
August 2004

Contents

List of Figures

List of Tables

List of Procedures

Notation

B_j the bid describing an agent's reported valuation for a network configuration, 41

$\Delta_{i,\text{vick}}$ the Vickrey discount deducted from an agent's reported valuation when determining its payment to the mechanism, 63

$f(\cdot)$ the social choice function which selects an allocation based on the agents' types, 49

$g(\cdot)$ the outcome function that chooses a network configuration based on the strategies used by the agents in reporting their preferences, 55

Γ the mechanism defining the procedure by which agent strategies are used to select a network configuration, 56

λ_j the allocation of items to agents, either supplying relay or sink service, or demanding a route; a partial network configuration, 41

$+$ the OR-constraint operator, expressing that both of its operand bids may be assigned winning, 46

$\oplus$ the XOR-constraint operator, expressing that *at most one* of its operand bids may be assigned winning, 46

P the route having $|P|$ nodes and $|P| - 1$ hops, 34

$p_{i,\text{vick}}$ the Vickrey payment an agent makes to the mechanism when it has winning bids, 63

$\phi(\cdot)$ the probability distribution an agent knows for the likely valuations of some other agent, 55

p_j the price of a bid indicating the value, or loss in value, an agent experiences from some network configuration, 41

$\sigma_i(\cdot)$ the strategy used by an agent in choosing a valuation to report, given its actual preferences over network configurations, 55

θ_i the type, representing the agent's privately-held information about its preferences over network configurations, 48

$u_i(\cdot)$ the quasilinear utility an agent experiences from a network configuration, its intrinsic valuation for the configuration less its payment, 51

V^* the surplus (sum of prices) of the surplus-maximizing combination of bids, 62

$\hat{v}_i(\cdot)$ the valuation an agent reports for an allocation, which may not be its *true* valuation, 50

$v_i(\cdot)$ the agent valuation function mapping an allocation (route selection) to a scalar measure of preference, 49

Chapter 1

Introduction

An ***ad hoc* network** (Perkins, 2001) is a collection of computing devices which form an impromptu communications system. *Ad hoc* networks can be **wireless**, if the devices communicate using radios or infrared transceivers. They can be **mobile**, meaning that the devices can physically move, creating and breaking links as they do so. They may also be **multihop**, meaning that the diameter of the network may be larger than the range of an individual wireless transceiver, thus requiring some nodes to **relay** traffic for others. Research into routing protocols for this environment is nearly a decade old, and has produced many solutions to the problem of reliable and efficient message delivery (Johnson et al., 2003; Perkins et al., 2003; Clausen and Jacquet, 2003).

An important dimension of the *ad hoc* routing problem which has received attention in recent years is that of **energy consumption**. The mobile devices which make up a network are powered by batteries or some other limited supply of energy. Expending this energy on routing tasks may shorten the device's active lifetime, or prevent it from completing other tasks assigned to it. On the other hand, conserving energy by *not* performing routing tasks may prevent the network from functioning. This conflict between **self interest** (with respect to energy) and **social utility** (with respect to communications) is depicted in Figure 1.1.

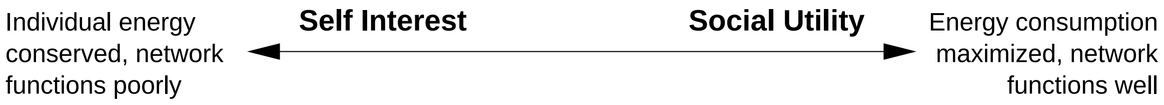

Individual energy conserved, network functions poorly
Self Interest
Social Utility
Energy consumption maximized, network functions well

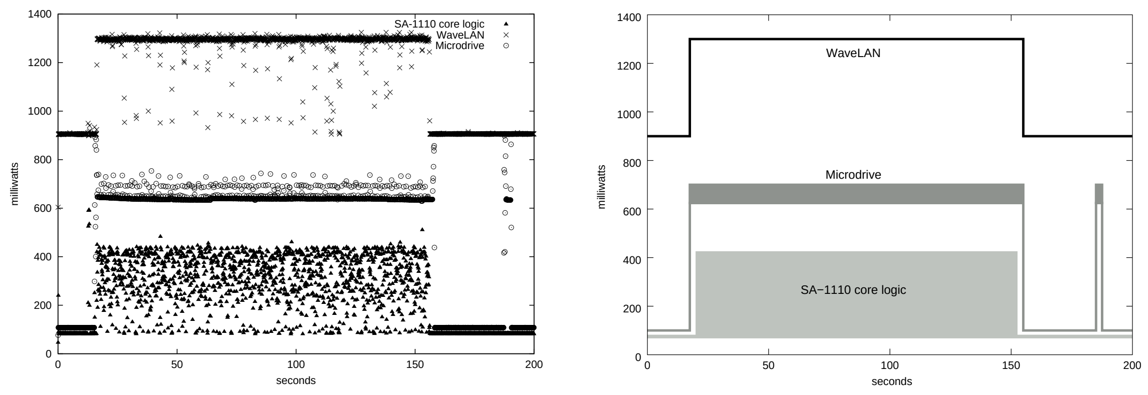
Figure 1.1: Self interest and social utility in *ad hoc* networks.

Wearable computers (Barfield and Caudell, 2001), handhelds, and other low-power mobile devices are all well-suited for *ad hoc* networking. Wearables, for example, support an on-the-go usage model which makes them better candidates for mobile communication than laptops. Stored energy is at a premium with these systems; **battery capacities** for handhelds and low-power wearables are an order of magnitude smaller than those for laptop computers. In order to provide a useful **service lifetime**, mobile devices employ **power management** techniques such as processor suspension and **clock scaling** (Martin, 1999). The most aggressive methods, those which **disable** the processor or wireless network interface, effectively **remove** a device from a network in which it participates. When the mobile devices are needed for message relay service, unilateral withdrawal from the network due to power management may prevent some messages from being delivered.

Figure 1.2: The Spot wearable computer.

The **Spot wearable computer** (Dorsey, 2004), shown in Figure 1.2, provides some insights into the energy costs of implementing *ad hoc* networks. Spot includes several **power monitors** (Dorsey and Siewiorek, 2002) which measure the energy consumption of subsystems such as the microprocessor, disk drive, and radio. Figure 1.3 shows a trace of the energy consumed by these three subsystems during a workload which stressed computation, disk reads, and wireless transmission. The trace reveals that the **transmit** power rate of the IEEE 802.11b radio is almost *twice* the read mode power of the disk drive, and more than *three times* the active mode power of the microprocessor. More importantly for *ad hoc* networks, the **idle state** power of the radio is *nine times* that of the disk or processor in their quiescent states.

The idle mode power rate of the 802.11b radio accounts for about **half** of all energy consumption in Spot. The Spot design contains an unusual number of memory and I/O components for a device of its class. For more conventional



(a) Sample data. (b) Simplified key.

Figure 1.3: Power trace of microprocessor, hard drive, and radio.

systems with fewer components, the radio would represent an even greater share of the energy budget. To improve the overall energy performance of such systems, clearly one must begin with the idle mode power of the radio. This thesis presents a method for controlling the amount of time the radio spends in idle mode while preserving *ad hoc* communications performance.

Energy **scarcity** induces the fundamental problem of power management in *ad hoc* networks: which nodes will bear the energy costs of **implementing** the network? This research looks to **economics** for the answer. We adopt a **game-theoretic** (Osborne and Rubinstein, 1994) approach to deciding how the nodes will distribute the burden of network implementation. Game theory is a set of tools which help us to understand the behavior of decision-making **agents**. In the *ad hoc* network setting, we will assume that a **power managing agent** at each node holds **preferences** over the possible configurations of the network. Factors such as the node's traffic-sourcing needs or history of energy consumption might figure into an agent's preference over which routes it can use, or whether it provides relay service for a given traffic flow. The agents work to satisfy **user goals**, such as achieving battery lifetime targets.

Users of mobile devices might be part of the same organization, or be entirely unaffiliated with one another. In the latter case, users have no inherent **incentive** to spend their batteries on the traffic needs of other users. We assume that some external **enforcement scheme**, such as a **credit** or **money** system, is available to provide this incentive. For example, agents might accrue credit by providing relay service, which they then spend in order to source traffic into the network. This thesis does not specify an enforcement scheme, but instead focuses on a method by which the agents can reach a **decision** about what configuration to adopt.

We recognized that the problem of selecting a route — a collection of **relays** and a **sink** — is an instance of an economic **combinatorial allocation problem**. In fact, the **Dynamic Source Routing** protocol implements exactly the **combinatorial reverse auction** when traffic sources choose their routes. Route selection is a *combinatorial* problem because only certain *combinations* of relays and sinks are valuable to traffic sources, namely, those which form a viable route. Our solution to this problem allows sources to submit **reports** about how much value they derive from individual routes. It also allows relays and sinks to report how much *loss* in value they experience from providing service to the network. We can then aggregate these reports to choose a network configuration that maximizes net value over the agents.

If the enforcement scheme used to ensure adherence to the chosen configuration is based on credit, there is an additional **strategic** problem to solve. We want the agents to submit **truthful** reports about their respective valuations. If the agents make **payments** using credit in order to implement a configuration, then there may be an incentive to **misreport** their true valuations. For example, an agent might strategically **underbid** so as to reduce its expected payment. We apply concepts from **mechanism design**, an area of game theory, to make the payment rules **strategy-proof**. That is, truthful reporting by the agents becomes their best strategy.

Strategy-proof bidding is useful for agents executing on mobile devices for several reasons. In contrast with weaker solution concepts, such as the **Bayesian Nash equilibrium**, strategy-proofness does not require the agents to develop **distributional information** about each others' behavior. This is critical for an *ad hoc* network, since **mobility** and other factors interfere with an agent's ability to observe the actions of its peers. Also, strategy-proofness simplifies the **computational** aspect of bidding, since the agents do not have to statistically model one another when choosing their bid prices.

We show that the dominance of the **idle state** in wireless interface energy consumption induces non-intuitive **energy costs**. Per-message cost models do not reflect the way energy is actually consumed in devices such as 802.11 radios. Instead, we argue for a model based on **time periods** in which a relay or sink is actively providing service. Given that a node is active *at all*, we show that the **marginal energy costs** of relaying additional traffic flows are small. As a result, a negotiation mechanism for the power managing environment must account for **overlap** — or **energy complementarity** — among traffic flows at relays or sinks. We describe the **combinatorial exchange**, which permits such overlap, and show how an **optimal solution** to the exchange is *required* to achieve **strategy-proofness**. Previous applications of mechanism design to networks do not have a way to account for overlap, and fail to be strategy-proof in the power managing environment.

Using these concepts, we have developed the **Exchange Power Management** design which implements **combinatorial exchange**-based route selection in *ad hoc* networks. Our design echoes the **on-demand** themes of protocols such as Dynamic Source Routing. Negotiations between nodes happen only when needed to choose routes for **active** traffic flows, and involve only the minimal set of sources, relays, and sinks which might be involved in those flows. Nodes that are **irrelevant** to active flows do not participate in negotiations, and use low-level power management to minimize their energy consumption.

We have also developed simple **valuation functions** which implement the concept of agent **preference**. Agents submit **bids** describing their preferred network configurations based on concepts such as their credit balance, or the duration of service they have provided to the network. We show through experimental simulation that these simple models can significantly reduce **variability** in node energy consumption.

1.1 Research Scope

This is an interdisciplinary thesis combining concepts from the fields of wireless networking and economics. As shown in Figure 1.4, the Exchange Power Management protocol developed though this research inherits from work in wireless LANs, *ad hoc* routing protocols, combinatorial auctions and exchanges, and mechanism design. The significance of this effort is in the application of economic and game-theoretic principles to **practical** network protocol design. To the best of our knowledge, this is the first work to apply exchanges and mechanism design to **real protocols** in the *ad hoc* network environment.

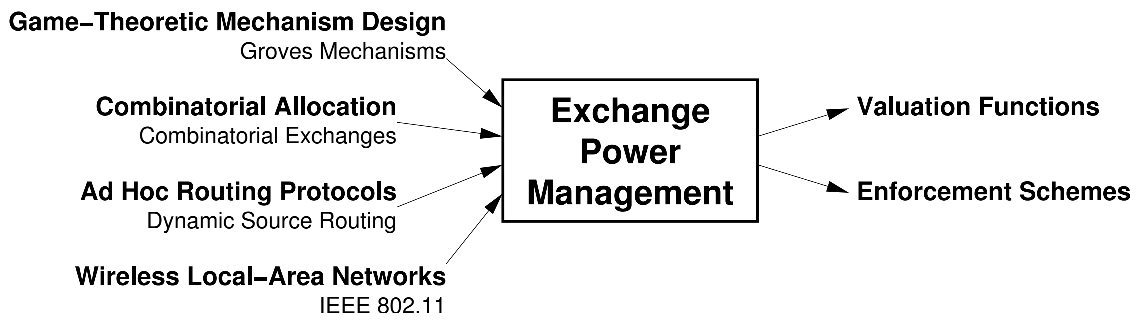


Figure 1.4: Scope of this research.

As described earlier, this research specifies a procedure by which agents can reach a group decision about the configuration of an *ad hoc* network. We have left the means by which that decision is **enforced** to future work, although in Chapter 3, we discuss some existing efforts which might be helpful in this regard. This thesis does supply an important constraint on the characteristics of an enforcement scheme, specifically regarding **budget balance** and credit systems.

For the purposes of this thesis, we have implemented two very simple **valuation functions** in order to provide examples of agent bid pricing. These models are effective in demonstrating the benefits of exchange-based route selection, but they operate on a relatively modest amount of information. The development of more sophisticated valuation systems is an interesting topic for subsequent research.

1.2 Contributions

This research is the first application of exchanges and mechanism design to the route selection problem in power managing *ad hoc* networks. Specific contributions to the field of *ad hoc* networking research include:

- Discovery of the correspondence between **source routing protocols** and **combinatorial reverse auctions**.

- Design of a **strategy-proof combinatorial exchange** concept for source route selection in power managing *ad hoc* networks. This concept is **expressive** for sources, relays, and sinks. It also introduces the concept of **unaffordable** routes, and accounts for **energy complementarity**.

- Design of the **Exchange Power Management** protocol to support exchange-based route selection in practical networks. We cast this protocol as a companion to Dynamic Source Routing, and take advantage of **low-level power management** features such as those provided by 802.11.
- Design of the **fast wakeup** technique, which reduces the latency of route discovery under power management by an order of magnitude.
- Design of a cross-layer **multihop power management architecture** supporting **power management suspension**. Combined with fast wakeups, we show how to reduce **average-case** application message delivery latency by an order of magnitude.
- Development of several improvements to the *ns* network simulator, including an implementation of **802.11 IBSS power management** and several additional management features.

1.3 Organization

This thesis first describes core concepts and related work before presenting our exchange-based power management protocol. We then provide details about the experimental environment and discuss results.

Chapter 2: Basic Technology introduces IEEE 802.11 wireless LANs, the Dynamic Source Routing protocol, combinatorial auctions and exchanges, and the game-theoretic Groves mechanisms.

Chapter 3: Related Work surveys prior efforts to study and improve energy performance and communications performance in wireless networks.

Chapter 4: System Design describes **multihop power management**, a cross-layer design which supports timer-based and exchange-based power management.

Chapter 5: Simulator and Workloads details improvements to the *ns* network simulator and describes experimental workloads.

Chapter 6: Experimental Results presents the energy and latency improvements observed in the simulation experiments.

Chapter 7: Conclusion summarizes this research.

Chapter 2

Basic Technology

This research is founded on four basic technologies, two from the field of wireless networking, and two from economics. A key insight of this thesis is the bridge between the fields; specifically, that **source routing protocols** are an instance of an **auction**. This Chapter provides an overview of all four technologies, develops the relationships between them, and defines terms used later in the thesis. Figure 2.1 shows the organization of the material.

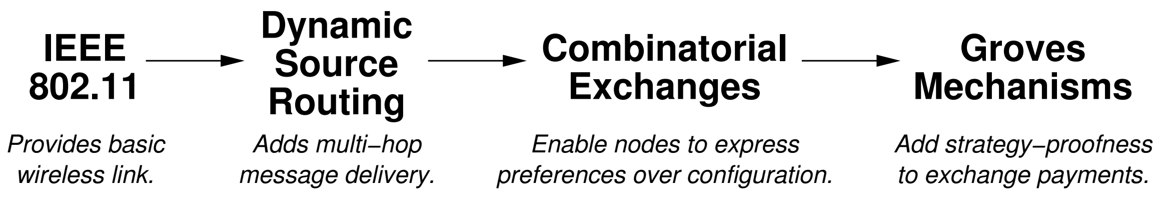

IEEE 802.11
Provides basic wireless link.
Dynamic Source Routing
Adds multi–hop message delivery.
Combinatorial Exchanges
Enable nodes to express preferences over configuration.
Groves Mechanisms
Add strategy–proofness to exchange payments.

Figure 2.1: Technology overview.

Section 2.1 describes the IEEE 802.11 wireless LAN protocol, a popular link technology used in contemporary wireless networks. Dynamic Source Routing, a mobile ad hoc routing protocol, is presented in Section 2.2 and its interaction with 802.11 is discussed. Section 2.3 introduces combinatorial exchanges, which allow nodes to express their preferences over which routes are used in a power-managing ad hoc network. Finally, Section 2.4 presents **mechanism design**, a technique for ensuring certain game-theoretic properties in strategic interactions such as the combinatorial exchange.

2.1 IEEE 802.11 Wireless LANs

The **IEEE 802.11** wireless LAN standard (IEEE, 1997) defines "a medium access control (MAC) and physical layer (PHY) specification for wireless connectivity for fixed, portable, and moving stations within a wireless local area." Originally designed for a 1Mbps and 2Mbps data payload communications capability, data rates up to 11Mbps were later added (IEEE, 1999), and 54Mbps products are available at the time of this writing (IEEE, 2003). Although several physical layers have been specified, including infrared and multiple radio-frequency spread spectrum methods, the most popular has been **Direct Sequence Spread Spectrum** (Rappaport, 1996) in the 2.4GHz industrial, scientific, and medical band.

In addition to framing, addressing, and other typical MAC tasks, 802.11 provides functionality unique to wireless networks such as **hidden terminal** avoidance and **power management**. The hidden terminal problem, shown in Figure 2.2, occurs when a receiver is within radio range of multiple transmitters which are

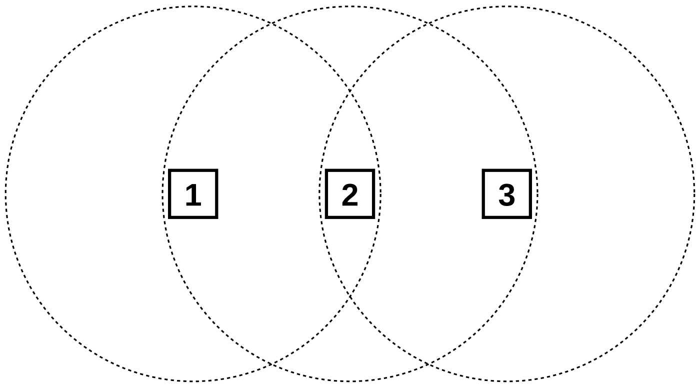


Figure 2.2: Hidden terminal problem.

mutually out of range of each other. In this example, suppose station 1 attempts to send a message to station 2. Station 3 cannot hear the transmission, because 3 is out of range of station 1. If 3 begins to transmit (to anyone), its message will collide at station 2 with those of the hidden terminal, station 1.

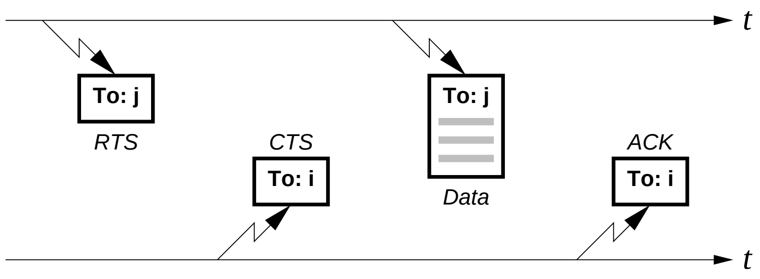


Figure 2.3: 802.11 RTS, CTS, Data, ACK exchange.

802.11 solves this problem through the use of **control frames**,[1] which precede and follow a **data frame** transmission. To send data from station i to station j, station i begins by sending a **request to send** (RTS) frame addressed to j. This alerts any station within range of i to suspend transmission long enough for the

[1]There are three additional control frames beyond those described here. These involve features specific to infrastructure wireless LANs, which are beyond the scope of this thesis.

data transaction to complete. Station *j* responds with a **clear to send** (CTS) frame. The CTS alerts any station within range of the receiver to defer transmissions until the data exchange occurs. Upon receiving the CTS, *i* sends the **data** addressed to *j*, which confirms receipt with an **acknowledgment** (ACK). This process is illustrated in Figure 2.3 (not drawn to scale). RTS, CTS, and ACK frames are only used for **directed**, or unicast, transmissions. **Broadcast** data frames are sent without handshaking or acknowledgment.

A station which sends directed Data frames to a neighbor must be able to receive CTS and ACK control frames from that neighbor. This implies that all directed links in an 802.11 network are **bidirectional**. Also, since directed transmissions are explicitly acknowledged, protocols at the network layer and above that know they are running on top of 802.11 do not need to provide their own acknowledgment facility. Both of these characteristics are exploited by Dynamic Source Routing (§2.2) and the power management design described in Chapter 4.

An 802.11 wireless LAN may connect to an **infrastructure network** through an **access point**. The stations in such a network, including the access point, belong to a **basic service set**, or BSS. The members of a BSS operate under the same **coordination function**, which determines when stations are permitted to transmit frames on the medium. The RTS-CTS handshake described above is part of the contention-based **distributed coordination function** (DCF). Medium access can also be scheduled by the access point, using the **point coordination function** (PCF).

2.1.1 Independent Basic Service Set Operation

When access to an infrastructure network is not provided, the 802.11 wireless LAN is called an **independent basic service set**, or IBSS. This mode of operation is sometimes referred to as an *"ad hoc"* network, but 802.11 only supports *single-hop* networks in which every station can hear every other station. To support a network of larger diameter, a routing protocol such as the one described in Section 2.2 is needed. This thesis uses the term "IBSS" to describe the infrastructureless operation of an 802.11 network, and uses "*ad hoc* network" to refer to generic infrastructureless networks of arbitrary diameter.

2.1.1.1 Beacon Management Frames

Stations join an IBSS by listening for **beacon** frames, which are periodically generated by stations already belonging to the IBSS. Beacons contain the operating parameters for the network, such as the network's name, the set of supported data rates, and the current channel. They also contain the value of the **beacon interval**, which is the nominal time between beacon broadcasts. Example intervals are 100 milliseconds or 200 milliseconds.

In an infrastructure network, beacons are transmitted by the access point. Beacon generation in an IBSS is performed using a randomized distributed algorithm. Starting from time zero, a series of **target beacon transmission times**, or TBTTs, are defined by the beacon interval. At each TBTT, every station suspends normal traffic and sets a short random timer. If the timer expires before any beacon frames have been received, the station generates a beacon for the IBSS. The idea is that only one beacon will be sent per TBTT. In a larger-diameter network, where every node cannot hear every other node, many beacons may be generated.

2.1.1.2 Timing Synchronization Function

IBSS stations mark the TBTTs using a local timer which is maintained by a **timing synchronization function** (TSF). The timer is required to be accurate to $\pm 0.01\%$. The 64-bit, microsecond-resolution timer value is included in all transmitted beacons. A station receiving a valid beacon sets its own timer to the greater of the beacon timer value or its own timer value. This method tries to maintain the synchronization of timers in an IBSS to within 4μs. Chapter 3 describes research into of the scalability of the TSF and its performance under network partitions and power management.

2.1.2 Energy Consumption

It is common to assume that radio energy consumption is dominated by the **transmit state—textbf**. Many examples from the literature, particularly those studying networks from an algorithmic perspective, adopt this assumption. This has led to a number of **transmit power control** schemes for wireless networks (Banerjee and Misra, 2002; Čagalj et al., 2002; Cruz and Santhanam, 2003; Eidenbenz et al., 2003; Kawadia and Kumar, 2003; Lloyd et al., 2002; Srinivasan et al., 2002). These approaches adjust the **radiated power** level of the wireless transmitter. The **range** of the radio increases or decreases with the radiated power (Rappaport, 1996).

The error in this assumption is that it mistakes *power* for *energy*. It is correct that the peak *power* rate for an 802.11 interface occurs in the transmit state. However, *energy* depends on both *power* and *time*: $E = P \times t$. To understand the total energy consumption of an 802.11 interface, we must consider the power rates for each state *and* the amount of time spent in each state.

The power behavior of several 802.11b interfaces has been studied through **high-resolution trace techniques** (Ebert et al., 2002; Feeney and Nilsson, 2001). These data are consistent with our own measurements (Dorsey and Siewiorek, 2002), and in some cases provide a more complete or precise picture of the different radio states. As with our own work, Feeney and Nilsson measured the **Lucent IEEE 802.11 WaveLAN** PCMCIA interface. Ebert et al. studied the **Aironet PC4800** PCMCIA interface, which supported variable radiated power levels. Table 2.1 shows the measured power rates for the entire interface in each of the possible radio states. For the Aironet device, the data for the 20mW radiated power level are given, as this level is closest to the 15dBm ($\sim$32mW) nominal rating for the WaveLAN device (Lucent, 1998).[2]

These data show **total interface power**: the power drawn by *all* electronics in

[2]The maximum permitted output power is 1W in the United States, and 100mW in Europe (IEEE, 1997). Transceivers for mobile applications typically radiate less than 50mW.

Device	**Doze**	**Idle**	**Receive**	**Transmit**
Lucent WaveLAN	47.4mW	739mW	901mW	1.35W
Aironet PC4800	75mW	1.34W	1.4W	1.74W

Table 2.1: 802.11b total interface power states by radio state.

the radio interface. (They are measurements from the **power rails** of the PCMCIA card itself.) While transmitting, total interface power is *much* greater than radiated power. For the WaveLAN device, $\frac{\text{TotalInterface}\,(1.35\text{W})}{\text{Radiated}\,(32\text{mW})}$ is more than 40:1. The ratio is almost 90:1 for the PC4800. The majority of device energy consumption is *not* being emitted onto the wireless medium. Instead, it is used to drive the firmware processor, modem, RF converter, RF amplifier, and other electronic components. In fact, many of these components are also needed to *receive* data, which explains why receive power is 67–80% of transmit power.[3]

The least intuitive measurement concerns the **idle** state, in which the interface is listening on the medium for new frames. The data show that idle power is comparable to receive power, and 55–77% of transmit power.[4] Listening for new frames requires many of the same baseband and RF components as frame reception, with the possible exception of the firmware processor. Section 2.1.3 describes a method by which these components can be disabled when the radio does not need to listen for new frames. This **doze** state forms the basis for **802.11 power management**.

The second element of interface energy consumption is the *time* spent in each

[3]High receive power is not peculiar to wireless LAN radios. The low-power **CC1000** transceiver (Chipcon, 2004) used in the **Cricket** location system (Smith et al., 2004) is an interesting example. At its lowest radiated power level, total interface power in the receive state is actually 10–40% *higher* than in transmit!

[4](Dorsey and Siewiorek, 2002) measured idle power at 69% of transmit power.

state. To understand the timing behavior of an 802.11 station, we present a detailed timeline of **directed data transmission**. Consider a common traffic workload used in *ad hoc* networking research: a **Constant Bit Rate** (CBR) application which generates 512-octet messages at an average rate of 4Hz. This workload might correspond to a media application such as **digital telephony**. For simplicity, we ignore other types of frame transmissions, such as the **beacon management frames** described in Section 2.1.1.1. We focus on the traffic source, which sends directed data frames to a neighboring station using the 802.11b 11Mbps PHY (IEEE, 1999).

As explained in Section 2.1, directed data transmissions are preceded by an **RTS-CTS handshake**, and followed by an **ACK**. There are also **wait states** before, during, and after the transmission procedure. Specifically, if the wireless medium is busy when a frame becomes ready for transmission, the station sets a random **backoff timer**. When the timer expires, the station samples the medium again to see if it has become available. Once the medium has been free for a **DCF interframe space** (DIFS), the transmitting station may send the RTS. Upon receiving the RTS, the neighboring station waits for a **short interframe space** (SIFS), and sends the CTS. The transmitting station then waits one SIFS before sending the Data frame, and the neighbor again waits a SIFS before returning the ACK. Finally, the transmitter waits one DIFS, and sets the random backoff timer again. The final random backoff permits other contending nodes to obtain access to the medium.

When using a **direct sequence spread spectrum** PHY, a SIFS is 10μs and a

DIFS is 50μs. The random backoff timer value is chosen according to $U[0, \mathrm{CW}] \times$ SlotTime. The **contention window** variable, CW, increases from 31 in ascending powers of 2 (minus 1) every time a station retries a transmission. SlotTime is 20μs.

Although the nominal data rate associated with 802.11b is 11Mbps, most frames are sent at a slower rate. The **PHY Layer Convergence Protocol** (PLCP) enables interoperability of 802.11 stations which support different data rates. A 144-bit **PLCP preamble** and a 48-bit **PLCP header** are prepended to every 802.11 frame. These 192 bits are *always* sent at 1Mbps. All **control frames** (*e.g.*, RTS, CTS, ACK) are sent at one of the rates in the **BSS basic rate set**. To ensure compatibility between 802.11 and 802.11b stations, this set typically contains the 1Mbps and 2Mbps rates. **Broadcast data frames** are also sent at one of the basic rates, but **directed data frames** may be sent at a higher rate supported by both sender and receiver, such as 5.5Mbps or 11Mbps in the case of 802.11b. The basic and supported rate sets are enumerated in the **beacon frames** periodically generated within the IBSS.

We can now compute the elapsed time t between the instant a station discovers the medium to be free until the time it completes its directed data transmission. In this example, 32 octets of header information have been appended to the 512-octet application payload, for a total data payload of 544 octets. The resulting data frame, with the **MAC header** and **frame check sequence** (FCS), is 4,768 bits long.

$$
\begin{aligned}
t = \text{DIFS} + {} & \frac{\text{PLCPLength (192 bits)}}{\text{1Mbps}} + \frac{\text{RTSLength (160 bits)}}{\text{2Mbps}} + \\
\text{SIFS} + {} & \frac{\text{PLCPLength (192 bits)}}{\text{1Mbps}} + \frac{\text{CTSLength (112 bits)}}{\text{2Mbps}} + \\
\text{SIFS} + {} & \frac{\text{PLCPLength (192 bits)}}{\text{1Mbps}} + \frac{\text{DataLength (4,768 bits)}}{\text{11Mbps}} + \\
\text{SIFS} + {} & \frac{\text{PLCPLength (192 bits)}}{\text{1Mbps}} + \frac{\text{ACKLength (112 bits)}}{\text{2Mbps}} + \text{DIFS}
\end{aligned}
$$

Given the definitions of SIFS and DIFS above, $t = 1.52$ms. The sending station spends 130μs in idle (8%), 496μs in receive (33%), and 897μs in transmit (59%). As an aside, this example partly illustrates why the user data rate experienced with 802.11b is less than the "11Mbps" advertised. The 544-octet payload (by itself) sent at 11Mbps only requires 396μs, yet the full transaction requires *at least* 3.8 times that long. (This simplified analysis ignores **backoff** and **contention** time, which adds even more to the total transmission time.) Reducing the data payload to a single octet only reduces the total transaction time by a quarter, to 1.13ms. Also, more than quadrupling the payload to the maximum of 2,312 octets only increases transaction time by about 80%, to 2.81ms.

Figure 2.4 shows the timeline for the 512-octet application payload transmission. The power levels are taken from Feeney and Nilsson's measurements of the Lucent WaveLAN device. Time zero is the instant at which the sending station finds the medium to be free. Following the ACK receipt, the sender waits a DIFS

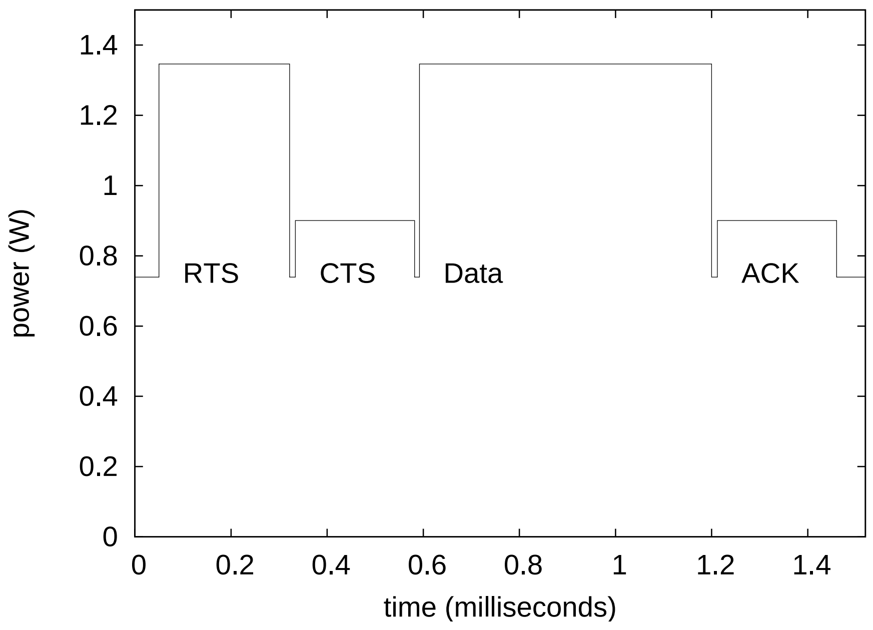


Figure 2.4: Power timeline for 802.11 directed data transmission.

before setting the random backoff timer. Integrating, we find that the total energy consumed is 1.75mJ, with 1.21mJ (69%) consumed in the transmit state.

Consider an environment in which no Beacon management frames or other transmissions occur. (This simplifies the analysis by ignoring contention delays and transitions to the receive state for frames **overheard** by the station.) For the 4Hz CBR workload, the transaction illustrated by Figure 2.4 occurs every 250ms. After the $t = 1.52$ms needed for the transaction have elapsed, the station then spends the next 248.48ms in **idle** waiting for the next application message. During this time, 184mJ are consumed, *even though no frames are being exchanged*. The total energy cost for the entire 250ms period is 185mJ, only 0.7% of which was consumed in the transmit state.

	Idle	**Receive**	**Transmit**
Power	739mW	901mW	1.35W
Time	249ms	496μs	897μs
Energy	184mJ	447μJ	1.21mJ

Table 2.2: Power, time, and energy per 250ms for 4Hz 512-octet CBR workload.

Table 2.2 summarizes the contributions of power and time to energy consumption. Even though the transmit power rate is nearly *twice* that of the idle state, the *energy* consumed by transmit is orders of magnitude *less* than idle. This is due to the dominance of the idle state in **time**. We say that the idle state **dominates** device energy consumption. The goal of **power management** should therefore be to reduce the amount of time *spent* in idle. Designs such as 802.11 power management (§2.1.3) attempt to spend some of that time in a low-power **doze** state when the interface is not actively exchanging data.

Suppose we doubled the work being done every 250ms; this might correspond to running a second copy of the CBR application. To find energy consumption with the new traffic flow added, we replace a $t = 1.52$ms idle interval ($E = 1.12$mJ) with the transaction from Figure 2.4 ($E = 1.75$mJ). Table 2.3 shows that total energy consumption over the 250ms period then rises from 185.38mJ to 186.01mJ, an increase of 0.34%. The Table also shows how energy consumption changes when increasing the number of flows from one to 10, one to 100, and one to 164. The 164-flow value is provided as a limit, but is not practically achievable. It corresponds to the case in which transactions occur back-to-back with no intervening random backoffs.

Table 2.3 provides a critical insight into the energy behavior of 802.11 interfaces.

Flows	**Energy**	**Δ-Energy**
1	185.38mJ	0%
2	186.01mJ	0.34%
10	191.04mJ	3.1%
100	247.62mJ	34%
164	287.86mJ	55%

Table 2.3: Energy consumption per 250ms for additional CBR traffic flows.

It says that the **energy cost** of processing an additional traffic flow (with similar characteristics) is **marginal**. That is, when doubling the number of application packets processed, the total energy consumption does *not* double. In fact, as the Table shows, it is *never* possible to double energy consumption under this power model by adding additional flows. Rather, the additional cost of adding flow $k+1$ to k existing flows is always less than 1%. We will return to the concept of **marginal energy cost** later in the Chapter.

In summary, we have described a point of confusion in some research related to the difference between *power* and *energy*. Using both our own measurements as well as those from the literature, we have described the power behavior of popular 802.11 interfaces. In particular, the **idle** power rate for these interfaces is more than *half* the **transmit** power rate. When we considered the dominant amount of time spent by the interface in the idle state, we showed that idle also dominates **energy consumption**. Finally, we showed that due to idle dominance, the energy cost of incrementally adding similar traffic flows is marginal.

2.1.3 IBSS Power Management

To address the high energy consumption associated with the idle state, 802.11 defines an **IBSS power management mode**. IBSS power management allows stations to place their transceivers in a low-power **doze** state. While dozing, stations can neither transmit nor receive frames, since most of the transceiver electronics are disabled. Periodically, stations **rendezvous** and, if traffic is pending, exchange frames. The rendezvous period is the **beacon interval** described in Section 2.1.1.1.

2.1.3.1 ATIM Management Frames

In a power-managing IBSS, all stations wake from doze mode just before each TBTT. At the TBTT, they execute the beacon generation algorithm, and remain awake for a fixed length of time to exchange **announcement traffic indication message** (ATIM) frames. The time following the TBTT in which ATIM frames may be sent is known as the **ATIM Window**. As an example, the ATIM Window might occupy the first 40 milliseconds of each 200-millisecond beacon interval.

A station transmits an ATIM frame during the ATIM Window if the station has **data frames** ready to transmit. The station *announces* the addresses of every destination for which it will send data during the current beacon interval. A directed ATIM is sent to each station to which unicast data will be transmitted. Directed ATIM frames are acknowledged with an ACK control frame, but are not preceded by an RTS-CTS handshake. Broadcast ATIM frames, like broadcast data frames, are not acknowledged, and are therefore less reliable. Data frames which become available for transmission after the ATIM Window ends may be transmitted if their destination address has been announced in the current interval. Otherwise, such frames are buffered until after the next ATIM Window, in which their destination addresses will be announced. ATIM transmission is depicted in Figure 2.5; control and non-ATIM management frames have been omitted for clarity.

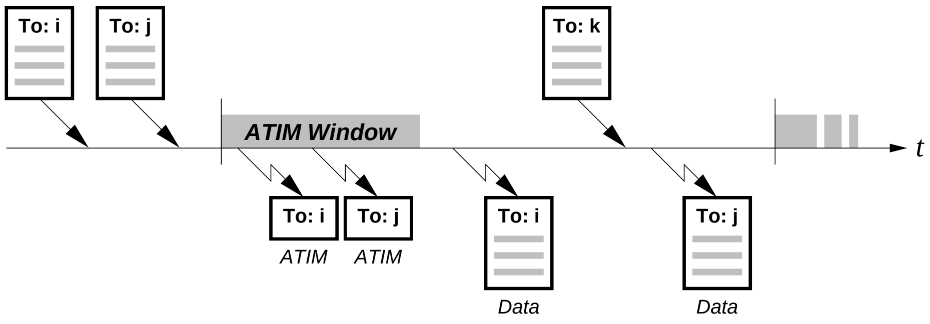


Figure 2.5: 802.11 ATIM frame transmission.

When a station receives an ATIM addressed to itself or the broadcast address, it knows that it should be prepared to receive data messages in the current beacon interval following the ATIM Window. When the ATIM Window ends, the station remains awake for the remainder of the current interval to receive frames. (The station may also send frames to any neighbor it believes to be awake, such as those neighbors which transmitted ATIM frames.) If a station does not receive any ATIM frames, and has no data messages of its own to transmit, it returns to doze mode following the ATIM Window.

2.1.3.2 IBSS Power Management Performance

The energy savings from using IBSS power management depend primarily on the size of the ATIM Window and the offered traffic load. Let $0 < w \leq 1$ be the fraction of the **beacon interval** occupied by the **ATIM Window**. Call P_{doze} and P_{idle} the power levels of the 802.11 doze and idle modes, with $P_{\text{doze}} < P_{\text{idle}}$. The minimum energy E_{min} consumed during a time t in which no **control** or **data** frames are exchanged is:

$$E_{\text{min}} = (w\, P_{\text{idle}} + (1 - w)\, P_{\text{doze}}) \times t + \beta(t)$$

The $\beta(t)$ term captures the small amount of energy consumed in transmitting or receiving **beacon frames** during t. For the purpose of computing a lower bound, let $\beta(t) \rightarrow 0$. Using the **WaveLAN** measurements from Table 2.1 as an example, let $P_{\text{doze}} = 47.4\text{mW}$ and $P_{\text{idle}} = 739\text{mW}$. Finally, let $w = \dfrac{\text{ATIMWindow (40ms)}}{\text{BeaconInterval (200ms)}} = \frac{1}{5}$. The minimum energy consumption rate for a device using IBSS power management is 186mW, a savings of 75%. Of course, a station cannot doze during an interval in which it transmits or receives data, so the **energy savings** decrease as **communications activity** increases. A station which transmits or receives data at least once every w seconds would realize no savings at all.

The cost of these energy savings is **latency**. A data message which becomes available for transmission during an interval in which the message's destination was not announced must be buffered until the end of the next ATIM Window. This means that the worst-case delay for a data message is *at least* the beacon interval, corresponding to the case of a message that becomes available just after the end of an ATIM Window. The average-case delay depends on traffic load; in a lightly-loaded scenario, the expected latency is *half* the beacon interval. Section 2.2.2 shows how these delays accumulate in ad hoc networks of larger diameter.

2.2 Dynamic Source Routing

As mentioned in Section 2.1.1, the 802.11 IBSS is designed for environments in which every station can hear every other station. In practice, such proximity between stations may not be practical. Depending on obstacles to signal propagation, the range of an 802.11 transceiver may be as little as tens of meters. If stations are willing to **relay** traffic for one another, then the communications range can be effectively extended. Stations participating in an **ad hoc routing protocol** become nodes in a **multihop ad hoc network**. This Section describes one such protocol, **Dynamic Source Routing** (Johnson and Maltz, 1996), or DSR.

DSR is an **on-demand**, or **reactive**, protocol. Unlike **proactive** routing protocols, which constantly exchange local or global topology information, on-demand protocols only do the work needed to service active **traffic flows**. This philosophy makes on-demand protocols a good fit for power-managing networks, since there is no fixed energy overhead associated with the periodic exchange of topology information. A power-managing node in a completely unladen DSR network would have communications energy consumption E_{min} (§2.1.3.2).

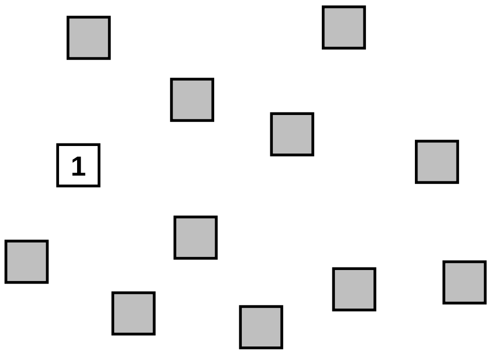
2.2.1 Source Routes, Route Discovery, and Route Maintenance

Traditional proactive algorithms such as **distance vector** or **link state** routing maintain **routing tables** at each node. When a message arrives at a routing node, the routing table is consulted to select the **next hop** for the message, given its **destination**. For such a scheme to work, each routing node needs to have an up-to-date view of the complete network topology.

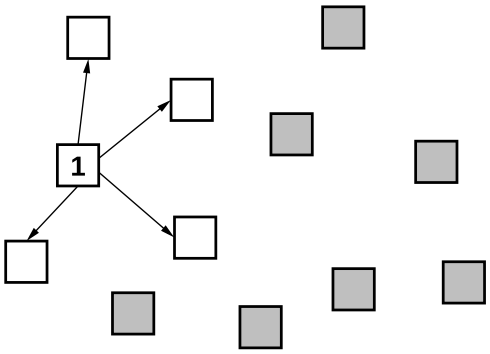
In a **source routing network**, only **traffic sources** need to know about the subtopology containing routes to their active **traffic sinks**. Once a source knows a route to a sink, it appends that **source route** to every message for that sink. To deliver a message, a relay simply examines the source route, and transmits the message to the next node listed in the route. DSR does not have distinguished routing nodes; all nodes can source, relay, or sink traffic flows at any time.

As shown in Figure 2.6(a), a DSR traffic source first needs to identify a set of one or more routes to the sinks of its active traffic flows. This **Route Discovery** process consists of **flooding** the network with **Route Request** messages asking for routes to a named destination. Route Requests are broadcast (Figure 2.6(b)); each node (that is not the named destination) receiving a Route Request rebroadcasts it if the particular Request has not been seen before. As each node rebroadcasts the Request, it appends its own address to the Request. Nodes which know how to reach the destination — either because they *are* the destination (Figure 2.6(c)), or have a **cached route** to it — return a **Route Reply** message to the source. Route

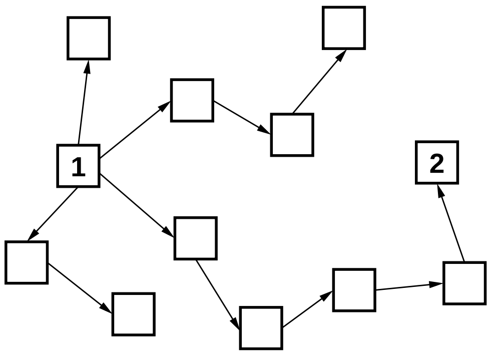
Replies are sent along the route which was formed as the Route Request was propagated (Figure 2.6(d)). Nodes store the routes they learn through this process in a **route cache**.



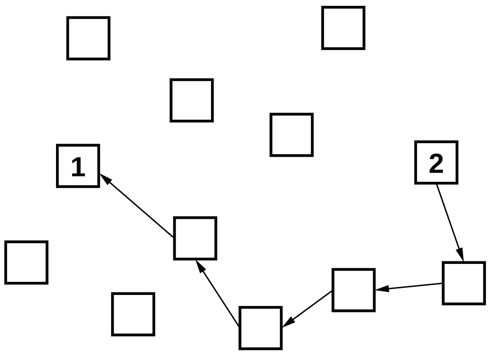
(a) 1 wants to discover 1 ⇝ 2.



(b) 1 broadcasts a Route Request.



(c) Route Request propagates to 2.



(d) Returning Route Reply to 1.

Figure 2.6: DSR Route Discovery.

Mobility is a problem for a source routing network, since **links** between nodes may break. DSR provides a **Route Maintenance** facility to inform sources of link breakage, and to **salvage** messages with broken source routes. A relay attempting to forward a message across a broken link will be notified by the MAC layer that the next hop is not available. This is an example of DSR exploiting the **link-layer**

acknowledgments described in Section 2.1. The relay then sends a **Route Error** message back to the source describing the broken link. The source, and any other nodes which observe the Route Error, then remove all routes containing the broken link from their cache. If the source has no more routes to a traffic sink after this process, the source may repeat Route Discovery. Relays encountering a broken link may attempt to salvage a message by checking their own route caches for routes to the message's destination, and forwarding the message along a new source route. It is common, however, that a message encountering a broken link will simply be **dropped**.

2.2.2 DSR and IBSS Power Management

When DSR is used with a non-power-managing 802.11 IBSS, the communications energy consumption of a node is dominated by the **idle** power rate. This is because nodes spend most of their time in idle **waiting** for communication to occur. Nodes can use IBSS power management to reduce their energy consumption, but this comes at a latency cost as described in Section 2.1.3.2. In fact, the latency costs of using IBSS power management are *worse* in a multihop network than in the single-hop environment for which 802.11 was designed.

DSR Route Discovery relies on the propagating broadcast of Route Request messages. A power-managing 802.11 station should not attempt to send data frames to an address which might not be awake, and this includes the broadcast address. Therefore, as broadcast Route Requests reach each new hop, they must wait until that station announces the broadcast address in the next ATIM Window. Figure 2.7 illustrates the problem with a timeline, not drawn to scale. Arrows indicate frame transmission, and are labeled with the message type and destination. For example, "*RREQ: **" is a DSR Route Request destined to the broadcast address, and "*RREP: 2*" is a Route Reply destined for address 2. For clarity, control and non-ATIM management frames are omitted.

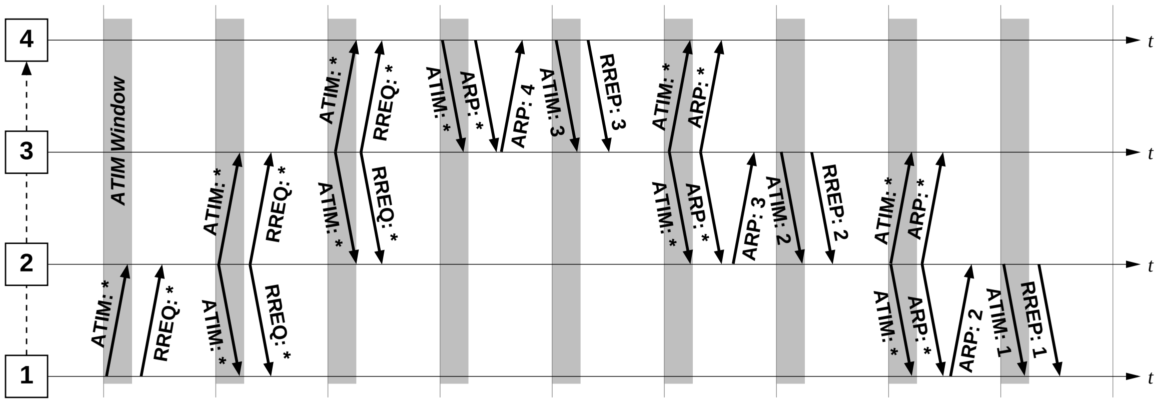


Figure 2.7: Route Discovery using IBSS power management.

The example of Figure 2.7 illustrates the worst-case scenario. Several stations are arranged in a line such that each can only communicate with its immediate neighbors. Initially, all route and **Address Resolution Protocol** (Plummer, 1982), or ARP, caches are empty, when DSR attempts to discover the route $1 \rightsquigarrow 4$. DSR names nodes using IP addresses, so an IP-to-MAC address **translation** must be available in order to send unicast messages to any DSR node; ARP provides the translation service. Because Route Replies are sent via unicast, nodes may have to perform an **ARP query** prior to relaying a Reply. IBSS power management treats receipt of a management frame, such as an ATIM, as evidence that the sender is awake; this is why the unicast ARP replies are transmitted immediately. The **worst-case delay** for DSR to discover a route P with $|P| - 1$ hops is $3\,(|P| - 1) \times$ BeaconInterval. At each hop, one beacon interval delay is incurred while waiting to propagate the Route Request, and two more delays are added on the return trip for the ARP exchange and transmission of the Route Reply. In this example, with a 200-millisecond beacon interval, a Route Discovery process that should complete in tens of milliseconds instead requires at least 1.8 *seconds*.

A similar latency problem affects **application message** delivery. Once a source route has been discovered, application messages are relayed along the route using a unicast transmission at each hop. The worst-case application message delivery latency is $(|P| - 1) \times$ BeaconInterval. If more than one message is generated per beacon interval, the average latency can be better than the worst case since multiple messages can be sent in an interval following a single traffic announcement.

The final performance issue to discuss is **fairness** of **energy distribution**. In a source routing network such as DSR, traffic sources determine which routes will be used. The logic used by sources in choosing a route is **private information**, meaning that it is opaque to any agent in the network other than the source. In practice, this logic simply chooses from among the **shortest routes** known, but in principle the logic could be arbitrary. We can say that sources have a **preference** for shorter routes because of their lower latency or reduced probability of breakage.

The problem is that the relays and sinks considered by a source have preferences as well. An energy-sensitive agent prefers not to be used for service too often, because this reduces the energy savings gained through power management. Other examples of agent preferences include sensitivities to **congestion** and **throughput**. DSR provides no way to incorporate relay and sink preferences into the choice of routes.

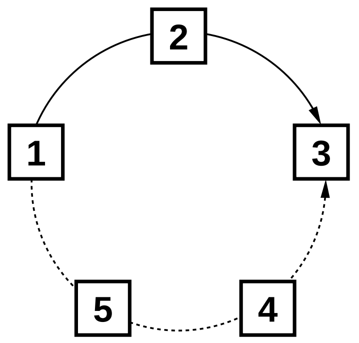


Figure 2.8: Choosing a source route.

In the example of Figure 2.8, source 1 knows the routes (1, 2, 3) and (1, 5, 4, 3). Given its preference for short routes, the source will choose (1, 2, 3) for every message to 3 until that route breaks. In a power-managing scenario, however, relay 2 will experience high energy consumption as it is continually prevented from entering the doze state. Nodes 4 and 5, on the other hand, will have low energy consumption approaching E_{min}.

A more sophisticated route selection method would take the energy concerns of relay 2 into account. Eventually, 2 might become so unhappy about providing relay service that it makes more sense to reconfigure the system and use the longer route through 4 and 5 instead. For this we need a method of **aggregating** node preferences in a way that balances **self interest** and **social utility**. Section 2.3 describes such a method which bears a remarkable similarity to the process DSR already uses.

2.3 Combinatorial Auctions and Exchanges

A central insight of this thesis is that DSR implements a **reverse auction**. A reverse auction consists of a **buyer** who wants to obtain some **items** at minimum cost, and a set of **sellers** who provide the items. A familiar example is **government procurement**, in which contracts are won by the "lowest bidder." In DSR, the traffic source is the buyer, and the relays and sinks are sellers. The "item" being sold is an individual node's **service** of forwarding or receiving messages.

In this Chapter, it is easiest to think of the auction as being held once for each message sent by a traffic source. An item may be interpreted as the service of a relay or sink to process an **individual** message. Chapter 4 introduces a different concept for items, in which they represent a commitment by relays and sinks to be available over a **time interval** to service an arbitrary number of messages.

Specifically, DSR is a **combinatorial reverse auction** (Sandholm et al., 2002). The auction is *combinatorial* because only certain *combinations* of items are valuable. For example, any combination of items that does not include the sink is valueless. We can think of a route as being a **bundle** consisting of the relays and the sink on that route. **Complementarity**, or **synergy**, exists between the items in a bundle.

Stated differently, the value of the items in the bundle is **superadditive**, meaning the value of the bundle is at least as much as the sum of the values of the constituent items.

The auction forms described in this Section, and in the remainder of this thesis, are all **one-shot auctions**. This means that agents independently submit **sealed bids**, which are (conceptually) only visible to the auctioneer. The auctioneer determines the winner or winners based only on those bids, and declares the results once and for all. There is no iterative bidding process, such as in an **ascending-price auction**. The one-shot interpretation is justifiable because DSR does not implement any sort of **sequential logic** in choosing routes.

2.3.1 Combinatorial Reverse Auctions

Conceptually, we can think of the DSR auction as happening in one of two equivalent ways. The first way is that relays and sinks **individually** offer to sell their service to a buyer. The buyer then specifies which **combinations** of items it wants, and selects the most affordable bundle. The other way is for the buyer to specify that it wants any bundle which includes the sink. A **meta-agent** then offers bundles which each contain relays and the sink, and the buyer selects the most affordable bundle. Both of these formulations are presented below.

2.3.1.1 Bundled-Demand Combinatorial Reverse Auctions

For the first case, the **auctioneer** (buyer) knows a set of items $M = \{1, 2, \ldots, m\}$ which correspond to a sink and relays found through Route Discovery. The buyer specifies the set of bundles it wants $\mathcal{U} = \{U_1, U_2, \ldots, U_l\}$, where each bundle $U_k = \{u_k^1, u_k^2, \ldots, u_k^v\}$ contains items $u_k \in M$ representing a sink and zero or more relays on a route to that sink. Sellers implicitly submit **asks** describing what they want to sell: $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$, $A_j \subseteq M$ and $|A_j| = 1$ (each agent can only sell its own service). Asks in this environment are **unit-price**. The auctioneer labels the asks A_j *winning* ($x_j = 1$) or *losing* ($x_j = 0$) so as to **minimize cost** under the constraint that the buyer receives *all* of the items in *any one* of its bundles:

$$\min_{U_k \in \mathcal{U}} \quad \sum_{j=1}^{n} x_j \quad \text{s.t.} \quad \sum_{j \mid A_j = \{i\}} x_j = 1, \quad i = u_k^1, u_k^2, \ldots, u_k^v, \quad x_j \in \{0, 1\}$$

Because the asks are all unit-price, and each ask only offers one item, this is the same as picking the **shortest route** from among several known routes — exactly what DSR does. The constraint guarantees that all of the relays and the sink in a route will be labeled winning. The fact that only the most affordable bundle (route) is selected is an example of **substitutability** between the bundles. The value of many bundles is never more than the value of a single bundle. This **subadditivity** reflects the fact that a traffic source only needs one route to a sink at a time.

2.3.1.2 Bundled-Supply Combinatorial Reverse Auctions

The second interpretation of DSR as a reverse auction uses a form more similar to that in (Sandholm et al., 2002). Again, the items $M = \{1, 2, \ldots, m\}$ are the relays and a sink. The buyer only specifies that it demands a bundle containing the sink $u \in M$. The relays and sink do not sell directly, but rather, **meta-agents** submit asks $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ which combine the services of a bundle of nodes. An ask $A_j = \langle(\lambda_j^1, \lambda_j^2, \ldots, \lambda_j^m), p_j\rangle$ contains a vector of offered items, where $\lambda_j^k = 1$ if k is on the route, and $\lambda_j^k = 0$ otherwise. Since all routes include the sink u, $\lambda_j^u \equiv 1$. Asks include a price $p_j = \sum_{k=1}^{m} \lambda_j^k$ equal to the length of the route. The auctioneer then labels the asks A_j *winning* ($x_j = 1$) or *losing* ($x_j = 0$) so as to **minimize cost** under the constraint that the buyer receives the item it demanded (namely, the sink):

$$\min \quad \sum_{j=1}^{n} p_j x_j \quad \text{s.t.} \quad \sum_{j=1}^{n} \lambda_j^u x_j = 1, \quad x_j \in \{0, 1\}$$

Since all bundles include the sink, the constraint says that the buyer receives exactly one bundle representing a route to the sink. This is a case of **no free disposal** with respect to the sink item, u, since the buyer is not willing to take additional units of u. Given the composition of the supply bundles, this is the same as not taking additional routes to u. The buyer may take additional *items*, however, since the supply bundles generally contain a number of relay items in addition to u. Asks are no longer strictly **unit-price**, but since the ask price is equal to the route length, cost minimization again selects the **shortest route**.

2.3.2 Combinatorial Exchanges

Section 2.3.1 presented formulations of DSR source route selection as a reverse auction. The forms implemented by DSR are restricted versions of the combinatorial reverse auction, which is itself a special case of the **combinatorial exchange**. From an economic standpoint, the restricted auction of DSR has several shortcomings:

1. **Inexpressive pricing for sellers.** In both reverse auction forms, relays and sinks only contribute one unit each to the cost of a route. However, if ask prices were **general** ($p_j \in \mathbb{R}$), sellers could indicate their individual **willingness** to provide service. For example, a relay with a preference against service could submit an ask with a large-magnitude price, decreasing the likelihood that it will be part of the cost-minimizing route. Such a preference could be based on **energy goals**, buffer and computational constraints, network **congestion**, or other factors.

2. **No unaffordable routes.** The definition of the reverse auction is that the buyer gets *everything* it wants. In the DSR interpretation, a source always gets to use one of the routes it knows to a sink. If relays and sinks have a strong preference against providing service, a designer may want to admit the possibility of an **unaffordable** route. That is, sources could indicate how much value they place on individual bundles of items, and if this value does

not exceed the cost, the bundle (route) is unavailable for use.

3. **Separate negotiations for each source-sink pair.** Each traffic source conducts a reverse auction for each of its active traffic sinks. When routes overlap, sellers negotiate independently with the separate buyers. This ignores the possibility for **complementarity** between traffic flows. For example, a relay could offer a discounted price if it were chosen to relay **multiple** concurrent flows. Such discounts reflect the **marginal cost of energy** for relaying additional flows (§2.1.2), and provide an incentive for **consolidating** the number of active relays.

We can address all of these concerns by extending the DSR route selection method to implement a **combinatorial exchange**. In an exchange, sellers still submit **asks**, but buyers also submit **bids** describing what they want to buy and what value they place on different combinations of items. Multiple buyers and sellers can participate in the same exchange, and sellers can offer multiple **units** of their service, one for each traffic flow they offer to service. Note that multiple buyers will participate in a single exchange only when the **intersection** of their demanded sets of relays and sinks is non-empty.

As with the reverse auction, the items $M = \{1, 2, \ldots, m\}$ are the services of the relays and sinks. (A source can also serve as a relay or sink.) Agents submit bids $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$, where a bid $B_j = \langle(\lambda_j^1, \lambda_j^2, \ldots, \lambda_j^m), p_j\rangle$ includes a vector of item **quantities** $\lambda_j \in \Lambda : \lambda_j^i \in \mathbb{Z}$, and a scalar **price** for the whole bundle $p_j \in \mathbb{R}$. For each item k, the vector indicates the number of **units** an agent is willing to buy ($\lambda_j^k > 0$) or sell ($\lambda_j^k < 0$). In the most general case, a bid can buy some items and sell others. Due to the structure of this particular problem, bids either only buy, or only sell items. Where the distinction is important, we will refer to B_j with all $\lambda_j^k \geq 0$ and $p_j \geq 0$ as "**bids**," and B_j with all $\lambda_j^k \leq 0$ and $p_j < 0$ as "**asks**." Traffic sources submit a bid for each route they want to use, while relays or sinks submit asks indicating the number of flows for which they are willing to provide service. The auctioneer labels the bids B_j *winning* ($x_j = 1$) or *losing* ($x_j = 0$) so as to **maximize surplus**, subject to a feasibility constraint:

$$\max \quad \sum_{j=1}^{n} p_j x_j \quad \text{s.t.} \quad \sum_{j=1}^{n} \lambda_j^i x_j \leq 0, \quad i = 1, 2, \ldots, m, \quad x_j \in \{0,1\}$$

The feasibility constraint simply says that agents cannot buy more units of an item than are available for sale. The result of the surplus maximization is an **allocation** $\lambda^* \in \Lambda$ that assigns items to agents as specified by the winning bids.

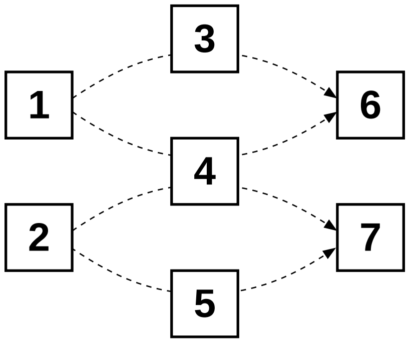


Figure 2.9: An exchange example.

The example of Figure 2.9 shows two traffic sources with overlapping demand sets. There are two traffic flows: $1 \rightsquigarrow 6$ and $2 \rightsquigarrow 7$. Node 1 knows the routes (1, 3, 6) and (1, 4, 6), while node 2 knows the routes (2, 4, 7) and (2, 5, 7). The items are $M = \{3, 4, 5, 6, 7\}$. Suppose the bids are as follows:

$B_1 = \langle(\lambda_1^3 : 1,\ \lambda_1^6 : 1),\ 3\rangle$	1′s bid for (1, 3, 6)
$B_2 = \langle(\lambda_2^4 : 1,\ \lambda_2^6 : 1),\ 3\rangle$	1′s bid for (1, 4, 6)
$B_3 = \langle(\lambda_3^4 : 1,\ \lambda_3^7 : 1),\ 2\rangle$	2′s bid for (2, 4, 7)
$B_4 = \langle(\lambda_4^5 : 1,\ \lambda_4^7 : 1),\ 2\rangle$	2′s bid for (2, 5, 7)
$B_5 = \langle(\lambda_5^3 : -1),\ -0.9\rangle$	3′s ask offering one unit of relay service
$B_6 = \langle(\lambda_6^5 : -1),\ -1\rangle$	5′s ask offering one unit of relay service
$B_7 = \langle(\lambda_7^6 : -1),\ -1\rangle$	6′s ask offering one unit of sink service
$B_8 = \langle(\lambda_8^7 : -1),\ -1.1\rangle$	7′s ask offering one unit of sink service
$B_9 = \langle(\lambda_9^4 : -1),\ -1\rangle$	4′s ask offering one unit of relay service
$B_{10} = \langle(\lambda_{10}^4 : -2),\ -1.5\rangle$	4′s ask offering *two* units of relay service

Node 1 is indifferent between the two possible routes represented by B_1 and B_2, and values them equally at price 3. Similarly, node 2 does not care which of B_3 or B_4 wins. Note that node 4 has submitted two asks: B_9 offers a price of -1 to relay one flow, and B_{10} offers a price of -1.5 to relay both flows. The surplus-maximizing solution assigns B_2, B_3, B_7, B_8, and B_{10} winning, with a surplus of $\sum_{j=1}^{10} p_j x_j = 1.4$.

One important way in which the exchange differs from the reverse auction has to do with **clearing**: a winning combination of bids *must* satisfy $\sum_{j=1}^{n} p_j x_j \geq 0$. In fact, no combination of winning bids can *ever* have a negative surplus, since the auctioneer could do better by assigning all bids *losing* ($x_j = 0$). In the reverse auction, one or more asks will *always* win; the only question is which combination minimizes cost.

In the current example, suppose node 1 had not submitted any bids. Out of the remaining bids $\{B_3, B_4, B_5, B_6, B_7, B_8, B_9, B_{10}\}$, no combination results in a positive surplus. Therefore, all bids are assigned losing. It is up to the system designer to determine what should happen when a source does not obtain any of its requested items. We say that such a source cannot **afford** any of its routes. One possibility is that the source is prevented from sending messages to destinations for which it has only **unaffordable** routes. Chapter 4 describes a design which uses this approach.

A consequence of the clearing problem is that buyers must be careful about the bid prices they submit. A bid price which is too low given the corresponding asks for the demanded items runs the risk of not clearing. A bid price which is too high may cause problems as well. In Section 2.3.1, we used **price** as a scalar measure of a relay or sink's **preferences** about providing service. In the exchange environment, we extend price to become a means of **accounting** for the work done by nodes in the network. When a relay or sink has a winning ask and provides service, it receives a **payment** from the exchange. The payment reflects the loss in **value** experienced by the node when it provides service to the network. For an energy-conserving node, the loss of value might be the loss of **energy**. For a memory- or compute-limited node, it might mean the loss of buffer space or processor cycles. It could even reflect the cost to **user attention** — at a sink, for example — when new messages arrive which could divert the user from his present task (Kraut et al., 2002). We can think of this payment as being made in an arbitrary-unit **credit**, which is then **spent** when that node is a source and has winning bids. Payments by winning sources reflect the value of the message delivery service — in energy or some other sense — that they obtain from the relays and sink.

If there are restrictions on whether, or how much, a node can spend in **deficit**, then each node has a motivation to accrue credit. For example, if a node cannot bid more than its current credit balance, then a node with a low balance may not be able to afford its routes. At the same time, a node with a sufficient balance does not want to bid more than it has to, since paying for large bids may substantially diminish that balance. Section 2.4 discusses **strategic manipulation** of bid prices by buyers trying to pay *less* for their routes, or by sellers trying to receive *more* payment for their service.

The actual **implementation** of credit is beyond the scope of this thesis. We are concerned with methods for **configuring** *ad hoc* networks given **preferences** over energy consumption. Credit is a means of **enforcing** an agreed-upon configuration. In practice, some sort of cryptographically secure instrument such as **digital coins** (Ferguson, 1993; Ferguson, 1994) might be used to implement payments. Chapter 3 describes existing work in the area of credit schemes for *ad hoc* networks which might also be useful.

2.3.3 XOR Constraints

Section 2.3.1.1 introduced the idea of **substitutability** between bundles. In the DSR environment, substitutability exists between routes, since a source only needs one route at a time. Additionally, in the combinatorial exchange environment, a seller offering multiple units of its service has substitutability between its own asks. In the example of Figure 2.9, node 4 wants either its ask for one flow (B_9) or its ask for two flows (B_{10}) to succeed, but not *both*.

Substitutability between bids can be expressed through the use of **XOR constraints**. Individual bids can be joined by exclusive-or ($\oplus$), indicating *at most one* of the bids in the $\oplus$-term should win. The $\oplus$-terms are joined by inclusive-or ($+$), indicating that *any* combination of $\oplus$-winners is acceptable. This **OR-of-XORs bidding language** (Sandholm, 2002) allows the agents to be fully expressive with their bids while keeping the bid representation compact. Using this language, the example of Figure 2.9 would be expressed as:

$$(B_1 \oplus B_2) + (B_3 \oplus B_4) + B_5 + B_6 + B_7 + B_8 + (B_9 \oplus B_{10})$$

2.4 Groves Mechanisms

The combinatorial exchange contains two basic problems: **winner determination** and **payment**. Winner determination solves the revenue maximization problem of Section 2.3.2 and identifies the winning bids. Payment specifies how much each agent pays to the exchange when its bids win. This latter problem is especially interesting if the agents in the exchange can **strategically manipulate** their bid prices in order to improve their net **utility**. For example, a buyer might artificially lower its bid price in order to pay less, or a seller might artificially raise its ask price in order to be paid more. These behaviors are consistent with an agent trying to build a positive **credit** balance in order to be able to afford routes.

Mechanism design is an area of **game theory** that studies how to solve optimization problems involving agents whose **preferences** over outcomes are **private**. Section 2.3.2 presented methods for aggregating agent **bids** to select a configuration of the ad hoc network. Bids can be thought of as **reports** of each agent's **preferences** over the final outcome. In this environment, the actual preferences themselves are **private information** known only to each agent, and not publicly observable. The exchange takes the *reports* as its inputs, not the private preferences directly. As such, agents can report preferences which are different from their *actual* preferences, as in the case of a buyer submitting a low bid price.

The mechanism design problem is to elicit **truthful** reports from the agents. This Section describes the **Groves mechanisms**, which improve the combinatorial exchange **payment rule** to cause the agents to report truthfully out of their own **self interest**. A thorough introduction to mechanism design and its application to exchanges is available in (Parkes et al., 2001), Chapter 2 of (Parkes, 2001), and Chapter 23 of (Mas-Collel et al., 1995), from which much of this Section is taken.

2.4.1 Social Choice Functions

In the exchange interpretation of *ad hoc* networks, preferences are the **value** a traffic source associates with a route, or the **loss** in value a relay or sink associates with providing service. Each agent $i = 1, 2, \ldots, I$ (including buyers and sellers) has a **type** $\theta_i \in \Theta_i$ observable *only* by agent i. For an agent in the *ad hoc* network environment, θ_i might include its knowledge of its own traffic sourcing behavior, its history of relay service to the network, its **model** for how it consumes energy while active, or its **projections** about future energy and communications needs.

It is difficult to talk about agent types in the ad hoc network environment without reference to the **past** or the **future**. For example, saying that an agent's type is "high-volume traffic source" implies that the agent *has* sourced, or *will* source many messages. This implies **interdependence** between runs of a route-selecting combinatorial exchange: an agent's history of winning and losing bids can affect how much traffic it has offered. We are careful to assume the restrictions of a **static game**, in which agents **observe** their types independently in each run of the exchange, but do not reason about how they **arrived** at those types. A key advantage of this assumption is **computational efficiency**; agents do not have to model past or future bidding behavior when deciding what they want in the **current** exchange. A **dynamic game** interpretation of the route selection problem, in which agents learn about each other over time, is left to future work.

Given its own type, the agent can assign a scalar **value** to every possible **allocation** of items using a **valuation function** $v_i : \Lambda \times \Theta_i \rightarrow \mathbb{R}$. The values assigned to different allocations $\lambda \in \Lambda$ by $v_i(\lambda, \theta_i)$ form a **preference ordering** over the allocations. An example of valuation might be a candidate relay's level of **happiness** with the allocation in which it is asked to provide service, given its history of service to the network. Another might be a source's relative happiness with the allocation in which it receives the route P. We do not specify a valuation function here, but do assume that each agent *has* such a function; Chapter 4 provides some concrete valuation examples.

Define a **social choice function** $f : \Theta_1 \times \Theta_2 \times \cdots \times \Theta_I \rightarrow \Lambda$ which selects an allocation $\lambda \in \Lambda$ based on the agents' types $(\theta_1, \theta_2, \ldots, \theta_I)$. For a combinatorial exchange, $f(\cdot)$ selects the surplus-maximizing feasible allocation of items λ^* described in Section 2.3.2. In some environments, the agents' valuation functions are taken to be **common knowledge**, so all that is needed to solve $f(\cdot)$ is the vector of agent type values. With allocation-based choice functions such as auctions and exchanges, it is common to view $f(\cdot)$ as operating on the agent valuation functions themselves: $f(v_1(\cdot, \theta_1), v_2(\cdot, \theta_2), \ldots, v_I(\cdot, \theta_I)) = \lambda^*$.

The allocation λ^* chosen by the combinatorial exchange using the valuation functions is **allocatively efficient**:

$$\forall \lambda \in \Lambda \quad | \quad \sum_{i=1}^{I} v_i(\lambda^*, \theta_i) \geq \sum_{i=1}^{I} v_i(\lambda, \theta_i)$$

Allocative efficiency is a desirable property for a social choice function to have; it says that there is no allocation λ that all agents weakly prefer to λ^*, and that some agent strictly prefers to λ^*:

$$\nexists \lambda \in \Lambda \quad | \quad \forall i \,.\, v_i(\lambda, \theta_i) \geq v_i(\lambda^*, \theta_i) \quad \wedge \quad \exists i \,.\, v_i(\lambda, \theta_i) > v_i(\lambda^*, \theta_i), \quad \lambda \neq \lambda^*$$

This property, also known as **Pareto optimality**, means no agent can be made happier without making at least one other agent less happy.

The difficulty in implementing a social choice function lies in the fact that $f(\cdot)$ cannot operate *directly* on the valuations $v_i(\cdot, \theta_i)$. Agent types (and valuation functions) are not publicly observable, so agents must submit **reports** about their valuations, which may be different from their *actual* valuations. Let $\hat{v}_i(\lambda, \theta_i)$ be the valuation agent i reports that it associates with allocation λ. In the exchange environment, valuation reports are expressed as **bid prices**. Specifically, bids are **samples** of $\hat{v}_i(\lambda, \theta_i)$ for the allocations λ agent i cares about: for bid B_j submitted by agent i, $p_j = \hat{v}_i(\lambda_j, \theta_i)$. An agent's reported valuation for the **surplus-maximizing allocation** λ^* solved by the exchange is simply the sum of its winning bid prices.

As explained earlier, agents have a specific motivation to report less value, or more *loss* in value, in trying to accrue credit. When $\hat{v}_i(\lambda, \theta_i) \neq v_i(\lambda, \theta_i)$, it may be the case that $f(\hat{v}_1(\cdot, \theta_1), \hat{v}_2(\cdot, \theta_2), \ldots, \hat{v}_I(\cdot, \theta_I))$ does not select the allocatively-efficient choice of $f(v_1(\cdot, \theta_1), v_2(\cdot, \theta_2), \ldots, v_I(\cdot, \theta_I))$. We say that the agents' **self interest** conflicts with the goal of the system designer to **implement** $f(\cdot)$. This self interest follows from the common assumptions that the agents are **utility maximizers**, and that the agents have **quasilinear preferences**. The quasilinear utility an agent associates with allocation λ is its **valuation** for λ less its **payment** to the exchange when that allocation is realized:

$$u_i(\lambda, \theta_i) = v_i(\lambda, \theta_i) - p_i(\lambda)$$

Quasi-linear utility means that agent i is willing to pay any amount up to its true valuation $v_i(\lambda, \theta_i)$ in order to obtain the allocation λ. The formulation of utility as a combination of *value* and *payment* leads to a corresponding formulation of the social choice function. We adopt the shorthand $\theta = (v_1(\cdot, \theta_1), v_2(\cdot, \theta_2), \ldots, v_I(\cdot, \theta_I))$ to represent the input to the social choice function. Let $\lambda(\theta) \in \Lambda$ be the **choice** of allocation selected by the function, such as the surplus-maximizing allocation $\lambda(\theta) = \lambda^*$ selected by the combinatorial exchange. Call $p_i(\lambda(\theta)) \in \mathbb{R}$ agent i's **payment** when the allocation $\lambda(\theta)$ is chosen. The **output** of the social choice function

$f(\cdot)$ then becomes:

$$f(\theta) = (\lambda(\theta),\, p_1(\lambda(\theta)),\, p_2(\lambda(\theta)),\, \ldots,\, p_I(\lambda(\theta)))$$

Since the choice rule $\lambda(\cdot)$ and payment rules $p_i(\cdot)$ are common knowledge, a **utility-maximizing** agent can **manipulate** its input to those rules to (possibly) **improve** its payment. Section 2.4.2 describes how an agent i can use **strategy** in choosing its input $\hat{v}_i(\lambda,\, \theta_i)$ so as to maximize its individual utility. We then show how to construct payment rules that remove the motivation for strategic manipulation.

2.4.2 Strategic Bidding

A classic example of strategic bidding illustrates the problem of self interest in the exchange environment. In the **first price sealed-bid auction**, buyers simultaneously submit bids for an item. Recall that auctions are a special case of exchanges. Formulated as an exchange, there are buyers $i = 1,\, 2,\, \ldots,\, I$, and a seller, "agent 0," who derives no value from the item. In the first-price auction, the choice rule $\lambda(\theta)$ assigns the item to the agent which reports that it values it the most. The payment rule $p_i(\lambda(\theta))$ causes the winning buyer to pay the amount of its reported valuation. In other words, the winner pays its **bid price**.

Suppose there are two buyers, 1 and 2, and that the bids are as follows:

$$
\begin{array}{ll}
B_0 = \langle(\lambda_0 : -1),\, 0\rangle & \text{Seller's ask.} \\[2ex]
B_1 = \langle(\lambda_1 : 1),\, \hat{v}_1(\lambda_1,\, \theta_1)\rangle & \text{Buyer 1's bid.} \\[2ex]
B_2 = \langle(\lambda_2 : 1),\, \hat{v}_2(\lambda_2,\, \theta_2)\rangle & \text{Buyer 2's bid.}
\end{array}
$$

We assume that any allocation for which an agent does not specify a valuation is **irrelevant** to that agent. For example, since buyer 1 submits no bids concerning the allocation λ_2 in which the item is assigned to buyer 2, $v_1(\lambda_2,\, \theta_1) = 0$. The choice rule is completely specified as:

$$
\lambda(\theta) = \begin{cases} (\lambda_0,\, \lambda_1) & \text{if } \hat{v}_1(\lambda_1,\, \theta_1) \geq \hat{v}_2(\lambda_2,\, \theta_2) \\[2ex] (\lambda_0,\, \lambda_2) & \text{otherwise} \end{cases}
$$

The allocation $(\lambda_0,\, \lambda_1)$, for example, assigns buyer 1's bid winning (along with the seller's ask). This λ is chosen when 1's valuation for the outcome in which 1 receives the item is at least as high as 2's valuation for the outcome in which 2 receives the item. Both outcomes satisfy **feasibility**: $\sum_{j=\{0,1\}} \lambda_j = 0$ and $\sum_{j=\{0,2\}} \lambda_j = 0$. The choice of outcome is based on **surplus maximization**; the allocation $(\lambda_0,\, \lambda_1)$ wins when $\sum_{j=\{0,1\}} p_j \geq \sum_{j=\{0,2\}} p_j$. Every outcome beyond these two is either infeasible (more of the item is demanded than supplied) or yields less surplus (*e.g.*, assigning only B_0 winning, with surplus 0). The accompanying payment rule for the buyers is:

$$p_1((\lambda_0,\ \lambda_1)) = \hat{v}_1(\lambda_1,\ \theta_1) \qquad p_1((\lambda_0,\ \lambda_2)) = 0$$

$$p_2((\lambda_0,\ \lambda_1)) = 0 \qquad p_2((\lambda_0,\ \lambda_2)) = \hat{v}_2(\lambda_2,\ \theta_2)$$

The payment rules for the buyers simply say that a winning buyer pays its bid price, while a losing buyer pays nothing. Using the quasilinear utility assumption, we can enumerate the buyers' net utilities for the possible outcomes:

	$\lambda(\theta) = (\lambda_0,\ \lambda_1)$	$\lambda(\theta) = (\lambda_0,\ \lambda_2)$
$u_1(\lambda(\theta), \theta_1)$	$v_1(\lambda_1,\ \theta_1) - \hat{v}_1(\lambda_1,\ \theta_1)$	0
$u_2(\lambda(\theta), \theta_2)$	0	$v_2(\lambda_2,\ \theta_2) - \hat{v}_2(\lambda_2,\ \theta_2)$

When a buyer wins, its utility is determined by its reported valuation. A lower reported value yields higher net utility. Lowering reported value also reduces the **likelihood** that a buyer will win the auction. Let the shorthand u_i, v_i, and $\hat{v}_i$ represent $u_i(\lambda(\theta),\ \theta_i)$, $v_i(\lambda(\theta),\ \theta_i)$, and $\hat{v}_i(\lambda(\theta),\ \theta_i)$, respectively. We can summarize the utility possibilities for buyer 1 as follows:

$$u_1 \begin{cases} < 0 & \text{if } \hat{v}_1 > v_1,\ \hat{v}_1 \geq \hat{v}_2 \quad \text{(1 bids more than its true valuation, and wins)} \\ \geq 0 & \text{if } \hat{v}_1 \leq v_1,\ \hat{v}_1 \geq \hat{v}_2 \quad \text{(1 bids \textit{at most} its true valuation, and wins)} \\ = 0 & \text{if } \hat{v}_1 < \hat{v}_2 \quad \text{(1 loses)} \end{cases}$$

A symmetric analysis holds for buyer 2's utility. Clearly, neither buyer should ever bid more than its true valuation, since a winning bid will always result in negative utility. However, the buyers *should* try to increase their utility by bidding $\hat{v}_i < v_i$. The problem a buyer must solve is to choose a $\hat{v}_i$ that is not so low that its bid loses.

When reporting its valuation for different allocations, a utility-maximizing agent will use a **strategy** $\sigma_i : \Lambda \times \Theta_i \rightarrow \Sigma_i$. In **game theory**, strategy is the basic manifestation of **agent choice**. The strategies Σ_i are all of the possible **actions** agent i might take in reporting a valuation, given its *actual* preferences. One strategy might be, "bid exactly the true valuation," $\sigma_i(v_i) = v_i$. Or, in the earlier example, if buyer 1 knows a **probability distribution** $\phi(v_2)$ for the likely valuations of buyer 2, a strategy might try to maximize **expected** utility: $\sigma_1(v_1) = \min\left(v_1,\ E_{\phi(v_2)}\left[v_2\right]\right)$. If buyer 2 simultaneously uses an expected-utility-maximizing strategy to choose $\hat{v}_2$, then the resulting reports form a **Bayesian Nash equilibrium**. In a Bayesian Nash equilibrium, each agent uses a strategy which maximizes its *expected* utility when the *other* agents use *their* expected-utility-maximizing strategies.

The **equilibrium** is a **solution concept** for a game that determines what strategies self-interested agents will use, given certain assumptions about their information and reasoning capabilities. Let $g : \Sigma_1 \times \Sigma_2 \times \cdots \times \Sigma_I \rightarrow \Lambda$ be the **outcome function** that chooses an allocation $\lambda \in \Lambda$ based on the strategies used by the agents. If, for all possible agent valuations v_i, there exists an **equilibrium strategy profile**

$(\sigma_1^*(v_1), \sigma_2^*(v_2), \ldots, \sigma_I^*(v_I))$ such that $g(\sigma_1^*(v_1), \sigma_2^*(v_2), \ldots, \sigma_I^*(v_I)) = f(v_1, v_2, \ldots, v_I)$, then $g(\cdot)$ **implements** social choice function $f(\cdot)$. Implementation is defined with respect to a particular solution concept; for example, $f(\cdot)$ might be implementable in Bayesian-Nash equilibria.

We can now define a **mechanism** $\Gamma = (\Sigma_1, \Sigma_2, \ldots, \Sigma_I, g(\cdot))$ as the collection of each agent's possible strategies, and an outcome function $g(\cdot)$. A mechanism defines the "rules of the game" by specifying the available strategies, and a procedure by which an outcome is chosen based on those strategies. The goal of a **mechanism designer** is to **implement** a social choice function $f(\cdot)$ based on actual agent preferences, *in spite of* the agents' strategies for reporting (or misreporting) their preferences. For example, in the first price auction, the goal is to allocate the item to the agent which *truly* values it the most, even though the agents have an **incentive** to tamper with their valuation reports and "bid low."

Let the shorthand $\sigma_{-i}(v_{-i}) = (\sigma_1(v_1), \ldots, \sigma_{i-1}(v_{i-1}), \sigma_{i+1}(v_{i+1}), \ldots, \sigma_I(v_I))$ be the strategies of agents other than i. Formally, the strategies $(\sigma_1^*(v_1), \sigma_2^*(v_2), \ldots, \sigma_I^*(v_I))$ are a Bayesian Nash equilibrium of the mechanism Γ if for all i and all $\theta_i \in \Theta_i$:

$$\forall \sigma_i' \in \Sigma_i \mid E_{\phi(v_{-i})}\left[u_i\left(g(\sigma_i^*(v_i), \sigma_{-i}^*(v_{-i})), \theta_i\right) \mid \theta_i\right] \geq E_{\phi(v_{-i})}\left[u_i\left(g(\sigma_i'(v_i), \sigma_{-i}^*(v_{-i})), \theta_i\right) \mid \theta_i\right]$$

This says that the agents choose the equilibrium strategies σ_i^* in order to maximize utility *in expectation*, given their beliefs about each other's probable types. Recall that in the *ad hoc* network environment, an agent's type might encapsulate its knowledge of its own traffic behavior, its history of participation in the network, or its communications and energy goals. It might be difficult for the agents to construct useful probability distributions over these parameters for one another.

The first price auction is **implementable** in Bayesian Nash equilibria. This is a fascinating result: when the agents use their equilibrium strategies, the auction still assigns the item to the buyer who values it the most, *even though* the buyers are not (necessarily) reporting their actual valuations! The equilibrium bidding strategy is for each agent to bid the **expected highest valuation** of the other agents, while not exceeding its own true valuation: $\sigma_i(v_i) = \min\left(v_i, \max_{j \neq i} E_{\phi(v_j)}\left[v_j\right]\right)$. When all the agents use this strategy, assuming they all know a *correct* distribution $\phi(\cdot)$ for each others' preferences, the item will be allocated (in expectation) to the buyer with the highest actual valuation.

Use of the Bayesian Nash equilibrium assumes that the agents have mutually correct information about each others' type and strategy distributions. In practice, it is probably the case that the agents instead have **asymmetric** and **incomplete** information about one another. One reason for this is that, in a mobile setting, the set of agents that any individual agent can **observe** is constantly changing. In addition, we are interested in minimizing the **computational demands** placed on the agents. A solution concept which requires the agents to perform complex statistical modeling of one another is not consistent with our goals.

Given these problems, the Bayesian Nash equilibrium is probably not appropriate for the the *ad hoc* network environment. We would like a solution concept in which an agent can choose its utility-maximizing strategy without having to reason about what the other agents will do. Section 2.4.3 presents such a concept.

2.4.3 Dominant Strategy Equilibria and Strategy-Proofness

A stronger solution concept than the Bayesian Nash equilibrium is implementation in **dominant strategies**. A strategy is **weakly dominant** for an agent if it gives at least as much utility as any of its other strategies for *all possible strategies* the other agents might use. The strategies $\sigma^* = (\sigma_1^*(v_1), \sigma_2^*(v_2), \ldots, \sigma_I^*(v_I))$ are a **dominant strategy equilibrium** of a mechanism $\Gamma = (\Sigma_1, \Sigma_2, \ldots, \Sigma_I, g(\cdot))$ if for all i and all $\theta_i \in \Theta_i$:

$$\forall \sigma_i' \in \Sigma_i \,.\, \forall \sigma_{-i} \in \Sigma_{-i} \quad | \quad u_i\left(g(\sigma_i^*(v_i), \sigma_{-i}(v_{-i})), \theta_i\right) \geq u_i\left(g(\sigma_i'(v_i), \sigma_{-i}(v_{-i})), \theta_i\right)$$

In a dominant strategy equilibrium, an agent maximizes utility (*not* expected utility) by using its dominant strategy, *no matter what* strategies the other agents use. All dominant strategy equilibria are in fact Bayesian Nash equilibria, though the converse is not true. If a social choice function $f(\cdot)$ can be implemented in dominant strategies — in other words, if $g(\sigma^*(\theta)) = f(\theta)$ — then it doesn't matter if the agents have **incorrect** information $\phi(\cdot)$ about one another. This is good for the mechanism designer, since if $f(\cdot)$ can be implemented in dominant strategies, then it can be implemented even if the designer does not know $\phi(\cdot)$.

Not all social choice functions are implementable in dominant strategies. In the special case of agents with **quasilinear preferences** (§2.4.1), it *is* possible to implement auctions and exchanges. The **Groves mechanisms** (Groves, 1973) enable implementation by restricting the **payment rules** $p_i(\cdot)$ of a social choice function $f(\theta) = (\lambda(\theta), p_1(\lambda(\theta)), p_2(\lambda(\theta)), \ldots, p_I(\lambda(\theta)))$. A Groves payment has the form:

$$p_{i,\text{groves}}(\lambda^*(\theta)) = -\left[\sum_{j \neq i} \hat{v}_j(\lambda^*(\theta), \theta_j)\right] + h_i(\theta_{-i})$$

Agent i's Groves payment is based on the reported valuations of the *other* agents for the surplus-maximizing allocation $\lambda^*(\theta)$; and $h_i(\cdot)$, a function that depends *only* on the preferences of the other agents. In other words, $h_i(\cdot)$ *cannot* depend on agent i's reported valuation. Later, we will give an example of $h_i(\cdot)$.

When Groves payments are used, **truth telling** $\sigma_i(v_i) = v_i$ is a dominant strategy for the agents. The proof refers to the definition of **allocative efficiency** introduced in Section 2.4.1. Let the shorthand $\lambda^*(v_i, v_{-i}) = \lambda^*(v_i(\cdot, \theta_i), v_{-i}(\cdot, \theta_{-i})) = \lambda^*(\theta)$ be the surplus-maximizing allocation based on the agents' true valuations. Call $\lambda^*(\hat{v}_i, v_{-i}) = \lambda^*(\hat{v}_i(\cdot, \theta_i), v_{-i}(\cdot, \theta_{-i}))$ the surplus-maximizing allocation when i makes the announcement $\hat{v}_i$. For conciseness, we drop the type notation for agent valuations, so that $v_i(\lambda^*(\hat{v}_i, v_{-i})) = v_i(\lambda^*(\hat{v}_i, v_{-i}), \theta_i)$.

The proof (by contradiction) is as follows. Suppose that truth-telling were *not* a dominant strategy for agent i. Then there must exist a v_i, $\hat{v}_i$, and v_{-i} such that i strictly prefers the outcome in which it reports $\hat{v}_i \neq v_i$. Using the definitions of quasilinear utility and the Groves payment rule:

$$u_i(\lambda^*(\hat{v}_i, v_{-i})) > u_i(\lambda^*(v_i, v_{-i}))$$

$$v_i(\lambda^*(\hat{v}_i, v_{-i})) - p_{i,\text{groves}}(\lambda^*(\hat{v}_i, v_{-i})) > v_i(\lambda^*(v_i, v_{-i})) - p_{i,\text{groves}}(\lambda^*(v_i, v_{-i}))$$

$$v_i(\lambda^*(\hat{v}_i, v_{-i})) + \left[\sum_{j\neq i} v_j(\lambda^*(\hat{v}_i, v_{-i}))\right] - h_i(\theta_{-i}) > v_i(\lambda^*(v_i, v_{-i})) + \left[\sum_{j\neq i} v_j(\lambda^*(v_i, v_{-i}))\right] - h_i(\theta_{-i})$$

$$\left[\sum_{j=1}^{I} v_j(\lambda^*(\hat{v}_i, v_{-i}))\right] - h_i(\theta_{-i}) > \left[\sum_{j=1}^{I} v_j(\lambda^*(v_i, v_{-i}))\right] - h_i(\theta_{-i})$$

$$\sum_{j=1}^{I} v_j(\lambda^*(\hat{v}_i, v_{-i})) > \sum_{j=1}^{I} v_j(\lambda^*(v_i, v_{-i}))$$

$$\sum_{j=1}^{I} v_j(\lambda) > \sum_{j=1}^{I} v_j(\lambda^*(\theta)) \quad \textit{contradicts allocative efficiency}$$

Therefore, truth-telling must be a dominant strategy for i.

A social choice function $f(\cdot)$ for which truth-telling is a dominant strategy equilibrium is said to be **dominant strategy incentive compatible**, or **strategy-proof**. Strategy-proofness is a powerful concept for practical environments such as the route-selecting exchange. It frees the agents from the **game-theoretic complexity** of reasoning about a strategy to use in manipulating the exchange. This means the agents do not need to know a distribution of each others' types $\phi(\cdot)$, and do not have to compute an expected-utility-maximizing valuation report.

We have shown that, for agents with quasilinear preferences, it is possible to implement a strategy-proof social choice function $f(\cdot)$ by using the Groves payment form. We still need to define the $h_i(\theta_{-i})$ term in $p_{i,\text{groves}}(\cdot)$. Let the notation

$\lambda^*_{-i}(\theta_{-i})$ represent the surplus-maximizing allocation when agent i is removed from the game. In an auction or exchange environment, this is the same as removing all of agent i's bids from consideration. The **Clarke mechanism** (Clarke, 1971) defines $h_i(\theta_{-i})$ as:

$$h_i(\theta_{-i}) = \sum_{j \neq i} \hat{v}_j(\lambda^*_{-i}(\theta_{-i}), \theta_j)$$

In other words, this $h_i(\theta_{-i})$ is the surplus that would be solved by the social choice function if i did not exist. The complete payment rule is therefore:

$$p_{i,\text{clarke}}(\lambda^*(\theta)) = -\left[\sum_{j \neq i} \hat{v}_j(\lambda^*(\theta),\, \theta_j)\right] + \left[\sum_{j \neq i} \hat{v}_j(\lambda^*_{-i}(\theta_{-i}), \theta_j)\right]$$

One interpretation of the "Clarke tax" $h_i(\theta_{-i})$ is that it causes agents to **internalize** the **externality** their reported valuations impose on others. If agent i's reported valuation changes the chosen allocation from what would be efficient in i's absence, then i pays an amount equal to the difference in surplus. On the other hand, if including i's reported valuation does not cause the efficient allocation to change, then i pays nothing.

As an example, consider the two-buyer auction problem from Section 2.4.2. When buyer 1 wins, buyer 2's (implicit) reported valuation is $\hat{v}_2((\lambda_0,\, \lambda_1),\, \theta_2) = 0$. If buyer 1 were removed from the auction, buyer 2 would become the winner, and the efficient allocation would have surplus $\hat{v}_2((\lambda_0,\, \lambda_2),\, \theta_2)$. Using Clarke payments, this means that when buyer 1 wins, it pays the amount of buyer 2's valuation! A symmetric analysis holds for buyer 2's winning payment, and the payment rule is:

$$p_{1,\text{clarke}}((\lambda_0,\,\lambda_1)) = \hat{v}_2(\lambda_2,\,\theta_2) \qquad p_{1,\text{clarke}}((\lambda_0,\,\lambda_2)) = 0$$

$$p_{2,\text{clarke}}((\lambda_0,\,\lambda_1)) = 0 \qquad p_{2,\text{clarke}}((\lambda_0,\,\lambda_2)) = \hat{v}_1(\lambda_1,\,\theta_1)$$

This is the **second price sealed-bid auction**, so named because the winning buyer pays the *second*-highest bid price. The second price auction, also known as a **Vickrey auction** (Vickrey, 1961), is **strategy-proof**; a buyer maximizes utility by always reporting its true valuation for allocations. Recall that in the **first price auction**, an agent could benefit by reporting a valuation which was *less* than its true valuation. This *decreased* the likelihood of winning, but when the agent *did* win, its utility *increased*. By comparison, in the second price auction, reporting a low valuation *only* decreases the likelihood of winning. The agent does not pay less when it wins.

2.4.4 The Vickrey Clarke Groves Mechanism

We can now describe the general **Vickrey Clarke Groves mechanism**, or VCG mechanism, for combinatorial exchanges. The VCG mechanism truthfully implements the surplus-maximizing (efficient) allocation choice λ^* in dominant strategies. It achieves this by using a Groves-Clarke payment rule, which we will redefine in terms of a **discount** to each agent. Let $V^* = \sum_{j=1}^{I} \hat{v}_j(\lambda^*(\theta),\,\theta_j)$ be the surplus of the surplus-maximizing allocation, and let $V^*_{-i} = \sum\limits_{j \neq i} \hat{v}_j(\lambda^*_{-i}(\theta_{-i}),\, \theta_j)$ be the surplus of the surplus-maximizing allocation when i's bids are removed. Define the **Vickrey discount** to i as $\Delta_{i,\text{vick}} = V^* - V^*_{-i}$. We can show that a **Vickrey payment** in which an agent pays its reported valuation for an allocation less its Vickrey discount is equivalent to the Groves-Clarke payment rule:

$$\begin{aligned}
p_{i,\text{clarke}}(\lambda^*(\theta)) &= -\left[\sum_{j \neq i} \hat{v}_j(\lambda^*(\theta),\, \theta_j)\right] + \left[\sum_{j \neq i} \hat{v}_j(\lambda^*_{-i}(\theta_{-i}), \theta_j)\right] \\
&= \left(\hat{v}_i(\lambda^*(\theta),\, \theta_i) - \left[\sum_{j=1}^{I} \hat{v}_j(\lambda^*(\theta),\, \theta_j)\right]\right) + \left[\sum_{j \neq i} \hat{v}_j(\lambda^*_{-i}(\theta_{-i}), \theta_j)\right] \\
&= \hat{v}_i(\lambda^*(\theta),\, \theta_i) - V^* + V^*_{-i} \\
p_{i,\text{vick}}(\lambda^*(\theta)) &= \hat{v}_i(\lambda^*(\theta),\, \theta_i) - \Delta_{i,\text{vick}}
\end{aligned}$$

The Vickrey discount is always non-negative. This follows from the fact that the surplus of the efficient allocation can never strictly increase by removing an agent i. If such an increase were possible, the same greater surplus would also be achievable by *keeping* i, but assigning all of its bids *losing*. Therefore, $V^* \geq V^*_{-i}$ always, and $\Delta_{i,\text{vick}} \geq 0$ always.

We can use this fact to claim that the VCG mechanism is **individual rational**, meaning that an agent's expected utility from participation is non-negative. This is a useful property for environments in which agent participation in the mechanism is **voluntary**. It says that an agent's utility will never strictly decrease as a result of participation. In other words, a buyer will never pay more than its bid price for an allocation, and a seller will never be paid less than its ask price.

Unfortunately, the VCG mechanism is not **budget balanced**. Weak budget balance exists when $\sum_{i=1}^{I} p_{i,\text{vick}}(\lambda(\theta)) \geq 0$. That is, the mechanism must not pay more to the agents than the agents pay to the mechanism. Stated in terms of discounts, $V^* \geq \sum_{i=1}^{I} \Delta_{i,\text{vick}}$. This result is a consequence of the Myerson-Satterthwaite impossibility theorem (Myerson, 1983), which states that no mechanism can simultaneously be **allocatively efficient**, **individual rational**, and **budget balanced**.

Lack of budget balance is not a serious problem for the ad hoc network environment; it simply restricts what kinds of **payment instruments** can be used. For example, **digital coins** will probably not be a good choice, since the auctioneer would need to be able to "print money" in order to supply the discounts. The ability of the auctioneer to pay out more than it takes in is the *only* constraint we place on a **money** or **credit** system in this thesis.

We are now prepared to revisit the combinatorial exchange route selection example of Figure 2.9 on page 42 and compute **Vickrey payments**. In the example, the surplus-maximizing combination of bids was B_2, B_3, B_7, B_8, and B_{10}, with a combined surplus of $V^* = 1.4$. In the VCG mechanism, each agent i will pay its winning bid price discounted by $\Delta_{i,\text{vick}}$. The way the discounts are computed is by solving a series of smaller exchanges with each agent removed in turn. This increases the computational complexity of the mechanism; in the worst case, with I agents, $I + 1$ separate winner determination problem must be solved. We can take advantage of the fact that the discount to an agent with only losing bids is zero, and only solve smaller exchanges for each *winning* agent.

For example, to compute the payment agent 1 makes after its bid B_2 is found winning, we compute its Vickrey discount $\Delta_{i,\text{vick}} = V^* - V^*_{-1}$. After removing all of 1's bids from $\mathcal{B}$, we solve a smaller exchange: $\mathcal{B}_{-1} = \{B_3, B_4, B_5, B_6, B_7, B_8, B_9, B_{10}\}$. No combination of the bids in $\mathcal{B}_{-1}$ yields a positive surplus, so no bids are assigned winning and $V^*_{-1} = 0$. Agent 1's reported valuation for the original surplus-maximizing allocation λ^* is simply the sum of its winning bid prices. Since B_2 was agent 1's only winning bid, $\hat{v}_1(\lambda^*) = p_2 = 3$. Therefore, agent 1's Vickrey payment is $p_{1,\text{vick}}(\lambda^*) = \hat{v}_1(\lambda^*) - \Delta_1 = 3 - 1.4 = 1.6$. The remaining agent payments are computed in a similar fashion:

$$\begin{aligned}
p_{2,\text{vick}}(\lambda^*) &= 2 - (1.4 - 1.1) &&= 1.7\\
p_{4,\text{vick}}(\lambda^*) &= -1.5 - (1.4 - 1.1) &&= -1.8\\
p_{6,\text{vick}}(\lambda^*) &= -1 - (1.4 - 0) &&= -2.4\\
p_{7,\text{vick}}(\lambda^*) &= -1.1 - (1.4 - 1.1) &&= -1.4\\
p_{3,\text{vick}}(\lambda^*) &= 0\\
p_{5,\text{vick}}(\lambda^*) &= 0
\end{aligned}$$

Like agent 1, agent 6 receives a large Vickrey discount because with the flow $1 \rightsquigarrow 6$ unavailable, agent 2 cannot afford any of the routes to 7, resulting in a zero surplus. Note that buyers pay *less* than their bid prices, sellers are paid *more* than their ask prices, and net payments to the mechanism are negative, at -2.3. We can define the **mechanism payment ratio**, R, to characterize how much "money" the VCG mechanism loses relative to the maximal surplus:

$$R = \frac{\sum_{i=1}^{I} p_{i,\text{vick}}(\lambda^*)}{V^*}$$

In this example, $R = \dfrac{-2.3}{1.4} = -1.64$. The mechanism does not collect any of the surplus indicated by the prices of the winning bids and asks. In fact, for every unit of surplus indicated, the mechanism must *pay out* more than one unit to the agents. The fact that $R < 0$ in this example is evidence that the VCG mechanism is not **budget balanced**.

2.4.5 Optimal Solutions and Strategy-Proofness

The **winner determination problem** for combinatorial auctions and exchanges is $\mathcal{NP}$-complete. In principle, each of the $2^{|\mathcal{B}|}$ combinations of bids must be examined in order to find the one which maximizes surplus. Chapter 4 describes how this complexity can be managed in practice using **heuristic search**. Approximations to the optimal $\lambda(\cdot)$ which are more computationally feasible, such as greedy methods (Lehmann et al., 1999), have been studied. These methods sacrifice **strategy-proofness**, and incent the agents to misreport their preferences in order to "help" the approximation algorithm reach a better allocation (Nisan and Ronen, 2000).

The design of this thesis is based on an **optimal mechanism**. That is, not only do we solve the winner determination problem optimally, we have formulated the exchange itself in a way that leads to an optimal allocation based on truthful agent reports. Specifically, as explained in Section 2.3.2, we require that buyers whose **demand sets** overlap participate in a *single* exchange. This requirement allows sellers to express **complementarity** between flows by submitting **multi-unit asks**. The **prices** of these asks reflect the **marginal energy cost** of additional flows, described in Section 2.1.2.

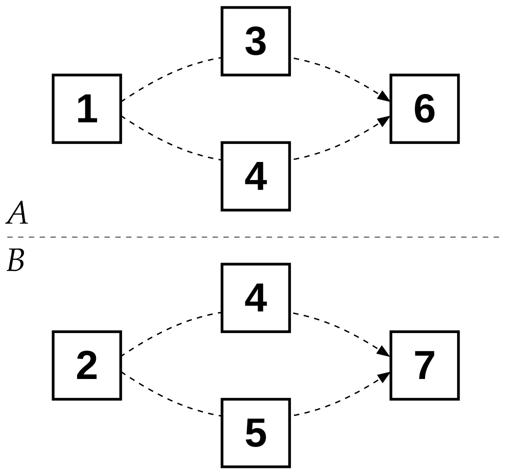


Figure 2.10: Solving with separate exchanges.

To see how this requirement helps, suppose the example from Section 2.3.2 were solved using a separate exchange for each traffic flow. This new configuration is illustrated in Figure 2.10. The separate exchanges A and B can be solved faster (though their solution is still $\mathcal{NP}$-complete) because they involve smaller numbers of bids. This method, though easier to compute, yields a **suboptimal solution**. In the example of Figure 2.10, the problem arises because agent 4 cannot express the complementarity that exists if it wins in *both* exchanges.

The **marginal energy cost of additional flows** says, given that a relay or sink is already providing service for k flows, adding flow $k + 1$ only **fractionally increases** energy consumption. When a relay or sink reports its loss in value to the mechanism in the form of an ask, it must be able to express these marginal energy costs. When overlapping demand sets are resolved through the use of a single exchange, relays express marginal cost relationships through **multi-unit asks**. In the original example, agent 4 fully expressed its costs by announcing a price of -1 to relay one flow, but only -1.5 to relay both flows. This reflects the fact that the energy to relay the second flow, given the first, is fractional.

When separate exchanges are used, agent 4 can only report its costs for the individual flows *separately*. It cannot describe its costs under one exchange based on the solution of another, since the exchanges are solved independently. In other words, agent 4 cannot report to exchange A, "my costs are p, but if I win in exchange B, I can offer you a better price." This leads to an **inefficient solution** in two cases. In the first, agent 4's single-flow asks might win in *both* exchanges. This essentially "overcharges" the mechanism, since 4's true costs to relay two flows are *less* than twice the cost of relaying one flow. In other words, the surplus of this solution could have been greater had 4 been able to express its actual, lesser costs using multi-unit asks. The other case occurs when 4 *loses* in one or both exchanges. By submitting separate one-flow asks, agent 4 appears less affordable than it actually is, so its asks may not maximize surplus. If multi-unit asks were available, the mechanism might have found that surplus increased by using 4 for multiple flows.

We can distribute the bids $\mathcal{B}$ from the earlier example among the smaller exchanges A and B. Exchange A contains the bids and asks relevant to $1 \rightsquigarrow 6$:

$$B_1 = \langle(\lambda_1^3 : 1,\ \lambda_1^6 : 1),\ 3\rangle \qquad B_2 = \langle(\lambda_2^4 : 1,\ \lambda_2^6 : 1),\ 3\rangle$$

$$B_5 = \langle(\lambda_5^3 : -1),\ -0.9\rangle \qquad B_7 = \langle(\lambda_7^6 : -1),\ -1\rangle \qquad B_9 = \langle(\lambda_9^4 : -1),\ -1\rangle$$

Exchange B contains the bids and asks for $2 \rightsquigarrow 7$:

$$B_3 = \langle(\lambda_3^4 : 1,\ \lambda_3^7 : 1),\ 2\rangle \qquad B_4 = \langle(\lambda_4^5 : 1,\ \lambda_4^7 : 1),\ 2\rangle$$

$$B_6 = \langle(\lambda_6^5 : -1),\ -1\rangle \qquad B_8 = \langle(\lambda_8^7 : -1),\ -1.1\rangle \qquad B_{11} = \langle(\lambda_{11}^4 : -1),\ -1\rangle$$

Agent 4 submits an ask to relay one flow to both exchanges. To reinforce the notion that these asks are **distinct**, we have named agent 4's ask B_{11} in exchange B. Note that agent 4's ask to relay two flows, B_{10} in the original example, does not appear here. The solution of exchange A cannot take into account possible overlap with exchange B, and vice versa.

The surplus maximizing combination of the bids and asks in exchange A assigns B_1, B_5, and B_7 winning. This corresponds to the route $(1, 3, 6)$, with a surplus of $V_A^* = 1.1$. *No* combination of the bids and asks in exchange B yields a positive surplus. No bids are assigned winning, and $V_B^* = 0$. The total surplus found by the mechanism using separate exchanges, $V_A^* + V_B^* = 1.1$, is less than the surplus of the single exchange, $V^* = 1.4$. This example shows that using separate exchanges in our environment yields a suboptimal solution.

We can now show that this suboptimal solution incents **untruthful reporting** by the agents. Suppose that after experimenting with the mechanism, agent 4 finds that submitting one-flow asks with price -0.75 to each exchange results in a better outcome. In other words, let $p'_9 = p'_{11} = -0.75$. In exchange A, route $(1, 4, 6)$ would be chosen with surplus $V_A^* = 1.25$, and in exchange B, route $(2, 4, 7)$ would be chosen with $V_B^* = 0.15$. Note that $V_A^* + V_B^* = 1.4$, which is *exactly the optimal result*. By lying to the mechanism, agent 4 improves its own **utility** *and* the **quality** of the overall solution.

Of course, this improvement is contingent on the reports of the *other* agents. If one, but not both, of agent 4's asks wins, agent 4's net utility could be negative (unless its **Vickrey discount** $\Delta_{4,\text{vick}}$ is sufficiently large), breaking **individual rationality**. Agent 4 cannot guarantee non-negative utility when it lies to the mechanism. When it knows a **distribution** $\phi(\cdot)$ of the other agents' types, it may obtain non-negative utility **in expectation**. Our goal in pursuing implementation in **dominant strategies** was to free the agents from having to reason about potentially advantageous manipulations. Using suboptimal solutions, the mechanism is no longer **dominant strategy incentive compatible**. In other words, it is not **strategy-proof**.

Our purpose in examining the rôle of suboptimal solutions is to argue that a

single-exchange solution for overlapping routes is *necessary* to ensure strategy-proofness. The use of separate exchanges is attractive for a variety of engineering reasons. As noted, the **computational costs** of winner determination may be reduced. The **communications overhead** of bid submission in a distributed environment may improve, as well. However, any solution that does not optimally account for **energy complementarity** at overlap nodes will fail to ensure truthful reporting by the agents.

2.5 Summary

This Chapter presented the four basic technologies underlying this thesis. We began by describing some of the control and management features of the IEEE 802.11 wireless LAN standard. The dominance of the **idle** state in the energy profile of an 802.11 transceiver was introduced, followed by a **power management** scheme to reduce the impact of high idle power.

Section 2.2 presented the basic features of Dynamic Source Routing, an **on demand source routing** protocol for multihop ad hoc networks. We showed that the 802.11 IBSS power management design introduces high **latency** to the route discovery process, as well as to the delivery of application messages. We also argued that the source-oriented route selection method could lead to **unfair** energy consumption among relays. We proposed that an improved route selection process would take into account the **preferences** of relays and sinks.

A main insight of this thesis is our interpretation of DSR route selection as a **combinatorial reverse auction**. We developed two forms of the reverse auction which cast the traffic source as auctioneer, and the relays and sink as sellers. The DSR auction is restrictive in the sense that it offers **inexpressive pricing** for sellers, and lacks the concept of an **unaffordable** route. Further, it misses opportunities to **consolidate** the set of active network nodes by holding independent auctions for each source-sink pair. We proposed a solution to these problems by generalizing route selection using **combinatorial exchanges**, which are allocatively efficient.

The combinatorial exchange formulation uses **credit** to account for the work done by network nodes. Relays and sinks accrue credit when providing their services, which reflects the loss in **value** they experience in terms of energy, buffer space, or even user attention. Traffic sources need to spend credit in order to use their routes, reflecting the gain in value they experience when their messages are delivered. When network agents have **quasilinear preferences**, meaning that their **utility** is a function of their happiness about a network configuration less their **payment** to realize that configuration, there is an incentive to misreport bid prices. We presented the **Groves mechanisms**, a game-theoretic instrument which shows how to constrain **payment rules** so as to make bidding **strategy-proof**. A Groves-based combinatorial exchange was shown to be allocatively efficient, **individual rational**, and strategy-proof, but *not* **budget balanced**. Finally, we showed that **suboptimal** mechanism solutions conflict with strategy-proofness.

Chapter 3

Related Work

This thesis focuses on the use of interaction mechanisms to coordinate power management among the nodes in an *ad hoc* network. The underlying assumption of the research is that power management should reduce the amount of time radio transceivers spend in the **idle** state. This Chapter briefly presents prior work which assumed that the **transmit power rate** must be optimized. Next, approaches to idle-state power management are described that employ out-of-band signaling. The bulk of this Chapter covers idle-state power management which uses information from higher protocol layers to control power mode transitions. Credit- and reputation-based techniques are presented which enforce behaviors in coordinated ad hoc networks. Finally, we conclude with theoretical work on the use of **economics** and **mechanism design** in networks of self-interested nodes.

Figure 3.1 shows the relationship between this thesis research and previous

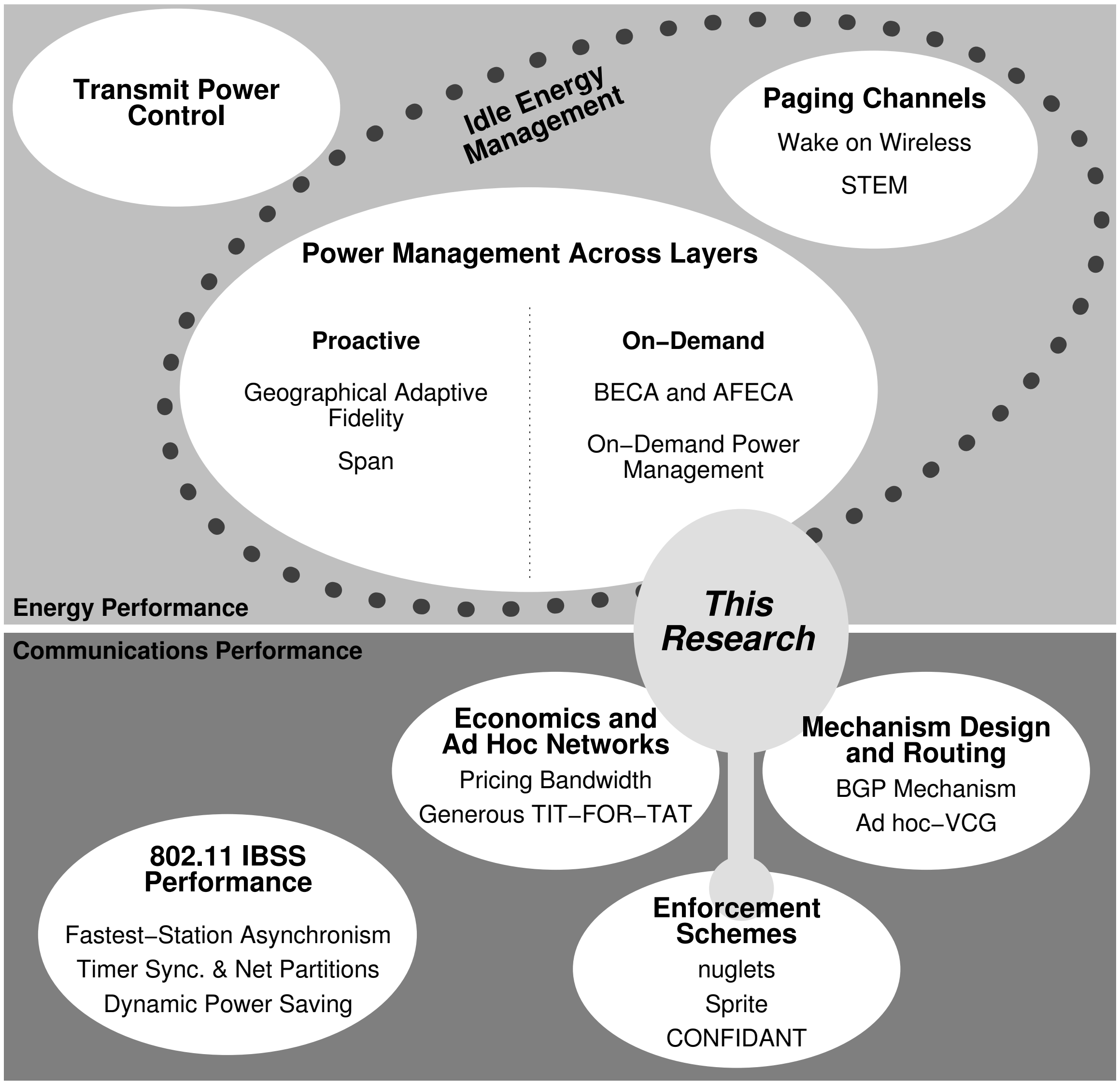


Figure 3.1: Taxonomy of related work.

work. The multihop power management design presented in Chapter 4 is first and foremost a method for controlling the energy consumption of nodes in an *ad hoc* network. By using combinatorial exchanges to negotiate over sources, relays, and sinks, our work leverages tools from the field of economics. Finally, enforcement schemes such as credit protocols are complementary to this work, in that they could be used to ensure that nodes follow their negotiated agreements. The following Sections describe these areas of related work in more detail.

3.1 Energy Performance

Most energy-related research in the area of mobile networking studies the problem of improving node energy consumption without degrading communications performance too much. This work falls primarily into two categories: efforts to reduce **transmitter power**, and efforts to manage the **energy costs** of high idle-state power. As explained in Section 3.1.1, the former type could be used in conjunction with the present research, but we focus on idle energy because of its dominance in 802.11 radios. Within the set of designs that attempt to reduce the energy costs of the idle state, there are two main approaches. The first, which adds **hardware** and **spectrum complexity** by using a companion radio, is described in Section 3.1.2. The second approach, which the work of this thesis adopts, uses information from higher protocol layers to improve the performance of low-level power management features. These cross-layer designs are detailed in Section 3.1.3.

3.1.1 Transmit Power Control

Intuitively, energy consumption by a radio transceiver depends on how many bits that transceiver sends or receives. In addition to the number of bits, the **radiated power level** used for transmission determines coverage radius of the transmitter. For example, as approximations, the **inverse-square** or **inverse-fourth-power** rules relating transmission power and distance are often used. As a result, many researchers have sought to manage energy consumption in a wireless ad hoc network by adjusting transmitter radiated power.

A change in radiated power can change the network **topology**. Increasing radiated power may increase connectivity, and create multihop paths with fewer hops. Decreasing radiated power may reduce interference between transmitters. In recent years, there has been great interest in studying the interaction between topology and radiated power (Banerjee and Misra, 2002; Lloyd et al., 2002; Čagalj et al., 2002; Cruz and Santhanam, 2003; Kawadia and Kumar, 2003; Zussman and Segall, 2003).

Transmitter power control methods are orthogonal to the approaches described in this thesis. The present research is founded on the observation that, for spread-spectrum radios as IEEE 802.11 transceivers, energy consumption in *idle* state is dominant, and that radiated power has second-order impact. Transmit power schemes may be more appropriate for low-power radios such as those found in sensor networks. Also, such methods could be used in conjunction with idle-state management designs for topology or interference control. For example, in 802.11 radios, actual radiated power (typically below 50mW) is smaller by an order of magnitude than total radio transceiver power (typically more than 1W). Transmit power could be varied over the range permitted by the 802.11 specification and not interfere with power management schemes such as the one presented in this thesis.

3.1.2 Paging Channels

As explained in Section 2.1.3.2, timer-based rendezvous power management trades off communications latency for idle-state energy savings. This thesis describes algorithmic methods that nodes may use to coordinate themselves in order to reduce this latency. A different approach, which substitutes hardware and spectrum complexity for coordination or timer complexity, is to use an out-of-band **paging channel**. Section 3.1.2.1 describes a design which pairs an 802.11b radio with a low-power paging radio to coordinate power management in infrastructure networks. Section 3.1.2.2 uses two identical radios, but operates the paging radio on a low duty cycle.

3.1.2.1 Wake on Wireless

The goal of **Wake on Wireless** (Shih et al., 2002) is to use a low-power control channel to notify wireless devices that they should wake up their 802.11b radios and receive messages. Using a PIC microcontroller and 19.2kbps 915MHz radio, which together draw less than 10mW during active use, a mobile host listens for **power-on messages**. These messages are generated by proxy hosts attached to the wired infrastructure LAN. Proxies receive commands from network presence servers, which act as intermediaries between "caller" and "callee" nodes.

Using an iPAQ mobile node, the authors were able to increase battery lifetime from 10 hours to 14 hours under a telephony workload. This is a 40% improvement over simply using the 802.11b radio in infrastructure power management mode. Countering these gains are the cost of the extra radio electronics on the mobile node and in the infrastructure. Because the low-power radios have a short range, a large number of transceivers must be installed on a site.

3.1.2.2 STEM

Rather than use a paging channel radio with low energy consumption, (Schurgers et al., 2002) uses a radio with a low duty cycle. In a multihop sensor network environment, **Sparse Topology and Energy Management** (STEM) places two identical radios at each sensor node. One radio controls the **wakeup plane**; periodically, nodes activate their wakeup plane radio to listen for **wakeup messages**. If a node receives a wakeup message, it then activates a **data plane** radio to handle the actual data exchange. The authors argue for a dual-radio (dual-frequency) design over single-channel design — such as 802.11 — in order to reduce wakeup message collisions. By combining dual radios with **Geographical Adaptive Fidelity**, described in Section 3.1.3.3, the authors report a 90% energy savings in simulation. The authors acknowledge that this design trades off energy for latency; in this regard, STEM seems to have all of the latency problems of an 802.11 *ad hoc* network (§2.2.2), on top of the complexity of a redundant radio.

3.1.3 Power Management Across Layers

The most promising architectures for power management in multihop networks seem to be those in which power state transitions (*e.g.*, from **doze** to **active**) at the MAC layer are governed by events at higher layers in the network protocol stack. Nodes which are not required for the delivery of messages should be permitted to use aggressive power management techniques. Nodes which are needed to service traffic flows should trade off energy savings for low-latency delivery performance. By using higher-layer information about the rôle of a node in the multihop network, a power management scheme can more appropriately configure the power mode of the underlying transceiver.

This Section presents several recent designs which use higher-layer knowledge, either of topology or communications activity, to control power management. Interestingly, the designs which focus on topology management seem to ignore activity. These designs elect a subset of nodes to stay in high-power, high-performance states regardless of the traffic level in the network. These **proactive** designs generally achieve good delivery latency at the cost of potentially high worst-case node energy consumption. On the other hand, the designs which use message activity to stimulate power mode transitions generally do not consider the broader network topology. These designs do not permit optimizations that require a wide view of the network, such as optimizing the number of nodes which are active for message relaying. Such **on-demand** approaches may waste less energy than proactive schemes, since the only nodes which operate in high-power modes are those needed to support active traffic flows. One tradeoff is that the latency of waking up the necessary relays may be high. Another is that, since node rôles are not **negotiated**, it is possible for some nodes to experience degenerately high energy consumption.

The following five designs are presented in chronological order of publication. Sections 3.1.3.1, 3.1.3.2, and 3.1.3.5 describe on-demand power management approaches, while Sections 3.1.3.3 and 3.1.3.4 are proactive.

3.1.3.1 BECA

The **Basic Energy-Conserving Algorithm**, or BECA (Xu et al., 2000), uses a timer-driven state machine to turn the radio on and off. (BECA, and the related designs in Sections 3.1.3.2 and 3.1.3.3, do not use MAC power management; they simply *turn off* the radio.) BECA requires the underlying routing protocol to retry routing requests, and permits nodes to turn off their radios for some multiple of this retry interval. Nodes periodically **wake** and listen for traffic; if none arrives during a retry interval, the node returns to **sleep**. If the node receives a routing request and participates in a route, or has traffic of its own to send, it remains awake until some time after the activity ceases.

Using the ***ns* network simulator** (Fall and Varadhan, 1998), the designers extended the **Ad hoc On-Demand Distance Vector** routing protocol (Perkins and Royer, 1999; Perkins et al., 2003), **AODV**, to support BECA. Route setup latency increases by as much as two orders of magnitude when using BECA relative to unmodified AODV. Energy consumption improves by 35%–40%, and **network lifetime** (defined as the time at which all nodes have exhausted their energy reserves) increases by 20%.

3.1.3.2 AFECA

A close relative of BECA is the **Adaptive Fidelity Energy-Conserving Algorithm**, or AFECA (Xu et al., 2000), which is an optimization to the timing of the BECA state machine. AFECA takes advantage of route redundancy in dense networks to increase sleep time when other route-equivalent nodes are nearby. Before going to sleep, an AFECA node passively eavesdrops on its neighbors to generate an estimate of their number, and uses this estimate to scale a randomized sleep duration.

Again simulating in *ns* with AODV, AFECA shows a 2–5% improvement in per-node energy consumption relative to BECA. Because AFECA may choose longer sleep times than BECA, route setup latency increases by as much as a factor of five. AFECA shows a clear improvement over BECA in terms of network lifetime, where AFECA improves longevity by 55% over an unmodified AODV network. AFECA is better able to exploit density in the network and can more aggressively power off redundant nodes.

3.1.3.3 Geographical Adaptive Fidelity

Geographical Adaptive Fidelity (Xu et al., 2001), **GAF**, replaces the neighborhood density estimation from AFECA with explicit location data, such as from GPS. GAF divides the physical space over which the network operates into **virtual grids**, where each node belongs to at most one grid. The size of each grid is a function of radio range, such that all nodes in a grid are within the transmission radius of all nodes in every adjacent grid. Within a grid, a **node ranking** process determines which nodes can **sleep** and which must stay active to support the routing protocol. The ranking rules attempt to create grids with only one active node; already-active nodes outrank inactive nodes, ties are broken first by estimated time remaining in the active state, and then by node identifier. There is some flexibility in estimating the active time remaining. A conservative estimate — say, projecting activity until the battery is exhausted — results in a situation where each node remains active until it expires, then is replaced by another. The designers of GAF choose an estimate of *half* the remaining battery lifetime in their experiments; thus, an active node consumes half of its energy before being replaced.

For high node mobility speeds and long pause times between movements, GAF is shown to consume up to 40% less energy per node (on average) than an unmodified on-demand routing protocol. With no pause between movements, these savings can increase to 50–60%. There is some question as to how sleep mode was treated in the authors' *ns* simulations. The energy model is based off of measurements reported for a pre-802.11 transceiver (Stemm and Katz, 1997), with a 25mW power value for sleep mode. This seems to suggest the use of something like 802.11 power management, rather than simply turning off the radio as in BECA or AFECA. The authors do not mention an implementation of 802.11 Power Management in their simulator, however. In any case, the power-cycling state machine of GAF is clearly descendant from BECA and AFECA, which were designed for the case of no useful underlying power management.

GAF transitions are not dependent on whether a node is currently being used to relay messages. As such, it is the responsibility of the routing protocol to deal with unexpected route breaks resulting from the GAF ranking procedure. Conversely, even in a completely idle network, GAF will cause some subset of nodes to remain active at all times. In this sense, GAF uses information about the local topology, but not information about communications activity to determine power state transitions.

3.1.3.4 Span

Span (Chen et al., 2001) is a contemporary of GAF, but does not make such explicit use of geographic information, and takes better advantage of lower-level power management. Although all Span nodes can send and receive messages at any time, only an elected subset must serve as relays. This subset is a **forwarding backbone**, and the members of this set are known as **coordinators**. Span attempts to choose a minimal set of coordinators such that all nodes are within range of at least one coordinator; this ensures connectivity. For performance reasons, coordinators do not operate in power management mode, although non-coordinators may. A non-coordinator node decides to try and become a coordinator if it discovers two neighbors who cannot reach each other through zero, one, or two coordinators. In the case where multiple equivalent nodes simultaneously attempt to become coordinators, Span attempts to ensure that not too many redundant nodes succeed. To achieve this, a randomized backoff is applied to their **coordinator announcement messages**; after the first few announcements are heard, the remaining candidates should decide that their participation is no longer needed. This backoff is scaled by, among other things, the number of neighbors a node senses, and a linear function of the energy remaining at a node. This algorithm tends to rotate the rôle of coordinator among the all network nodes, but in the case of "bridge" nodes which join connected components of the network, such nodes will always be pressed into service.

The authors define network lifetime as the time from initialization to the time when fewer than 90% of messages can be delivered. Using *ns* simulation with greedy geographic forwarding, Span doubles network lifetime in simulation over an unmodified ad hoc network. Latency using Span is within a factor of two to four of 802.11 with no power management. For comparison, latency with 802.11 IBSS power management is shown to be two orders of magnitude larger than without power management. As with GAF, there is always a set of coordinator nodes active in a Span network, regardless of traffic load.

3.1.3.5 On-demand Power Management

In **On-demand Power Management** (Zheng and Kravets, 2003), information from the DSR layer is used to trigger power management transitions at the 802.11 layer. Upon the reception of a DSR packet, nodes set a **keep-awake timer** which suspends the use of power management at the MAC layer. Route Reply and data messages are considered good indicators of “commitment” to a route, while Route Request messages are not. Accordingly, Route Replies and data messages cause the keep-alive timer to be extended, while Route Requests have no effect on this timer. The power management state of neighbor nodes is determined via passive inference. For example, if traffic has not been overheard from a neighbor recently, that neighbor is assumed to be in power management mode (or has moved). Using *ns* with the 2Mbps 802.11 MAC, the authors report power savings of up to 50% based on the Span power model (Chen et al., 2001), which is similar to that used in this thesis.

End-to-end latency is noted as a problem during route setup, before the power management state of neighbor nodes is known. For route discovery messages, several graphs in the paper illustrate a delay comparable to that experienced under normal IBSS power management. No coordination among the nodes is used, although it is mentioned in the context of load balancing for future work.

This is the most recent work in the area of using network-layer information to control MAC power state transitions. Conceptually, it is very similar to BECA (§3.1.3.1), but uses 802.11 Power Management rather than simply turning the radio off. Like BECA and AFECA, packet handling is the stimulus for power mode transitions, and there is no coordination among nodes in the larger network topology. We simulate this design in our comparative experiments; our implementation is described in Chapter 5.

3.1.3.6 Cross-Layer Power Management Summary

Table 3.1 compares five designs which extend existing ad hoc routing protocols. The *PM* column indicates whether or not a protocol takes advantage of MAC-layer power management. The chief factor in improving network lifetime seems to be the ability to minimize the number of **active** nodes. With BECA, AFECA, and On-demand Power Management, redundant nodes can be simultaneously active, while GAF and Span nodes "take turns" operating in high-consumption states. GAF and Span better manage redundancy because power management is *coordinated* among the nodes.

The **multihop power management architecture**, described in Chapter 4, is competitive with the best of the designs presented in Table 3.1. Experimental results show an average energy savings of 56—69%. We attribute this good energy performance to the on-demand nature of our protocol, which does not need to waste energy maintaining a high-performance forwarding backbone. Not only is average latency comparable to the case of no power management, but with improve-

Name	PM	Method	Energy	Latency
BECA	no	on-demand, timer-driven state machine; power down radio	35–40% energy savings, 20% network lifetime extension	20× increase in route setup latency
AFECA	no	on-demand, timer-driven state machine, sleep time influenced by neighbor density estimation	40–45% energy savings, 55% network lifetime extension	30× increase in route setup latency
GAF	no	proactive, timer-driven state machine, elect one active node per physical grid	40–60% energy savings, 200–300% network lifetime extension	2× increase in average latency
Span	yes	proactive, coordinator election; active nodes provide forwarding backbone	200–250% network lifetime extension	3× increase in average latency
On-demand	yes	on-demand, timer-driven PM suspension	50% energy savings	average latency similar to no PM, route setup latency similar to IBSS PM

Table 3.1: Summary of cross-layer power management designs.

ments to the 802.11 **ATIM** feature, we can achieve the low route setup latency of the proactive designs. In addition to good performance on these metrics, we will show how the use of negotiation among traffic sources, relays, and sinks can improve **worst case** as well as **average case** energy consumption.

3.2 Communications Performance

Several efforts from the broader field of ad hoc network communications performance are relevant to the current work. Section 3.2.1 presents studies of the scalability and performance of several 802.11 features. Methods to force nodes to behave cooperatively, thereby maximizing network communications performance, are described in Section 3.2.2. Section 3.2.3 shows recent theoretical work using the economic concept of *equilibrium* to derive bandwidth allocation and relay behavior in *ad hoc* networks. Finally, Section 3.2.4 presents recent work in the application of **mechanism design** to network route selection.

3.2.1 IEEE 802.11 IBSS Performance

This Section presents recent work studying the scalability of the 802.11 **timing synchronization function** (§2.1.1.2), as well as the performance of the rendezvous-based **power management** method (§2.1.3) under varying connectivity and load conditions. These results are not strictly related to the design of coordinated power management protocols, but we have found these papers helpful in extending the *ns* network simulator.

3.2.1.1 Fastest-Station Asynchronism

Clock synchronization among IEEE 802.11 stations in an *ad hoc* network is important for frequency-hopping coordination as well as for the use of power management. A formal analysis of the 802.11 **timing synchronization function** appears in (Huang and Lai, 2002). The authors prove that for single-hop Direct Sequence Spread Spectrum networks of size 100 nodes, the station with the fastest timer drift

— up to 0.01%, as permitted by the specification — will be out of synchronization with 15% of stations on average. Such network density is higher than the cases simulated in this thesis, but for completeness, station **timer drift** and the timer update algorithm have both been implemented.

3.2.1.2 Timer Synchronization and Network Partitions

The authors of (Tseng et al., 2002) extend the 802.11 **beacon** and **ATIM Window** features, described in Sections 2.1.1.1 and 2.1.3.1, respectively, to address clock synchronization issues. Specifically, they focus on asynchronism resulting from multihop network partitions, and from missed beacon receptions resulting from the use of power management. As a solution, multiple beacons are transmitted during ATIM Windows, and periodically stations suspend power management in order to discover and possibly resynchronize with neigbors. This design is evaluated in a custom simulator with parameters similar to *ns*: 2Mbps 802.11 with a 250m coverage radius. It is not clear how realistic this MAC implementation is, nor is a specific routing protocol named. Improvements in energy consumption are shown, but no analysis of network performance costs is given.

3.2.1.3 Dynamic Power Saving Mechanism

The 802.11 specification considers the duration of the ATIM Window to be a static parameter of the network. In (Jung and Vaidya, 2002), the ATIM Window size is adjusted dynamically (within the range [2ms, 50ms] for a 100ms beacon interval) based on load. For example, if a node cannot send an ATIM frame to all pending destinations because the current ATIM Window size is too small, then the Window size is increased. Although the authors use *ns* for evaluation, it appears that the experimental environment is restricted to single-hop ad hoc networks.

3.2.2 Enforcement Schemes

Section 3.1.3 presented techniques for reducing energy consumption in *ad hoc* networks. Nodes enter low-power states either by explicit coordination with other nodes, or by using timers based on communications activity. In either case, it is assumed that all nodes are willing to correctly implement the power-saving algorithm, and will correctly relay messages when expected.

A branch of research in recent years has considered the question of what to do if the designer *cannot* assume all nodes to be cooperative. One obvious reason why a node might not be willing to relay messages is the energy cost involved. A rational power-managing agent which controls node behavior might reason that relaying the messages of others provides no utility, while it does incur a drain on local energy reserves. Another scenario involves criminal organizations hoping to steal network service by tampering with their communications hardware; this is the specific motivation for the design in Section 3.2.2.1.

In any case, the techniques in this Section do not specifically address power issues. Rather, using a variety of enforcement schemes, the following designs attempt to *maximize* participation in the network *in spite of* a node's own (possibly conflicting) goals. These designs are presented to show that enforcement of group decisions in ad hoc networks is being studied. Enforcement is a complement to the work presented in this thesis: after nodes have executed a negotiation mechanism to determine a power-managing network configuration, *enforcement* provides part of the incentive to ensure that the nodes actually adhere to their agreements.

3.2.2.1 Nuglets

Nuglets (Buttyán and Hubaux, 2000) are a type of currency used to implement market-style interaction within a mobile ad hoc network. In this model, relays must be compensated (using *nuglets*) for their forwarding services. Nodes are motivated to accrue currency through self-interest; in order to transmit their *own* messages later on, a sufficient number of *nuglets* must be available for spending.

The motivation for *nuglets* is based on several assumptions about users (Buttyán and Hubaux, 2001a). Users are taken to have complete control over the functions of their mobile devices, and in particular may tamper with these functions in order to better satisfy local goals. This tampering is limited to the network layer and above; the physical and link layers are assumed to be impervious to modification. Finally, it is acknowledged that most users lack the sophistication needed to perform this tampering, but that some groups (*e.g.*, criminal organizations) do possess the requisite capabilities.

Two payment models are discussed: the **Packet Purse Model** and the **Packet**

Trade Model. Briefly, the former requires traffic sources to append a number of *nuglets* to each message such that each transit node which forwards the message can be paid. The latter places the payment burden on the traffic sink; each relay along the route buys the message from the previous node. Most attention in the *nuglets* literature is devoted to the Packet Purse Model. For example, one proposed extension to this model involves each relay executing a dynamic auction with its neighbors while forwarding a message, the goal being to obtain the best price for the next hop.

Performance evaluation of *nuglets* is accomplished through simulation, with the goal of maximizing network throughput. In (Buttyán and Hubaux, 2001a), pricing is determined using utility functions over node battery levels and *nuglet* reserves. The battery costs of transmitting a message are taken as proportional to the square of the transmit distance, and nodes do not move. The authors claim that, except for very low node battery levels, the *nuglet* mechanisms do not introduce substantial overhead (throughput remains within 5% of the non-*nuglet* network). In (Buttyán and Hubaux, 2001b), the authors consider network performance as a function of various decision rules which nodes may implement when forwarding messages. This time, battery considerations are ignored, and the decision rules reduce to inequality tests on the number of available *nuglets*. Here, the authors claim that the best strategy for transit nodes is to always forward (as opposed to forwarding only if available *nuglets* fall below some threshold).

Nodes participating in a *nuglet* network are assumed to include a tamper-proof module which implements the currency exchange mechanism. The authors reject the use of digital coins (Ferguson, 1993; Ferguson, 1994) on the grounds that such protocols typically require the services of a trusted **bank**. Instead, a **tamper-proof security module** on each node maintains the local *nuglet* counter in such a way as to enforce correct operation of the payment protocol. This adds complexity to the routing protocol in some cases; for example, the module must participate in route selection when using dynamic auctions. Also, because the module refuses to permit message transmission when no *nuglets* are available, it seems possible for nodes to "starve" if they happen not to be in a position to forward messages. Finally, because the *nuglet* counters are only synchronized periodically, it is possible for nuglets to be "lost" when nodes with a mutual outstanding balance move out of communications range of one another.

3.2.2.2 CONFIDANT

The *nuglets* system uses budget balance as an indicator of how well-behaved a node has been in the network. Because credit exchanges — based on budget balance — are one-to-one interactions, the designers of **CONFIDANT** (Buchegger and Boudec, 2002) claim that *nuglets* cannot address network-wide issues such as traffic diversion around misbehaving regions. CONFIDANT is a reputation-based system which uses announcements to indicate the trustworthiness of individual nodes. A trust table is maintained at each node, and this table is consulted by a routing protocol such as DSR. The routing protocol might steer traffic around known misbehaving nodes, or might penalize misbehaving nodes by not relaying their messages.

Through simulation (presumably *ns*), the authors have determined that a DSR network using CONFIDANT can function with as many as 60% of the nodes misbehaving. Misbehavior was limited to "no forwarding." Although the authors list power savings as one reason why a node might intentionally misbehave, energy is not addressed in the simulated results. It is unclear whether CONFIDANT alone could provide a minimum communications performance guarantee in the presence of individual power management.

3.2.2.3 Sprite

Sprite (Zhong et al., 2003) is a credit system which uses a centralized **Credit Clearance Service** for enforcement. Here, nodes periodically report their sourcing and relay activities to the centralized service, which then issues credit based on node behavior. Sources pay relay nodes for their service, but the amount of payment is related to the success of each individual relay activity. For example, in a multihop route, suppose that one node refuses to relay a message. Such a node would receive zero payment, the preceding node would receive a payment β, and all of the nodes preceding *that* node would receive $\alpha > \beta$.

Various precautions are taken to provide a disincentive for collusion against the Credit Clearance Service. The authors use game-theoretic analysis to claim that nodes have an optimal strategy of truth-telling when reporting receipts to the central service, and that this reporting is collusion-resistant. As with *nuglets*, the goal of this design is to maximize participation rather than to address any specific energy issue.

3.2.3 Economics and Ad Hoc Networks

Theoretical work on the conflict between self-interested users and the system-wide performance of ad hoc networks has emerged in the last year. These efforts seek to identify an **equilibrium** operating point for the network, such as a socially-optimal allocation of bandwidth or a stable relay-refusal rate. These approaches are abstract, and do not propose specific coordination protocols, nor do they examine interactions with existing protocols. Nevertheless, they point to an increasing interest in the use of economics and game theory to improve system behavior in networks of self-interested agents.

3.2.3.1 Pricing Bandwidth

Inspired by work on pricing in wired networks, (Qiu and Marbach, 2003) presents an iterative algorithm for maximizing joint user utility in bandwidth allocation. The authors note that their algorithm could be used to address battery issues. Specifically, they anticipate that traffic sources would adjust their message rate if their own energy reserves were low, or if their traffic flows transited nodes whose reserves were low.

Strictly speaking, this work describes an algorithm for a certain kind of utility maximization in an ad hoc network. It is not a protocol for exchanging credit like those of Sections 3.2.2.1 and 3.2.2.3. Nevertheless, it shows how economic reasoning can be used, assuming enforcement, to achieve resource-sharing goals in an ad hoc network.

3.2.3.2 Generous TIT-FOR-TAT

A sequential game modeled closely on the Iterated Prisoner's Dilemma problem is used in (Srinivasan et al., 2003) to control the rate at which forwarding nodes relay messages for others. The authors claim to be the first to "[apply] game theory to the problem of cooperation among nodes in an *ad hoc* network."[1] Variations on the TIT-FOR-TAT strategy are known to be successful for this problem (Axelrod, 1984). Here, nodes can choose to accept relay requests or not; this decision is made strategically. The authors derive the conditions for Pareto optimality of this acceptance rate subject to energy constraints. A **Generous TIT-FOR-TAT** (GTFT) strategy is described, in which a node chooses to relay or not relay based on what it has observed other nodes to do previously. (The strategy is "generous" in that it occasionally accepts relay requests even if selfish behavior was previously seen.)

The parameter being optimized by GTFT is the likelihood that a given node will deny a relay request from a given traffic source. Nodes are categorized ac-

[1]The prospectus for this thesis applied these concepts as well (Dorsey, 2001).

cording to energy capacity, and individual traffic flows take on the category of the lowest-capacity source or relay on a route. Relays decide to accept or deny a relay request for a flow in category j based on their history of interaction with flows of category j. For example, a node might refuse to relay a category-j flow if it has previously relayed more category-j traffic than has been relayed for it. The authors show that their algorithm converges to a sequential equilibrium where no source can obtain additional relay capacity for itself by deviating from the GTFT strategy.

Analysis of GTFT occurs in a highly idealized MATLAB environment. No specific network or link layer protocols are analyzed, nor is a specific topology or mobility pattern described. More seriously, the energy model used assumes that transmit-state power dominates, which is inconsistent with the behavior of popular spread-spectrum interfaces (§2.1.2). The fraction of successful relay requests is presented according to the energy categories described earlier. Nodes with the lowest energy capacity succeed about 10% of the time, while the highest-capacity nodes succeed about 80% of the time. Assuming that this fraction correlates with application message delivery ratio, a GTFT network would exhibit an end-to-end reliability much worse than that of conventional networks.

3.2.4 Mechanism Design and Routing

Mechanism design, introduced in Chapter 2, is a natural fit for network problems in which the routing nodes incur a **cost** for providing their service. This thesis is the first work to apply mechanism design to **power management** in ad hoc networks. It is also the first to apply mechanisms using a **real** ad hoc routing protocol tested under realistic conditions. This section presents recent theoretical work applying mechanism design to the problems of **interdomain routing** (§3.2.4.1) and **transmit power control** in ad hoc networks (§3.2.4.2). We organize the latter example under Communications Performance because it implements **topology control** rather than power management.

3.2.4.1 BGP Mechanism

The **interdomain routing** problem on the Internet is to choose routes between separate **administrative domains**. Currently, this problem is solved using the **Border Gateway Protocol**, BGP, which is a variation of a **distance vector** routing protocol. BGP routers proactively maintain routing tables for every known-reachable administrative domain. The table entries basically say, for each of a router's neighbors, "when you ask me to relay a message to a destination domain, I will use the route *P*." In BGP, routing table exchanges occur every time a link is added to or removed from the network. In addition, each router must know how to reach every other domain. These characteristics would make BGP poorly suited for use in mobile networks where links change frequently.

A mechanism for selecting **lowest-cost routes** in BGP has been proposed (Feigenbaum et al., 2002). In this method, routers include **prices** in the routing tables they exchange. Each router solves an **all-pairs lowest-cost path** problem using its view of the entire network, and the prices reported by the other routers. When a router receives a message addressed to some other administrative domain, it relays the message to the next hop on a lowest-cost path to that domain. The router eventually receives **Vickrey payments** based on the number of messages it has relayed. Although BGP is a very different kind of routing protocol from the one used in this thesis, the BGP mechanism is an interesting application of **strategy-proof pricing** to a real-world protocol. Several **distributed algorithmic mechanism design** aspects of this application have also been studied (Shneidman and Parkes, 2003; Feigenbaum and Shenker, 2002). Computationally-efficient Vickrey payments for a more abstract network environment are discussed in (Hershberger and Suri, 2001).

3.2.4.2 Ad hoc-VCG

A recent approach to the problem of **transmit power control** in ad hoc networks, **Ad hoc-VCG** (Anderegg and Eidenbenz, 2003) uses a **VCG mechanism** to select **minimum-transmit-power** routes. The design uses an abstract on-demand routing concept which lets the traffic **destination** choose which route the source will use. The sum of the transmit power levels of the nodes on a route is the **cost** of the route; the destination selects the cost-minimizing route.

The authors describe their assumptions about energy as follows:

> *The total energy of a routing path is the sum on the emission energy levels used at the source and at each intermediate node. We ignore other types of energy consumption such as listening to signals as they tend to be magnitudes smaller than the emission energy which grows with an exponent of one to six in the distance from one intermediate node to the next.*

Though no specific radio technology is named, a reference to modern “wireless cards” strongly suggests 802.11. Section 2.1.2 explained that transmit power is *not* a dominant source of energy consumption in spread spectrum systems such as 802.11. For these systems, adjusting transmit power achieves **topology control**, which is different from **power management** (the subject of this thesis). Regarding the magnitude of the “other types of energy consumption,” Section 2.1.2 showed that measured power rates for an 802.11 interface in the “listening” (idle) state are 55–77% of the transmit-state power (Feeney and Nilsson, 2001; Dorsey and Siewiorek, 2002; Ebert et al., 2002).

Using Ad hoc-VCG, traffic sources broadcast **Route Requests** as in DSR. Each node that propagates a Route Request chooses a **transmit power** $P_{\text{tx}} \leq \infty$; a large enough P_{tx} permits all destinations to be reached in a single hop. The node then appends to the Route Request a **report** of what transmit power level it used to send the Request. A node receiving the Route Request adjusts this reported level to what it thinks should be the **minimum** transmit power that could have been used. When the Route Request reaches the destination node, it contains a list of nodes on a route and the reported transmit power costs associated with each. The destination chooses the route for which the sum of transmit power costs is minimized. It then uses a VCG mechanism to compute **Vickrey payments** to each of the relays. A relay's payment is derived by computing the cost-minimizing route when that relay is removed from the network, and using the cost difference as a **Vickrey discount**.

Analysis of this VCG-based idea is limited to characterizing the **overpayments** made to the relays due to the Vickrey discounts. Evaluation occurs over **static topologies** where network **edge weights** are based on the distance between nodes and a **distance-power gradient** α relating transmit power to **range**. A receiver is within range if the received power level $P_{\text{rx}} = \frac{K}{d^{\alpha}} P_{\text{tx}}$, with $1 < \alpha \leq 6$, exceeds a threshold determined by the radio. By examining the minimum-cost routes between random source-destination pairs, the average overpayment for low $\alpha = 1.5$ was found to be 16–25%. For higher α, overpayment reaches as much as 500%. In sparse networks, it was found that sources could often have incurred lower costs by transmitting **directly** to destinations, rather than by paying relays.

Unlike the design of this thesis, Ad hoc-VCG is *not* an actual protocol. The authors describe their work as a "first step in designing a practical protocol that achieves truthfulness and cost-efficiency." The system we describe in Chapter 4 *is* a practical protocol which achieves these things.

Ad hoc-VCG includes some features which are unconventional for a VCG mechanism. The transmit power level reporting system violates the requirements for a

direct revelation mechanism, in which agents **observe** their types and then **directly reveal** them to the mechanism. Transmit power level reports are manipulated by other agents **by design**. If a node claims it used transmit power $\hat{P}_{\text{tx}}$ to transmit a Route Request, a relay propagating that Request modifies the report to be $\hat{P}'_{\text{tx}} \leq \hat{P}_{\text{tx}}$, where $\hat{P}'_{\text{tx}}$ is the minimum power level the relay **claims** could have been used. This is doubly complicating: nodes do not know their own **type** (true minimum transmit power level), and do not know what their eventual "report" to the mechanism will be! This opens the door for **multiple layers** of manipulation by the nodes, which **conflicts** with the definition of direct revelation mechanisms. Our design assumes that no tampering with agent reports occurs on the way to the mechanism; it would be possible to use cryptographic measures to ensure this.

Ad hoc-VCG has a number of other limitations not shared by our design. Like the **combinatorial reverse auction** interpretation of DSR from Chapter 2, it lacks the concept of an **unaffordable route**. Also, it is assumed that the source truthfully reports its own transmit power level! A manipulation exploiting this assumption can happen when the source increases transmit power such that it can directly reach the destination. If the source then **underreports** its transmit power level, this direct link becomes the cost-minimizing route. The source benefits from this approach when the power costs of the single hop are less than the Vickrey payments the source would have to pay to relays. The authors justify the truthful-source assumption by referring to the tradition of assuming that the **auctioneer** is not **self-interested**. Unfortunately, in Ad hoc-VCG this rôle is played by the **destination**, not the source. In our design, we make no such assumption, and allow sources to use their **dominant strategies** when reporting to the mechanism.

Ad hoc-VCG is the first published example of mechanism design for ad hoc networks. As a transmit power control design, it is fundamentally different from the work of this thesis, which focuses on **power management**. Because separate mechanisms are used for each **source-sink pair**, Ad hoc-VCG cannot account for **overlap** between routes. As we showed in Section 2.4.5, this means that Ad hoc-VCG is not **strategy-proof** in the power-managing environment. Ad hoc-VCG is also a conceptual design, rather than the practical protocol we present in Chapter 4. Finally, we have raised several questions about the **manipulability** of the mechanism itself, and have argued that our design does not share these issues.

3.3 Summary

This Chapter has presented related work from a range of disciplines, including **power management**, **enforcement schemes**, and **economics**. This thesis inherits from, and in some cases builds upon, these efforts. Our work is the first to combine practical protocols, realistic power models, and game-theoretic concepts to address the problem of energy consumption in mobile *ad hoc* networks.

We described several **proactive** and **on-demand** techniques for cross-layer power management in *ad hoc* networks. Proactive designs select a subset of nodes to remain active at all times and provide relay service. These offer good route setup latency, since the relays are already awake. Because relays may be active even if they are not servicing active traffic flows, the energy consumption of proactive schemes will typically be higher than in on-demand designs. Published energy savings for recent proactive schemes were in the range of 40–60% relative to non-power-managing networks. On-demand designs can achieve good energy savings, particularly in low-load networks, by only waking those relays needed for active traffic flows. The cost of existing on-demand designs is high **route setup** latency, which can be orders of magnitude worse than in non-power-managing networks

Enforcement schemes for *ad hoc* networks, both *with* and *without* a **central bank**, were presented. Existing credit systems deal in per-message payment, while in our environment, payment only occurs during periodic solutions of a **VCG mechanism**. Nevertheless, credit instrument design is complementary to our work, as credit provides the "strength" behind the **incentive compatibility** of any Groves mechanism for our environment.

Finally, we discussed recent applications of mechanism design to routing in the Internet and *ad hoc* networks. Existing designs do not support the **combinatorial exchange** concept required for an **optimal mechanism solution** in the power-managing environment. We have shown that in cases of route overlap, nodes experience a **marginal cost of energy** for additional traffic flows. Only a mechanism solution which considers this cost structure can be **strategy-proof**.

Chapter 4

System Design

To evaluate the **combinatorial exchange** concept for *ad hoc* route selection, we have developed a practical implementation of **exchange-based power management**. This Chapter presents the software architecture needed to support our design, and gives a full description of the protocol itself. Section 4.1 introduces our power management philosophy, and Section 4.2 details the cross-layer power management architecture. Section 4.3 shows how this architecture can be used to implement a simple **timer-based** power management scheme. In Section 4.4, we define the **Exchange Power Management** protocol, the core of this thesis. Section 4.5 describes the simple **agent valuation functions** we have developed to demonstrate exchange-based power management. Finally, in Section 4.6 we discuss future improvements to the **fault tolerance** and **scalability** of our protocol design.

4.1 Multihop Power Management Concepts

The basic premise underlying the **multihop power management** design is that low-level power management features **conflict** with good communications performance. Chapter 2 described the latency caused by 802.11 IBSS power management. In the multihop setting, we showed how the worst-case latency for DSR Route Discovery was $3\,(|P|-1) \times \text{BeaconInterval}$. We also showed that the worst-case latency for application message delivery is $(|P|-1) \times \text{BeaconInterval}$. These delays, which can be several seconds in length, are unacceptable in a modern network.

The approach we adopt is to allow low-level power management to operate while the network is quiescent, but to **suspend** power management while traffic is active. This **on-demand** philosophy achieves energy savings for nodes that are not servicing traffic flows, and high performance for active sources, relays, and sinks. The fundamental problem to solve at each node is how to choose the **intervals** during which that node will suspend power management.

This Section presents the basic tools needed to support multihop power management designs. We define concepts such as **power management suspension** and **fast wakeups** which are used by any instance of the architecture. Sections 4.3 and 4.4 build on these tools by providing specific methods for choosing the intervals of power management suspension.

4.1.1 Power Management Suspension

Section 2.1.3 described the IEEE 802.11 IBSS power management design, which lets stations enter a low-power **doze** state. Periodically, the stations **rendezvous** by waking up and exchanging **traffic announcements**. Stations that will send or receive data remain awake for the remainder of the **beacon interval**, while the others return to doze.

This design is **memoryless**: a station's traffic activity in the current beacon interval has no effect on its behavior in subsequent intervals. The idea behind **power management suspension** is that events such as the transmission or reception of data are **indicators** that a station is likely to be active in the future. Upon observing such an event, the station may **suspend** power management by no longer entering the doze state after each ATIM Window. This behavior continues until a **resume** event occurs. Section 4.3 implements the suspend and resume events using **timers**. For example, processing a DSR Route Request causes a short period of suspension, while processing an application message results in a longer suspension. Section 4.4 adds **negotiated** periods of suspension based on the results of a combinatorial exchange.

While a station is suspending, it still obeys all the rules of IBSS power management. Only **control** and **management** frames may be sent during the ATIM Window. Data frames for neighboring stations using power management must first be announced with an appropriate ATIM. The only difference is that suspending stations never enter the doze state.

The 802.11 layer is also supplied with the MAC addresses of neighboring stations that are themselves suspending power management. These addresses are stored in a **suspending neighbors list**. Directed data frames destined for a suspending neighbor do not need to be preceded by an ATIM, and can be sent as soon as possible (subject to the ATIM Window frame restrictions). This behavior is explicitly permitted by Section 11.2.2.1 of the 802.11 specification:

> *The estimated power-saving state of another [station] may be based on the power management information transmitted by that [station] and on additional information available locally, such as a history of failed transmission attempts. The use of RTS/CTS in an IBSS may reduce the number of transmissions to a [station] that is in [power-saving] mode. If an RTS is sent and a CTS is not received, the transmitting [station] may assume that the destination [station] is in [power-saving] mode. The method of estimating the power management state of other [stations] in the IBSS is outside the scope of this standard.* (IEEE, 1997)

It is worth noting that the part of this provision dealing with RTS/CTS is **invalid** for multihop networks. A station which fails to return a CTS should be treated as **out of range**; we adopt this assumption in our design.

To reduce congestion in the ATIM Window, directed ATIM frames are not sent to neighbors which are known to be suspending power management. Combined with the techniques described in Sections 4.1.2 and 4.1.3, it is possible to almost completely eliminate the use of directed ATIMs.[1]

Combining power management suspension with the **list of suspending neighbors**, stations can exchange data with the same latency they would experience *without* power management. The only exception to this occurs during the ATIM Window, where data frames are not permitted. These techniques solve the latency problem for multihop application message delivery. As long as all of the nodes along a route are suspending (and are known by their neighbors to be suspending), end-to-end latency is comparable to the case of **no power management**. Again, the only exception to this occurs if the message is generated by a traffic source *during* an ATIM Window, as the message must be delayed until after the Window ends.

4.1.2 Fast Wakeup

Section 2.2.2 revealed an important problem in the interaction betewen DSR **Route Discovery** and 802.11 **IBSS power management**. The propagating broadcast of **Route Requests**, and the subsequent unicast return of **Route Replies**, experienced much higher latency under power management than without. We showed that the cause of this higher latency was the **ATIM** traffic announcement process.

We have developed an extension to 802.11 which permits Route Discovery to

[1]An example of when directed ATIMs are still used happens after a DSR route breaks. The traffic source may send out subsequent application messages on another cached route containing new relays, some of which are not yet known to be suspending power management.

occur with much lower latency. This extension, called **fast wakeup**, does not require changes to any 802.11 frame type, and does not break compatibility with existing 802.11 implementations. The method changes the handling of ATIM management frames sent to the **broadcast** address. Such frames would be scheduled whenever a DSR Route Request was passed to the MAC layer, if a broadcast announcement had not already been made in the current **beacon interval**. Procedure 1 shows the new handling of broadcast ATIM frames.

Procedure 1 HANDLE-ATIM(*a*)

```
1: Let $a$ be a received ATIM frame
2: Let $s$ be the number of broadcast ATIM frames sent in this beacon interval
3: Let $r$ be the number of broadcast ATIM frames received in this beacon iterval
4: if $address(a) = \text{Broadcast}$ then
5:     $r \Leftarrow r + 1$
6:     if $s = 0$ then {only generate or propagate one broadcast ATIM this Interval}
7:         if $r = 1$ then
8:             SET-ATIM-HOLDOFF(0) {helps to avoid broadcast ATIM collisions}
9:         ATIM-ENQUEUE($address(a)$) {propagate ATIM to the broadcast address}
```

We implement a **random holdoff** on the transmission of broadcast ATIM frames. The purpose of the holdoff is to try and avoid undetectable collisions between broadcast ATIM frames. Since neither **RTS-CTS** nor **ACK** are used for broadcast ATIM frames, **hidden terminal** problems (§2.1) apply. The holdoff timer is set by Procedure 2 when a broadcast ATIM becomes available for transmission. In the procedure, we choose a random timer value of up to Holdoff-Interval (4ms in our implementation) or the remainder of the ATIM Window minus the time required for an ATIM transmission,[2] whichever is least. To this random holdoff we add a **minimum** holdoff time, and scale the whole value by the worst-case station timer drift. (Max-TSF-Drift is 0.01%, per Section 11.1.2.4 of the 802.11 specification.) Since a station does not know its own drift, this final adjustment ensures that the holdoff will not go past the end of the effective ATIM Window.

Procedure 2 SET-ATIM-HOLDOFF(t)

```
Let $t$ be the minimum holdoff time
Let $W$ be the time remaining in this ATIM Window
Let $A$ be the time required to transmit an ATIM frame
**if** the holdoff timer is already running **then**
    Return.
**if** $W - A < t$ **then** {insufficient time left in the Window}
    Abandon ATIM transmission and return.
$\mathit{max\text{-}holdoff} \leftarrow \min(W - A - t, \text{Holdoff-Interval})$ {don't go past Window end}
Set holdoff timer to $\dfrac{t + U[0, \mathit{max\text{-}holdoff}]}{1 + \text{Max-TSF-Drift}}$
```

At the beginning of every ATIM Window, if broadcast **data frames** are scheduled, we enqueue a broadcast ATIM and call Procedure 2 specifying a minimum holdoff of 2 milliseconds. This minimum holdoff is used to address **skew** issues related to the **loosely-synchronized station timers** (§2.1.1.2). Not all stations will have precisely aligned ATIM Windows, so a "faster" station might enter its Window ahead of its neighbors. Accordingly, we wait for the minimum 2-millisecond holdoff to expire before sending broadcast ATIMs to increase the probability that neighboring stations are prepared to receive ATIM frames.

[2] A includes enough time for one **backoff** given the current **contention window** size, an ATIM frame transmission, an Acknowledgment frame transmission, and the requisite **inter-frame spacing** between the frames. Since this duration is based on a unicast ATIM transaction, it actually overstates the amount of time needed for a broadcast ATIM transmission.

The goal of fast wakeup is to reach **every** station in the IBSS in a **single** ATIM Window. With every station awake, and knowing that every *other* station is awake, DSR Route Discovery can complete as quickly as it would **without** power management. As with the latency improvement we claimed for power management suspension, there is a caveat for the latency improvement of fast wakeup. If a DSR Route Request becomes available in the post-ATIM Window portion of the current beacon interval, fast wakeup does not occur until the ATIM Window of the **next** beacon interval. Following *that* Window, Route Discovery proceeds at full speed.

It is reasonable to ask about the **scalability** of the fast wakeup technique. The **radius** of the IBSS reached by propagating broadcast ATIM frames is a function of the **size** of the ATIM Window and the level of **congestion** in the ATIM Window. Suppose a broadcast data frame, such as a Route Request, is enqueued for transmission at or before the start of an ATIM Window. In our implementation, there is a holdoff on the broadcast ATIM transmission of 2 milliseconds plus an additional random delay of up to 4 milliseconds. Following that holdoff, the broadcast ATIM frame becomes the **highest-priority** transmission among all enqueued ATIM frames. Supposing a 40-millisecond ATIM Window, which we use in our work, the fast wakeup technique has at least 34 milliseconds in which to reach as many stations as possible. This is a fairly long time by 802.11 measures. For example, without power management, unicast DSR messages require about 10 milliseconds to traverse a network 1,000 meters in diameter.

We take several steps to manage congestion in the ATIM Window. As explained at the end of Section 4.1.1, directed ATIM frames are rare when **power management suspension** and **suspending-neighbor lists** are used. This means that the ATIM Window is mostly free to accommodate broadcast ATIM transmissions. Also, we limit each station to a **single** broadcast ATIM frame transmission per ATIM Window. This echoes the DSR concept of **controlled flooding** for Route Request propagation.

When the radius covered by fast wakeup does not encompass the entire IBSS, Route Discovery can still complete faster than it would without fast wakeup. An example scenario happens when a traffic source initiates Route Discovery late in an ATIM Window. Broadcast ATIMs propagate out to a limited radius, but stations beyond that radius return to the **doze** state following the ATIM Window. When the Route Requests are subsequently propagated, they can reach any station[3] within the fast wakeup radius. At the fast wakeup frontier, DSR will enqueue a broadcast Route Request frame at the MAC layer, which will trigger another fast wakeup in the *next* ATIM Window. Generally, this second fast wakeup is able to reach the remainder of the IBSS stations, permitting Route Discovery to complete in a total of **two** beacon intervals, which is less than the $3\left(|P|-1\right)$ intervals required without fast wakeup.

[3]Of course, this does not mean Route Requests reach *every* station within the radius. Route Requests are transmitted as **unreliable broadcasts**, meaning that frame collisions can interfere with propagation. Also, DSR nodes use a **controlled flooding** technique to limit the spread of Route Requests.

Fast wakeup not only speeds the distribution of broadcast Route Requests, it also solves the problem of latency with the unicast **Route Reply** messages. In the worst case, with cold **ARP caches**, Route Replies account for $2(|P|-1)$ beacon intervals of latency in the Route Discovery process. With fast wakeup, a node can transmit a broadcast **ARP request** without delay, since the node has already announced the broadcast address in the current beacon interval. The directed **ARP reply** can be transmitted by the target to the requester because the requester is known to be awake, since it made a traffic announcement in the current beacon interval.

Finally, the directed Route Reply can be sent as soon as an ARP address translation is available, for two reasons. For beacon intervals in which a station transmits a broadcast ATIM frame, *if and only if* a directed ATIM to a destination did not fail in the current Interval, then that destination is assumed to be **awake** following the ATIM Window. Therefore, following a fast wakeup, the directed Route Reply can be sent immediately.

The second reason has to do with the **suspending neighbors list**. If, upon processing a Route Request, a node suspends power management, and knows that the **predecessor** node in the route has also suspended power management, then the node can inform the MAC layer that the predecessor is suspending. When returning the directed Route Reply, this predecessor node is a **next-hop** in the unicast route, so the Reply frame can be transmitted to it without delay. This second method presumes that an **IP-to-MAC translation** for the predecessor node's address is available at the time the Route Request is processed. Section 4.1.3 presents a method to ensure that such a translation *is* available.

4.1.3 ARP Snooping

The DSR implementation at a network node processes all DSR messages addressed to that node or to the broadcast address. It also **promiscuously** examines all DSR messages **overheard** on the wireless medium. The address information contained in these messages can be used to "**pre-heat**" the **ARP cache**.

Each DSR message bears the **IP address** of the most recent node to transmit the message. DSR messages are of course sent as the payload of a MAC-layer **data frame**. We assume that DSR has access to the MAC frame and its headers; this is reasonable for a **kernel-mode network-layer** implementation. Each DSR message either **received** or **overheard** by a node therefore contains both a **source route header** and a **MAC header**. These headers provide an **IP-to-MAC translation** for the address of the transmitting node.

Given the information provided in each DSR message, the **ARP snooping** technique is obvious. For every DSR message received or overheard by a node, we insert an address translation into the ARP cache for the sender of that message.

ARP snooping **totally eliminates** the need for explicit ARP requests. A DSR node never *transmits* a directed data frame to a neighbor from which it has not previously *received* at least one frame. For example, during Route Discovery, nodes send broadcast Route Requests (which do not require address translation) before sending directed Route Replies. A node which transmits a Route Reply to a neighbor **necessarily** has received a Route Request from that neighbor, and therefore has already added an address translation for the neighbor to its ARP cache.

4.1.4 Route Fidelity

Power management suspension, introduced in Section 4.1.1, reduces application message delivery **latency** using **suspending neighbor lists**. When a station knows that the **next hop** in a route is suspending power management, it can send **data frames** to that station without delay. We have not specified *how* stations come to know the power management state of their neighbors. Sections 4.3 and 4.4 describe methods by which power management suspension is configured, and through which neighboring stations can learn about each others' states.

An attribute that both methods share is their need for **consistency** in route usage. The **on demand** nature of both methods means that the nodes which are needed to support active **traffic flows** suspend power management, while other nodes continue to use power management. For this reason, it is desirable for a traffic source to choose a route and use it consistently; only the relays on that route need to incur the higher **energy costs** of suspension.

DSR chooses a route for each outgoing message **independently**. This means that the route chosen by DSR can vary from message to message as a traffic source learns about additional routes. We override this behavior using the **route fidelity** technique, which intercepts DSR route selection and uses custom selection logic. For example, Section 4.4 describes a method by which routes are selected according to the results of a **combinatorial exchange**; route fidelity allows the source to adhere to those results.

Nodes not on the route used by the route fidelity technique **resume** power management, either by timing out or by implementing the results of an exchange. If a source needs to **change** to a new route, as happens when its previous route **breaks**, the nodes on the new route may have resumed power management. This means the new route cannot be immediately used with low **latency**; an application message sent to a power-managing node must wait for the **ATIM** announcement process to complete.

When a source changes to a new route from the one previously chosen by route fidelity, the source triggers a **fast wakeup**. This is essentially a **forged** broadcast ATIM, which serves to wake up all stations in the IBSS. The nodes on the new route can then suspend power management, while the other nodes eventually resume power management.

4.2 Multihop Power Management Architecture

We have developed a **cross-layer power management architecture** which supports the concepts described in Section 4.1. Our design requires modifications to Dynamic Source Routing, the DSR route cache, the ARP cache, and the MAC layer. The design is modular, so different **multihop power management** methods can be easily tested. Sections 4.3 and 4.4 describe such methods.

Figure 4.1 illustrates the interaction between multihop power management and the various layers of the protocol stack. The design is driven primarily by **events** within DSR. For example, we **intercept** lookups into the DSR route cache, and **observe** messages sent, received, or overheard by DSR. In response to these events, we perform actions on the DSR route cache, the ARP cache, and the MAC layer. In the Figure, actions in *(parentheses)* are used only by the method of Section 4.4.

4.2.1 Network Layer Interactions

Half of the actions in the multihop power management design involve Dynamic Source Routing and the DSR route cache. DSR **exposes** the route cache to the power management implementation directly.

Get Route

> This action intercepts DSR route cache lookups for normal application traffic. DSR names a destination, and the power management instance supplies a route of its choosing, if one is available. This is the method by

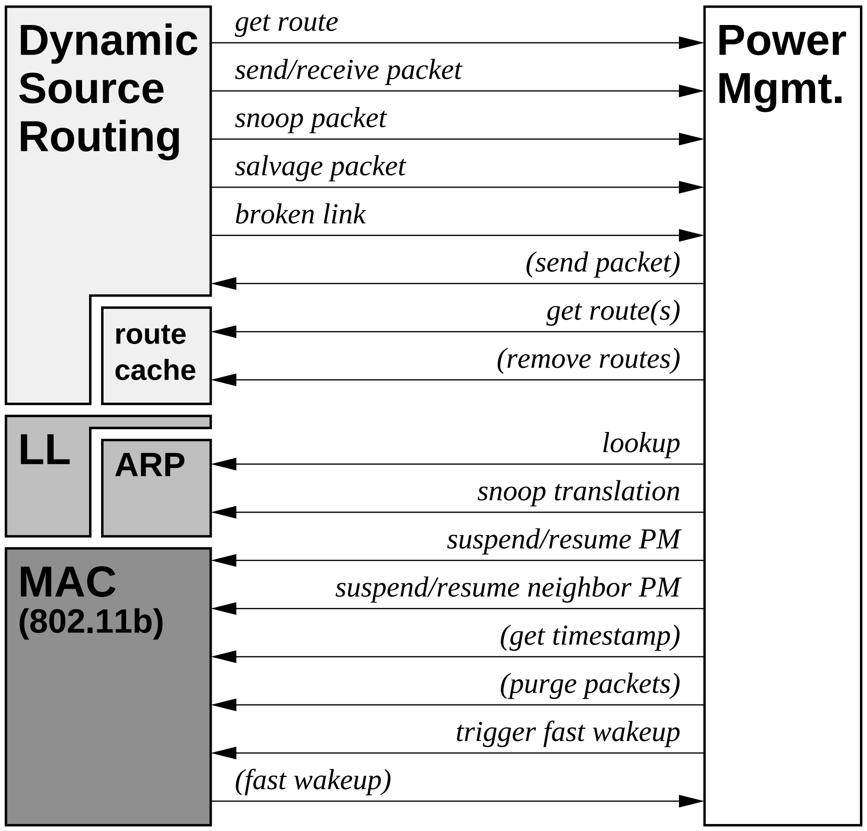


Figure 4.1: Multihop Power Management layer interaction.

which **route fidelity** is implemented.

Send/Receive Packet

Every message sent (as a traffic **source**) or received (as a **relay** or **sink**, or messages to the **broadcast** address) by DSR is shown to the power management instance. Power management can use the message type to set suspension timers, and can use address information in the message to configure the **suspending neighbors list**. Power management can notify DSR that it has **consumed** the message, meaning that DSR does not need to perform additional processing on the message. This is useful for messages passed between power management instances running on different nodes, using DSR as a **transport**.

Snoop Packet

A DSR node **promiscuously** monitors the wireless medium, and processes messages that it overhears. Such messages are shown to the power management instance, which can use them for **ARP snooping**, or to infer links in the local network **topology**.

Salvage Packet

In some circumstances, a DSR node may want to know a route to a destination, but not because it is sourcing traffic for that destination. One example is a node trying to **forward** a message along a **source route**, where the next-hop link has broken. If the node knows another route to the message's destination, it can **salvage** the message by replacing the defunct source route with a new, hopefully better, route. The **salvage route** is probably not going to be long-lived, so it is not necessary to trigger **fast wakeup** or apply **route fidelity** constraints. We accept that the salvage route may contain power-managing nodes, and may not provide low **latency**.

Broken Link

A DSR node learns that a link has **broken** either directly from the link layer, or by receiving a **Route Error** message. For example, 802.11 notifies DSR whenever it sends an **RTS** but does not receive a **CTS**, when it sends a

data frame but does not receive an **acknowledgment**, or when it sends a directed **ATIM** and does not receive an acknowledgment. The names of the nodes on either side of the broken link are shown to the power management instance. Power management can then check to see if the broken link affects any of the routes being maintained under **route fidelity**. Also, broken link notification can be used to update the power management instance's image of the local network topology.

Send Packet (to DSR)

Power management instances running on different nodes may **coordinate** themselves by passing messages. The method described in Section 4.4 uses this technique. Power management may construct a message with a DSR **source route**, which DSR will deliver either to an immediate neighbor, or to more distant nodes using a **multihop route**.

Get Route or Routes (from DSR route cache)

A power management instance intercepts DSR **route lookups**, and returns routes under **route fidelity** constraints as described earlier. In order to provide such routes, power management must have access to the DSR **route cache**. The route cache exposes two operations to power management. The first simply returns a single route to a named destination (if one is known) using whatever **selection criteria** the cache implements (*e.g.*, a shortest route). The other operation returns *all* known routes to a destination, thus permitting power management to implement its own selection.

Remove Routes (from DSR route cache)

The DSR route cache may maintain routes for a long time, and some routes may become **stale**. Power management may request that all routes to a named destination be **evicted** from the cache, so that known-good routes may be discovered.

4.2.2 Link Layer Interactions

The **link layer** contains the **ARP cache**, which records translations between **IP addresses** (used by DSR) and **MAC addresses** (used by 802.11). The ARP cache is exposed to the power management instance for **lookup** and **insert** operations.

Lookup

In our **multihop power management** design, the MAC layer maintains a **suspending neighbors list**. This list records the MAC addresses of neighboring stations that are known to be suspending power management. As described in Section 4.2.3, the power management instance notifies the MAC layer when a particular neighbor has **suspended** or **resumed** power management. To do this, power management must know an **IP-to-MAC address translation**, since DSR names nodes by their IP addresses. This translation is available from the ARP cache.

Snoop Translation

As described in Section 4.1.3, when DSR receives or overhears a message from another DSR node, the message contains both an IP and MAC address for that node. Power management can **manually insert** an IP-to-MAC address translation into the ARP cache based on the message addresses.

4.2.3 MAC Layer Interactions

In Section 4.1.1 we argued that multihop communications performance could be improved in power-managing networks through **power management suspension**. Multihop power management instructs the MAC layer[4] to **suspend** and **resume** its own power management features. It also informs the MAC layer about the power management status of neighboring stations.

Suspend and Resume Power Management

In response to on-demand activity in the network, the power management instance may determine that it should direct the MAC layer to **suspend** power management. For example, after observing incoming messages, or in response to a negotiation procedure, a node might be needed to service an active traffic flow. For performance reasons discussed in Section 4.1.1, the node's MAC implementation should no longer enter the **doze state** after each **ATIM Window**. When the period of on-demand activity has passed, the power management instance may direct the MAC layer to **resume** use of the doze state.

[4]Technically, medium access control is a **sublayer** of the link layer (Tanenbaum, 1996). It is easier for presentation to treat it separately.

Suspend and Resume Neighbor Power Management

The complement to power management suspension which allows latency improvements to be realized is the **suspending neighbors list**. This list is maintained within the MAC layer; addresses are added to or removed from the list at the direction of the power management instance. The manner by which power management learns the state of its neighbors is method-defined. Section 4.3 uses common-knowledge suspension timer values, while Section 4.4 adds explicitly-negotiation suspension intervals.

Get Timestamp

Each 802.11 station maintains a **timing synchronization function** timer, which is loosely synchronized across all **IBSS** stations. This 64-bit, microsecond-resolution timer may be useful to power management instances. For example, the method of Section 4.4 uses timers to elect the **root** of a **spanning tree**, and to recognize **stale** messages from old negotiations. In the 802.11 specification, this timer is permitted to have a **drift** of $\pm 0.01\%$; the exact drift is not known to the station.

Purge Packets

Messages may be buffered by 802.11 while waiting for the wireless medium to become available, or while waiting for **ATIM** announcements for power-managing stations to complete. The power management instance may decide that its own buffered messages are **stale**, and are not worth transmitting. For example, the method of Section 4.4 **abandons** the negotiation procedure if it is taking too long. Negotiation messages transmitted after abandonment serve no purpose, and are purged. Messages are purged by **type**; for example, “CBR” or “DSR.”

Trigger Fast Wakeup

A power management instance may signal the 802.11 implementation to transmit a broadcast **ATIM frame** in the next **ATIM Window**, even if it has no broadcast **data frames** to send. This “forged” traffic announcement wakes all IBSS stations. This is desirable when a traffic source changes routes, since the some of the nodes on the new route may be dozing.

Fast Wakeup (to Power Management)

The final interaction between the existing protocol stack and **multihop power management** is delivered by 802.11 to the power management instance. At the conclusion of an **ATIM Window** in which a station either **sent** or **received** a broadcast ATIM, the MAC layer notifies power management that a **fast wakeup** (§4.1.2) was observed. The power management instance described in Section 4.4 treats this event as the signal to start a negotiation procedure.

4.3 Local Power Management

Using the actions presented in Section 4.2, we can implement a simple **timer-based power management** design in the style of **BECA** (§3.1.3.1) or **On-demand Power Management** (§3.1.3.5). This design, called **Local Power Management**, or LPM, uses only *local* information about traffic flows when making **power management suspension** decisions. When a node processes a message, such as a DSR routing message or a CBR application message, it directs the MAC layer to suspend power management. After a period of **inactivity**, power management is allowed to resume. The purpose of LPM is to establish a baseline performance level for the **multihop power management architecture**. In a sense, LPM simply enhances DSR by improving **latency** under power management. LPM does *not* use **exchange-based negotiation** techniques; these will be introduced in the design of Section 4.4.

4.3.1 Active Destinations

Section 4.1.4 introduced the concept of **route fidelity**, which enforces **consistency** across the **source routes** used for successive application messages. An active traffic source has one or more **active destinations** to which it has recently sent traffic. LPM associates **state** with each active destination, such as whether a route has been discovered for it, the route itself, and the time since the last use of that route.

The first time DSR requests a route to a destination (using the **get route** action, Section 4.2.1), LPM begins by checking the **DSR route cache** to see if a route is known. If one *is* known, the route preferred by the route cache is established as the **active route** for the named destination. The MAC layer is instructed to trigger a **fast wakeup** (§4.2.3), and the active route is returned to DSR. The purpose of the wakeup is to transition the network to a low-latency state, since a new route is about to be used.

If a route is *not* known to the new active destination, DSR behaves as it normally would in the event of a route cache miss: it begins **Route Discovery**. By enqueueing a broadcast **Route Request** message at the MAC layer, DSR itself triggers a fast wakeup. Upon receiving a **Route Reply**, DSR performs the **get route** action for any buffered messages awaiting source routes. This time, since a cached route *is* known, LPM can establish an active route as described above.

Once the state for an active destination has been configured, subsequent **get route** actions by DSR are handled by returning the active route. After a short delay, 250 milliseconds in our implementation, an active route is **finalized**. A route is finalized by asking the DSR route cache for its preferred route to the active destination. The finalized route may be different from the original route if the source has learned better routes in the interim. For example, the first route discovered may not be the shortest one; subsequent **Route Replies** may reveal better routes. Nodes which participate in DSR Route Discovery will still be awake at the time of finalization; Section 4.3.2 describes this in more detail. As such, if the finalized route differs from the original active route, the new relays are still in a **low-latency**

state and can accept the application traffic.

When a link **breaks** due to **mobility** or other factors, a traffic source with an active route containing that link must choose a new active route. A source learns about such a link by receiving a **DSR Route Error**, or by an indication from the **MAC layer** that transmission to a neighboring station has failed. In both cases, all active routes are examined to see if they contain the broken link. For those that do, the associated active destination state is **torn down**. If DSR subsequently performs a **get route** action for such a destination, the request is handled as for a *new* active destination, described above.

When a DSR node discovers that it cannot pass a message to the next hop named in a source route, it tries to **salvage** the message. Using the **salvage packet** action (§4.2.1), DSR asks for a new route to the message destination. If the destination is already an **active destination** for the node, then LPM prefers to respond with the **active route** for that destination. Otherwise, LPM consults the DSR route cache for its preferred route. If a route is known, the MAC layer is instructed to trigger a **fast wakeup** and the route is returned to DSR. Otherwise, DSR **drops** the message.[5]

If an active route has not been used for some period of time, the active destination **times out**. In our implementation, timeout occurs after 500 milliseconds of inactivity. Timeout values should be chosen to correspond with the duration of

[5]DSR relies on higher protocol layers to ensure **reliable delivery**. Workloads such as the CBR application used in this thesis do not **retry** messages that are dropped in this manner.

power management suspension implemented by the relays and sink on an active route, discussed in Section 4.3.2. After this period of inactivity, the relays and sink will **resume** power management, so the source should tear down its active destination state. Shorter timeout values result in more aggressive **energy savings**, while longer values are more tolerant of **bursty application traffic**. The topic of adapting timeout values to different traffic characteristics is left to future work.

4.3.2 Message Handling

When a DSR node sends or receives a message, it passes a copy of the message to LPM using the **send/receive packet** actions (§4.2.1). LPM responds by instructing the MAC layer to **suspend power management** (§4.2.3). After a period of inactivity, LPM instructs the MAC layer to **resume** power management. This period can be different for each message type, but in our implementation, both DSR and CBR messages cause 500 milliseconds of suspension.[6] All nodes use the same suspension periods, and it is **common knowledge** among the nodes that they all use the same periods. Note that when the MAC layer receives the resume instruction, it may not *immediately* return to the **doze** state. If the resume is ordered in the post-**ATIM Window** portion of the **beacon interval**, the station will stay awake until (at least) the end of the *next* ATIM Window.

The **source route** contained in a message processed by DSR yields information

[6]In particular, the CBR suspension period is defined to be the **active route timeout** (§4.3.1).

about the power management state of neighboring nodes. For example, a node which sends a message (*e.g.*, a DSR Route Reply or a CBR message) along a source route knows the identity of the **next hop** on that route. Because the per-type suspension periods are common to all nodes, the sender knows that the neighbor will suspend its own power management upon receiving the message. The sender instructs its own MAC layer to add the neighbor to the **suspending neighbors list** (§4.2.3). When adding a neighbor to the list, the neighbor is named by its **MAC address**, obtained from the **ARP cache** (§4.2.2). The sender then sets a timer for that neighbor, which in the absence of further interaction with the neighbor, will expire after the appropriate suspension period. Upon expiry, the MAC layer is instructed to remove the neighbor from the suspending neighbors list.

A node which receives a message from a neighbor updates the suspending neighbor list as well. The **previous hop** on the source route is passed to the MAC layer for neighbor suspension, and again a timer is set. There is an additional case involving **DSR Route Request** messages which applies when such messages are *received*, but not when they are *sent*. The previous hop on the route whose discovery is in progress can be treated as a suspending neighbor. The next hop, to which the Request will next be sent, is of course *unknown* (and there may in fact be *many* next hops).

When a node is informed by the MAC layer that a link to its neighbor has **broken**, it may have to revise its beliefs about the **power management state** of that neighbor. For example, when a node sends a CBR message to its neighbor, it assumes that the neighbor will suspend power management for the duration associated with CBR messages. If the CBR message is *never received*, then this assumption becomes invalid. LPM responds to broken neighbor links by **canceling** the neighbor suspension timer. Specifically, each neighbor has *several* timers, one for each message type (*e.g.*, CBR, DSR). LPM looks at the type of the message which encountered the broken link, and cancels the timer for that type. The **resume neighbor power management** action is sent to the MAC layer when the timers for *all* types have expired.

Finally, every message **received** or **overheard** (promiscuously) on the medium is examined to provide a **snoop translation** (§4.2.2) to the **ARP cache**. The **IP address** of the sender is listed in the DSR **source route**, while the **MAC address** is found in the **MAC header** passed up from the MAC layer. This address pair is an **IP-to-MAC address translation**, which is manually added to the ARP cache to help with neighbor power management suspension.

4.3.3 Summary

We have described **Local Power Management**, a timer-based instance of the **multihop power management architecture**. LPM uses *local* information about traffic activity to configure **power management suspension**. Source routes are chosen by the DSR route cache (using a **shortest-route criterion** in our implementation).

We impose **route fidelity** constraints on the route selection to allow the network to converge to a small set of active nodes.

LPM improves the performance of DSR in power-managing environments by reducing the **latency** of both Route Discovery and application message delivery. Route Discovery performance is improved through the **fast wakeup** technique. Application messages benefit from power management suspension and **neighbor** power management suspension lists.

Timer	**Duration**
Active route finalize	250ms
Active route timeout	500ms
CBR PM suspension	500ms
DSR PM suspension	500ms

Table 4.1: LPM timer durations.

Table 4.1 summarizes the LPM timers. All LPM nodes implement the same timer values, and it is **common knowledge** among the nodes that these values are used by all nodes.

LPM demonstrates the possibilities for improved latency in on-demand power-managing multihop 802.11 networks. It does not attempt to **shape** the energy consumption of individual nodes. Rather, node energy consumption is still a function of how often a node is chosen by traffic sources to be active. In Section 4.4, we describe a protocol that allows nodes to report their **preference** for activity, and thereby influence the selection of **source routes**.

4.4 Exchange Power Management

Exchange Power Management, or XPM, is another instance of the **multihop power management architecture**. Like **Local Power Management** (§4.3), XPM uses techniques such as **route fidelity** and **fast wakeups** to improve the performance of DSR in power-managing networks. The contribution of XPM is its use of a **combinatorial exchange** to select source routes, rather than letting sources choose their routes independently.

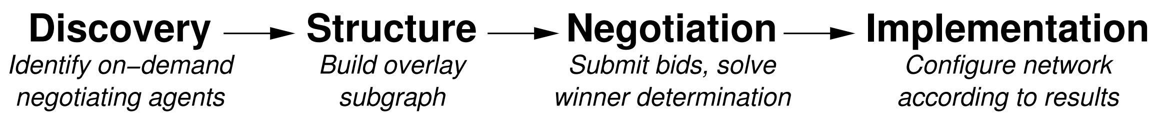

Discovery
Identify on–demand negotiating agents
Structure
Build overlay subgraph
Negotiation
Submit bids, solve winner determination
Implementation
Configure network according to results

Figure 4.2: Four-phase on-demand negotiation framework.

XPM uses a four-phase on-demand **negotiation framework**, shown in Figure 4.2. When routes need to be selected, active sources begin by entering the **Discovery** phase, in which they determine the routes for which they will **bid**. Once routes are known, nodes enter **Structure**, in which the relays and sinks on those routes are signaled to participate in a negotiation. An **overlay subgraph** linking the negotiating nodes is constructed. With this overlay structure in place, nodes submit bids and asks in the **Negotiation** phase. The winner determination problem is solved, and agent payments are computed; the results are then distributed among the agents. Once the results are known, the agents enter **Implementation**, where sources set their active routes, and relays and sinks configure power management. These phases are not **globally synchonized**; instead, they are defined by **precondi-**

tions at each node. Nodes independently progress through the sequence of phases as these preconditions are satisfied.

In this thesis, we focus on specific protocols and algorithms in each phase which are relevant to the problem at hand. However, the phases are essentially **orthogonal** to one another. For example, the **greedy spanning tree** algorithm we describe for Structure could be replaced by a **balanced** tree formation algorithm. The **heuristic search** method we describe for winner determination could perhaps be replaced by a **linear programming** approach. Rather than having relays and sinks implement a power management state, they could configure a **bandwidth reservation** for traffic shaping. So, while the results of this thesis focus on an instance of the framework which supports negotiated power management, the framework could be applied in a variety of other problem domains.

4.4.1 Discovery Phase

The prerequisite for entering **Discovery** is that a node does not know any routes for at least one of its **active destinations**. Discovery is entered following the receipt of a **fast wakeup** (§4.1.2). A node that already knows routes to its active destinations skips Discovery and proceeds to **Structure** (§4.4.2), as does a non-source node (which has no active destinations).

As with LPM, DSR performs the **get route** action (§4.2.1) to request a route to a named destination. If that destination was not already active, XPM begins by instructing the **DSR route cache** to **remove routes** (§4.2.1) to that destination. For a destination d, this is the equivalent of discovering that all links $(\cdot, d)$ have broken.[7] This purge avoids generating bids for **stale** routes, since routes which resided in the cache for a long time may no longer work. Unlike the basic DSR route selection problem in which only *one* working route must be found, we would like to identify (and bid on) *several* working routes at once.

After the route cache has been purged, the state for the new active destination is established. We again set up a **finalize timer** (§4.3.1) for the destination. Since it is helpful to discover as many routes as possible, the timer is set to a longer duration — 750 milliseconds — than in LPM. Expiry of the finalize timer will cause the source to begin contacting sellers in preparation for bidding.

After setting up the active destination state, XPM reports to DSR that no cached routes to the destination are known. This causes DSR to begin **Route Discovery**, in which broadcast **Route Request** messages are propagated throughout the network (§2.2.1). The **controlled flooding** technique used by DSR to limit propagation determines which routes will actually be discovered. Let $\langle P, n \rangle$ be a Route Request arriving on route $P = (p_1, p_2, \ldots, p_{|P|})$ having sequence number n. DSR normally **prunes** the Request if it has already seen a Request from source p_1 having sequence number n. This behavior is reasonable if the only goal is to quickly discover the **shortest route** to a destination.

[7]This approach is inefficient if cached routes to other active destinations happen to traverse d. A more elegant solution is left to future work.

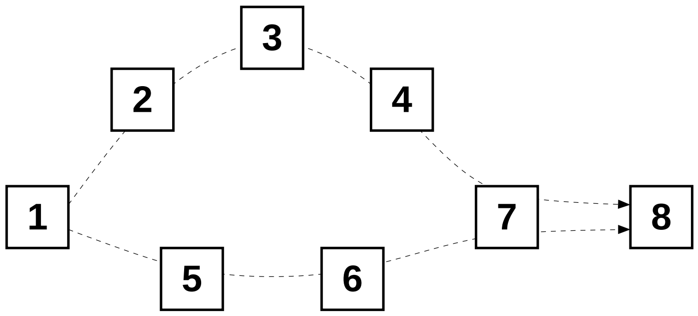


Figure 4.3: DSR may fail to discover both routes for the flow $1 \rightsquigarrow 8$.

With XPM, the goal is instead to discover several **diverse routes**. Diversity is important because it allows sources to bid over a variety of **alternative** bundles. DSR's pruning rule fails to achieve diversity in cases such as the one shown in Figure 4.3. Suppose that source 1 broadcasts a Route Request for destination 8. If the Request propagates along the lower route (relays 5, 6, and 7) *faster* than it propagates along the upper route (relays 2, 3, 4, and 7), then *only* the lower route will be discovered (and vice versa). This is because 7 will refuse to propagate Route Requests it receives following the first. From a **bidding** perspective, this is a problem. Suppose the lower route is **unaffordable** to 1, but the upper route *is* affordable. If the upper route is never discovered, then source 1 will be unable to send its messages!

XPM solves this problem by making a small change to the way DSR prunes Route Requests.[8] When Route Request $\langle P, n \rangle$ arrives, its route $P = (p_1, p_2, \ldots, p_{|P|})$ is compared against all other Route Requests from p_1 having sequence number n. Let $\langle Q, n \rangle$ be a previously-received Route Request with $Q = (p_1, q_2, \ldots, q_{|Q|})$. In

[8]Actually, the change applies whenever any **multihop power management** instance, including **LPM**, is used.

other words, a Route Request from the same source p_1 with the same sequence number n. The Route Request $\langle P, n \rangle$ is **accepted** if $\forall \langle Q, n \rangle : \dfrac{|Q \cap P|}{|Q|} \leq T$, where $0 < T \leq 1$ is a **diversity threshold**. T determines the maximum amount of **commonality** that may exist between P and any previously-seen route. In our implementation, $T = \frac{3}{4}$ seems to do a good job of pruning Requests near the source (which reduces congestion), while encouraging diversity at more distant nodes.

In the example of Figure 4.3, suppose that node 7 processes a Route Request from 1 on route $(1, 5, 6)$, which it accepts and propagates to 8. Shortly afterwards, it receives another Request having the same sequence number on $(1, 2, 3, 4)$. The only intersection between the routes is at node 1, so $\dfrac{|(1, 5, 6) \cap (1, 2, 3, 4)|}{|(1, 5, 6)|} = \frac{1}{3}$. If $T \geq \frac{1}{3}$, then the new Request will also be accepted and propagated.

4.4.2 Structure Phase

The prerequisite for entering **Structure** is that a node knows one or more routes for each of its **active destinations**. Pure relays and sinks trivially satisfy this requirement. Upon observing a **fast wakeup**, nodes which do not need to discover routes immediately enter Structure.

Sources which *do* need to discover routes may enter Structure as soon as they receive **Route Replies** for their active destinations. Immediately upon entering Structure, an active traffic source begins the process of building a **spanning tree**. In constructing this tree, XPM nodes **elect** a **leader** to solve the **winner determination**

problem and compute **Vickrey payments**. The tree itself describes the method by which **bids** are collected *from*, and **mechanism results** are distributed *to* the agents.

The **root** of the spanning tree is a natural choice to solve the mechanism. Section 4.4.3 describes a **wave algorithm** for passing bids to the root and distributing results. The important design questions are *which* nodes should comprise the **vertex set** of the tree, and what **criteria** should determine the choice of a root node?

The result of Section 2.4.5 helps to answer the former question. When **overlap** exists among the routes being negotiated, then all agents on those routes must participate in a **single** mechanism. So, for example, if two separate sources each know a set of routes on which they want to bid, and those routes have any nodes in common, then both sources and all of the relays and sinks on their routes must be **vertices** in a single tree. This requirement determines the **minimum** vertex set. XPM constructs trees having exactly this minimum set.

The choice of root is more complicated. The root solves an important, but complex, **winner determination problem**. Although we do not model the energy (and other) costs of this solution, the root will be operating its **processor** and **memory hierarchy** in a high-performance mode for some period of time. It might be desirable to **randomly rotate** the rôle of root among the nodes so that a single node does not incur disproportionate costs. Conversely, in the special case where some nodes have access to **infrastructure power**, it might make more sense to delegate computational tasks to those nodes. XPM assumes the general case of a self-contained mobile network in which *any* node can be the root.

4.4.2.1 Greedy-ST Spanning Tree Formation

Many distributed algorithms are known for computing spanning trees in graphs with both **weighted** (Gallager et al., 1983; Gafni, 1985; Awerbuch, 1987; Faloutsos and Molle, 1995; Przytycka and Higham, 1996) and **unweighted** (Perlman, 1985; Métivier et al., 2001) edges. In the former environment, the goal is to find a spanning tree whose total weight is **minimal**. In general, nodes are assumed to know their graph neighbors, and all graph nodes are assumed to participate in the algorithm. Typically, the tree formation procedure may be initiated by any node, or even by many nodes at once.

For the purposes of XPM, edges are unweighted, so there is no notion of a "minimal" tree. Additionally, nodes enter the **Structure** phase *not knowing* the identities of their neighbors. Most significantly, *not all* nodes will participate in the tree formation procedure. Finally, we preserve the requirement that the procedure should be able to initiate in a distributed fashion.

Figure 4.4 shows some properties of the trees formed by XPM. Shaded vertices are neither sources, relays, nor sinks for active traffic flows. As such, these nodes *do not* participate in the tree formation procedure. The Figure shows **multiple** trees, which correspond to **independent mechanisms**. This occurs when sets of active flows have *no* **overlap**. (§2.4.5 explained why sets *with* overlap must be combined into a single mechanism.) These properties contribute to the **scalability** of XPM

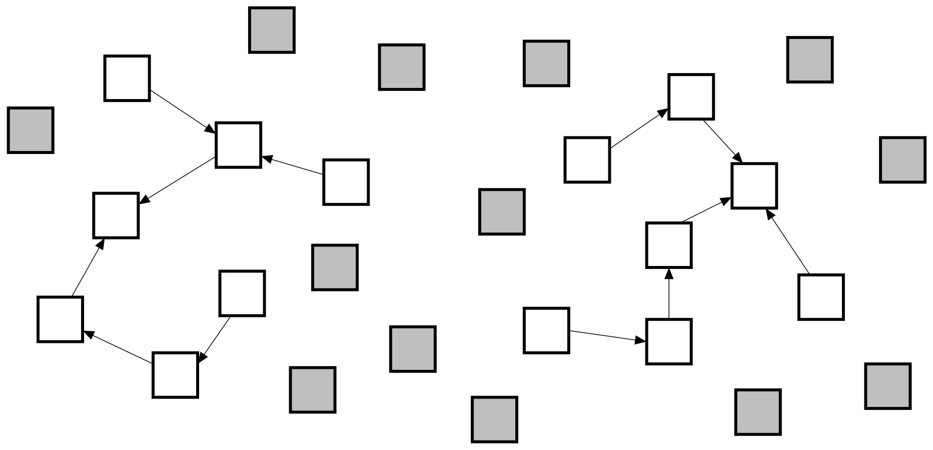


Figure 4.4: Spanning tree membership.

by preventing **irrelevant** nodes (those not participating in any active route) from doing work related to mechanism solutions.

One property that we do not require for spanning trees is **balance**. A tree is balanced when some nodes are not too much further from the root (measured by edge count) than other nodes. Typically, spanning tree algorithms achieve balance by only joining subtrees of similar size. This requires **coordination** and **locking** within each subtree, which adds to **protocol complexity**. Our algorithm instead permits nodes to determine when neighbors should be conquered based solely on **local information**. This method does not result in balanced trees, but rather, **greedily** conquers subtrees when such opportunities are discovered. For this reason, we call the XPM spanning tree formation procedure **Greedy-ST**.

An XPM source which enters **Structure** initiates tree formation by **notifying** the relays and sinks on its **demanded routes** that they will participate in the mechanism. This occurs by sending a **Notify** message along each route for which the source will submit a bid. A Notify is a type of **XPM message**, sent as the payload of a **DSR message**. All XPM messages have the 16-octet header shown in Figure 4.5.

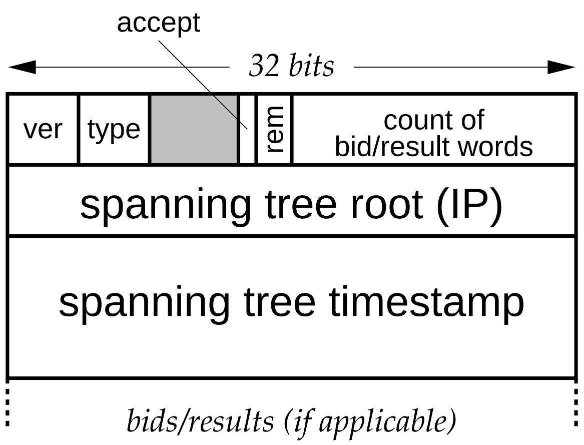


Figure 4.5: XPM protocol message header.

XPM messages contain a four-bit **protocol version** (`0x1`), and a four-bit **message type** (*e.g.*, `0x0` for a Notify). They also contain the 32-bit **IP address** of the **root** to whose tree the message sender belongs. A 64-bit **timestamp** for the sender's tree, expressed in 802.11 **time units** (1024μs), is also included. This information is used in the subtree **conquering** procedure. The remaining fields will be discussed later in this Chapter.

The tree timestamp is the basis upon which the tree root is chosen. Upon observing a **fast wakeup**, each XPM node samples its **802.11 station timer** (§2.1.1.2) using the **get timestamp** action from Section 4.2.3. These **microsecond-resolution** timers are **loosely synchronized**. The sample is taken at the end of the **ATIM Window** in which the fast wakeup occurred. Due to timer **drift**, these samples are not all taken at precisely the same time, nor are the sampled values identical. As a result, we treat the sampled values as **randomly drawn** on some small interval. The XPM tree timestamp has the same **format** as the 802.11 timer, and initially takes the sampled **value** of that timer.

The **conquering rule** used by Greedy-ST is simple: the **oldest timestamp** wins. Therefore, the root of a tree produced by Greedy-ST is the node with the oldest timestamp. (In the event of a tie, we decide by comparing node identifiers: the least numerical address wins.) Since all tree nodes begin as the root of a **singleton** tree with randomly-drawn timestamps, the rôle of root in the **final** tree should be randomly assigned. This satisfies our requirement that **no single node** consistently bears the costs of solving the mechanism.

A source which needed to perform **Discovery** enters **Structure** when it receives a **Route Reply** for one of its **active destinations**. The source adds the discovered route to its **route cache**, then sends a **Notify** message along the new route. The Notify precedes any application traffic on the route,[9] and informs the relays and sink that they will be participants in a negotiation.

When the **finalize timer** for a new active destination expires, a source assumes that it has learned all of the routes DSR is able to discover. If the source has not already entered Structure, it does so at this time. The source then asks the DSR route cache for **all** routes to the finalized active destination using the **get routes** action from Section 4.2.1. A source which does not need to perform Discovery enters

[9]This alerts the relays and sink that a new negotiation is beginning, so that they do not treat subsequent application messages as **infringing** on an existing negotiation. An optimization would be to **piggyback** the Notify on the first outgoing application message.

Structure and performs this lookup *immediately* upon observing a **fast wakeup**.

The implementation of the route cache affects the efficiency of this lookup. Early DSR cache designs which store **entire** routes — known as **path caches** — answer this request by scanning the cached routes $\mathcal{P}$ in $\Theta(|\mathcal{P}|)$ time. Newer **link caches** (Hu and Johnson, 2000) may be less efficient since the routes must be **computed** from knowledge of individual links.

From these cached routes, the source selects a **subset** for which it will actually submit bids. The routes for which bidding will occur comprise the source's **demand set**. The source may decide not to bid on routes that are **redundant** or **too long**. In general, keeping the demand set **compact** is good for the **communications** and **computational complexity** of the negotiation procedure.

Once a source determines its demand set, it sends **Notify** messages along *each* demanded route. In addition to informing the relays and sinks on those routes of their participation in a mechanism, these messages serve two other purposes. First, they verify that the routes *actually work*. Second, they begin the process of forming a **spanning tree**.

If a Notify message encounters a broken link, the node which discovers the break returns a **DSR Route Error** to the source. The source then removes the affected routes from its route cache *and* from its **demand set**. Subsequently, the source will not submit bids for routes that do not appear in its demand set.

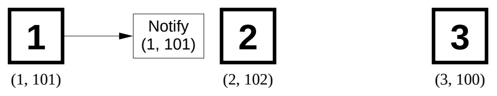
When a Notify is successfully received, there are three possibilities with respect to spanning tree formation. Let q be a received Notify message, and let *root* and *timestamp* describe the tree to which the receiver belongs. The Notify could **conquer** the receiver, if $\mathit{timestamp}(q) < \mathit{timestamp}$ or $\mathit{timestamp}(q) = \mathit{timestamp} \wedge \mathit{root}(q) < \mathit{root}$. The Notify could be from the **same tree**, if $\mathit{timestamp}(q) = \mathit{timestamp} \wedge \mathit{root}(q) = \mathit{root}$. Otherwise, the receiver can **conquer** the node which sent the Notify.

When a receiver is **conquered** by a Notify, it becomes a **child** of the node which sent the Notify. It acknowledges this new relationship by returning a **Respond** message to its new parent. The **accept** bit (Figure 4.5) is set in the Respond message to indicate that the child *accepts* its new position in the parent's tree.

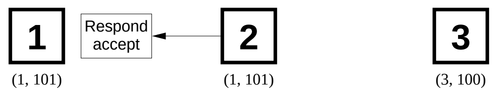
If the sender and receiver are in the **same tree**, then nothing changes as a result of the Notify. If a parent-child relationship already existed between the nodes, then it is preserved. If there was no previous relationship, then the nodes are now **cousins**, reflecting their common tree ancestry. In either case, the receiver returns a **Respond** message with the **accept** bit *cleared*. We call such a message a Respond **reject**, meaning that the receiver *rejects* the sender's conquering attempt.

In the final case, the receiver finds that it can **conquer** the sender. The receiver first returns a Respond reject. It then sends an **Update** message to the sender, which provides the sender with *updated* tree information. Updates are handled in much the same way as Notify messages, except that Updates are only sent to **immediate neighbors**, and are never propagated.

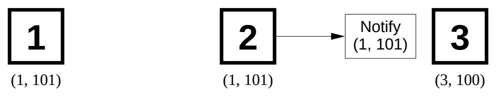
We call **Notify**, **Update**, and **Respond** messages **Structure messages**. These are the only message types needed to form the spanning tree. In Section 4.4.3, we will introduce three **Negotiation messages** (**Submit**, **Result**, **Revoke**) which enable the bidding process.



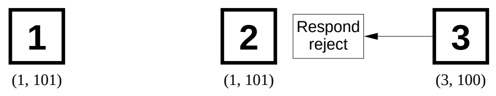
(a) 1 sends Notify to 2.



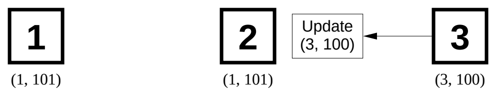
(b) 2 accepts 1 as its parent.



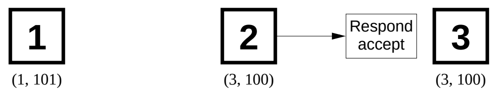
(c) 2 propagates Notify to 3.



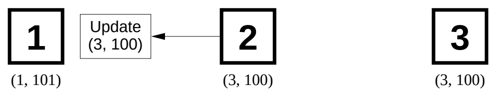
(d) 3 rejects 2 as its parent.



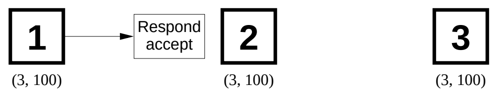
(e) 3 conquers 2 with an Update.



(f) 2 accepts 3 as its new parent.



(g) 2 conquers 1 with an Update.



(h) 1 accepts 2 as its parent.

Figure 4.6: Subtree conquering process.

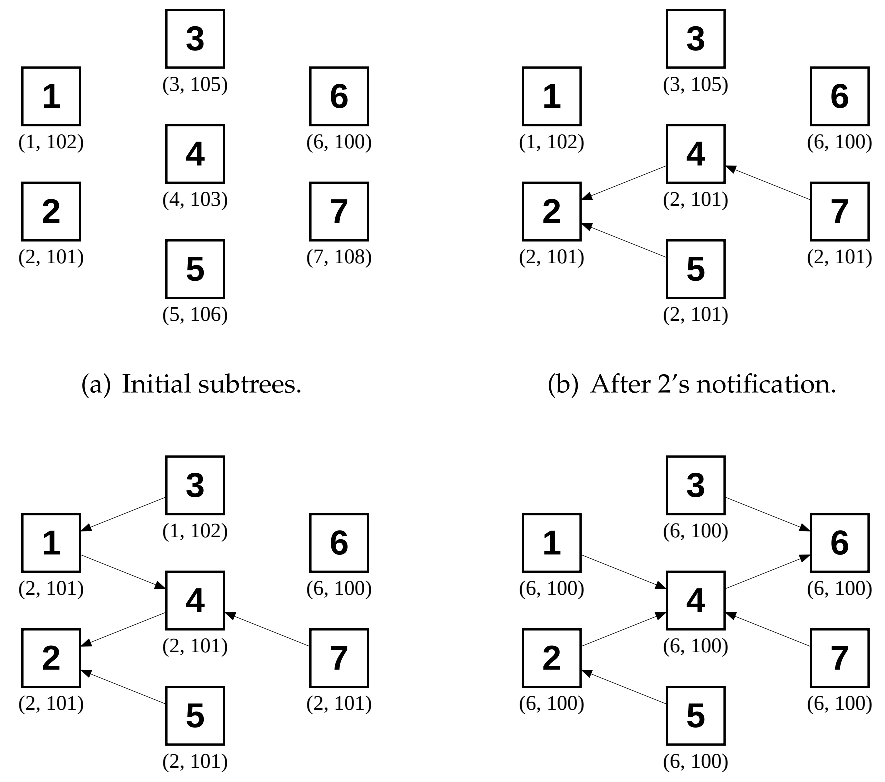
Figure 4.6 shows all of the possible actions in the tree formation process. Source 1 is notifying the relay and sink on the route $(1, 2, 3)$. All three nodes begin as the roots of singleton trees with timestamps sampled around the end of an **ATIM Window** in which a **fast wakeup** was observed. For example, node 2 begins as the root of its own tree, with timestamp 102. Figures 4.6(a) and 4.6(b) show 1 conquering 2. The Notify is propagated to 3 (Figure 4.6(c)), which returns a rejection

(Figure 4.6(d)). 3 then sends a **conquering Update** to 2 (Figure 4.6(e)), which is accepted (Figure 4.6(f)). Since 2's tree membership has changed, 2 sends an Update to it's former neighbor 1 (Figure 4.6(g)). This Update conquers 1, which accepts 2 as its new parent (Figure 4.6(h)). The final tree is rooted at 3, with all nodes having the same root and timestamp.

For examples such as the one just presented, the **best-case** message complexity occurs when the Notify message **conquers** each node it encounters. Two messages are sent across each edge: the **Notify**, and the **Respond accept**. Let $\mathcal{P}$ be a source's **demand set**. The best-case message count is $\sum\limits_{P\in\mathcal{P}} 2\,(|P|-1)$, yielding complexity $\Omega\left(\sum_{P\in\mathcal{P}} |P|\right)$. For a vertex set V, this is "$\Omega(|V|)$."

The **worst-case** occurs when each node encountered by every Notify **conquers** the preceding node. The per-edge message count includes the **Notify** and **Respond reject**, *plus* a sequence of **conquering Update** and **Respond accept** exchanges on *each preceding edge* going back to the source. The worst-case message count is $\sum\limits_{P\in\mathcal{P}} \left[2\,(|P|-1) + 2\sum\limits_{p=1}^{|P|-1} p\right]$. This is an arithmetic series, which can be rewritten using Gauss' "trick:" $\sum\limits_{P\in\mathcal{P}} [(|P|+2)(|P|-1)]$. The resulting complexity is $O\left(\sum_{P\in\mathcal{P}} |P|^2\right)$, or "$O(|V|^2)$."

Greedy-ST has higher $O(\cdot)$ complexity than some distributed spanning tree algorithms because of the **notification** step. Traditional problem formulations have the nodes all beginning with knowledge of their incident edges. XPM nodes do *not* necessarily have this knowledge, and will not know *all* of their edges until they receive *all* of their Notify messages. Greedy-ST optimistically proceeds with conquering even if it might later discover additional edges. In future work, it might be interesting to **postpone** conquering until all Notify messages have propagated. After this point the **vertex** and **edge** sets V and E are known, and a **message-optimal** algorithm — $O\left(|E| + |V| \log |V|\right)$ — could be used (Awerbuch, 1987).



(a) Initial subtrees.

(b) After 2's notification.

(c) During 1's notification.

(d) Final tree.

Figure 4.7: A multiple-source tree formation example.

Figure 4.7 shows how the notification and conquering process works with **multiple sources**. Again, all nodes begin as the root of singleton trees (Figure 4.7(a)). Source 1 demands the routes $(1, 3, 6)$ and $(1, 4, 6)$, while 2 demands $(2, 4, 7)$ and

$(2, 5, 7)$. Suppose 2 sends its Notify messages out first, resulting in the subtree of Figure 4.7(b), in which arrows point from **child** to **parent**. When 1′s Notifies are sent, 3 is conquered but 4 is *not* (Figure 4.7(c)). Eventually, 1′s Notifies reach 6, which has the oldest timestamp. **Conquering Updates** propagate throughout the tree, leaving 6 as the root of the tree shown in Figure 4.7(d).

Every **edge** known to a node is classified by its **type** in the tree. The types **Parent**, **Child**, and **Cousin** name relationships that have already been described. An edge across which a node has sent a Notify or Update has type **Updating** until a response is received. If a node expects to receive a conquering Update from a neighbor, the edge to that neighbor has type **Conquering**. All other edges have type **Unclassified** until a message is sent to, or received from, such edges.

Messages sent by a node are **enqueued** in an **outgoing message queue** before transmission. A message is **blocked** in the queue if its destination is a neighbor from which a node expects contact. For example, if the neighbor edge has type **Conquering**, then no messages are sent to that neighbor until a **conquering Update** arrives. If a Notify or Update is outstanding to a neighbor, then the edge must have type **Updating**, so no messages can be sent until a **Respond** arrives.[10]

Similarly, an **incoming message queue** buffers received messages before processing. Some messages are *never* processed. For example, those messages whose **tree timestamps** indicate they were sent during an **earlier run** of the negotiation

[10]To break deadlocks, we permit Respond messages to Updating neighbors.

procedure are **discarded**.

Procedure 3 PROCESS-IMMEDIATELY(q)

```
1: if $\nexists e \in edges$ s.t. $type(e) =$ Updating then
2:   if $type(edges[sender(q)]) =$ Conquering then
3:     if $type(q) \neq$ Notify $\wedge\, type(q) \neq$ Update then
4:       Return false. {we're expecting a conquering Notify or Update}
5:   Return true. {it's safe to process $q$}
   {Even if edges are Updating, we make some exceptions:}
6: if $type(edges[sender(q)]) =$ Updating then
7:   if $type(q) =$ Notify $\vee\, type(q) =$ Update then
8:     if we appear to be able to conquer $sender(q)$ then
9:       if our root/timestamp changed since our last message to $sender(q)$ then
10:        Return true. {break a potential deadlock}
11:    else {$sender(q)$ can conquer us, or is in the same tree}
12:      Return true.
13:  else if $type(q) =$ Respond then {we're expecting this}
14:    Return true.
     {Negotiation messages from an Updating edge are never processed.}
15: else if $type(edges[sender(q)]) =$ Conquering then
16:  if $type(q) =$ Notify $\vee\, type(q) =$ Update then {we're expecting this}
17:    Return true.
18: else
19:  if $type(q) =$ Notify $\vee\, type(q) =$ Update then
20:    if $q$ does not conquer our root/timestamp then {$q$ is "harmless"}
21:      Return true.
22: Return false.
```

Messages which are not discarded are tested to see if they can be processed immediately. Procedure 3 returns **true** if a message q can be processed, given the current edge classifications. When no edges have type **Updating**, all messages are admitted *unless* they arrive on a **Conquering** edge. Once an edge is labeled Conquering, we only accept a conquering **Notify**[11] or **Update** from that neighbor.

There are a few special cases in which a node accepts messages even if it *is* in

[11]Typically, **Update** messages are used for conquering. If a node happens to have an outgoing **Notify** queued for a neighbor that it will conquer, there is no need to *also* send an Update.

the process of updating some of its neighbors. The first of these is a **deadlock-avoidance** measure. If two neighbors send Notify/Update messages at one another, we don't want both nodes to **block** waiting for a response. Another case involves the receipt of a **Respond**. These *should* be arriving from **Updating** edges, so we always accept them.

Some Notify/Update messages will not cause a node to be conquered. These "**harmless**" messages can be accepted from most edge types. The node will simply return a **Respond**, and possibly a **conquering Update**, to the neighbor.

Procedure 4 HANDLE-NOTIFY(q)

```
1: $received\text{-}tree\text{-}messages \leftarrow received\text{-}tree\text{-}messages \cup \{sender(q)\}$
2: NOTIFY-UPDATE($q$)
3: if $q$ should be propagated then {prune Notifies with redundant route suffixes}
4:    $update\text{-}type(edges[next\text{-}hop(q)]) \leftarrow type(edges[next\text{-}hop(q)])$
5:    $type(edges[next\text{-}hop(q)]) \leftarrow$ Updating
6:    Enqueue $q$ to $next\text{-}hop(q)$.
```

Once a **Structure message** has been accepted from the incoming queue, it is passed to a per-type **handler**. For example, Procedure 4 shows the treatment of **Notify** messages. Greedy-ST maintains a set of neighbors from which it has received Structure messages. An edge e is **classified** when $type(e) \neq$ Unclassified $\wedge$ $neighbor(e) \in received\text{-}tree\text{-}messages$. When a classified edge **breaks**, as detected by **link layer acknowledgments**, the spanning tree formation procedure is restarted. Section 4.6.2 describes a more graceful treatment of broken tree edges.

Notify messages are handled in much the same way as **Update** messages (Procedure 5), except that Notify messages can be **propagated**. When a Notify is pro-

Procedure 5 HANDLE-UPDATE(q)

```
1: $received\text{-}tree\text{-}messages \leftarrow received\text{-}tree\text{-}messages \cup \{sender(q)\}$
2: NOTIFY-UPDATE($q$)
```

cessed at a relay on some source's **demanded route**, in general it will be sent to the next hop on the route. We can improve the efficiency of Notify distribution in some cases. For example, Notifies from the same source having a common **route suffix** can be **pruned**.

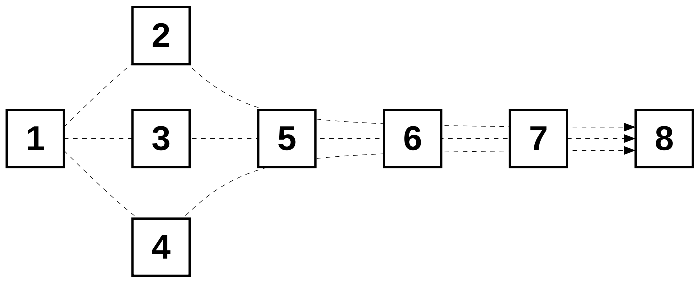


Figure 4.8: Some Notify messages for routes on the flow $1 \rightsquigarrow 8$ may be pruned.

Figure 4.8 shows three routes for the flow $1 \rightsquigarrow 8$. Suppose the Notify for the route through 2 reaches 5 first. The subsequent Notify messages from 1 all have a route suffix $(5, 6, 7, 8)$ in common with the first Notify, and so do not need to be propagated. If the Notify that passed through 3 or 4 can **conquer** 5, then 5 will use the normal Update process to inform its neighbor 6 of its new tree membership.

Procedure 6 shows the common handling of **Notify** and **Update** messages. Such messages arriving from a node's own tree typically do not result in any state changes; the node simply returns a **Respond reject** to the sender. If the neighbor was not previously classified as a "relative" in the tree, its edge takes type **Cousin**.

Messages that **conquer** the receiving node cause the node's *root* and *timestamp*

Procedure 6 NOTIFY-UPDATE(q)

```
1: if $root(q) = root \wedge timestamp(q) = timestamp$ then {same tree}
2:   if $type(edges[sender(q)])$ is not Updating, Parent, Child, or Cousin then
3:     $type(edges[sender(q)]) \leftarrow$ Cousin
4:   Enqueue a Respond (reject) to $sender(q)$.
5: else if $q$ conquers our root/timestamp then
6:   $root \leftarrow root(q)$
7:   $timestamp \leftarrow timestamp(q)$
8:   $type(edges[sender(q)]) \leftarrow$ Parent
9:   Let $e$ be the old parent edge, if it exists.
10:  if $e \neq edges[sender(q)]$ then {parent has changed}
11:    Revoke bids, if submitted.
12:    $type(e) \leftarrow$ Child
13:  for all $n$ s.t. $type(edges[n]) =$ Child $\vee$ $type(edges[n]) =$ Cousin do
14:    if $q$ will not be propagated or $n \neq next\text{-}hop(q)$ then
15:      $type(edges[n]) \leftarrow$ Updating
16:      Enqueue an Update to $n$.
17:  Enqueue a Respond (accept) to $sender(q)$.
18: else {we appear to be able to conquer $sender(q)$}
19:  $type(edges[sender(q)]) \leftarrow$ Child
20:  Enqueue a Respond (reject) to $sender(q)$.
21:  if we're not already trying to update $sender(q)$ then
22:    $type(edges[sender(q)]) \leftarrow$ Updating
23:    Enqueue an Update to $sender(q)$.
```

to change. If the message came from an edge that was not previously the node's **parent**, then the node must **revoke** any bids it may have sent to the old parent. This process is described in Section 4.4.3.1. The old parent then becomes a **child** of the node, and all Child or Cousin edges receive **Update** messages to inform them of the new tree membership.

Procedure 7 HANDLE-RESPOND(q)

```
1: $received\text{-}tree\text{-}messages \leftarrow received\text{-}tree\text{-}messages \cup \{sender(q)\}$
2: if $accept(q)$ then
3:    if $root = root(q)$ then
4:       $type(edges[sender(q)]) \leftarrow$ Child
5:    else {$q$ is stale}
6:       Schedule a correcting Update to $sender(q)$.
7: else {$q$ is a rejection}
8:    if $root = root(q)$ then {$sender(q)$ is our relative in the tree}
9:       $type(edges[sender(q)]) \leftarrow$ Cousin
10:   else if $q$ conquers our root/timestamp then
11:      $type(edges[sender(q)]) \leftarrow$ Conquering {await a conquering Update}
12:   else {$q$ is stale}
13:      Schedule a correcting Update to $sender(q)$.
```

Respond messages are handled as shown in Procedure 7. Typically, receipt of a Respond from a neighbor informs a node that the neighbor is a "**relative**" in the same tree, or is about to **conquer** the node. If the **root** and **timestamp** of the Respond message are not consistent with a node's beliefs about the state of the tree, the node may schedule a **correcting Update** to the neighbor. Bid submission will not proceed until the node and its neighbors agree upon the tree status.

Nodes learn their incident edges by receiving **Notify** messages along the **demanded routes** of traffic sources. They may **infer** additional edges when they

overhear messages on the wireless medium. DSR passes overheard messages to XPM using the **snoop packet** action (§4.2.1). If a neighbor transmits an XPM message that is overheard, and an edge to that neighbor was not previously known, then a new tree edge is added. The type of the edge is **Cousin**, if the XPM message bears a root and timestamp from the same tree; otherwise, the edge is **Unclassified**.

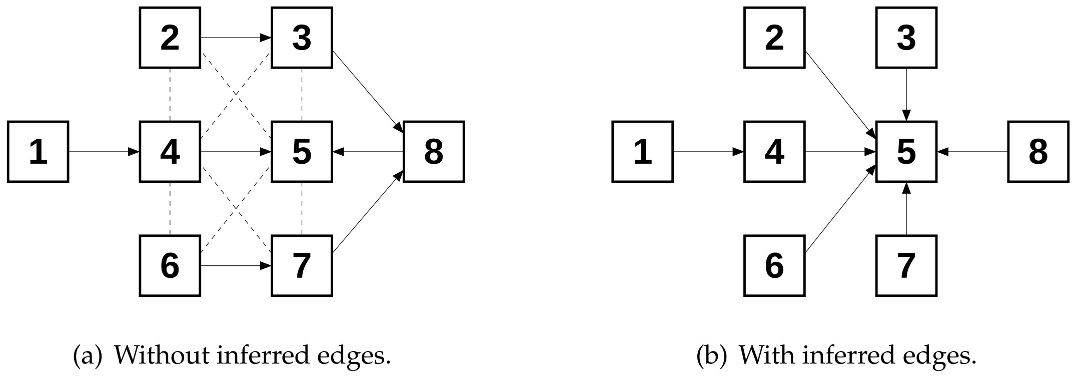


Figure 4.9: Inferred spanning tree edges.

Figure 4.9 shows how the use of inferred edges is helpful. Suppose 1 sends Notify messages along the routes $(1,2,3,8)$, $(1,4,5,8)$, and $(1,6,7,8)$, resulting in the tree of Figure 4.9(a). The **average path** between any node and the root 5 is 2 edges in length. However, as the Notify messages proceed along their routes, nearby nodes might overhear them. For example, 4 might overhear the Notify on $(1,2,3,8)$ when it is transmitted by 2, and later by 3. This allows 4 to **infer** the edges $(2,4)$ and $(3,4)$, shown in dashed lines. Using the inferred edges, it may be possible to construct a tree with **shorter** paths. For example, the average path in the tree of Figure 4.9(b) — which uses inferred edges — is only 1.14 edges in length.

This example shows that the use of inferred edges permits the formation of more **balanced** trees. **Edge inference**, which we have implemented for Greedy-ST, is a useful technique for *any* spanning tree algorithm.

4.4.3 Negotiation Phase

The prerequisite for entering the **Negotiation** phase is that a node has **classified** all of its incident edges, and has received **bids** from all of its **children**. The former requirement states that a node has resolved its relationships with its neighbors; it is neither updating nor being conquered by other nodes. The latter requirement is trivially satisfied by **leaf nodes** in the tree, as they have no children. Leaves begin the bid submission process, by passing bids towards the **root** of their tree. Section 4.4.3.1 describes this algorithm in more detail. Once all bids reach the root, the root solves the **mechanism** by executing a **winner determination algorithm** (§4.4.3.2), and computing agent **payments** (§4.4.3.3). The results of the mechanism are then distributed among the agents.

4.4.3.1 Bidding Wave Algorithm

Mechanisms provide a **centralized** solution to a **distributed** optimization problem. The exchange-based mechanism of this thesis requires all reports of agent **preference** to be gathered in one place. Some agent, the "**auctioneer**," then solves a **surplus-maximizing** combination of bids, and determines agent payments to the mechanism. The problem we must address in the **distributed** environment is that of collecting the bids at a single node.

Given a **spanning tree** produced by the algorithm of Section 4.4.2.1, there is a straightforward $\Theta(|E|)$ algorithm for bid submission. Beginning with the **leaf nodes** in the tree, each node submits bids to its parent. A parent combines the bids received from its children with its own bids, and submits the aggregate to *its* parent, and so forth. Bids traverse each of the $|E|$ tree edges. Once the root receives bids from all of its children, the algorithm is complete and the root may proceed with solving the mechanism. A **distributed algorithm** of this form is called a **wave algorithm** (Tel, 2000).

Section 2.3.3 described the **OR-of-XORs bidding language**, which lets agents express **substitutability** between bids. Each agent's bids are contained in an **OR term** (which contains one or more **XOR terms**, which contain one or more **bids**). When parents combine their bids with those of their children, they are simply joining OR terms to produce a larger OR term. Ultimately, the root generates a *single* large OR term which is the input to the mechanism.

Drawing from the earlier examples of Figures 2.9 (§2.3.2) and 4.7, Figure 4.10 shows how bids are passed to the root. As an example of **bid aggregation**, consider agent 2, whose own bids are $B_3 \oplus B_4$. Agent 2 must wait to submit its bids until it has received bids from all of its children, namely, agent 5 in this case. Once 5 submits its bid B_6, 2 combines the bids to produce the OR term $(B_3 \oplus B_4) + B_6$,

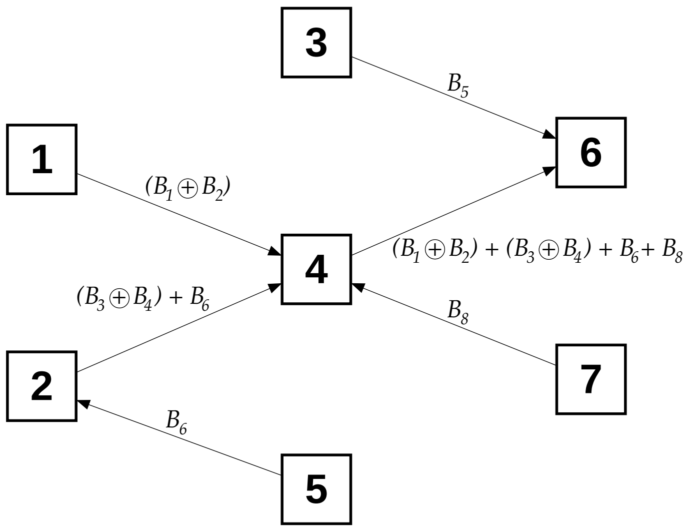
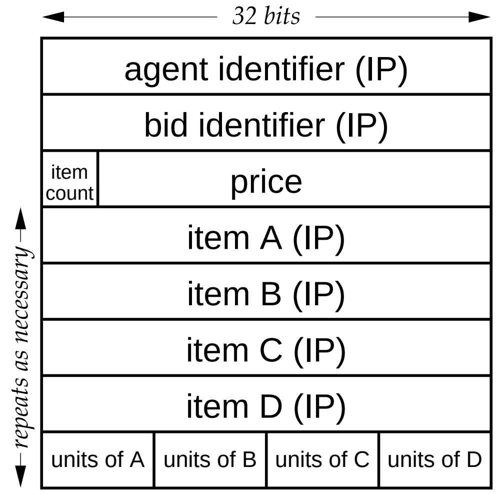


Figure 4.10: Bid submission and aggregation.

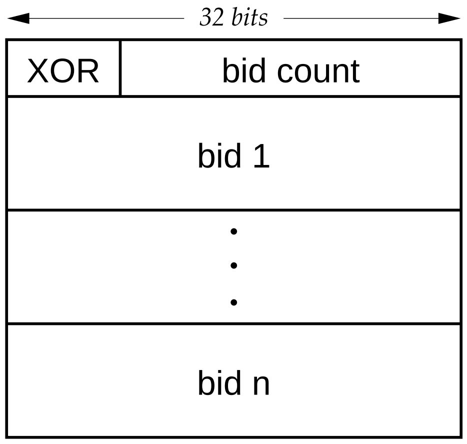
which it submits to its parent, node 4.

XPM uses a compact **bidding syntax** to encode the **OR-of-XORs language**. Bids are transmitted in the body of an XPM **Submit** message. As Figure 4.11(a) shows, each bid contains an identifier for the agent which submitted the bid, as well as a unique identifier used by the submitting agent to help distinguish among multiple bids. Bids can specify up to 15 items i for which $\lambda_j^i \neq 0$; for all other items k, $\lambda_j^k \equiv 0$. Prices p_j are encoded as a 28-bit signed integer in units of $\frac{1}{1000}$. Items i and the 8-bit signed integer units λ_j^i are then packed in groups of four.

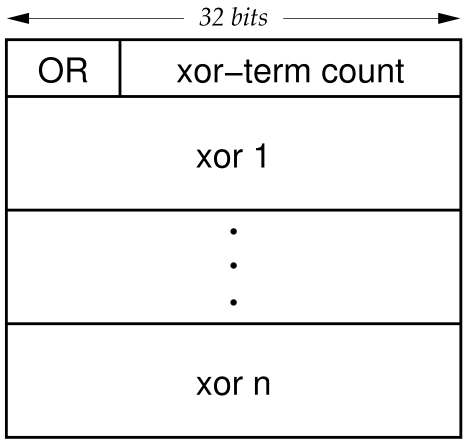
Bids are marshaled into an **XOR term** as shown in Figure 4.11(b). XOR terms are, in turn, marshaled into an **OR term** as shown in Figure 4.11(c). The smallest expression by an agent, a bid or ask for a single item, requires 20 octets for the bid itself, an additional 4 octets for the enclosing XOR, and 4 more octets for the OR wrapper, for a total of 28 octets. Expressed in 32-bit units, this size is 7 **words**.



(a) Bid representation.



(b) XOR representation.



(c) OR representation.

Figure 4.11: OR-of-XORs bidding language syntax.

XPM messages indicate the **count** of bid words they contain in the 16-bit unsigned integer field shown in Figure 4.5 (§4.4.2.1).

Although we do not explore the use of **cryptographic techniques** in this thesis, we note that bids could be **encrypted**. Encryption of the mechanism inputs is one of many techniques that can help make an implementation **robust** against **ma-**

nipulation in a distributed environment (Shneidman and Parkes, 2003). Generally speaking, **distributed algorithmic mechanism design** (Feigenbaum and Shenker, 2002) studies how the distributed nature of mechanisms introduces opportunities for agent manipulation. In the specific case of XPM, cryptographic protection of the **OR term** submitted by each agent could reduce the chances of bid alteration. No agent other than the root needs to be able to **interpret** another agent's report, so no functionality is lost by encrypting these reports. Applications of cryptography, as well as other distributed algorithmic mechanism design results, to XPM are interesting subjects for future work.

Procedure 8 HANDLE-SUBMIT(q)

```
1: $descendant\text{-}bids[sender(q)] \leftarrow descendant\text{-}bids[sender(q)] \cup bids(q)$
2: if $remaining(q) > 0$ then {there are additional parts to this multipart Submit}
3:     Return.
4: if $\exists e \in edges$ s.t. $type(e)$ is Conquering or Updating then
5:     Return.
6: if $\exists e \in edges$ s.t. $type(e) = \text{Child} \wedge descendant\text{-}bids[neighbor(e)] = \varnothing$ then
7:     Return. {awaiting bids from $neighbor(e)$}
   {All edges are classified, and we've received bids from all children.}
8: Signal Bids-Ready.
```

Procedure 8 shows the handling of **Submit** messages. A parent receives a Submit from one of its **children**, which may contain bids from *many* **descendants**. It is useful to remember which descendants exist in the **subtrees** rooted at each child. This information can later be used to factor out per-subtree **mechanism results**.

As descendant bids are **aggregated** with parents' own bids, it can be the case that the size of the encoded bids grows beyond the maximum size of a single

Submit message.[12] It is useful to support the concept of a **multipart Submit**, in which bids are **segmented** across multiple back-to-back Submit messages, then **reassembled** at the receiving parent. The first $n-1$ parts of an n-part Submit have a nonzero value in the 2-bit unsigned **remaining** field of the XPM header (Figure 4.5, §4.4.2.1). For each message, this value indicates the number of parts to follow; when a Submit with a value of zero is received, bid transmission is complete.

If the **tree relationships** between a node and all of its neighbors are resolved, and bids have arrived from all of the node's **Child** edges, then the node is ready to submit bids to the mechanism. It signals that all bids are ready, which indicates that the prerequisites for the **Negotiation** phase have been satisfied. The node constructs a **Submit** message containing its own bids as well as those of its descendants, and enqueues the message for its parent.

It is useful for a node to be able to **cancel** bids it has previously submitted. For example, in Figure 4.7(b) (§4.4.2.1), suppose node 4 sends a **Submit** message to its parent, 2, containing bids from 4 and 7. In Figure 4.7(d), 4's **tree membership** *and* **parent** have changed. To prevent nodes (such as 2) from doing wasteful work, 4 sends an XPM **Revoke** message to its former parent.

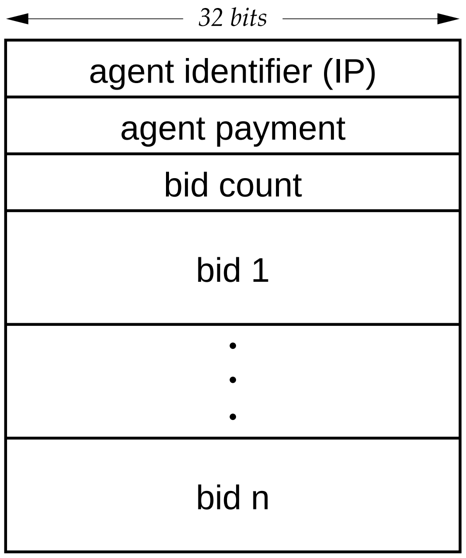
Procedure 9 shows how **Revoke** messages are processed. Upon receiving a Revoke, a node should delete any bids received from the sender of the Revoke (including those submitted by **descendants**). If any bids have been submitted to

[12]Given the maximum 802.11 **data frame payload**, and the various header sizes, a single Submit can carry 530 words of encoded bids.

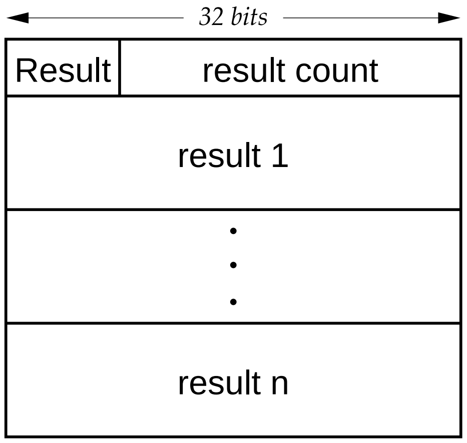
Procedure 9 HANDLE-REVOKE(q)

```
1: $descendant\text{-}bids[sender(q)] \leftarrow \varnothing$ {delete bids from $sender(q)$}
2: if $type(edges[sender(q)])$ is Parent, Child, or Cousin then
3:    if $\exists\, e \in edges$ s.t. $type(e) = Parent$ and we've already submitted bids then
4:       Enqueue Revoke to $neighbor(e)$. {send bid revocation to our parent}
5:    if $root = id$ then {we're the root of our tree}
6:       Signal Bids-Invalid. {abort an in-progress mechanism solution}
```

a node's parent, the Revoke **propagates** upward to that parent. If the node is the root of its tree, it may be in the process of solving a **mechanism**. Since the inputs to that mechanism (the bids) have changed, this solution should be **aborted**.



(a) Result representation.



(b) Result set representation.

Figure 4.12: Mechanism result syntax.

Following the solution of the mechanism, the root informs all of the **agents** of their winning bids (if any) and their **payments** to the mechanism. This information is distributed along the **spanning tree** edges using XPM **Result** messages. Each agent receives a mechanism result of the form shown in Figure 4.12(a). The result contains the agent's **identifier**, and the amount of its **Vickrey payment**, $p_{i,\text{vick}}$. A number of **bids** may be included, as well. These are copies of any of the agent's winning bids or asks (*i.e.*, $B_j \in \mathcal{B}$ s.t. $x_j = 1$). Additionally, for **sellers**, copies of the winning bids demanding items offered by the agent's winning asks are included. These copies allow sellers — **relays** and **sinks** — to see which **routes** they serve, and thereby configure their **suspending neighbors list** (§4.1.1) accordingly. Agent results are packed into a **result set**, shown in Figure 4.12(b), which forms the body of the **Result** message.

Procedure 10 HANDLE-RESULT(q)

```
1: $own\text{-}bids \leftarrow results(q)[id]$ {factor out own results}
2: for all $n$ s.t. $type(edges[n]) = Child$ do
3:     Enqueue Result containing $results(q)[n]$ to $n$. {factor out child $n$'s results}
4: Signal Results-Ready.
```

Procedure 10 shows the handling of **Result** messages. A node receiving a Result from its parent first factors out its own **agent result** from the enclosed **result set**. Then it factors out **agent results** for each of the subtrees rooted at its children. (Recall that it knows the subtree memberships from the **bid submission** stage.) Result messages are then sent to each child with result sets appropriate to those children and their respective **descendants**. Following this, the node is ready to enter **Implementation**.

Like agent bid submissions, agent results could be **encrypted**. Once the root determines the winning bids and agent payments, agent i's results need only be readable by agent i.[13] Again, the use of cryptography in XPM is a topic for future research.

4.4.3.2 Branch-on-Bids Winner Determination

The **winner determination problem** for combinatorial exchanges, described in Section 2.3.2, is $\mathcal{NP}$-complete (by reduction from **weighted set packing**). The **Branch-on-Bids** algorithm (Sandholm, 2003) uses **heuristic search** to manage this complexity. For the scenarios we have simulated, we were able to optimize BOB sufficiently to feel comfortable with its performance in practical environments.

BOB looks for a **surplus-maximizing** combination of bids in a **search tree**, shown in Figure 4.13. (This tree has *nothing to do with* the spanning tree of Section 4.4.2.) The vertices of the tree are bids $B_j \in \mathcal{B}$. Each vertex has two descendent edges corresponding to the cases in which B_j is assigned *winning* ($x_j = 1$) or *losing* ($x_j = 0$). The $2^{|\mathcal{B}|}$ leaves of the tree represent all **possible** combinations of the bids $\mathcal{B}$. In principle, we could perform a depth-first examination of the tree, enumerate all of the combinations, and pick the one with the greatest surplus.

Fortunately, it is possible to bound the search by exploiting the **constraints** between bids, and by employing **heuristics**. As an example of the former, suppose that in Figure 4.13, B_1 and B_3 were joined by XOR constraints: $B_1 \oplus B_3$. Then, when searching the subtree in which $x_1 = 1$, we could prune the search of any subtrees

[13]To further reduce the amount of **information leakage**, it might be desirable to eliminate the practice of passing sellers a copy of each winning bid that involves them. The **suspending neighbors list** could be configured using other means.

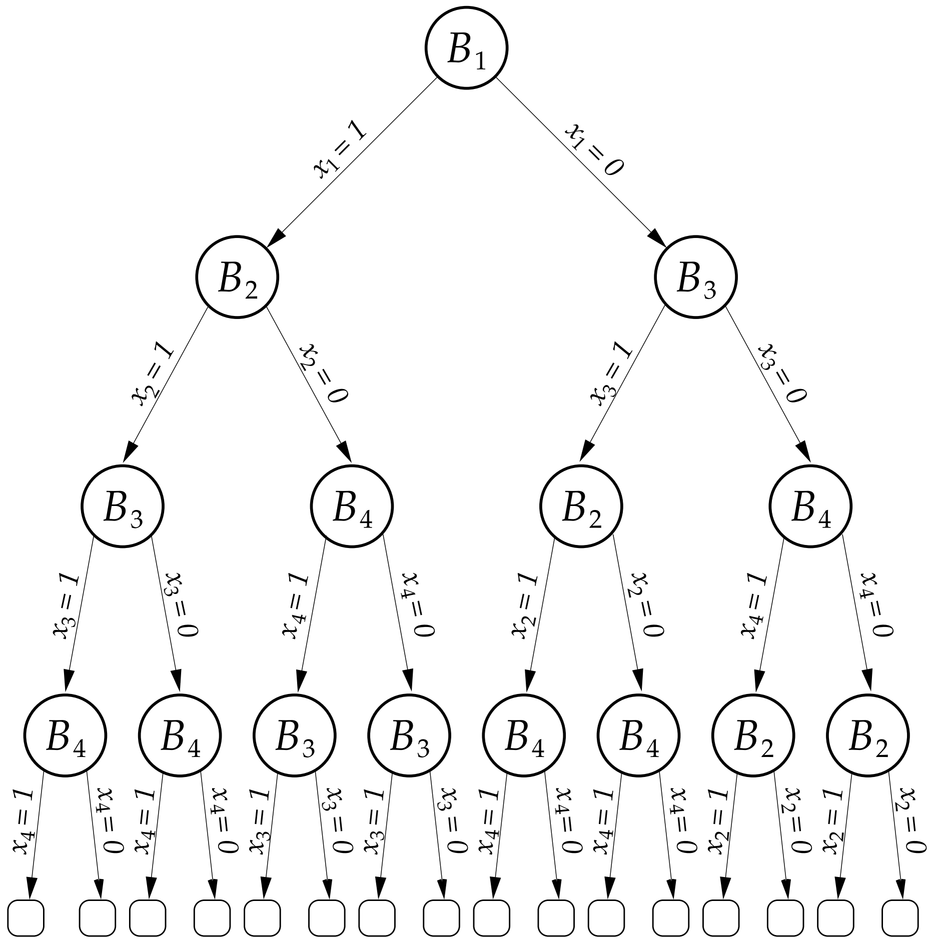


Figure 4.13: Branch-on-Bids winner determination.

in which B_3 is assigned winning ($x_3 = 1$).

Heuristics improve search performance by **estimating** the additional surplus available in a subtree (Russell and Norvig, 1995). An **admissible** heuristic never *overestimates* this surplus. BOB uses admissible heuristics to decide which subtrees cannot yield a greater surplus than the best combination of bids found so far.

During the search, the set of bids labeled winning so far are stored in the set IN. The best allocation found so far, IN^*, is kept up-to-date at each search node, as is the surplus of that allocation, $\tilde{f}^*$. Initially, $IN = \varnothing$, $IN^* = \varnothing$, and $\tilde{f}^* = 0$. At each search node, let g be the surplus of the bids which were assigned winning ($x_j = 1$) so far. The allocation Λ resulting from the winning bids so far records the allocations for each item i: $\Lambda_i = \sum_j \lambda_j^i x_j$. Procedure 11 is run at each search node.

Our implementation uses the **graph data structure** optimization (GRA) from (Sandholm, 2003). Each bid corresponds to a **vertex** in a graph, where the edges between vertices indicate constraints between bids. For example, bids B_j and B_k which are joined by an XOR constraint share an **XOR edge** (j, k). If B_j and B_k do not share an XOR edge, but there exists an item i for which $\lambda_j^i \neq 0$ and $\lambda_k^i \neq 0$, then the vertices share an **OR edge** (j, k).

When a bid B_k is assigned winning ($x_k = 1$), it is **deleted** from the graph (line 25 of Procedure 11). Any bids sharing an XOR edge to B_k are *also* deleted, since they can no longer be assigned winning. A bid B_j sharing an OR edge to B_k is deleted if there exists an item i for which $\lambda_j^i > 0$ and $\left[\lambda_j^i + \Lambda_i + \sum_{B_l \mid \neg deleted(B_l) \wedge \lambda_l^i < 0} \lambda_l^i\right] > 0$. In other words, B_j can be deleted if the amount of i it demands, plus the amount of i available from assigned-winning bids, plus the amount available from *unassigned* bids is an **infeasible** allocation.[14] (B_j demands more of i than can be supplied.)

After the subtree corresponding to the assignment of bid B_k winning has been searched, the subtree in which bid B_j is assigned losing ($x_k = 0$) is searched. The neighbors of B_k which were deleted prior to searching the assigned-winning case

[14]An additional case in which neighboring bids are deleted appears in (Sandholm, 2003), applicable when there is **no free disposal** of items. The rule is symmetric with the one already presented, but applies to asks for item i. It says that if B_j is offering an amount of i that, given the demand for i in already-allocated bids and the demand in unallocated bids, would result in a net surplus of i, then B_j is deleted. We permit free disposal in our implementation, and do not use this rule.

Procedure 11 BOB(Λ, g)

```
1: if $\Lambda_i \leq 0$ for $i = 1, 2, \ldots, m$ then {$\Lambda$ is a feasible allocation}
2:   if $g \geq f^*$ then
3:     $IN^* \leftarrow IN$, $\tilde{f}^* \leftarrow g$
   {upper revenue bound for remaining bids, $h \geq 0$:}
4:   $h \leftarrow \sum_{i=1}^{m}$(ESTIMATE-SUPPLY($i$) $- \Lambda_i$)$\times$ESTIMATE-REVENUE($i$)
5:   if $g + h < \tilde{f}^*$ then
6:     Return. {bounding}
   {lower revenue bound for remaining feasible bids, $L \geq 0$:}
7:   $L \leftarrow \max_{B_j \in \mathcal{B}} p_j$ s.t. $\neg deleted(B_j)$, $p_j > 0$, and $\Lambda_i + \lambda_j^i \leq 0$ for $i = 1, 2, \ldots, m$
8:   if $g + L > \tilde{f}^*$ then
9:     $IN^* \leftarrow IN$, $\tilde{f}^* \leftarrow g + L$
10:  Choose branching $B_k$ s.t. $\neg deleted(B_k)$ and $\forall_{B_j \mid \neg deleted(B_j)} p_k \geq p_j$. If $\nexists B_k$, return.
11: else {$\Lambda$ is currently infeasible}
12:   if $\nexists B_j \in \mathcal{B}$ s.t. $\neg deleted(B_j) \wedge p_j > 0$ then {no available bids remain}
13:     if $\nexists \mathbf{B} \subseteq \mathcal{B}$ s.t. $\forall B_j \in \mathbf{B} \mid \neg deleted(B_j) \wedge IN \cup \mathbf{B}$ is feasible then
14:       Return.
     {least revenue decrease for items with infeasible allocations, $H \leq 0$:}
15:     $H \leftarrow \sum_{i=1 \mid \Lambda_i > 0}^{m} p_j \mid \max_{B_j \mid \neg deleted(B_j) \wedge \Lambda_i + \lambda_j^i \leq 0} \lambda_j^i$
16:     if $g + H < \tilde{f}^*$ then
17:       Return. {bounding}
18:   else
19:     Choose the shortest $B_k$ s.t. $\neg deleted(B_k) \wedge \exists i \mid \lambda_k^i > 0 \wedge \Lambda_i > 0$, if it exists.
20:   if a branching bid $B_k \in \mathcal{B}$ has not been chosen then
21:     Choose branching $B_k$ s.t. $\min_{B_k \mid \neg deleted(B_k)} \left[\max_{i \mid \Lambda_i > 0} \left|\Lambda_i + \lambda_k^i\right|\right]$. If $\nexists B_k$, return.
22: $IN \leftarrow IN \cup \{B_k\}$ {$x_k \leftarrow 1$}
23: for $i = 1, 2, \ldots, m$ do
24:   $\Lambda_i \leftarrow \Lambda_i + \lambda_k^i$
25: Delete $B_k$ and neighboring bids as appropriate.
26: BOB($\Lambda$, $g + p_k$) {branch $B_k$ in}
27: $IN \leftarrow IN \setminus \{B_k\}$ {$x_k \leftarrow 0$}
28: for $i = 1, 2, \ldots, m$ do
29:   $\Lambda_i \leftarrow \Lambda_i - \lambda_k^i$
30: Restore neighbors of $B_k$.
31: BOB($\Lambda$, $g$) {branch $B_k$ out}
32: Restore $B_k$, return.
```

are **restored** to the graph before searching the $x_k = 0$ subtree (line 30 of Procedure 11). Bid B_k is not itself restored to the graph until both the $x_k = 1$ and $x_k = 0$ subtrees have been searched (line 32 of Procedure 11). Delete and restore operations use the constant-time methods described in (Sandholm, 2003), such as the use of cross pointers between bids, and the removed-edge sets E' and E''.

Line 1 of Procedure 11 divides the procedure at each search node into the case in which the current allocation Λ is **feasible**, and that in which it is **infeasible**. We only update the best allocation found so far, IN^* and $\tilde{f}^*$, when Λ is feasible. This is important because it is not acceptable for BOB to declare an **infeasible** allocation to be the solution.[15]

In line 4 of Procedure 11, $h \geq 0$ is a heuristic upper bound on the revenue achievable from the bids that have not yet been assigned winning or losing. It is computed by first determining the maximum possible supply of each item i from the unassigned bids, ESTIMATE-SUPPLY(i). This supply is calculated by adding the maximum amount of i offered by any bid in each **XOR term**. This sum is optimistic: no more than one bid per XOR term can be found winning, and in general, not all XOR terms will have a winning bid. Therefore, ESTIMATE-SUPPLY($\cdot$) does not violate admissibility. Procedure 12 shows the maximum supply determination.

Given the maximum supply of an item i from the remaining unassigned bids, we can compute the upper bound of the supply of i at the current search node.

[15]This restriction does not appear in (Sandholm, 2003). The algorithm presented there is described for combinatorial **auctions**, which have a different feasibility constraint.

Procedure 12 ESTIMATE-SUPPLY(i)

```
1: Let $X \in \mathcal{X}$ be an XOR term s.t. $X \subseteq \mathcal{B}$, $\bigcup_{X \in \mathcal{X}} X = \mathcal{B}$, and $\bigcap_{X \in \mathcal{X}} X = \varnothing$
2: $a \leftarrow 0$
3: for all $X \in \mathcal{X}$ do
4:     $a_X \leftarrow 0$
5:     for all $B_j \in X$ do
6:         if $\neg deleted(B_j)$ then
7:             if $\lambda_j^i < a_X$ then
8:                 $a_X \leftarrow \lambda_j^i$
9:     if $a_X < a$ then
10:        $a \leftarrow a_X$
11: return $-a$
```

Subtract out the allocation $\Lambda_i \leq 0$ from the bids assigned winning so far. The result is an upper bound on the number of units of i available in a subtree of the current search node.

To determine the additional revenue achievable from the estimated amount of i remaining, we use an estimate of per-unit revenue. When initializing the bid graph, we compute for each bid B_j an estimate of its per-unit revenue potential, $\frac{p_j}{\sum_{i=1}^{m} \lambda_j^i}$.[16] In our ad hoc network environment, bids ($p_j \geq 0$) only demand items ($\lambda_j^i \geq 0$), and asks ($p_j < 0$) only supply items ($\lambda_j^i < 0$), so $\forall_{B_j \in \mathcal{B}} \frac{p_j}{\sum_{i=1}^{m} \lambda_j^i} \geq 0$.

An upper bound on the additional *net* revenue from item i can be computed using these per-unit revenue estimates. Find the bid demanding i with the greatest per-unit estimate, and the ask supplying i with the greatest per-unit estimate, and compute the net revenue which would result from trading one unit of i using this

[16]This is similar to the $\frac{p_j}{|S_j|^\alpha}$ heuristic from Section 3.1.2 of (Sandholm, 2003), which is presented for combinatorial auctions. Note that since $\sum_{i=1}^{m} \lambda_j^i$ can be negative in the exchange environment, α cannot be used.

maximum-revenue pair. Procedure 13 shows this net revenue estimate, ESTIMATE-REVENUE(i).

Procedure 13 ESTIMATE-REVENUE(i)

```
1: $d_i \leftarrow \max_{B_j \in \mathcal{B}} \frac{p_j}{\sum_{i=1}^{m} \lambda_j^i}$ s.t. $\neg deleted(B_j)$ and $\lambda_j^i > 0$ {max demand price per unit $i$}
2: $s_i \leftarrow \max_{B_j \in \mathcal{B}} \frac{p_j}{\sum_{i=1}^{m} \lambda_j^i}$ s.t. $\neg deleted(B_j)$ and $\lambda_j^i < 0$ {max supply price per unit $i$}
3: return $\max(d_i - s_i, 0)$
```

We have described ESTIMATE-SUPPLY(i) $- \Lambda_i$, the upper bound on the net units of i available below the current search node, and ESTIMATE-REVENUE(i), the upper bound on the net revenue from trading one unit of i. The product of these estimates is an upper bound on the revenue available from trades involving item i in the subtree below the current search node. The combined estimated revenue from all items is the heuristic h solved on line 4 of Procedure 11.[17]

Given the heuristic upper bound h on the additional revenue achievable in a subtree of the current search node, line 6 of Procedure 11 **bounds** the search. If the optimistic revenue achievable below the current search node is not as great as the best revenue seen so far, the search is pruned. The search efficiency gained through pruning depends on the tightness of the upper bounds used to compute h. Improvements to this heuristic are a topic for future work.

Procedure 11 also uses a **lower** bound on subtree revenue, $L \geq 0$. Line 7 finds the unassigned bid B_j with the greatest $p_j \geq 0$ which can be feasibly added to the allocation Λ. (If no such bid exists, $L \leftarrow 0$.) The idea behind L is that if we

[17]This formulation of h is a generalization of the heuristic described for multi-unit combinatorial auctions in Section 4.1 of (Sandholm, 2003).

can trivially increase surplus by adding a single bid, then we should ignore any subtrees that don't provide at least that increased amount of surplus.

L is used on line 9 of Procedure 11 to update the best allocation seen so far, IN^* and $\tilde{f}^*$. Note that when IN^* is updated, it does not include the bid B_j that was used to compute L. That bid will not be added to IN^* until we reach a search node in which B_j has been assigned winning. The purpose of updating $\tilde{f}^*$ is to avoid searching fruitless subtrees until we either assign B_j winning, or find some other combination of bids that further increases surplus.[18]

The final aspect of Procedure 11 which is unique to the **feasible-allocation** case concerns how a **branching bid** is chosen on line 10. When initializing the algorithm, we **sort** the bids in **descending order** of price. At each search node, if the allocation Λ is feasible, we choose an unassigned bid that has **maximum price** in this ordering. The goal here is to increase surplus as much as possible, as described in Section 4.2 of (Sandholm, 2003). Branching bids which yield high-surplus allocations allow BOB to **prune** later subtrees more effectively.

When the allocation Λ is *infeasible*, BOB tries to add bids or asks that would *improve* feasibility. Line 12 of Procedure 11 checks to see if all bids ($p_j > 0$) have been **deleted** from the graph. When this is the case, it is no longer possible to increase the **demand** for items. BOB therefore considers the contributions of **asks**, if they exist, which could produce a feasible allocation. When *no combination* of remain-

[18]This use of L requires a change to the test on line 2 of Procedure 11 from the algorithm presented in (Sandholm, 2003), which compares $g > \tilde{f}^*$. The change is needed so that when we eventually reach a search node at which the lower bound surplus $\tilde{f}^*$ is realized, we properly update IN^*.

ing asks can make the allocation feasible (line 13), there is no reason to search the current subtree.

Otherwise, we compute a lower bound $H \leq 0$ on the *decrease* in revenue that would result from making the current allocation **feasible**. The intuition behind H is that we can assign some asks *winning* that would make the currently-infeasible allocation feasible. Adding these asks ($p_j < 0$) to the current surplus g will *reduce* surplus. The surplus of a feasible allocation based on the current allocation is *maximized* when this **reduction** is *minimized*. H, computed on line 15, gives the amount of this minimal reduction. If the surplus $g + H$ is not as good as the best surplus found so far, $\tilde{f}^*$, we **prune** the search in line 17.

If undeleted bids *are* available, it can actually be useful to try and branch on a **bid** even though the current Λ is infeasible (line 19). The reasoning is as follows. Consider a branching bid B_k which demands an item i for which the current allocation is **infeasible**: $\Lambda_i > 0$. We call such a B_k an **overlap bid**, referring to the overlap between the items demanded by B_k and the items demanded by **assigned-winning** bids. By branching on an overlap bid, we cause descendant search nodes to branch on **multi-unit asks** *early*. We expect combinations involving multi-unit asks to reach high-surplus allocations sooner, in general, than those which have no demand overlap among bids.

When we choose an overlap bid for branching, we pick the **shortest** such bid. This bid demands the fewest items; it is the B_k for which the count of items i having λ_k^i is minimal. To break ties among shortest bids, we choose the one which demands the greatest number of items i for which $\Lambda_i > 0$.

Of course, an overlap bid may not **exist**, or there may simply be no **undeleted** bids at all. If we reach line 20, and have not pruned the search or selected a branching bid, we choose a branching bid that moves Λ closer to **feasibility**. Line 21 chooses a branching bid that most reduces the **magnitude** of the **net demand** for some item i whose allocation is currently infeasible. In our environment, this typically means finding an ask which increases the supply of i in the allocation Λ by just enough to achieve $\Lambda_i \leq 0$.

A variation of this technique is described in Section 4.2 of (Sandholm, 2003), in which the goal is to minimize $\max_{i|\Lambda_i>0} \Lambda_i + \lambda_k^i$. That is, it tries to minimize the **allocation** of i rather than the **magnitude** of the allocation. In our environment, minimizing the allocation tends to branch on **multi-unit asks** offering more units than are necessary to make some Λ_i feasible. Branching on an ask for fewer units (with a **lower-magnitude price**) would decrease surplus by a smaller amount, thus improving the future pruning behavior of the algorithm.

The net effect of the branching bid selection rules is to have the allocation Λ **oscillate** between feasibility and infeasibility. When the current allocation is feasible, we try to increase **surplus**. When it becomes infeasible, we try to choose branching bids that will restore **feasibility**.

Before handing bids to BOB, we perform several **preprocessing** steps to try and reduce wasted work. The first step removes those bids which demand at least one item that is not being supplied. This is generally useful when computing the **Vickrey discount** to a seller (§4.4.3.3). Removing the supply of that agent's item from the exchange may cause some buyers' bids to become **infeasible**. These bids will never be part of a **surplus-maximizing solution**, so it is safe to remove them from the input.

The next step removes those bids that are **unaffordable**. Some bids may demand only items for which a **single unit** is being supplied.[19] It is easy to determine whether such bids can possibly contribute **positive surplus** to a solution. The bid price plus the sum of the (single-unit) ask prices for the demanded items must be positive. If not, the bid may be eliminated from the BOB input.

The final step removes **excess asks**, those which supply more units of an item than are being demanded by all bids. We permit **free disposal** in our **combinatorial exchange**, so technically we can permit such asks. However, the structure of asks in our environment is such that a **multi-unit ask** has a price with magnitude *at least as great* as any ask offering fewer items. We also require that an agent offering k units of service submit asks for $1, 2, \ldots, k-1$ units. Therefore, a solution involving an excess ask can *never* have a surplus greater than a solution using an ask for the same item, but which offers fewer units.

These preprocessing steps are $O(|\mathcal{B}|^2)$, compared with the $O(2^{|\mathcal{B}|})$ of actual win-

[19]This preprocessing step does not work for multi-unit asks. We would have to consider the possible contributions to surplus of other bids which demand the same item(s), which is essentially winner determination.

ner determination. In the current implementation, we perform the steps prior to constructing the **graph data structure** used by BOB. Nodes in this graph are joined by **OR edges** when **item overlap** exists between bids. By performing the preprocessing in this graph, we could reduce the amount of searching that occurs when deciding whether to cull a given bid. For each of the three steps, only a bid's neighbors reachable by OR edges need be examined.

We conclude the description of BOB by showing how it solves the earlier example of Figure 2.9 from Section 2.3.2. That example had ten bids, but our implementation begins by **preprocessing** out two of them. Bid B_4, which demanded the route $(1, 5, 7)$, is **unaffordable**. The corresponding supply comes from B_6 (relay 5) and B_8 (sink 7). Since $p_4 + p_6 + p_8 = -0.1$, B_4 can be eliminated. After removing B_4, there is no demand for relay 5, so B_6 becomes an **excess ask**, and is removed.

Figure 4.14 shows the complete search tree examined by BOB. Search nodes marked with an "$\times$" indicate **bounding** or **pruning**. The current surplus g is labeled at each node; when the allocation at that node is **infeasible**, the surplus is shown in parentheses. In this example, BOB discovers the surplus-maximizing combination, with $\tilde{f}^* = 1.4$, *first*. It then backtracks, testing the combinations in which some $x_j = 0$. Most of these are simply **infeasible**. However, after assigning $x_3 = 0$, **heuristic pruning** is used to avoid searching that subtree, since the best allocation without B_3 has surplus 1. The algorithm then searches the subtree in which $x_2 = 0$, which does not yield a greater surplus than 1.4.

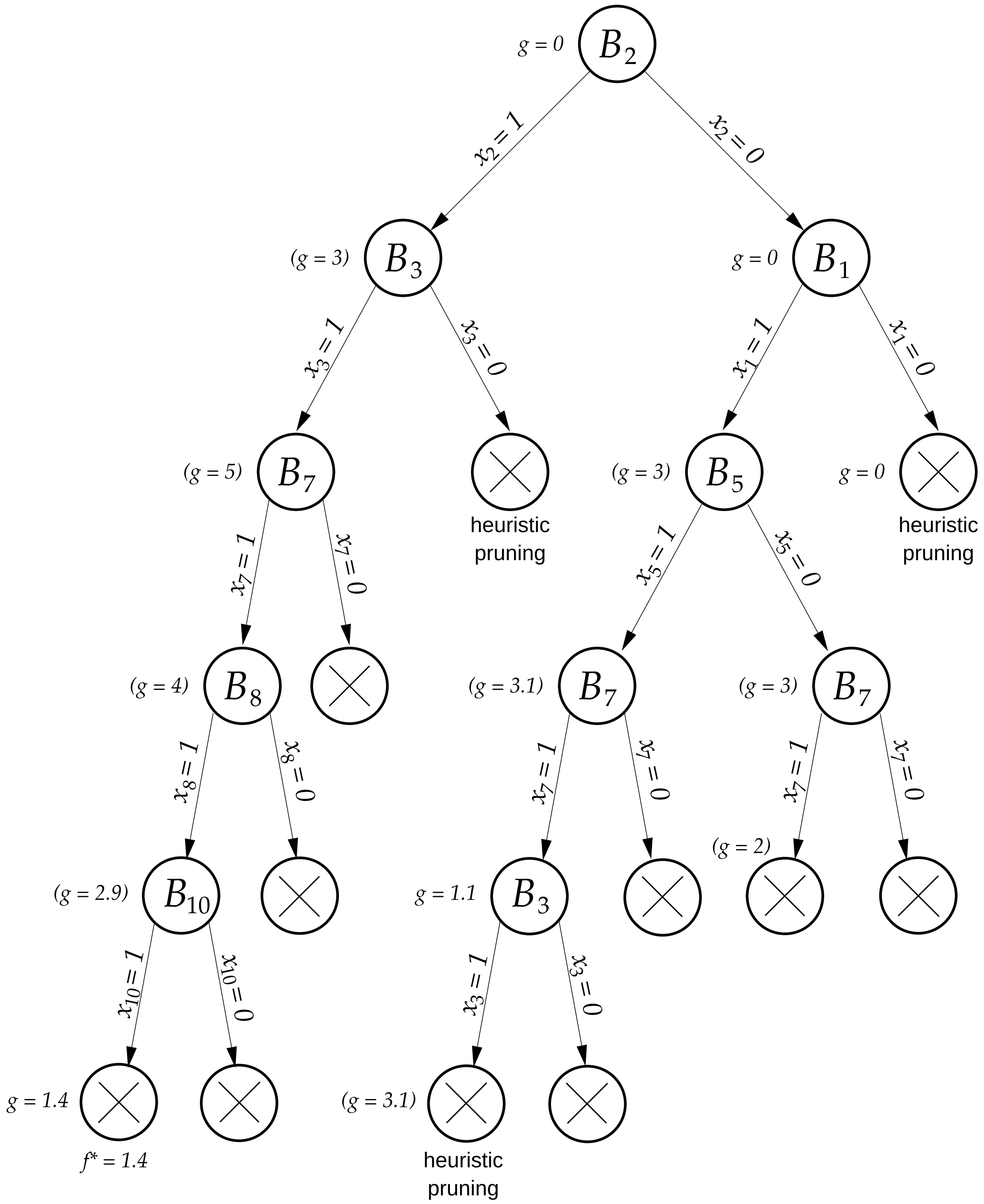


Figure 4.14: BOB search tree for the example of Figure 2.9.

This example visits 21 search nodes, compared with the $2^{|\mathcal{B}|+1} - 1 = 511$ nodes required for full search. Our implementation, running on a 1GHz mobile PowerPC 7450, completes in 2.8ms.

4.4.3.3 Vickrey Payment Computation

An agent's payment to the **Vickrey Clarke Groves mechanism** (§2.4.4) is computed by solving the **winner determination problem** with that agent's reports removed. In principle, this means solving a smaller exchange for *each agent*. However, we only actually have to compute **Vickrey payments** for those agents who had *winning* bids in the original exchange. An agent with no winning bids contributes nothing to the surplus of the original exchange, so removing its bids will not change the winning surplus. Therefore, its **Vickrey discount** is zero, and so is its payment.

After solving the winner determination problem using the algorithm described in Section 4.4.3.2, we solve smaller exchanges for each *winning* agent. These exchanges are smaller (*i.e.*, $|\mathcal{B}_{-i}| < |\mathcal{B}|$), not only because the winning agent's bids have been removed, but also because of **preprocessing**. Typically, removing a winning buyer's bids causes some asks to become **excess asks**, which are then culled. Removing a winning seller's asks causes some bids to become **infeasible**, which are also culled. As a result, BOB generally executes more quickly during Vickrey payment computation than during initial winner determination.

4.4.4 Implementation Phase

The prerequisite for entering **Implementation** is that a node knows its own **mechanism results**. The **root** learns its results after solving the **winner determination problem**, while the other nodes learn their results after receiving an XPM **Result** message (§4.4.3.1). Upon entering Implementation, a node configures its **active routes**, if any. It also configures timers for its own **power management suspension**, as well as for its **suspending neighbors list**.

A traffic source which receives its mechanism results first marks all of its active routes as having **invalid negotiations**. Since a node participates in one and *only* one mechanism at a time, its mechanism results completely specify the **active routes** it may use. If the source had no winning bids, it sets a timer for the duration of the **negotiation interval**, which is 20 seconds in our implementation. When this timer expires, the source will start the negotiation procedure again, at which point it might be able to win a route. If the source had winning bids, it uses them to replace whatever **active route** state it was maintaining for a given destination. Winning routes will be used until they break, until the end of the **negotiation interval**, or until their destination ceases to be **active**.

If a destination becomes **inactive** in the middle of the Implementation phase, the source no longer requires the services of the nodes on the negotiated route to that destination. The source can trigger a **restart**, which allows nodes to **renegotiate**. If there are no other active traffic flows, this allows nodes to resume power management earlier than if they waited for the current negotiation interval to expire. If the destination becomes inactive within the **renegotiation hold time** — 5 seconds in our implementation — before the scheduled end of the negotiation interval, then the interval is allowed to expire and no early restart is triggered.

Implementation is a **terminal state** for the four-phase **negotiation procedure**. A node eventually enters Implementation, either by correctly receiving its mechanism results, or by **abandoning** the procedure. Nodes abandon the **Structure** or **Negotiation** phases[20] after failing to receive an expected message within some fixed period of time. The timeout in Structure is 750 milliseconds, while Negotiation is slightly longer — 1 second — to account for time spent solving the **mechanism**. Table 4.2 summarizes the XPM timers. As with LPM, all nodes implement the same timer values.

Timer	**Duration**
Active route finalize	750ms
Structure phase timeout	750ms
Negotiation phase timeout	1s
Negotiation interval	20s
Inactivity renegotiation hold	5s

Table 4.2: XPM timer durations.

Even under stressful mobility conditions, the procedure will always **terminate**. If a node terminates by abandoning, then the mechanism in which that node participates may fail to be solved, or some nodes may fail to receive their mechanism

[20]Nodes automatically proceed from Discovery to Structure following the expiry of a **finalize** timer (§4.4.1).

results. Traffic sources which discover that the procedure has abandoned may **restart** it by triggering a **fast wakeup**.

Sources are *not* blocked from sending application messages while the negotiation procedure is in progress. A source which had a winning route in the previous negotiation continues to use that route while a new negotiation is proceeding. A source which does not have a previously-winning route uses whatever route is selected by the **DSR route cache**, subject to **route fidelity** constraints (§4.1.4).

The only time a source is blocked from sending is when it has an **unaffordable route**. That is, when it has bid on routes to a destination, but none of the bids were found **winning** during winner determination. This ensures that the negotiation process interferes minimally with normal application message delivery. The **results** of the mechanism are what determine the usability of routes.

4.5 Valuation Functions

Exchange Power Management provides a **framework** for agents in a mobile *ad hoc* network to express their **preferences** over configurations. So far, we have assumed that the agents have *some* way of computing their preferences, but have not described specific methods. This Section presents basic **valuation functions** which agents can use to determine their preferences. These functions demonstrate some of the fundamental capabilities of **exchange**-based power management. More sophisticated functions are an interesting subject for future work.

Conceptually, the valuation function is the realization of agent **type**, θ_i (§2.4.1). Literally, it is the function $v_i(\cdot, \theta_i)$ that, given a type, maps **configurations** of the network to some scalar **value**. In our environment, an agent's type encapsulates **private information** that the agent holds regarding its energy and communications status. For example, it might contain the amount of time a node has spent **suspending power management** on behalf of other nodes. Or, it might maintain the balance of **credit** the node holds with respect to the mechanism.

Valuation functions in our environment form the basis for two kinds of **reports** by the agents: **bid** prices and **ask** prices. In the examples of this Section, bid prices are primarily a function of **route length**, while ask prices are primarily a function of how much an agent "owes" the network. The individual functions in Sections 4.5.1 and 4.5.2 refine these ideas further.

The philosophy underlying bid pricing is that an agent must capture the **value**, or **work**, associated with message delivery. Intuitively, this is somehow related to the number of **relays** which must provide service in order to implement the route. We must be careful with this idea, however. Consider a naïve model in which each individual route P is valued at $S \times |P|$ for some **scaling factor** $S \geq 1$. Here, *longer* routes are valued more than *shorter* routes, which is contrary to the common notion of route quality!

The model we adopt instead uses route length as a general **estimate** of the work required to deliver messages to a destination. A source could use many methods to estimate this work. For example, it might compute an estimate of the **network diameter** based on the contents of its **route cache**. Or, it could compute the **mean route length** among routes it knows to a specific destination. In our implementation, after a source selects the routes for which it will submit bids, it picks the **longest** of these routes. The length of this route is passed to the valuation function, which computes a bid price. This price is applied to **all** bids to the destination, reflecting the value the source associates with **any** route that delivers messages to that destination.

Every agent submits one or more **asks**, which indicate the agent's reported **costs** for providing relay or sink service to the network. The valuation function determines the costs for forwarding or receiving a **single** traffic flow. Given this **single-unit ask price**, XPM proceeds to compute the prices for **multi-unit asks** as necessary.

Procedure 14 MAKE-SUPPLY

```
1: $p \leftarrow 0$ {price for $u$-unit ask}
2: $\dot{p} \leftarrow$ single-unit ask price {from valuation function}
3: for $u = 1, 2, \ldots, U$ do {$U$ is the greatest number of units to be supplied}
4:     $p \leftarrow p + \dot{p}$
5:     Create an ask for $u$ units with price $p$.
6:     $\dot{p} \leftarrow \dot{p} \times (1 - \text{Discount})$
```

Procedure 14 shows the method by which multi-unit ask prices are computed. An ask offering u units of relay or sink service has a price that is based on a **volume discount** applied to the $1, \ldots, u-1$ smaller asks. In our implementation, the discount is $\frac{9}{10}$. As an example, if the single unit ask price is -1, the prices for subsequent multi-unit asks are -1.1, -1.11, -1.111, and so forth. This method was chosen because it is easy to compute, and loosely reflects the concept of **marginal energy cost** (§2.1.2).[21] Methods which more accurately capture the energy costs of multiple flows should be examined as part of future research.

In the current implementation, we do not consider **complementarity** between sourcing and relaying or sinking at the same node. A node which is a winning traffic source must **suspend power management**. If such a source were to be a relay or sink as well, its additional energy costs would be **marginal** given that it is already suspending. To fully express this complementarity, agents would have to be able to submit bids that **demand** routes and **supply** their own relay or sink service. This is permitted by the definition of a **combinatorial exchange**, and the XPM bidding protocol syntax supports it. The current BOB implementation (§4.4.3.2) employs several optimizations that assume bids *only* demand items or *only* supply them, but not both. Exploration of more expressive bidding is left to future work.

4.5.1 Time Balance

The first valuation function we describe was designed to show the capabilities of XPM under the constraint that the **application message delivery ratio** is maximized. That is, the ratio of messages *received* at the application layer to the number of messages *sent* should be maximal. This is a common metric applied to rout-

[21] This particular discounting method probably overstates the **incremental costs** of the first few additional traffic flows, and understates them after about the tenth.

ing protocols, so we would like to see what influence, if any, XPM can have on performance given this constraint.

Section 4.4.4 explained that the only time messages are **blocked** in XPM is when a source has an **unaffordable route**. To maximize delivery ratio, we need a valuation function that produces bid prices which can afford *any* route. These are necessarily large bid prices, which result in large **mechanism payments** when a source has winning bids.

An agent that can afford any route is essentially **insensitive** to the cost of its routes. Rather than having **quasilinear preferences** (§2.4.1), such an agent experiences **direct** utility $u_i(\lambda, \theta_i) = v_i(\lambda, \theta_i)$ unrelated to its payment $p_i(\lambda)$. For agents with these preferences, the **VCG mechanism** provides no additional properties (*e.g.*, **strategy-proofness**) not already offered by the combinatorial exchange. In fact, an XPM for such agents could dispense with computing Vickrey prices altogether, as the only important information solved by the mechanism is the set of winning bids and asks.

We call this valuation function the **Time Balance** model. Bid prices are derived by scaling the (maximum) route length. Ask prices are an exponential function of a **balance** which measures the amount of time an agent has spent supplying service to the network against the amount of time in which it has demanded service.

Let $|P_{\max}|$ be the length of the **maximum-length route** passed to the valuation function during bid preparation. Under the Time Balance model, the bid price is computed as $p = S_t \times (|P_{\max}| - 1)$, for a scaling factor $S_t \gg 1$. In our implementation, $S_t = 5{,}000$, which seems to be large enough for nearly all routes.

Let $W = \left(w_1, w_2, \ldots, w_{|W|}\right)$ be the sequence of time windows in which a node has **suspended** power management to serve as a relay or sink. The total time the node has spent suspending while providing service is $\sum_{i=1}^{|W|} w_i$, in seconds. Let $R = \left((P_1, r_1), (P_2, r_2), \ldots, \left(P_{|R|}, r_{|R|}\right)\right)$ be the sequence of tuples describing the routes this node has used as a source, and the time windows in which each route was used. For each route P_i, $|P_i| - 1$ other nodes (the relays and a sink) spent r_i seconds suspending power management on behalf of this node. The total time other nodes have spent suspending power management for this node is $\sum_{i=1}^{|R|} (|P_i| - 1) \times r_i$, in seconds. Define the **time balance** b_t as:

$$b_t = \sum_{i=1}^{|W|} w_i - \sum_{i=1}^{|R|} (|P_i| - 1) \times r_i$$

When b_t is negative, this node "owes" the network service. Otherwise, this node is "owed" service by other network nodes. Note that b_t accounts for suspension activity *while* the negotiation procedure is executing, as well as *after* a negotiated configuration has been reached. The single-flow ask price is computed as $p = -A_t^{b_t}$, where $A_t > 1$ is a constant which determines how aggressively prices rise as the node's balance becomes increasingly positive. In our implementation, $A_t = 1.003$, which keeps prices from growing too quickly.

Figure 4.15 shows the single-flow ask pricing as a function of the time balance.

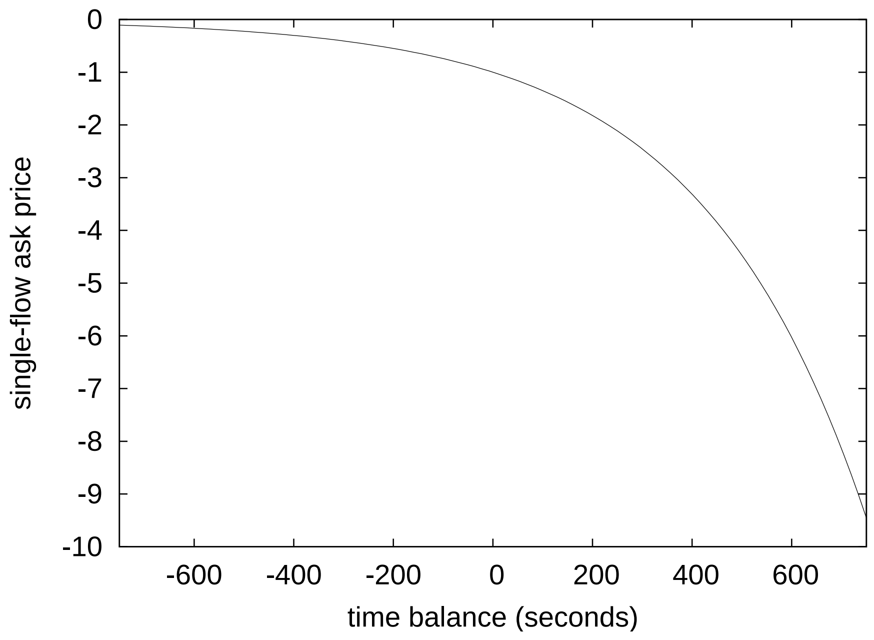


Figure 4.15: Time Balance ask pricing.

The ask price is -1 when the balance is zero, and asymptotically approaches zero as this node "owes" more to the network. When the node has a **positive** balance with respect to the rest of the network, it rapidly becomes less **affordable** to other sources. Larger values of A_t cause the node to be more aggressively expensive. In our current implementation, all nodes use the same A_t. It would be interesting to **dynamically** choose A_t based on a node's beliefs about its own traffic-sourcing behavior (*e.g.*, high-volume source, bursty source) or some other feature.

4.5.2 Credit Balance

To make use of the **incentive compatibility** properties of the VCG mechanism, agents need to care about how much they pay to implement an outcome. The basis for this mechanism is a **payment rule** which removes the incentive for an agent to **strategically manipulate** its valuation reports in the hope of paying less (or being paid more). We want a valuation function that, unlike the Time Balance model, is sensitive to payment amounts.

The basic concepts underlying the **Credit Balance** model are similar to those from the Time Balance model. Bid prices are mainly proportional to route length, and ask prices are a function of the **credit balance** $b_c = -\sum p_{i,\text{vick}}$. Limiting the measure of a node's **contribution** to the network to just this credit balance, however, can be inaccurate with respect to the node's actual energy consumption.

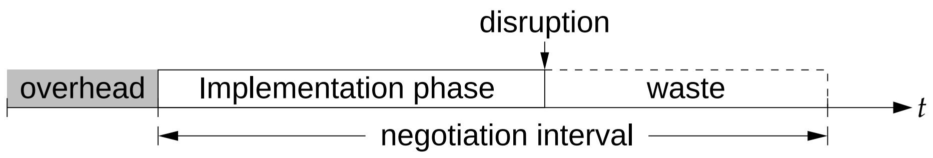


Figure 4.16: Overhead and waste in credit-based contribution assessment.

Figure 4.16 shows the two factors which lead to the inaccuracy: **overhead** and **waste**. In XPM, agent payments are based on a network configuration that lasts for a **negotiation interval** (§4.4.4). Essentially, the agents are paying for (or are paid for) the service that occurs during the **Implementation phase**. But before the Implementation phase is reached, some nodes must spend time in the Discovery, Structure, and Negotiation phases, during which they are **suspending power management**. This time is not accounted for in the agent payments; we call such time **overhead**.

During the Implementation phase, a **disruption** may occur which causes the agents to **restart** the negotiation process. Examples of disruptive events include route breakage or the arrival of a new traffic flow. Following a disruption, the current negotiation is abandoned, but the agents have *already paid* (or have been paid) for a full negotiation interval of service. These payments are partially wasteful, since a full interval was not used. We call the time between a disruption and the end of the disrupted negotiation interval **waste**.

Overhead and waste are opposites. One measures work that is unpaid; the other measures work that was paid for, but not performed. We can combine these measures to make Credit Balance ask pricing more closely reflect Time Balance pricing (which automatically accounts for overhead, and is **insensitive** to wasted payments). The idea is for certain periods of **waste** to *pay for* periods of **overhead**. Specifically, an agent with winning asks and *no* winning bids — a "pure" relay or sink — adds the wasted remainder of a disrupted negotiation interval to the counter Waste. All agents add their periods of overhead to the counter Overhead. Call the difference, Overhead − Waste (in seconds), an agent's **net overhead**.

Credit Balance ask pricing is a **piecewise** function of the credit balance:

$$p = \begin{cases} -A_c^{b_c} & \text{if } b_c < 0 \\ -Lb_c - 1 - \dfrac{\text{Overhead} - \text{Waste}}{\text{NegotiationInterval}} & \text{if } b_c \geq 0 \text{ and Overhead} > \text{Waste} \\ -Lb_c - 1 & \text{otherwise} \end{cases}$$

The constant A_c is similar to that used in the Time Balance model, but smaller, since the magnitude of b_c is generally much less than that of b_t. In our implementation, $A_c = 1.1$. Again, the ask price asymptotically approaches zero as b_c becomes increasingly negative. When $b_c = 0$, the ask price is -1.

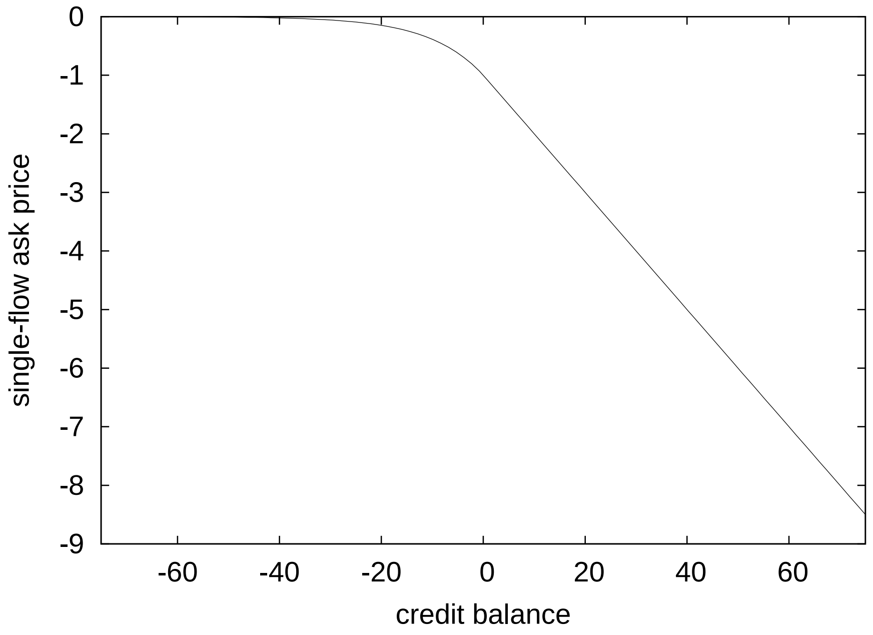


Figure 4.17: Credit Balance ask pricing without surcharge.

When the credit balance is non-negative, we switch to a **linear** function of b_c, with $L > 0$. For our implementation, $L = 0.1$. In the linear region, prices rise less aggressively than if we were to use a pure exponential function. This helps to keep relays and sinks **affordable** as their credit balances increase. If the **net overhead** is positive, then we try to "sell off" some of the overhead by applying a **surcharge** to the ask price.[22] The basis for the surcharge is a **reference price** of 1 unit of credit per **negotiation interval**. This price is scaled to the actual amount of overhead being

[22]The amount of overhead sold in any ask is capped at 5 seconds.

"sold." If an ask whose price has the surcharge applied is found **winning**, then we adjust the net overhead counter by the amount of overhead that was "sold." When the net overhead is non-positive, no surcharge is applied. Figure 4.17 shows the single-flow ask price for this latter case.

Bid prices are again computed by **scaling** the length of a maximum-length route. Let the **base price** of a bid $p_{\text{base}} = S_c \times (|P_{\max}| - 1)$, with $S_c \geq 1$. The scaling factor S_c is much smaller than the S_t used in the Time Balance model, reflecting the sensitivity of agents to **overpayment**. In our implementation, $S_c = 1.5$.

A consequence of the more conservative bid prices used in the Credit Balance model is that some bids may be **unaffordable**. That is, their price is too low to yield positive surplus given the corresponding asks. When an agent's bids are unaffordable, it is **blocked** from sending messages in the current negotiation. We employ several techniques to help an agent whose **base price** is too low. The justification for this is a practical interest in keeping **delivery ratio** high. We concede that a game-theoretic interpretation of these techniques may be a challenge to develop.

The first technique we use to increase the chances that a bid will be **affordable** is the use of a "**tip**." Call $b'_c = b_c - p_{\text{base}}$ the **remaining balance** after accounting for a payment equal to the base (bid) price.[23] The bid price is then computed as:

[23]Due to the use of **Vickrey discounts**, the bid price is the *most* an agent will pay when it wins.

$$p = \begin{cases} p_{\text{base}} & \text{if } b'_c < 0 \\ p_{\text{base}} + \min\left(b_c \times \text{Tip},\, b'_c\right) & \text{otherwise} \end{cases}$$

The Tip > 0 is a fraction of the current credit balance that an agent is willing to pay to increase its chances of winning. In our implementation, Tip $= \frac{1}{10}$. An agent offers *at most* its remaining credit balance in a single tip.

The second technique is a **multiplier** of the bid price just computed. The multiplier begins with unit value, and is incremented by $\frac{1}{2}$ every time a source has losing bids. The effect of the multiplier is to **successively increase** its bid price with each losing negotiation, in the hopes of eventually winning. The multiplier is reset to the unit value at the conclusion of a negotiation which is allowed to expire. An interpretation of the multiplier might be an agent which **revises** its beliefs about the **value** of its routes over time. Because this "learning" implies some sort of **interdependence** between sequential runs of the mechanism, we suggest that it is probably better to avoid such behavior in future valuation functions. As a practical matter, the **tip** and **multiplier** techniques do reduce the number of unaffordable routes experienced by agents.

4.5.3 Summary

Neither the Time Balance nor Credit Balance valuation functions incorporate the concept of **energy**. For example, neither involves a measure of "remaining battery life." This is a useful abstraction, since it frees the agents from having to compare **energy profiles** across heterogeneous devices. By expressing **contribution** to the network in terms of time spent **suspending power management**, or in the even more abstract units of **credit**, these valuation functions are *independent* of the specific hardware implementation.

The notion of **balance**, rather than a "gas gauge" form of resource measurement, is useful for a number of reasons. Some earlier designs (Chen et al., 2001; Xu et al., 2001) based their selection of **active** nodes on **expected lifetime**. A node which is closer to exhausting its battery is less likely to be chosen for service. This concept is perhaps most applicable to environments in which all nodes are inserted into an environment at time zero, and the network must function for as long as possible. **Sensor networks** are an example of systems which have this property.

By contrast, **balance** is a **persistent** measure of a node's contribution to the network. We envision an environment in which nodes continually enter and leave the network. Here, there is no concept of network **longevity**; there is only the steady-state energy consumption rate of the nodes. A node's balance can be preserved when it leaves and later returns to the network. Balance still makes sense if a node's battery is **recharged**, or if it switches to an alternate radio implementation.

Finally, balance is attractive because it provides a link between a node's own **traffic sourcing activity** and its **total communications energy consumption**. A node which sources more must be able to pay for its traffic, so it must provide more service to the network, thus increasing its energy consumption. A node which wants to **reduce** its energy consumption now has a way to do so: simply find a way to source fewer messages. This is a major difference between a **negotiated** design like XPM and a **non-negotiated** design like LPM. With LPM, an "unlucky" node could find itself being made to provide relay service for many sources, thus increasing its energy consumption even if its own communication needs are modest. With XPM and balance-based valuation functions, a potential relay can examine its **contributions** to the network, and conclude that it does not "owe" the network additional service. It can then report a high ask price indicating its preferences, and effectively **limit** its additional energy consumption.

4.6 Future Improvements

The **Exchange Power Management** design presented in this Chapter is the first application of a **strategy-proof mechanism** to the problem of **power management** in a mobile *ad hoc* network. XPM tries to find a balance between game-theoretic interests such as **incentive compatibility** and network protocol interests such as **communications efficiency**. Incentive compatibility constrains the set of **optimizations** we can perform, both to the communications and compute performance of our protocol. Communications efficiency determines the **scalability** of the protocol, and can limit its usefulness in some environments.

Section 2.4.5 showed how the result of (Nisan and Ronen, 2000) applies to the power managing network environment. **Suboptimal mechanism solutions** cannot guarantee strategy-proofness; agents may be incented to lie about their preferences in order to "help" the winner determination algorithm reach a better result. We showed that a mechanism based on the solution of **separate**, independent combinatorial exchanges was indeed suboptimal. We then argued that, for overlapping routes, the use of a **single** combinatorial exchange was necessary to ensure strategy-proofness.

This observation informs the manner in which XPM handles **disruptive events** during a negotiation interval. Since an **optimal solution** to the winner determination problem is required to preserve incentive compatibility, XPM reacts to **new traffic flows** and **route breakage** by computing a *new* optimal solution. At any time, the agents are either implementing the results of an optimal mechanism, or they are trying to construct the solution to such a mechanism.

XPM uses the **fast wakeup** event (§4.1.2) as a signal to **restart** the negotiation procedure. Nodes receiving a fast wakeup reconstruct their **spanning trees** "from scratch," submit new bids, and implement the results of a new, optimal mechanism. Since fast wakeup is a **global** signal, even nodes participating in mechanisms *unaffected* by a new or changed route must re-run the procedure.

From a protocols perspective, triggering the solution of possibly many mechanisms throughout the network based on a local event (*e.g.*, a broken link) raises scalability concerns. Yet many **local** means of reacting to such events fail to yield an **incentive compatible** outcome. Section 4.6.1 argues against some local repair techniques, and presents an alternative which preserves strategy-proofness without triggering network-wide renegotiation.

This improved renegotiation method utilizes the **spanning tree overlay** from Section 4.4.2.1 in a different way. Rather than **discarding** the tree after a negotiation completes, the new method may **reuse** the tree during renegotiation. This new application calls for a more graceful handling of broken tree edges. Section 4.6.2 describes one possible approach.

Finally, under conditions of high **node mobility**, a negotiation procedure for route selection may not be able to keep up with the rate of **topology change**. Although such an environment is not the subject of this thesis, Section 4.6.3 describes a method for gracefully **failing over** to LPM under high mobility. This approach completely sacrifices the energy-shaping advantages of XPM, but allows DSR to continue experiencing the latency benefits of **multihop power management**.

4.6.1 Interim Mechanism Solutions

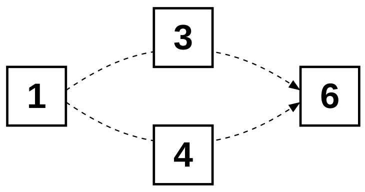
XPM provides a facility for selecting a subset of network nodes to support active traffic flows. Section 2.1.2 showed that, for an 802.11 interface, energy consumption in the **idle** state dominates. Accordingly, XPM **sellers** are offering to spend some fixed interval of time (*e.g.*, 20 seconds) **suspending** power management while they relay or sink messages. While suspending, these nodes incur the high energy costs of the idle state, rather than the lower costs of the power management **doze** state.

Ideally, once a negotiation has been reached, the network would remain **static** for the remainder of the **negotiation interval**. No links would break, no new traffic flows would become active, and no existing traffic flows would terminate. In reality, all of these events are possible. When such changes occur, we can apply corrective measures from one of the following categories:

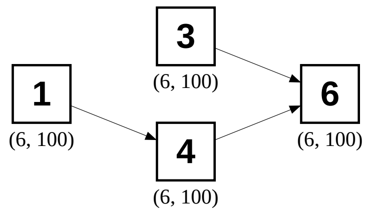
- **Renegotiate optimally.** When overlap exists between an existing negotiation and a new or changed set of routes, solve a new, single exchange **optimally**. The solution must be based on the reports of all agents in the union set of the existing negotiation and the new or changed routes.
- **Renegotiate suboptimally.** Solve a local, smaller mechanism which only requires reports from the agents on the new or changed routes.
- **Don't renegotiate.** Wait until the existing negotiation expires, then add the agents on the new or changed routes to the next solution. Until that time, either permit sources to choose routes at their discretion, or block them from sourcing messages.

In the order presented, these approaches exhibit increasing **communications efficiency**, and therefore, increasing **scalability**. XPM currently uses an approach from the first category, and solicits bids from all affected agents in order to solve the exchange optimally. As explained earlier, *no* approach from the second category can guarantee **strategy-proofness**. The third category either reduces to LPM — the sellers cannot express their preferences, and are not compensated for their costs — or induces extremely high **delivery latency** due to blocking.

This thesis is about strategy-proof mechanisms, so our approach comes from the first category. We can improve upon XPM by **reusing** and **augmenting** the spanning tree from the existing negotiation, rather than generating a whole new tree. The agents on the new or changed routes pass bids to the (old) root, which retains the bids used to reach the existing solution. The exchange is solved optimally using a combination of old and new bids. This **interim mechanism solution** is then distributed among the agents, which implement the new configuration for the remainder of the existing **negotiation interval**.



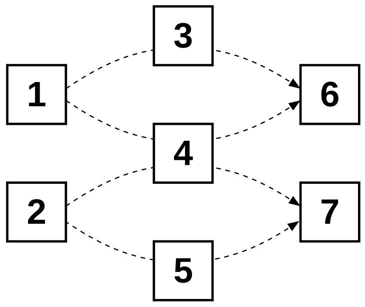
(a) Existing flow 1 ⇝ 6.



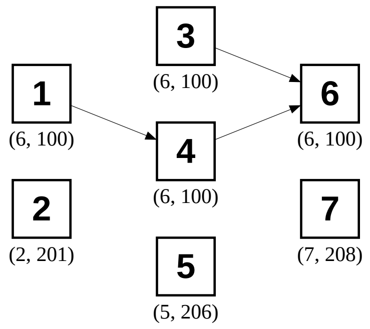
(b) Existing tree edges.

Figure 4.18: Existing flow and tree edges.

Adapting a familiar example, the agents in Figure 4.18(a) execute a mechanism to allocate a route for 1 ⇝ 6. The nodes form the spanning tree depicted in Figure 4.18(b), with the root 6 having the oldest timestamp, 100. Bids are passed to 6, which solves the exchange optimally, and distributes the results.



(a) Adding flow 2 ⇝ 7.



(b) Augmented vertices.

Figure 4.19: Augmenting an existing negotiation.

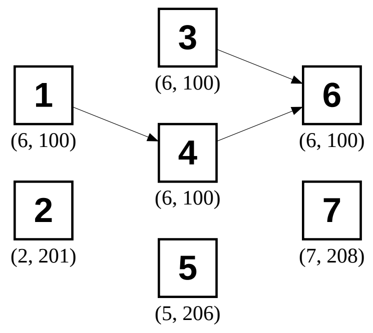
Before the end of the negotiation interval, suppose that source 2 decides to bid on one of the routes for 2 ⇝ 7, shown in Figure 4.19(a). This might happen because the flow 2 ⇝ 7 has recently become active. Or, 2 might have previously known a different set of routes to 7 which broke, and the newly-discovered replacement routes overlap with the negotiation from Figure 4.18. In any case, the problem is to **augment** the existing tree with the new **vertices** 2, 5, and 7 shown in Figure 4.19(b).

Under XPM, the **fast wakeup** which preceded 2's **Route Discovery** is interpreted by the other agents as a signal to **restart** the negotiation procedure. Agents 1, 3, 4, and 6 would re-run Structure, joined shortly thereafter by 2, 5, and 7. All seven agents would submit bids to a common root, which would compute an optimal solution.

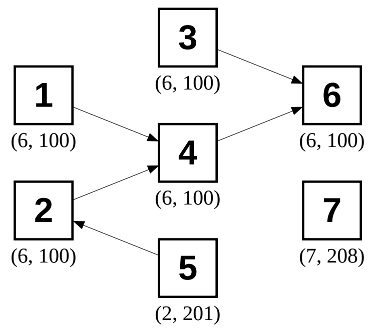
In the improved approach, we *no longer interpret fast wakeup as a global negotiation restart signal*. Upon receiving a fast wakeup, nodes suspend power management as before. Traffic sources whose demand set has not changed *do not* immediately initiate the process of building the spanning tree. As we will show, nodes whose **existing negotiations** do not overlap with new or changed routes continue to use their negotiations, and do not process new Structure or Negotiation messages. Only those sources having new or changed routes send **Notify** messages along those routes. Such sources begin the tree construction process as in XPM, as the **root** of a singleton tree, with a **timestamp** sampled at the time of the fast wakeup.

A Notify for a new or changed route may reach a node which participated in an existing negotiation. Call this an **overlap node**, referring to the overlap between the existing negotiation and the new or changed routes. Overlap nodes remember their **tree state** from the existing negotiation. That is, they know the root and timestamp for their tree, as well as the **edge classifications** (*e.g.*, Parent, Child) for their neighbors. This is a departure from XPM, in which tree state is discarded once an agent receives its **Result** message and enters **Implementation**.

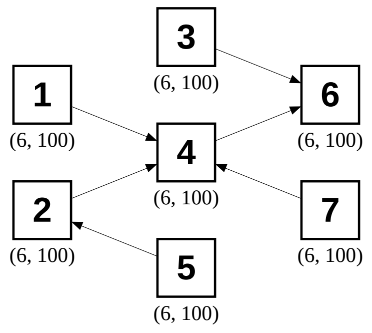
Tree augmentation begins at overlap nodes. XPM already uses a tree formation algorithm in which older subtrees **conquer** younger subtrees, based on timestamps. We can use exactly this algorithm for augmentation. Sources with new or changed routes begin with a timestamp that is **later** (younger) than the one remembered by nodes in existing negotiations. An overlap node will have an older timestamp than the Notify messages it receives from sources with new or changed routes. Therefore, the overlap node will conquer the nodes on those routes having later timestamps.



(a) Before 2's notification.



(b) During 2's notification.



(c) Final tree.

Figure 4.20: Augmenting the spanning tree.

Figure 4.20 illustrates spanning tree augmentation. Following the fast wakeup triggered by source 2, nodes 2, 4, and 5 are the roots of singleton trees with timestamps sampled at the time of the fast wakeup. Nodes 1, 3, 4, and 6 — which are already implementing a negotiation — do not update their root or timestamp information, and remember their existing edge classifications. The initial state, before 2 sends its **Notify** messages, is shown in Figure 4.20(a). In Figure 4.20(b), 2 has sent Notifies to 4 and 5. 5 agrees to be a child of 2, but 4 has an older timestamp from the existing tree, and conquers 2. 2 will next conquer 5, and ultimately 7 will be conquered after receiving the propagated Notifies from 4 and 5. The final tree appears in Figure 4.20(c).

Note that only a *subset* of the nodes processed **Structure** messages in the example. Nodes 1, 3, and 6, which do not appear on the new or changed routes, did not process any **Notify**, **Update**, or **Respond** messages. This is an improvement over XPM, in which *all* nodes on active routes process Structure messages.

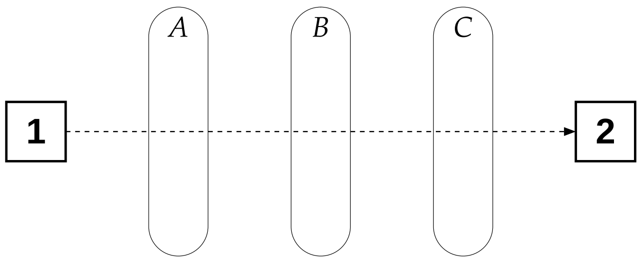


Figure 4.21: Flow 1 $\rightsquigarrow$ 2 overlaps existing negotiations A, B, and C.

Figure 4.21 illustrates the case of **multiple overlap**. The new flow 1 $\rightsquigarrow$ 2 contains relays from three existing negotiations, labeled A, B, and C. In this situation, the existing tree with the oldest timestamp will **conquer** the other trees (as well as the augmented vertices).

The precondition for entering **Negotiation** is similar to that in XPM. A node which has received a Notify from a source with new or changed routes, and which has classified all of its **incident edges**, enters Negotiation after receiving bids from all of its children. **Leaf nodes** begin this **wave algorithm** as in XPM, by sending **Submit** messages to their parent. Bids are aggregated at parents, and eventually propagate up to the root.

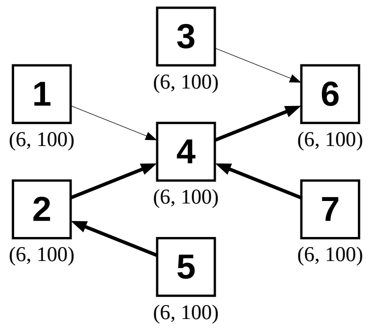


Figure 4.22: Edges traversed by Submit messages in the augmented tree.

Figure 4.22 shows in bold the tree edges across which Submit messages are passed in the augmented tree. Nodes which do not lie on a tree path between **overlap nodes** and the **root** do not process Submit messages. This again improves upon XPM, in which *all* nodes on active routes process Submit messages.

We can reduce the number of bids that must be **encoded** in Submit messages passing between overlap nodes and the root. In XPM, a parent combines *all* of its childrens' bids with its *own* bids, and passes the aggregated bids towards the root. In the improved design, a parent only needs to include bids submitted by agents at **overlap nodes** or **augmented tree vertices**. Bids from overlap nodes must be considered because such nodes must express their preferences for relaying additional traffic flows. In the example of Figure 4.18, node 4 reported its costs for relaying a single flow. When the second flow was added in Figure 4.19, 4 completely specifies its preferences by submitting an additional **ask** for two flows.

In Figure 4.22, node 4 must include all bids and asks from its **augmented vertex descendants**, 2, 5, and 7. Node 4 also includes its *own* asks, offering to relay one or two flows (joined by **XOR constraints**). Node 4 does *not* include any asks from its child 1, since 1 is neither an overlap node nor part of the augmented vertex set. Once 6 receives the new bids from 4, it adds them to the set $\mathcal{B}$. If an agent — such as 4 — already submitted bids in the existing negotiation, its old bids are **replaced** when it submits new ones. The existing bids of agents that do not submit replacements are preserved.

Not requiring bid submissions from all agents improves communications efficiency. It does, however, requires us to specify how the existing bids will be transformed into results in the **interim mechanism**. The existing prices reflect valuations for a *full* negotiation interval, but the interim mechanism solves a configuration that will be used for only the *remaining fraction* of that interval. If we can assume that agent valuations **scale** directly with time, then it is easy to incorporate the existing bids into the new solution. Suppose the interim mechanism is solving a configuration that will last for the remaining fraction $s < 1$ of the current interval. All existing bid prices are multiplied by *s by the root*. Agents submitting new bids scale their prices by *s themselves*. These agents can compute the value of s by examining the tree timestamp. The negotiation interval is **common knowledge**, and the tree timestamp marks the (approximate) start of the existing interval. Therefore, $s = 1 - \frac{\text{CurrentTime} - \text{TreeTimestamp}}{\text{NegotiationInterval}}$. In this manner, *all* bids express values for a common subinterval.

Given the transformed existing bids and the newly-submitted bids, the root can solve the **winner determination problem**. The **Branch-on-Bids** variant described in Section 4.4.3.2 computes an optimal solution "from scratch." Depending on the extent to which new bids override existing bids, it might be possible to use **incremental winner determination** techniques to reduce the amount of computation at this step (Kastner et al., 2002; Sandholm, 2002). Incremental methods apply to combinatorial allocation problems when the set of bids $\mathcal{B}$ is **perturbed**, for example by adding bids or changing bid valuations. The goal is to take an existing solution, make some changes according the perturbations, and arrive at a new solution in less time than solving the full problem again.

Once a new solution is reached, we compute **Vickrey payments** as in Section 4.4.3.3. Call agent i's payment in the existing negotiation $p_{i,\text{vick}}$, and its payment in the interim mechanism $p'_{i,\text{vick}}$. Our interpretation of interim payments is that i pays $(1-s) \times p_{i,\text{vick}} + p'_{i,\text{vick}}$ for the *entire interval*. That is, the original payment is scaled to the time for which the original solution was in effect. For the remaining fraction of the negotiation interval, s, the agent pays its Vickrey payment in the interim mechanism.[24]

There are some special cases in which the solution to the interim mechanism does not affect agents implementing an existing result. If an interim mechanism is solved with the **same bids** as the existing solution, but with prices scaled by s, then the interim Vickrey payments $p'_{i,\text{vick}} = s \times p_{i,\text{vick}}$. Therefore, agent i's **total payments** for the interval do not change due to the interim mechanism. Another

[24]If we have to solve a *second* interim mechanism, then we scale the payment of the first interim mechanism to reflect its **effective subinterval**, and so forth.

case concerns agents with no winning bids or asks in either the *existing* or *interim* solutions. These agents make have $p'_{i,\text{vick}} = p_{i,\text{vick}} = 0$. A third case occurs when the introduction of new bids does not change the mechanism solution. In the example of Figure 4.19, suppose source 2 cannot **afford** any of its routes. The **winning agents** in the interim mechanism are the same as those in the existing solution. When any of these cases occur, it may be possible to improve the distribution of **Result** messages. Specifically, agents which have not explicitly submitted new bids to the interim mechanism and whose results **do not change** do not need to receive a new Result message. They may continue to implement the existing result for the remainder of the negotiation interval.

In the general case, the interim solution will cause some agents' **Vickrey payments** for the total negotiation interval to change. These agents' bids may change from *winning* in the existing solution to *losing* in the interim solution (or vice versa). Such agents must receive a Result message informing them of the **status** of their bids and the new amount of their Vickrey payments. We also require that agents which have submitted **new bids** to the interim mechanism receive a Result message, whether or not their status or payments change.

So far, we have emphasized improvements in **message efficiency** made possible by the **interim mechanism** approach. These improvements do not necessarily translate into **energy savings**, however. Consider a node which is implementing an existing negotiation in which it had no winning bids. This node has resumed

power management. During the negotiation interval, the node observes a **fast wakeup**, and suspends power management while waiting to see if an interim negotiation will occur.

The first opportunity a node has to learn that it will participate in an interim mechanism happens if it receives a **Notify** or **Update**. The first of these must arrive before the **Structure timeout** (750 milliseconds in our implementation). However, a node will not *necessarily* receive a Structure message, as in the case of nodes 1 and 3 in Figure 4.20. The next opportunity a node has to learn that it is participating in an interim mechanism is during **bid submission**. For example, in Figure 4.20, the first message received by node 6 is a **Submit** from node 4. Again, nodes 1 and 3 do not receive these messages. The final opportunity occurs as **Result** messages are distributed, which may happen several seconds after the fast wakeup was observed.

Nodes should not need to wait, suspending power management, for the worst-case Result delay to find out if they are **participants** in an interim mechanism. Non-participants should **resume** power management as soon as possible. We can achieve this by borrowing the application of a **keep-alive message** from **multicast routing** in *ad hoc* networks (Jetcheva, 2004). A node which is implementing an existing negotiation, and which receives a Notify or Update as part of an interim negotiation, sends a keep-alive message to all of its neighbors *in the existing tree*. (Section 4.6.2 discusses how to handle broken tree edges.) Those nodes in turn propagate the keep-alive to *their* tree neighbors, and so forth, until all members of the existing tree have received the message.[25] This process must complete before the Structure timeout. If a node receives neither a **Notify**, **Update**, nor a **keep-alive** before this timeout, it may assume that it is not a participant in an interim mechanism. As a result, the **energy cost** of a fast wakeup to an **irrelevant** node (*i.e.*, one which will not participate in the interim mechanism) is *at most* the Structure timeout duration spent in the **idle** state.

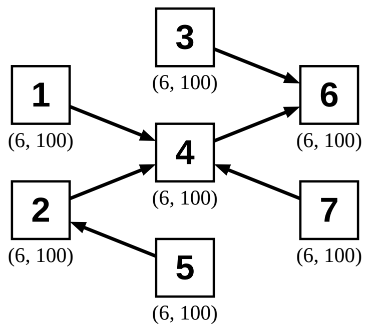


Figure 4.23: Worst-case edge traversal by Result messages.

Participants in the interim mechanism suspend power management until they receive their **Result** message. In the worst case, Result messages will traverse all $|E|$ edges of the spanning tree. This corresponds to the case in which every agent has at least one bid that changes from *winning* to *losing* (or vice versa), or whose **Vickrey** payment changes. Figure 4.23 illustrates this case; a Result message traverses each bold edge (but in the direction opposite of the arrow, since the arrows point from **child** to **parent**).

[25]With suitable **pruning**, at most one keep-alive traverses each **tree edge** in this step. Multiple nodes at different positions in the tree can initiate keep-alive dissemination concurrently.

Upon receiving a new Result message, nodes enter **Implementation** as in XPM. Implementation following an **interim mechanism solution** only lasts for the remainder of the existing **negotiation interval**. For those agents which do not need to receive Result messages, we can allow them to resume their previous Implementation status following a **timeout**.

4.6.2 Spanning Tree Repair

XPM constructs a **spanning tree overlay** "from scratch" following a **fast wakeup** event. Although the tree typically exists for less than a second, it is still possible for a **tree edge** to break before a **Result** message traverses it. If we use **interim mechanism solutions** (§4.6.1), the probability of broken edges increases since the tree may be **reused** some time after it was initially constructed. XPM responds to edge breaks by restarting the tree formation process via a fast wakeup. This is a **global restart**, which is inefficient; any time an edge breaks in *any* tree, *all* trees are reconstructed. We would like a more **localized** means of responding to tree edge breaks. One possible approach is to **repair** the tree by having the nodes in the **subtree** beneath the broken edge **rejoin** the tree via other edges.

Tree repair relies on information obtained during **tree formation**. We assume that each tree node knows the **root** and **timestamp** of its tree, as well as the **classifications** of its **incident edges** (*e.g.*, Parent, Child). The problems we must solve are to **detect** broken edges, and to **reconnect** disconnected subtrees. The former is easily solved, both **reactively** and **proactively**. The latter requires more work, but has already been solved in other problem domains.

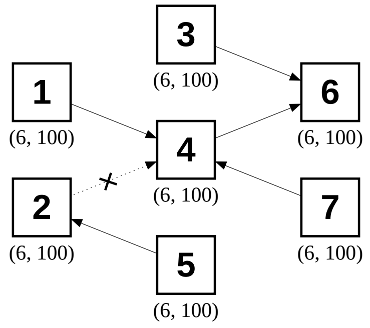


Figure 4.24: A broken tree edge $(2, 4)$ disconnects the subtree containing 2 and 5.

Figure 4.24 shows an example of a broken tree edge. When the link $(2, 4)$ fails, nodes 2 and 5 are disconnected from the tree. XPM detects such a break using **link layer acknowledgments**. The first time any node attempts to send a message across the broken edge, the link layer will report that the transmission failed. For example, either 2 or 4 might send a **Structure message** across the link. Or, once Structure has completed, 2 (the child) might send a **Submit** or **Revoke**, or 4 (the parent) might send a **Result**. Relying on link layer acknowledgments is a **reactive** approach; the break is not discovered until a node needs to use it.

The reactive approach just described introduces no **overhead**, but there are circumstances in which it would be useful to discover the break earlier. For example, suppose the link $(2, 4)$ fails *after* a Submit message traverses it, but *before* the Result arrives. Node 4 may have already sent a Result to 7 before detecting the break. At this point, even if nodes 2 and 5 could be **reconnected** to the tree, say, as **descendants** of 7, it is *too late*. Once 7 receives its Result, it enters **Implementation**. In

XPM, Implementation is a **terminal state** for the protocol; no additional Structure or Negotiation messages may be processed. Also note that 2 will *not* detect the break, and will **time out** waiting for Negotiation to conclude. 2 infers that the protocol has **failed** somehow, but does not know specifically that one of its incident edges has broken.

We could perhaps detect the break earlier using **proactive** means. **Keep-alive messages**, as used in **multicast protocols** (Jetcheva, 2004), allow *both* endpoints of a broken link to detect the failure within some bounded delay. Consider a change to XPM in which, as soon as a node classifies one of its incident edges as type **Child**, it begins to **periodically** send keep-alive messages across that edge. Since these are **directed** messages, the sender can detect when the transmission fails. Similarly, if the period (plus some **margin**[26]) elapses and the child does not receive a keep-alive, then the child can infer the failure as well.

This change allows the parent to detect the break in *at most* the time between periodic keep-alive transmissions. The child detects the break in *at most* this period plus a small margin. It is important that the child be able to detect the break, since we want the nodes in a **disconnected subtree** to be able to initiate the process of **reconnecting**.

Figure 4.25: A broken tree edge $(1,2)$ from source 1′s demand set.

[26]If a keep-alive message is scheduled for transmission during the **ATIM Window**, then it will be **deferred** until the end of the Window. **Contention** may cause additional delays.

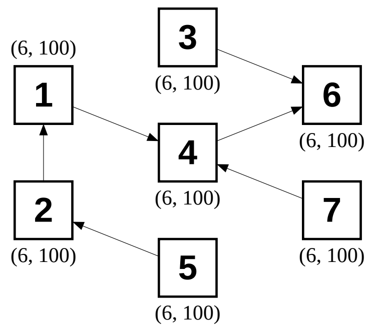
The **edge set** of the spanning tree consists of edges from **demanded routes** as well as **inferred edges** discovered by promiscuous listening on the wireless medium. The loss of an inferred edge can never (by itself) disconnect a tree; such an edge can always be replaced by one or more links from the sources' **demand sets**. The loss of an edge from any source's demand set is more serious. Figure 4.25 shows a simple example: traffic source 1 demands only the route $(1, 2)$. If the link $(1, 2)$ breaks, the tree becomes **disconnected**; there is no way to reconnect it using only tree nodes. Further, because the broken edge comes from a route in source 1's demand set, even if we *could* reconnect the tree, the break affects the outcome of the negotiation. Source 1 can no longer bid on the route $(1, 2)$.

In XPM, a broken tree edge causes a restart of the spanning tree formation procedure. If the broken edge was part of a source's demand set, then that source will discover the break when it sends out its new **Notify** messages along each demanded route. The source can then modify its demand, and not submit bids for the affected route or routes.

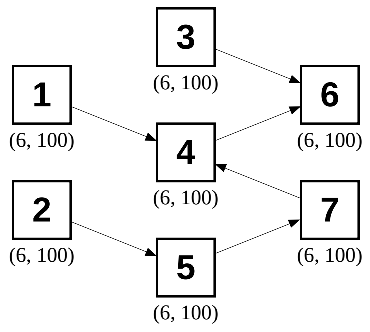
In the revised protocol, we can **proactively** inform affected sources that their demand set may need to change. A node which has an incident broken tree edge, and which is the *closer* to an affected source of the two nodes formerly joined by the edge, can send a **DSR Route Error** back to the source.[27] This gives sources the chance to withhold **irrelevant bids** from the mechanism.

[27]Nodes can remember the routes on which they received **Notify** messages. If a tree edge from one of those routes is observed to break, the proactive Route Error is sent to the sources which sent Notify messages along those routes.

We have several options for dealing with the tree disconnection problem of Figure 4.25. We can allow the tree formation process to **time out**, and let one or more traffic sources **restart** the procedure. We can attempt to **reconnect** the disconnected subtrees by recruiting "helper" nodes from **outside** the tree. We could allow bidding only from nodes still connected to the **root**; disconnected nodes would be **blocked** until a subsequent negotiation. The choice of how to handle this specific situation is an open problem, and should be studied as part of future research.



(a) Rerooting the subtree.



(b) Reparenting the nodes.

Figure 4.26: Two spanning tree repair techniques.

Returning to an earlier example, suppose it *is* possible to reconnect a subtree. We can use concepts from **Adaptive Demand-Driven Multicast Routing**, or ADMR (Jetcheva, 2004), to perform the repair. Upon discovering that it is the root of a disconnected subtree, node 2 in Figure 4.26(a) can attempt to **reroot** the entire subtree by finding a new parent (*e.g.*, node 1) in its former tree. This can be accomplished by soliciting neighbors across **Cousin** edges for a **conquering Update**. If no such edges exist, a **broadcast solicitation** identifying the tree 2 is trying to rejoin may be issued. During this rerooting process, 2 could use the equivalent of ADMR **repair notification** messages to prevent its descendants from attempting a repair themselves.

If it is not possible to reconnect the subtree at the subtree root, it may still be possible to reconnect using *other* subtree nodes. Figure 4.26(b) shows that the edge $(5,7)$ could be used to make 7 the new parent of 5. In turn, 5 becomes the new parent of 2. This **reparenting** process uses the existing XPM **subtree conquering** procedure.

We observe that the loss of a tree edge having type **Child** does not prevent a node from participating in the bid submission **wave algorithm**. The precondition for entering **Negotiation** is that bids have been received from all Child edges. Therefore, after removing a broken Child edge, a node must only wait for the *remaining* children to submit bids before passing a combined **Submit** towards the root. As in XPM, if the tree structure later changes such that a node gains or loses descendants, the node may always **Revoke** its earlier submission and **Submit** revised bids.

Finally, with the use of **interim mechanism solutions**, all tree nodes must *know* that they are participating in a negotiation. Section 4.6.1 described how **keep-alive messages** could be propagated throughout a tree to inform tree nodes of their participation. If a broken tree edge is encountered at this stage, nodes in the disconnected subtree will not be informed! Further, since they are not informed, they cannot initiate the **tree repair** techniques described above. One possible (but **unreliable**) solution might be to send a **propagating broadcast** throughout the network, inviting members of the named tree to verify the integrity of their incident tree edges. Broken links could then be repaired using the aforementioned methods. Alternative handling of this case is a topic for future research.

4.6.3 High-Mobility Adaptation

Even with the **interim mechanism solution** (§4.6.1) and **tree repair** (§4.6.2) techniques just presented, it is possible for XPM to be overwhelmed by high **node mobility**. Although specific thresholds are not known, if links break faster than the spanning tree can react, then the nodes will never be able to successfully execute the mechanism. As link breakage occurs more frequently, more **Structure messages** are transmitted to try and reestablish the tree, thus increasing **congestion**. We would like to detect a point of **diminishing returns** associated with the use of negotiated routes. Beyond this point, the nodes could agree to use **unnegotiated** routes until mobility conditions become more favorable.

When nodes move slowly, it is more likely that some will become "stuck" in **distinguished positions** within the network. These nodes will bear a disproportionate amount of the network relay load. XPM helps these nodes by allowing them to express their **preferences** for providing service. When the nodes move quickly, the likelihood of a node becoming "stuck" in this manner decreases. As such, it is reasonable to consider methods other than negotiation for choosing routes.

Nodes might discover that tree edges are failing “too often” by several means. They can measure the frequency with which their own incident edges fail. They can monitor the overall density of Structure messages, both **received** and **overheard**. They could keep statistics on Route Error messages or **fast wakeup** frequency. These and other data could be used to infer whether the local network has become too **unstable** for XPM to be helpful.

A source which concludes that mobility conditions are prohibitive might be permitted to switch to **Local Power Management** (§4.3). Even if the source is not in a position to observe the problem, a relay or sink on one of the source’s routes could send back a report. The source could inform its affiliated relays and sinks that it believes switching to LPM is justified. A technique such as XPM **Notify propagation** could be used for this purpose. If enough relays and sinks concur with the source’s assessment, the source would subsequently be permitted to use its own discretion in choosing routes. This would be permitted for some fixed interval, in the spirit of the XPM **negotiation interval**. After the interval expires, the source would be required to negotiate properly, or again appeal to the relays and sinks for continued use of LPM.

It may be possible for the relays and sinks to receive some compensation for their costs under this approach. These nodes could simply report prices to traffic sources. The sources would then be obliged to pay in the amount of the reported price to each node whose service it employs. Since there is no way to make such a method **strategy-proof**, we hesitate to proceed further in this exposition. We conclude by saying that **failover techniques** for XPM in high-mobility environments are a topic worthy of additional study.

4.7 Summary

This Chapter presented a practical framework for **power management** in a mobile *ad hoc* network. We presented several techniques for improving the performance of protocols such as **Dynamic Source Routing** in power-managing environments. Specifically, we showed how **power management suspension** and **fast wakeups** could reduce the latency of application message delivery and Route Discovery. We then described an architectural framework for implementing these concepts in a protocol stack containing DSR and 802.11.

We then described **Local Power Management**, a design which provides DSR nodes with the energy savings of 802.11 power management, but without the high latency. LPM is not a **negotiated** design, so nodes cannot express **preferences** over network configurations. This means that LPM does not offer any **energy-shaping** features, but it is still useful as a baseline for improved DSR performance in power-managing environments.

Building on LPM, we introduced **Exchange Power Management**, a mechanism-based design for negotiation among power-managing nodes. Using DSR as the actual **routing protocol**, XPM permits nodes to express their **preferences** using bids and asks in a **combinatorial exchange**. Once routes have been discovered, XPM forms a **spanning tree** over the sources, relays, and sinks on **overlapping** routes. This tree is used to collect bids in a **wave algorithm**. The root of the tree solves the **winner determination problem** using heuristic search, and computes **Vickrey payments** from the agents. The mechanism results are passed down along the tree, and sources proceed to use those routes which correspond to their winning bids.

We provided some simple examples of agent **valuation functions** based on the concept of **balance**, which measures an agent's **contribution** to the network. One example emphasized **delivery ratio**, a common metric in routing protocol analysis. This design necessarily sacrificed the **incentive compatibility** of the mechanism, but still provided a way for relays and sinks to express their preferences for providing service. The other example assumed agents which were sensitive to their payment amounts, which led to the idea of an **unaffordable route**.

Finally, we described some interesting directions for future improvements to various XPM procedures and algorithms. A method for **renegotiating** a mechanism which has been disrupted was proposed. Second, a technique for **repairing** spanning trees whose edges break, rather than reforming the tree "from scratch," was explained. Finally, concepts for responding to **high node mobility** were mentioned, including failing over to LPM in situations where the XPM procedure cannot react quickly enough.

Exchange Power Management is the first design to combine 802.11 power management, Dynamic Source Routing, combinatorial exchanges, and VCG mechanisms. It is the first application of mechanism design to the mobile *ad hoc* network environment based on practical protocols. It is the first game-theoretic power management design to appreciate the energy consumption behavior of *actual* wireless network interfaces. By understanding the rôle of the **idle state** in total energy consumption, we have identified the requirement that a **strategy-proof** mechanism for this environment must solve a **single exchange** for overlapping routes. XPM therefore implements the first strategy-proof mechanism for power-managing mobile *ad hoc* networks.

Chapter 5

Simulator and Workloads

To measure the performance of the **multihop power management architecture**, we have implemented the designs of Chapter 4 in simulation. This Chapter describes our experimental environment, detailing the **simulator** and **workloads** we have used in this research. Section 5.1 presents our enhancements to the ***ns* network simulator**, a popular tool used by many ad hoc networking researchers. Section 5.2 explains the **communications** and **movement patterns** to which we have applied our design.

5.1 Simulator

Since the beginning of this research, no vendor of IEEE 802.11 wireless network interfaces has offered an implementation of **IBSS power management** (§2.1.3). An informal survey of vendors on this matter revealed that IBSS power management performance did not justify the **complexity** of implementation, or that there was insufficient **market demand** for the feature. We chose **simulation** as the method of evaluation in lieu of actual system measurement. This is a common approach used by mobile *ad hoc* networking researchers, since it permits rapid experimentation with larger networks than would otherwise be practical.

We can offer some insight into these beliefs about complexity and demand. First, most commercial and consumer applications of 802.11 involve **infrastructure** networks, *not* the *ad hoc* networks supported by IBSS mode. Most *ad hoc* applications today are found in military, research, or demonstration environments.[1] Only a small population of users would benefit from IBSS power management were it to be released. As we described in Section 2.1.3, IBSS power management adds to the **latency** of communication. This thesis describes techniques to improve latency in **multihop** environments, but these techniques are not part of the 802.11 specification. Finally, the **complexity** of supporting IBSS power management at the firmware and device driver levels is non-negligible. As an informal measure, when we added IBSS power management to the existing 802.11 implementation in the *ns* simulator, the 802.11 code more than **tripled** in size.

The experimental environment for this research is an extended version of the ***ns* network simulator** (Fall and Varadhan, 1998), originally developed in a collaboration between UC Berkeley, LBL, USC/ISI, and Xerox PARC. *ns* is an object-oriented

[1]In fact, early versions of the **WaveLAN** 802.11 product only supported *ad hoc* operation through a non-compliant **demonstration** mode, suggesting how the designers imagined it would be used.

discrete event simulator written in C++ with an OTcl command interface. Several network protocols at the application, transport, network, and link layers have been implemented in *ns*. The simulator supports both wide-area and local-area networks.

5.1.1 Monarch Extensions

The Monarch Project at Rice University, formerly at Carnegie Mellon, has published an extended version of *ns* 2.1b1, which introduced support for mobile, multihop wireless networks (Monarch, 1999). The present research is based on release 1.1.2 of the Monarch extensions. The major enhancements provided by the Monarch version are a **radio propagation** model underlying the wireless MAC, and the implementation of **multihop routing protocols** for the mobile *ad hoc* environment. We have ported the Monarch code base to Mac OS X and newer distributions of Linux with the Intel C++ Compiler. Patches for these updates have been published on the Internet and are in use by several researchers around the world.

5.1.1.1 Radio Propagation

Support for position and mobility in a wireless network simulation is important because wireless network interfaces have limited range. The Monarch extensions simulate this with a simple radio propagation model. The model assumes an antenna **reference distance** of 100 meters; within this distance, received signal power is given by the **Friis free space equation**,

$$P_{\mathrm{r}}(d) = \frac{P_{\mathrm{t}} G_{\mathrm{t}} G_{\mathrm{r}} \lambda^2}{(4\pi)^2 d^2 L}$$

A reference for the Friis equation is Section 3.2 of (Rappaport, 1996). Given a transmitter power, the gains of the transmitting and receiving antennae, and miscellaneous system losses, the Friis equation says that received power falls off with the inverse square of distance. For transmitter-receiver distances beyond the reference distance, a **two-ray ground reflection** model is used:

$$P_{\mathrm{r}}(d) = P_{\mathrm{t}} G_{\mathrm{t}} G_{\mathrm{r}} \frac{h_{\mathrm{t}}^2 h_{\mathrm{r}}^2}{d^4}$$

Given a transmitter power, transmitter and receiver gains, and the heights of the transmitting and receiving antennae above a flat ground plane, the two-ray model says that received power falls off with the inverse of distance raised to the fourth power. Section 3.6 of (Rappaport, 1996) explains this model in greater detail.

Under this model, the maximum **coverage radius** of a simulated radio is 250 meters. The model does not account for **obstacles** or other sources of **signal attenuation**. It also does not model **multipath** effects, such as **fast fading** (also known as **Rayleigh fading**).

5.1.1.2 Multihop Routing

The motivation for the Monarch *ns* extensions was to provide a simulation environment in which to evaluate multihop *ad hoc* routing protocols. Several such protocols have been implemented in *ns* , including **Destination-Sequenced Distance Vector** (Perkins and Bhagwat, 1994), the **Temporally-Ordered Routing Algorithm** (Park and Corson, 1997), **Ad hoc On-Demand Distance Vector routing** (Perkins and Royer, 1999), and **Dynamic Source Routing** (Johnson et al., 2003). The present research is based on DSR, which has been documented extensively in this simulation environment (Broch et al., 1998; Maltz et al., 1999; Maltz et al., 2000).

The version of *ns* used in our experiments implements a DSR **path cache**, called **MobiCache**, which stores entire routes. Newer **link caches** have been developed (Hu and Johnson, 2000) since this version of *ns* was released. As Section 4.4.2.1 explained, patch caches simplify the process of determining a source's **demand set**. We have not experimented with trying to compose demand sets from individual links, although this might be an interesting subject for future work.

5.1.2 802.11 Implementation Improvements

The Monarch *ns* implementation of the **IEEE 802.11** standard (IEEE, 1997) is largely complete with respect to **medium access**. That is, the **contention-based** statistical multiplexing algorithms are in place, as are the procedures to mitigate **hidden terminal** effects (§2.1) and **retransmit** lost frames. In addition to these, however, 802.11 defines a number of **management features** which are necessary for stations to join networks, implement **power management**, and perform other activities. These management features add substantially to the complexity of an 802.11 implementation. In our work, we have more than *tripled* the lines of code associated with the 802.11 protocol.

5.1.2.1 Timer Synchronization

Every station in an 802.11 network implements a **microsecond-resolution timer** which is used to coordinate frequency hopping, contention-free medium access, and power management. **Frequency Hopping Spread Spectrum** has become a less popular 802.11 PHY since the advent of the **Direct Sequence Spread Spectrum**, or DSSS, 802.11b standard (IEEE, 1999). We therefore ignore the use of station timers for frequency hopping. Similarly, contention-free medium access is only available in infrastructure networks; we also ignore the use of timers for this purpose. The remaining item, power management, relies on station timers to coordinate periodic power cycling of the transceiver electronics. The details of this procedure are described in Section 5.1.2.3.

Section 11.1 of the 802.11 specification (IEEE, 1997) describes the timer synchronization method for infrastructure and IBSS environments. In an infrastructure network, timer synchronization is simple: the access point periodically broadcasts a beacon, and the beacon contains a **timestamp**. Stations always adopt the timestamp in beacon frames received from an access point.

In an IBSS, there is no distinguished station which provides a master timer reference. Stations therefore implement a distributed **timing synchronization function** (§2.1.1.2), or TSF, which is intended to maintain timer synchronization in a BSS "to within 4μs plus the maximum propagation delay of the PHY for PHYs of 1 Mb/s, or greater." In the distributed TSF, *all* stations periodically attempt to transmit a beacon frame. When the time for beacon transmission — known as a **target beacon transmission time**, or TBTT — arrives, each station contends to send the beacon. The algorithm, described in Section 11.1.2.2 of the 802.11 specification, requires stations to do the following at each TBTT:

1. Suspend the decrementing of the backoff timer for any pending non-beacon or non-ATIM transmission.

2. Calculate a random delay uniformly distributed in the range [0ms, 1.24ms] (for DSSS PHYs).

3. Wait for the period of the random delay.

4. If a beacon arrives before the random delay has expired, cancel both the remaining delay and the pending beacon transmission. Resume decrementing the ATIM timer.

5. If the random delay has expired and no beacon has arrived during the delay period, send a beacon.

We implement an **alternate backoff timer** used for beacon delays and ATIM backoff. At a TBTT, we pause the normal backoff timer, and switch to the alternate. After beacon transmission or reception (in a non-power-managing network), or at the end of the ATIM Window (in a power-managing network), we reset the alternate timer and resume using the normal one.

The separation between TBTTs is fixed throughout the life of the network, and is known as the **beacon interval** (§2.1.1.1). This interval is a configurable parameter of the network. Previous research involving simulated IBSS environments has used values of 100 milliseconds, (Heindl and German, 2001; Huang and Lai, 2002), 200 milliseconds (Chen et al., 2001), and 400 milliseconds (Zheng and Kravets, 2003). Our experiments use a beacon interval of 200 milliseconds.

Beacons include a timestamp which is equal to the value of the sending station's timer at the time the beacon is placed on the wireless medium. The timestamp is an encoding of the station timer value in **time units**. A TU is "a measurement of time equal to 1024μs" (IEEE, 1997). When a station receives a beacon, it compares the received timestamp with the value of its own timer. If the received timestamp is greater than the local timer value, then the local timer is set to the value of the received timestamp. Beacons also include several network **parameters**, including the size of the **beacon interval** and **ATIM Window**, both expressed in TU.

The introduction of beacons has minimal impact on simulated performance, relative to the original Monarch *ns*. At worst, they periodically increase the level of **congestion** in the network by a small amount. Our experiments have not shown this change to noticeably affect DSR.

5.1.2.2 Timer Implementation

Time is continuous in *ns*; most events in the simulator are dispatched by real-valued timers. Call the time scale on which these timers operate **absolute time**. Physical movement and signal propagation are two aspects of the simulator which occur in absolute time. A station's timer may be faster or slower than absolute time. For an interval $\Delta\tau$ measured using a station's timer, the corresponding absolute interval Δt is given by $\Delta t = \frac{\Delta\tau}{1+d}$. The timer **drift** d is, for each station, chosen from a uniform random distribution on $[-0.01\%, 0.01\%]$. This interval corresponds to the maximum drift permitted in Section 11.1.2.4 of the 802.11 specification. In our implementation, all timers based on the timer, such as the TBTT timer and ATIM Window length timer, are subject to this drift.

When sending beacons or comparing the timestamp of a received beacon to a local timer, it is necessary to know the station timer value associated with the present moment in absolute time. This requires two pieces of state: the absolute time of the last timer update, t_{up}, and the value of the station timer at that update, τ_{up}. Given the absolute time t_{now}, which is always available from the simulator, the associated station timer value is:

$$\tau_{\text{now}} = \left[\left(t_{\text{now}} - t_{\text{up}}\right) \times (1+d)\right] + \tau_{\text{up}}$$

5.1.2.3 IBSS Power Management

When a station uses **power management** (§2.1.3), it periodically wakes and sends **ATIM management frames** to those neighbors for which it wants to send messages. Stations that will send or receive data during the current **beacon interval** then stay awake for the *entire* interval. The implementation issues surrounding power management involve the additional **timers** for the ATIM facility, **buffering** for data frames, and **power state transitions** in and out of the low-power **doze state**.

The ATIM Window starts at a TBTT and lasts for a fixed duration as measured by the station timer, described in Section 5.1.2.2. In previous simulation work, this duration has been set at 40 milliseconds (Chen et al., 2001; Zheng and Kravets, 2003), and a variable (but non-compliant) 2—50 milliseconds (Jung and Vaidya, 2002). We use a 40-millisecond ATIM Window in our experiments.

Once the ATIM Window begins, only **beacon**, **ATIM**, and **acknowledgment** frames (sent in response to a directed ATIM) may be transmitted until the end of the Window. No ATIM frames are sent until a station has either sent or received a beacon following a TBTT, as described in Section 5.1.2.1. If a **timer update** causes a station to advance its timer past a TBTT — in other words, if it "jumps over" the TBTT — then the ATIM Window starts **late**. In such cases, the ATIM Window length relative to the TBTT does *not* change. Instead, the timer which defines the end of the Window is abbreviated so that the station ends its Window in (loose) sync with its neighbors.

When the MAC layer receives a **datagram** from a higher protocol layer for transmission, it may need to **buffer** the datagram due to power management. For example, the transmitting station might be asleep, or the destination (including the **broadcast** destination) might be asleep. In general, such datagrams will be buffered until the next **beacon interval**, when the destination can be **announced**, although problems in the announcement process (*e.g.*, **congestion**) may require longer durations.

Our implementation assumes that the MAC layer can buffer an arbitrary number of datagrams to an arbitrary number of destinations. In each ATIM Window, as many destinations are announced as possible. To extract the best performance from this implementation, we altered the **queue** between the **link** and MAC layers. The Monarch *ns* design releases one datagram to the MAC layer at a time: once a message is successfully transmitted, or a transmission failure occurs, the next message is handed down. When messages to a variety of destinations are available, this can result in very poor **latency**; in the worst case, only one datagram can be transmitted per beacon interval. Our implementation passes *all* datagrams to the MAC layer as soon as they are ready for transmission, thus making better use of the traffic announcement process.

During the ATIM Window, our implementation announces the **broadcast** address (if it requires announcement) before any other addresses. Broadcast ATIM frames are not acknowledged, but **directed** ATIM frames are. If no **ACK** for a directed ATIM is received after several **retries**, then the destination for the ATIM is marked as **unreachable**. Following the ATIM Window, buffered datagrams are processed in queue order. Messages to unreachable destinations which reach the head of the queue are *not* transmitted; instead, we trigger a higher-layer callback to indicate that the transmission attempt has failed.

There are several cases in which a station must stay **awake** following the ATIM Window, at least one of which is hard to justify. If a station sends an ATIM frame, or receives one destined for itself or the broadcast address, it must stay awake. Correspondingly, if a station hears an ATIM from its neighbor, or sends a directed ATIM which is **acknowledged**, then it knows the neighbor will be awake. In addition, Section 11.2.2.3 of the 802.11 specification requires that if the station transmits a **beacon**, "it shall remain in the Awake state until the end of the next ATIM Window." We find this clause very difficult to motivate; it means that a station which frequently sends beacons, but *never* processes actual datagrams, can still be awake in many beacon intervals. Our implementation honors this clause, and in Chapter 6 we show the energy costs of this requirement.

Section 11.2.2.3 also requires a dozing station to "enter the Awake state prior to each TBTT." In our implementation, a station wakes 3 milliseconds prior to the TBTT, as measured with its own **station timer**. This prevents the station from missing beacon or ATIM frames transmitted near the TBTT, even if its own timer is slightly out of sync with those of its neighbors.

When a station is using the **multihop power management architecture**, described in Chapter 4, we make several changes to the behavior of the 802.11 implementation. In general, these are compatible with the existing specification. The first is for the **suspending neighbors list** (§4.1.1), which allows stations to skip the buffering and ATIM process for neighbors which are known to be awake. The second involves the propagation of broadcast ATIM frames using the **fast wakeup** technique (§4.1.2). The third concerns the rule about remaining awake after beacon frame transmission, which we ignore. (A station which has another reason to stay awake, such as having received an ATIM, can still do so.) These changes result in improved latency and energy performance without significantly departing from the 802.11 specification.

5.1.2.4 802.11b High-Rate PHY

Monarch *ns* implements the original 802.11 specification, which supported data rates of 1 megabit per second and 2 megabits per second. The much more common version in use today is **802.11b** (IEEE, 1999), which is backwards compatible and adds support for higher data rates, 5.5 and 11 megabits per second. Owing to the ubiquity of the newer standard, we modified *ns* to support the higher rates.

Multirate support in 802.11 is implemented using the **PHY Layer Convergence Protocol**, or PLCP, described in Section 15.2 of the 802.11 specification for **DSSS** PHYs. Each 802.11 frame is prepended with a PLCP header which describes the data to follow. Included in the header is information about the **modulation scheme**, **data rate**, and **length** of the frame, as well as a **CRC-16 frame check sequence**. Although the PLCP preamble and header were accounted for in the frame sizes used by Monarch *ns*, the **timing** calculations made for frames were incorrect.

To permit interoperability between stations that support different data rates, the PLCP preamble and header are *always* sent at 1 megabit per second as required by Section 15.2.3 of the specification. The existing code assumed that all frames were sent in their entirety at 2 megabits per second, which would **understate** the time required to send a frame. Our implementation corrects this.

An 802.11 network has **basic rates**, which are the data rates usable by *all* stations, and **supported rates**, which a station is not required to support before joining the network. Both the basic and supported rate sets are encoded in the **beacon frame**. In our implementation, the basic rate set includes the 1 megabit per second and 2 megabit per second rates, while the supported rate set contains the 11 megabit per second rate.

Section 9.6 of the 802.11 specification requires that certain frames always be sent at one of the basic rates. These include all **control** frames and any frame sent to the **broadcast address**. The latter encompasses all beacons, broadcast ATIM frames, and broadcast data frames. In our implementation, these frames are sent at 2 megabits per second. **Directed** data frames may be sent at any rate known to be supported by the receiver; in our implementation, such frames are sent at

11 megabits per second.

5.1.3 Power Model

To collect information about energy consumption in the simulated radio, we added hooks to the 802.11 implementation which record **power state transitions**. For example, frame reception is implemented by starting a timer as the first bit of a frame is received. The timer is set to expire when the frame reception should complete. Our **power model** records the transition to the receive state at the time the timer is set, and records a return to the idle state when the timer expires.

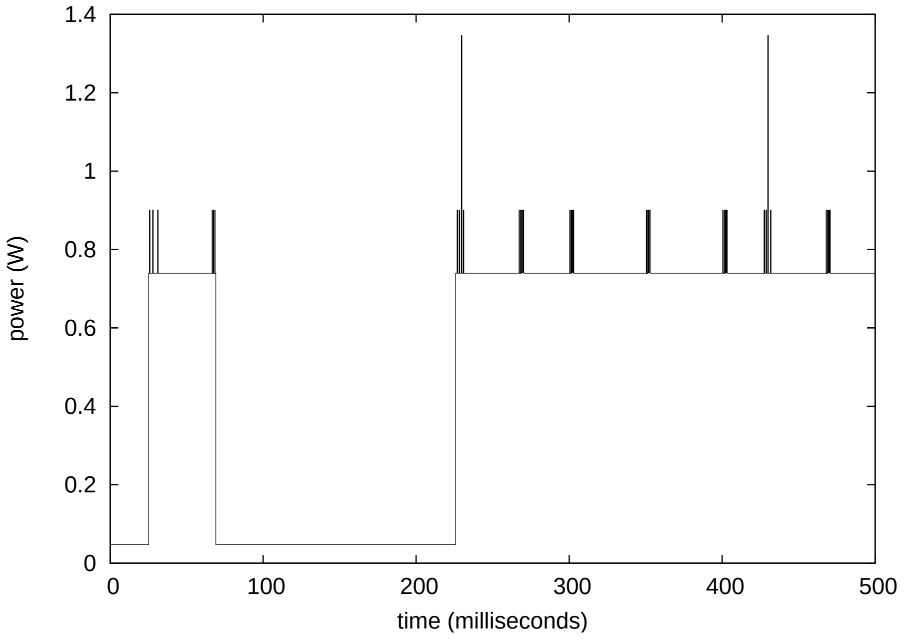


Figure 5.1: Power trace using Feeney & Nilsson rates.

The power model itself does not know about specific **power rates**; it only records the amounts of time spent in each state. Power rates are applied in postprocessing. Figure 5.1 shows an example trace produced by the model, using rates from (Feeney and Nilsson, 2001).

In *ns*, frames which arrive with insufficient signal strength to be decoded (according to the propagation model) still trigger the receive timer, but are then discarded. Our implementation records these periods spent in the receive state. This means that distant transmitters can affect a station's energy consumption as the faint frames are "heard."

This is not the case when the interface is in the **doze** state, however. All incoming frames are immediately **discarded** when in doze, and no power state transitions are recorded until the station emerges from doze. Our implementation assumes an instantaneous transition between idle and doze. Published data for 802.11 interfaces indicates transition times of 250 microseconds (Kamerman and Monteban, 1997), and recent simulator work has assumed 800 microseconds (Jung and Vaidya, 2002). These transitions are small relative to the length of a beacon interval, and will not significantly affect communication or energy performance. For completeness, a future version of our power model will incorporate less ideal transition behavior.

5.1.4 On-demand Power Management Implementation

We have implemented **On-demand Power Management** (§3.1.3.5) in our version of *ns* to provide a direct comparison with previous work. On-demand Power Management is the most recent work to use network-layer information to control MAC power state transitions. It is similar in some respects to **Local Power Management**, described in Section 4.3. For example, power state transitions are controlled by uncoordinated timers. It differs in several key respects:

- **No route discovery improvement.** DSR Route Discovery incurs the latency associated with 802.11 IBSS power management, as broadcast Route Requests are delayed at each hop. Route Replies may return more quickly than under basic IBSS power management, as some nodes may **suspend** power management. By comparison, LPM and XPM use **fast wakeup** (§4.1.2) to reduce the latency of Route Discovery.

- **Suspending neighbor list based on MAC information.** Every received or overheard MAC frame, including control and management frames, is examined for the status of the **power management** bit in its **frame control** field. The power management suspension status of neighboring stations (§4.1.1) is updated accordingly, as the bit indicates whether the sending station is currently using power management. By comparison, LPM only uses messages received at the DSR layer to update the **suspending neighbors list**. This reduces the size of the list, since only neighbors with which a node is communicating will appear in the list.

- **2-stage transmit failure handling.** When the MAC layer fails to send a message to a neighbor, the response to that failure depends on the status of the suspending neighbor list. If the neighbor was believed to be suspending power management, it is now believed to be using power management. If it was already using power management, it is now considered unreachable. By comparison, transmission failures under LPM are *always* treated as link failures (as in normal DSR).

- **No change to beacon/doze interaction.** Section 5.1.2.3 described a requirement of the 802.11 specification that stations transmitting a beacon frame during a beacon interval must remain awake for the entire interval. On-demand power management honors this requirement. By comparison, LPM and XPM ignore the requirement and achieve lower energy variability in idle networks.

To implement On-demand Power Management in our simulator, we corresponded with the primary author of (Zheng and Kravets, 2003) to clarify some design points described in her paper. We also reviewed her implementation of the design, originally developed for a different version of *ns*. We then implemented On-demand Power Management as an instance of our own **multihop power management architecture**, since most of the data structures and cross-layer actions are similar.

Several concepts are described in the paper which the author did not implement in her code. These include the use of periodic “HELLO” messages, and neighbor state inference based on node degree and beacon frames. Our implementation follows the code in these instances.

Timer	**Duration**
Route Request keepalive	0s
Route Reply keepalive	5s
CBR message keepalive (source)	2s
CBR message keepalive (relay)	2s
CBR message keepalive (sink)	2s
Refresh interval	5s

Table 5.1: On-demand Power Management timer durations.

We adopted the timer values described in the paper, which are reproduced in Table 5.1. The **refresh interval** is not implemented as a timer, but rather is used when a transmission failure occurs. When a stations sends an RTS frame but does not receive a CTS from its neighbor, it checks the suspending neighbor list. If the neighbor hasn't been heard from in longer than the refresh interval, then one of two things happens. If the neighbor was previously suspending power management, it is considered to have resumed power management. Otherwise, its list entry is removed.

The code uses a different value for the refresh interval: 900 seconds. This value means "forever" in many simulators based on the Monarch *ns* distribution. We used the value from the paper in our implementation.

5.2 Workloads

In order to evaluate the performance of the simulated power management schemes, we need synthetic **workloads** which span a range of **communications** and **mobility** types. Realistic workload generation could warrant a separate thesis; we have tried to select plausible workloads similar to those already published for this type of research. Our communications workload exercises the network under varying amounts of **concurrent** traffic. The mobility models, some of which are new, attempt to capture the inherent **structure** of human movement in organized environments.

A major difference between our workloads and those of previous work is the **duration** of each simulation. Our experiments run for 10,000 seconds (2.8 hours) of simulated time, an order of magnitude longer than is typical (Maltz et al., 1999; Chen et al., 2001; Xu et al., 2001; Zheng and Kravets, 2003). Our goal was to mitigate **transient** energy effects, and focus on the **steady state** of the network.

Our experiments each involve 50 mobile nodes, a number used in other research (Broch et al., 1998; Zheng and Kravets, 2003). There are no special **rôles**; any node can be a source, relay or sink. Nodes are confined to an area defined by the mobility model for the duration of the simulation. Each node has an **infinite supply** of energy, and can participate in communications at any time during the simulation.

5.2.1 Traffic Loads

The most popular **traffic load** applied to simulated ad hoc networks is the **Constant Bit Rate** application (Broch et al., 1998; Maltz et al., 1999; Chen et al., 2001; Xu et al., 2001; Zheng and Kravets, 2003). A CBR traffic source generates fixed-size packets at semi-regular intervals. A real-world application with these characteristics might be **digital telephony**[2] or some other form of **streaming media**.

The CBR application is popular for a number of reasons, not the least of which is the fact that it ships with the *ns* simulator (Monarch, 1999). Also, it is *simple*, with no **acknowledgment** or **retry** facility. Unlike a **TCP** source, CBR does not **adapt** to congestion, nor does it implement **flow control**. This makes the CBR message rate more predictable, especially in the face of **delays** caused by Route Discovery or **route breakage** resulting from mobility.

The parameters we use for an individual CBR traffic flow are identical to those found elsewhere (Maltz et al., 1999). The CBR application generates 512-octet messages at an average rate of 4Hz. The inter-message spacing is randomly drawn from a uniform distribution on 250ms $\pm$ 125ms.

The CBR traffic load generator that ships with *ns* randomly starts all flows in the first 180 seconds of simulation time (Monarch, 1999). This is unsuitable for our purposes, as we want to study the long-term behavior of the network. Instead, we space traffic flow starts **throughout** the life of the simulation. Each network node is initially assigned some number of messages to send during the simulation. Then, this message count is divided among several flows; each flow contains a random number of messages drawn from a uniform distribution on [100, 500] messages. At 4Hz, this corresponds to flows lasting 25 seconds to just over two minutes.

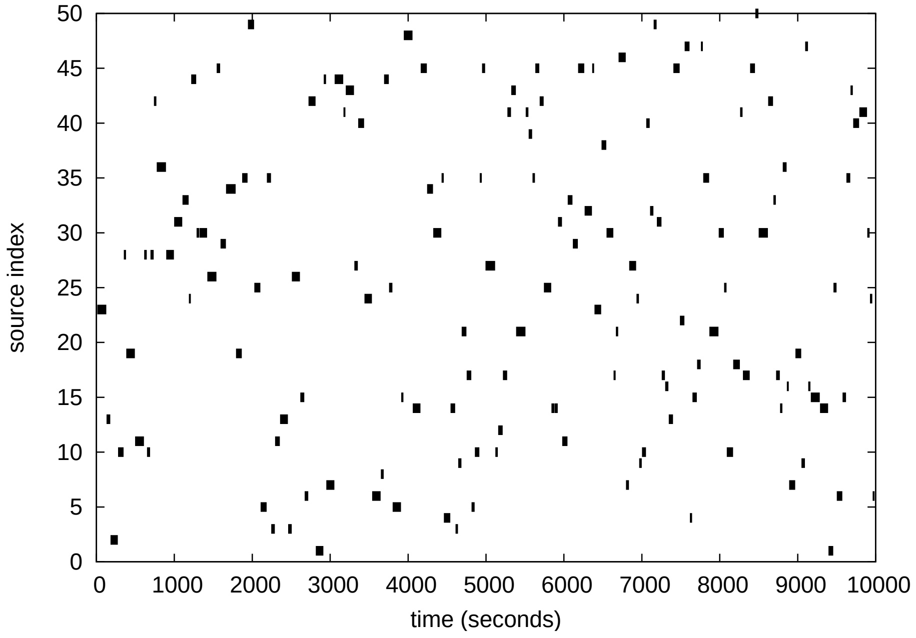
[2]Since the CBR data flow is **unidirectional**, this is an imperfect example.

The source for these values is the author's **mobile phone bill**; the overwhelming majority of calls were observed to fit into this interval.

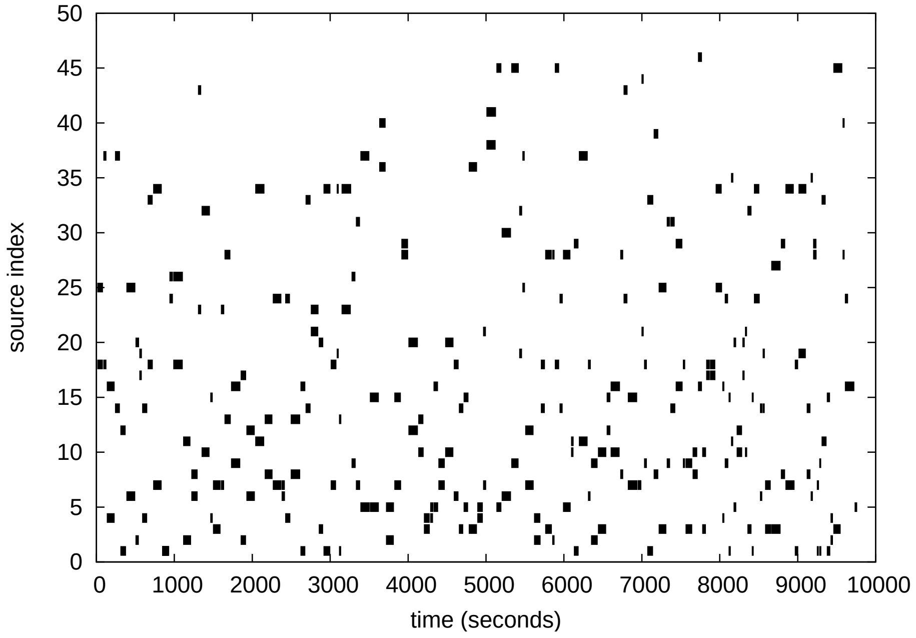
When scheduling traffic, we control the number of **concurrent** flows that are active at any given time. This is a useful way to set a bound on how "busy" the network can be. By contrast, the *ns*-supplied CBR generator has no such control. In our experiments, we evaluate three concurrency levels: 0-flow, 1-flow, and 2-flow. The 0-flow traffic load is actually an **idle** or **unladen** network; no traffic sources are ever active. This is an important case to consider when measuring the **energy performance** of a network. It establishes baseline energy consumption independent of traffic activity, and can reveal **wasted** energy in a power management scheme.

In the 1-flow traffic load, *at most one* traffic flow is active at any given time. Each node is assigned a number of messages to source over the entire simulation, randomly drawn from [100, 1,400). This interval is sized to allow all messages to be sent in the time allotted for the simulation. Figure 5.2(a) shows an example of the times during which each node is active as a traffic source. The vertical axis corresponds to a node's **index** or address; the graph comprises 50 stacked timelines showing periods in which each node is a traffic source. In this example, 471 seconds of the total 10,000 seconds (4.7%) are spent with no active traffic flow; otherwise a traffic flow is in progress.

Similarly, the 2-flow traffic load has at most two flows active at any time. Our traffic generator tries to keep two flows active for most of the simulation. Because



(a) 1-flow traffic load.



(b) 2-flow traffic load.

Figure 5.2: Example traffic loads.

more flows can be used, we assign larger numbers of messages to the nodes drawn from [100, 2,500). Figure 5.2(b) shows the periods during which each node is a source; 647 seconds (6.5%) are spent with no active flows.

Our traffic loads exhibit less concurrency than some earlier work (Maltz et al., 1999). To evaluate the effectiveness of a **power management** scheme, we argue that performance under **low** and **moderate** load is more important than that under **heavy** load. For example, suppose that all nodes are active as sources or sinks. There is *nothing* a power management scheme can do to improve energy consumption! Every node will be awake, and experience the worst-case energy consumption. Further, it is unrealistic to expect in any network of appreciable size that *all* (or even *most*) nodes have messages to send at any given time.

There are at least two interesting questions to ask about a power management design. Does it behave **reasonably** under low load? Does energy consumption increase in a **controlled fashion** as load increases? Reasonability refers to the **correlation** between communication activity and energy consumption. We expect a node that performs no communication tasks to experience close to the **minimum possible** energy consumption. A node that processes more messages should consume more energy than a node that processes fewer messages. Consumption is **controlled** if it gradually increases with load. It is undesirable for a node to quickly approach the worst case energy consumption under moderate communication activity. Chapter 6 will show that the 802.11 **IBSS power management** design is

unreasonable, and that only **negotiated** designs such as XPM are controlled.

5.2.2 Mobility Models

The second facet of our experimental workloads is the model for **node mobility**. During the simulation, nodes are constantly moving about in a plane. We want to know how different types of mobility affect the **communications** and **energy** performance of the network.

In keeping with the original motivation for this research — energy consumption in **wearable** and **handheld** devices — we examine node movement at a **walking pace**, up to one meter per second. Our models have **node densities** of 50—100 nodes per square kilometer, which is comparable to previous work (Maltz et al., 1999). The models vary by the degree of **structure** they exhibit. The structure of a mobility model encompasses the **constraints** on node movement; when nodes are free to move at **random**, we say there is no structure to their mobility. If nodes are forced to stay in a particular region or move along defined paths, then their mobility is structured.

5.2.2.1 Unstructured Mobility Model

The first model we present is the popular **random waypoint** movement pattern used by many researchers (Broch et al., 1998; Maltz et al., 1999; Chen et al., 2001; Xu et al., 2001; Zheng and Kravets, 2003). Nodes are placed randomly in a plane, which for our experiments has size 800m $\times$ 600m. They pick a random **destination** in the plane, and move towards it at a speed chosen randomly from the interval (0, 1] meters per second. Upon reaching its destination, a node **pauses** for a random time drawn from the interval (0, 60] seconds. After the pause time expires, a new random destination is chosen, and the process repeats. This cycle runs continuously for the entire life of the simulation.

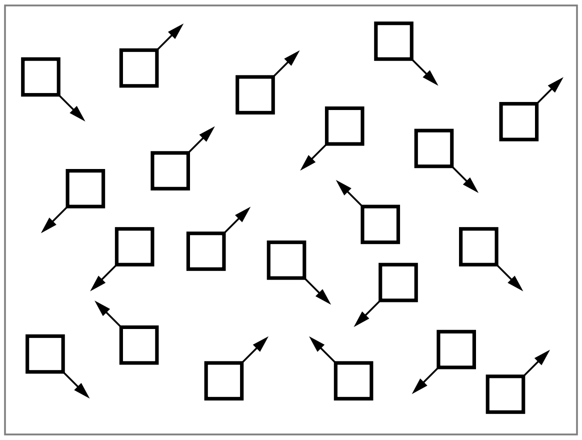


Figure 5.3: Unstructured movement pattern.

Figure 5.3 depicts the random waypoint movement pattern, not to scale. This **unstructured mobility model** is ubiquitous in mobile *ad hoc* network research, and we are obliged to include it here. The model is considered to be a good test of the underlying routing protocol and its ability to deal with **route breakage** caused by mobility. It models unorganized environments, such as **deserts** or **open plains**, in which mobile nodes are not constrained to follow particular paths.

5.2.2.2 Moderately Structured Mobility Model

Unstructured mobility is of theoretical interest for the stress it places on routing protocols. It is difficult to argue that such movement corresponds to any real physical scenario. Measurements of real-world wireless networks with mobile users show that **dwell time** is often substantial (Shaffer, 2003; Shaffer and Siewiorek, 2003). Users move to a "favorite site," then *stay* there for a long time before moving to another favorite site.

We have developed additional mobility models which attempt to capture this behavior. The first, which we call the **moderately structured mobility model**, is a small variation on the unstructured model. We divide the same 800m $\times$ 600m plane into five **zones**, shown in Figure 5.4. One or more nodes are placed in the center zone (radius 150 meters), the remaining nodes are distributed evenly across the remaining four zones. Nodes move *in their zone* according to the **random waypoint** algorithm described earlier, but never leave their zone. This reflects scenarios in which nodes are assigned to a particular area, such as **soldiers** on patrol.

A node near the middle of the network is more likely to appear on a route than a node at the **periphery**. In this mobility model, a node in the center zone can never reach the periphery. We expect such a node's energy consumption to be higher than that of nodes in peripheral zones.

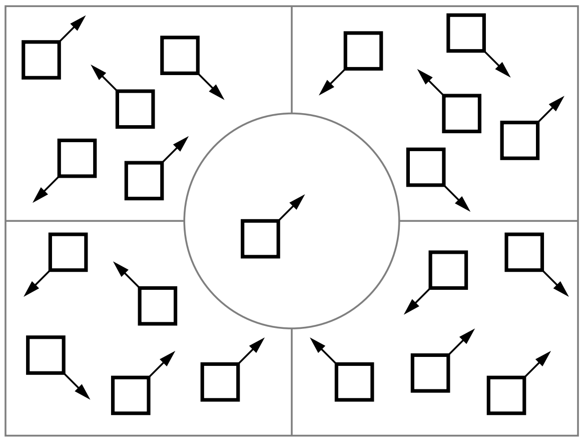


Figure 5.4: Moderately structured movement pattern.

5.2.2.3 Highly Structured Mobility Model

Finally, the **highly structured mobility model** is intended to capture the properties of **metropolitan** or **campus** environments. Nodes congregate in small spaces, and move from space to space along predefined paths. Figure 5.5 shows the model; this particular geometry might occur in an academic building.

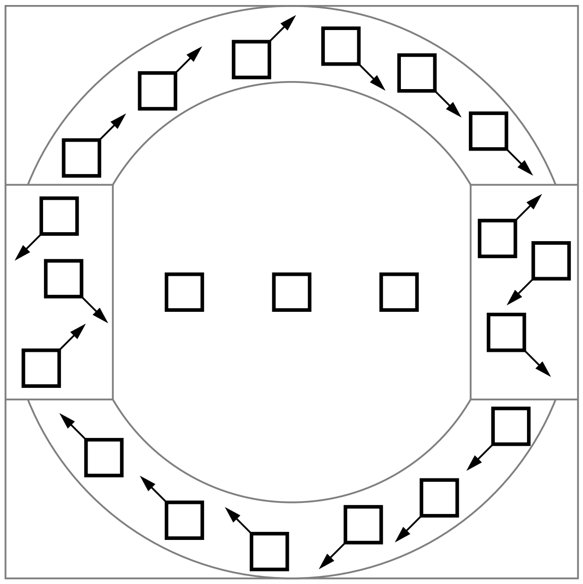


Figure 5.5: Highly structured movement pattern.

The plane is now 1km $\times$ 1km. Nodes are placed in small zones at the western and eastern ends of the plane; while in these end zones, they move according to the random waypoint algorithm described earlier. These zones might correspond to lecture halls or dining spaces. Occasionally, a node will leave its end zone and proceed at one meter per second along one of the **arc-shaped corridors** to the *other* end zone. These corridors might be hallways, sidewalks, or bridges. In addition, there are several **stationary** nodes placed in the middle of the plane. These might be users in **offices** whose positions are **static**.

Several routes are possible between the end zones: one uses the stationary nodes, the others use the arc-shaped corridors. The route involving the stationary nodes will tend to be chosen because it is **shorter** than the corridor routes. We expect the stationary nodes to have higher energy consumption than those nodes which are permitted to move.

Chapter 6

Experimental Results

This Chapter presents the measurements collected from experimental trials of the **multihop power management architecture**. Using the simulator and workloads from Chapter 5, we have studied the energy and communications performance of the design under a variety of conditions. Section 6.1 presents the energy consumption experienced by nodes under several power management schemes. In Section 6.2, we examine the interaction between power management and **latency**, while Section 6.3 shows the effects on message **delivery ratio**. Section 6.4 gives statistics on the performance of the **Exchange Power Management** negotiation procedure. Finally, Section 6.5 profiles our **Vickrey Clarke Groves mechanism** implementation, and Section 6.6 summarizes the experimental results.

The results of this Chapter reflect more than a thousand experimental trials spanning a range of workloads, which are organized by three parameters:

Traffic Load

We vary the maximum number of **concurrent traffic flows** which may be active at a given time. A traffic flow is a 512-octet $\times$ 4Hz CBR application. We present data for 0-flow, 1-flow, and 2-flow traffic loads (§5.2.1). The 0-flow load corresponds to an **idle** network. For most of the time in a 2-flow load, there are two **concurrently-active** traffic sources.

Mobility Model

We test each of the **unstructured** (§5.2.2.1), **moderately structured** (§5.2.2.2), and **highly structured** (§5.2.2.3) models. The maximum node speed under all models is one meter per second, which is a walking pace.

Power Management

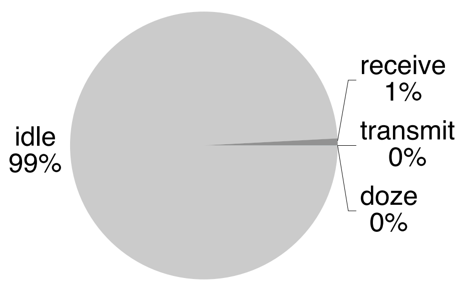
We apply five different power management schemes to each workload scenario. **IBSS** is the standard 802.11 "*ad hoc*" mode, *without* power management. **IBSS PM** is the power management mode *as defined by the 802.11 specification*. **ODPM** is the timer-based On-demand Power Management, the implementation of which we described in Section 5.1.4. **LPM** is our own timer-based Local Power Management scheme described in Section 4.3. Finally, we compare our Exchange Power Management using two separate valuation functions: **XPM Time** uses Time Balance valuation (§4.5.1), while **XPM Credit** uses Credit Balance valuation (§4.5.2).

For each combination of the three traffic loads and three mobility models, we randomly generated thirty scenarios, for a total of 270 scenarios. Each scenario involves 50 nodes and runs for 10,000 seconds of simulated time. We then applied all of the six power management schemes to each scenario, for a total of 1,620 trials. The results show that Exchange Power Management **reduces** worst-case node energy consumption, and offers **comparable** or **better** average-case energy performance than 802.11 power management. We also see that both **Route Discovery** and **application message delivery latency** improve substantially when using the multihop power management architecture.

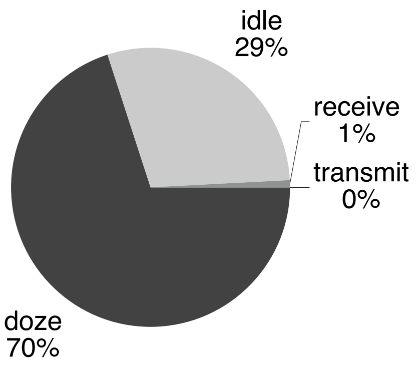
6.1 Energy Consumption

Using the power model described in Section 5.1.3, we have collected statistics on the **energy consumption** of nodes in the simulated trials. The simulator records the amount of **time** nodes spend in the various **power states** (*e.g.*, **doze**, **receive**). In postprocessing, we multiply these times by **power rates** to produce **energy** measurements.

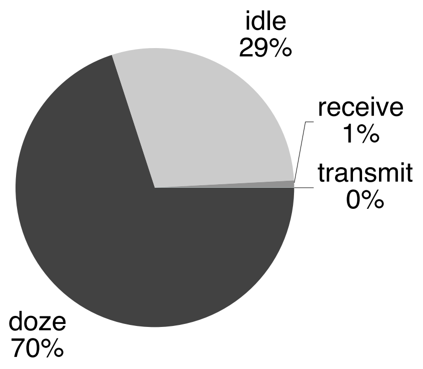
Figure 6.1 shows an example of the time spent in the four power states. These measurements are for an **idle** network using the **unstructured** mobility model. Each pie represents the sample mean times over 30 trials. Figure 6.1(a) shows that for **IBSS mode** (no power management), the **idle** state dominates. Adding power management (Figures 6.1(b) and 6.1(c)) shifts some of that idle time to the **doze** state, where the power rate is lower. The remaining examples, all instances of **mul-**



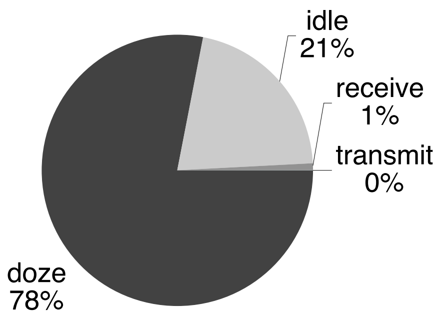
(a) IBSS.



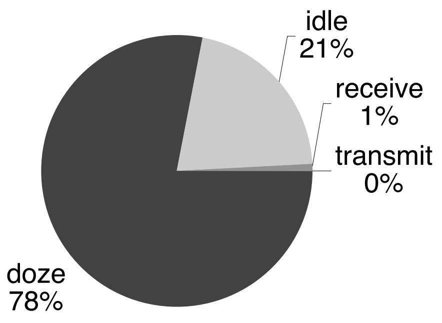
(b) ODPM.



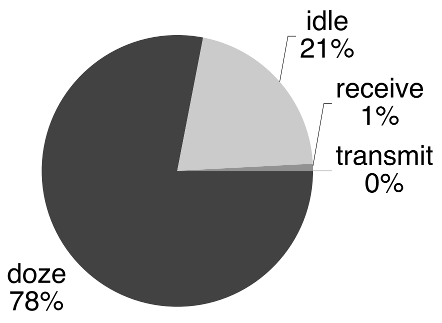
(c) IBSS PM.



(d) LPM.



(e) XPM Time.



(f) XPM Credit.

Figure 6.1: Power state times for unstructured, idle traffic load.

tihop power management, shift even more time to doze because of an improvement to the way **beacon frames** are treated under power management (§5.1.2.3).

The measurements of Figure 6.1 provide a critical insight into energy conservation for interfaces such as 802.11 transceivers. A power management scheme saves energy by reducing time spent in the **idle** state. Multihop power management reduces this time from 99% (in IBSS mode) to 21% in this example. In fact, $21\% \approx \frac{\text{ATIMWindow (40ms)}}{\text{BeaconInterval (200ms)}}$, which is the best case since there are no active traffic flows.[1]

Given the power state times, we can apply a variety of **power rates** to see how each would affect **energy consumption**. In this Section, we consider three sets of rates, one taken from actual measurements, the other two provided for hypothetical purposes. The **Feeney & Nilsson** rates, taken from (Feeney and Nilsson, 2001), were measured from the popular **WaveLAN** 802.11b interface. Starting from these rates, we want to know what would happen if the dominant power states were to become less costly. The $\frac{1}{2}$ **idle power** set is the same as Feeney & Nilsson, but the idle power rate is reduced by *half*. We then reduce idle power to *one tenth* its Feeney & Nilsson value, and reduce **doze** power to *one one-hundredth* in the $\frac{1}{10}$ **idle power,** $\frac{1}{100}$ **doze power** set. These hypothetical rates will not be discussed in the text, but are shown in plots of energy performance. The plots in Sections 6.1.1 and 6.1.3 show that while the **magnitudes** of energy consumption change, the relative **rela-**

[1]The measurement is not exactly 20% because, in our implementation, stations wake from doze 3ms prior to the **TBTT**, as described in Section 5.1.2.3. For this example, multihop power management stations spend about 43ms in idle during each Beacon Interval.

tionships between power management modes are preserved. This indicates that our techniques should still be effective for future wireless interfaces with better idle power performance. Table 6.1 shows the rates used for each set.

State	**Feeney & Nilsson**	$\frac{1}{2}$ **idle power**	$\frac{1}{10}$ **idle power,** $\frac{1}{100}$ **doze power**
doze	47.4mW	47.4mW	474μW
idle	739mW	370mW	73.9mW
receive	901mW	901mW	901mW
transmit	1.35W	1.35W	1.35W

Table 6.1: Power rate sets.

The following Sections present the **range**, **quartiles**, and **mean** of per-node energy consumption in our experiments. For each set of results, the Feeney & Nilsson rates are featured, followed by the same results under the hypothetical rates.

6.1.1 Per-Node Energy Range

We begin by characterizing the **variability** of energy consumption experienced by an individual node. A power management scheme that exhibits low variability is desirable when estimating the portion of the system **energy budget** that should be allocated to communications. Energy consumption in 802.11 interfaces is a **bounded variable**: given a workload, there is a nonzero **minimum** and finite **maximum** amount of energy that the interface can consume. When a variable is bounded in this fashion, the **range** is a suitable **index of dispersion** (Jain, 1991).

Section 2.1.3.2 derived the **minimum** energy consumption $E_{\min}$ for a station which processes no **control** or **data** frames, which we repeat here:

$$E_{\text{min}} = (w\, P_{\text{idle}} + (1 - w)\, P_{\text{doze}}) \times t + \beta(t)$$

The $\beta(t)$ term represents the additional energy consumed by the periodic transmission and reception of **beacon management frames** during t. For example, in Figure 6.1, $\beta(\cdot)$ was visible as the 1% of time spent in the **receive** state.

If a station does not use power management, $w \equiv 1$, and $E_{\text{min}} = P_{\text{idle}} \times t + \beta(t)$. For the purpose of describing a **lower bound**, let $\beta(\cdot) \rightarrow 0$. In our 10,000-second simulations, using the Feeney & Nilsson rates, such a station *cannot* consume fewer than $E_{\text{min}} = 7{,}390\text{J}$. When using power management, our simulated stations spend a 40 millisecond **ATIM Window** plus a 3 millisecond **early wakeup** in the idle state every 200 milliseconds, giving $w = \dfrac{\text{ATIMWindow (40ms) + EarlyWakeup (3ms)}}{\text{BeaconInterval (200ms)}} = 0.215$. The resulting $E_{\text{min}} = 1{,}961\text{J}$ means that the best-case **energy savings** from power management are about 73.5%.

The maximum energy consumption experienced by an 802.11 station depends on the time the station spends in **idle** waiting for communication, plus the energy to exchange the frames themselves. The worst case occurs when the station is *always* awake and waiting for communication. Let $\mathcal{C}(t)$ represent the *additional* energy, above that consumed in idle, required to transmit and receive non-beacon frames. The maximum energy consumption during time t is therefore:

$$E_{\text{max}} = P_{\text{idle}} \times t + \mathcal{C}(t) + \beta(t)$$

Both $\mathcal{C}(t)$ and $\beta(t)$ are bounded. Section 2.1.2 showed that there is a limit to the number of messages a station can send in a fixed amount of time, based on the **medium access control** algorithm. This leads to an upper bound on the additional energy consumed due to frame transmission. For the **CBR application**, we showed that $\mathcal{C}(t)$ could add **at most** 55% to energy consumption, but that typically the contribution was much smaller — less than 1%. Likewise, since beacons are only transmitted periodically, their contribution is largely a function of their frequency. Figure 6.1, in which beacons are the *only* frames being exchanged, showed that beacon activity takes up about 1% of the total time. The additional energy $\beta(\cdot)$ is similarly small.

A station that does not use power management *always* incurs the worst-case energy consumption. This occurs because such a station is always in the idle state, and can never **doze**. In fact, as $\mathcal{C}(\cdot) \to 0$, $E_{\text{max}} \to E_{\text{min}}$.

Power managing stations typically experience lower energy consumption than E_{max}, because they can enter the **doze** state. The amount of time they spend in **idle** is determined by **communications activity**. A station which expects to send or receive frames in the current **beacon interval** must remain awake in the idle state for the entire interval. Therefore, it is easy to make a power managing station experience $E_{\max}$: simply make it transmit or receive at least one message during every beacon interval.

In this Section, we will show examples of energy consumption approaching $E_{\max}$. A contribution of **Exchange Power Management** is that it provides nodes with a way to **limit** $E_{\max}$ to a lower value. Once a node raises its **ask price** sufficiently high, it may no longer be **affordable** as a relay or sink. At this point, it will no longer have to stay awake and wait for communications activity. We will show how this can serve as a "cap" on energy consumption.

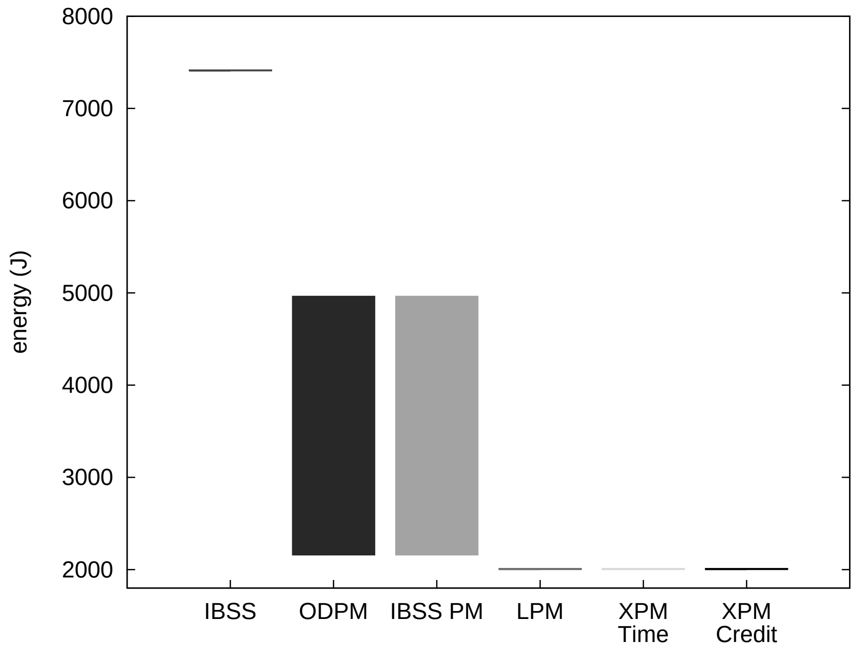
We refer to $(E_{\min}, E_{\max})$ as the **range** of energy consumption values experienced by a node. Call $E_{\max} - E_{\min}$ the **magnitude** of the range. Figures 6.2—6.10 show the sample ranges measured during simulation. Sample ranges with small magnitudes are desirable because they indicate **low variability**. As variability decreases, the ability to effectively **estimate** — and plan for — the energy consumption of the interface improves.

In this and subsequent Sections, we organize the data for our experimental trials first by **traffic load**, then by **mobility model**. The sequences begin with simple workloads and progress towards more complex ones. The data for the 0-flow (idle) traffic load are presented first, followed by the 1-flow and 2-flow loads. For each traffic load, the unstructured mobility model scenarios are presented first, followed by the moderately structured and highly structured models. Thirty **random scenarios** were generated for each combination of traffic load and mobility model, and six power management modes were applied to each scenario. Each Figure summarizes the results of 150 trials.

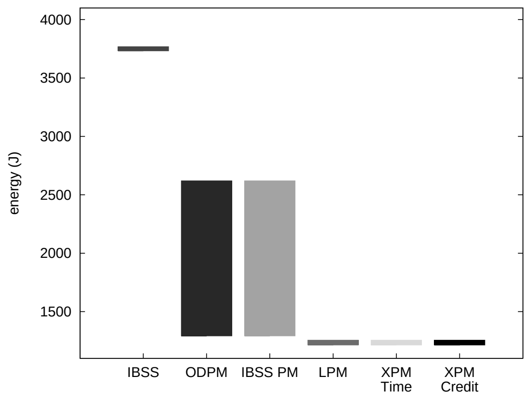
The order in which we present the power management nodes was chosen to reflect their general energy consumption, greatest to least. IBSS mode (using no power management) typically has the highest energy consumption. Next, ODPM and IBSS PM often have very similar energy behavior, with ODPM showing slightly higher worst case and average case consumption. These are followed by the new designs introduced in this thesis, LPM and XPM.

Figure 6.2 shows the energy ranges for each of the power management schemes in an **idle network**. Using the Feeney & Nilsson power rates in Figure 6.2(a), the range for **IBSS mode** (using no power management) has a very small magnitude. As the idle power rate drops in the hypothetical power models of Figures 6.2(b) and 6.2(c), the magnitude grows. This is due to the increased significance of the transmit and receive states, as some stations process more beacons than others. For example, a station with many neighbors might receive many beacon frames after a TBTT, while a station in a sparse region of the network receives relatively few.

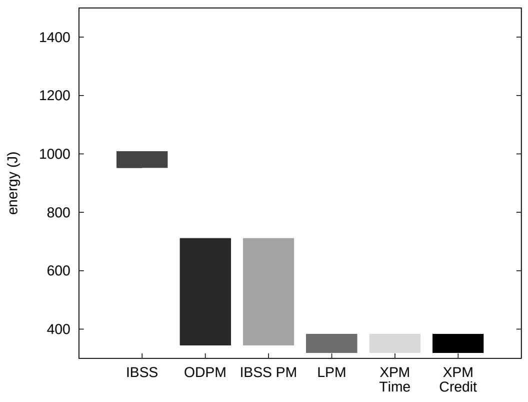
The ODPM and IBSS PM ranges have a conspicuously large magnitude. Section 5.1.2.3 explained that stations which transmit **beacon** frames must remain awake for the entire **beacon interval**. This means that stations with "fast" timers, and those in **sparse** regions of the network, will tend to win the contention-based **beacon generation algorithm** (§5.1.2.1) more often, and will spend relatively more



(a) Feeney & Nilsson.



(b) $\frac{1}{2}$ idle power.



(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

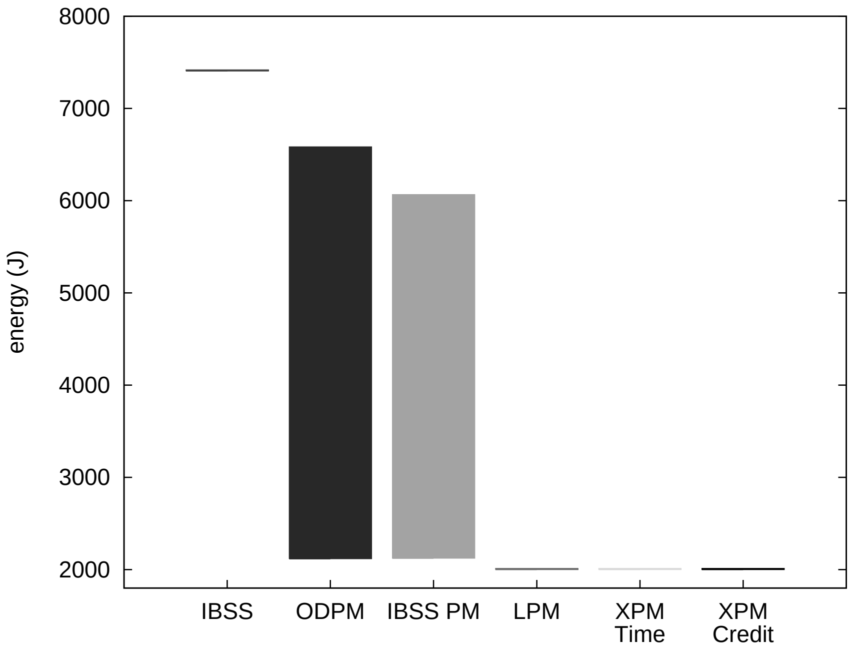
Figure 6.2: Unstructured energy range for idle traffic load.

beacon intervals awake. These stations will experience higher energy consumption than stations which send fewer beacons.

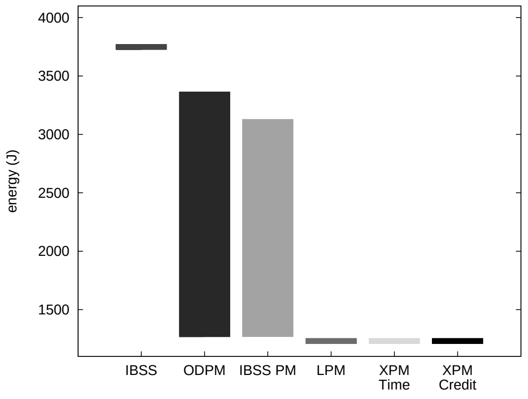
When using the **multihop power management architecture**, our 802.11 implementation permits beacon-transmitting stations to doze after the ATIM Window. As such, the LPM and XPM ranges are all identical for the idle traffic load; the number of beacons a station sends has minimal impact on energy consumption. We note that the sample minimum energy for the LPM and XPM samples is slightly higher than $E_{\text{min}} = 1{,}961$J, derived earlier. The difference is explained by the additional energy consumed when exchanging **beacon frames**, $\beta(\cdot)$.

Figures 6.3 and 6.4 show similar representations for the idle traffic load in the **moderately structured** and **highly structured** mobility models. The variation between the ranges is accounted for by differences in geometry across the mobility models. For our models, as the degree of **structure** increases, so does **sparseness**. Stations in sparse regions of the network will tend to send more beacons and experience higher energy consumption, particularly when using ODPM or IBSS PM mode.

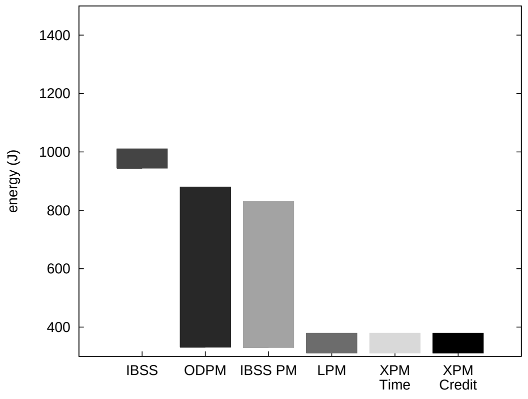
Figure 6.5 shows the 1-flow traffic load in the **unstructured** mobility model. This representation plots **energy** *vs.* **messages sent**. Each node is assigned to a **bucket** of size 100 messages, corresponding to the number of messages the node was able to send as a **source** during the trial. For each bucket, the maximum and minimum points are plotted, and the region enveloped by those points is shaded.



(a) Feeney & Nilsson.

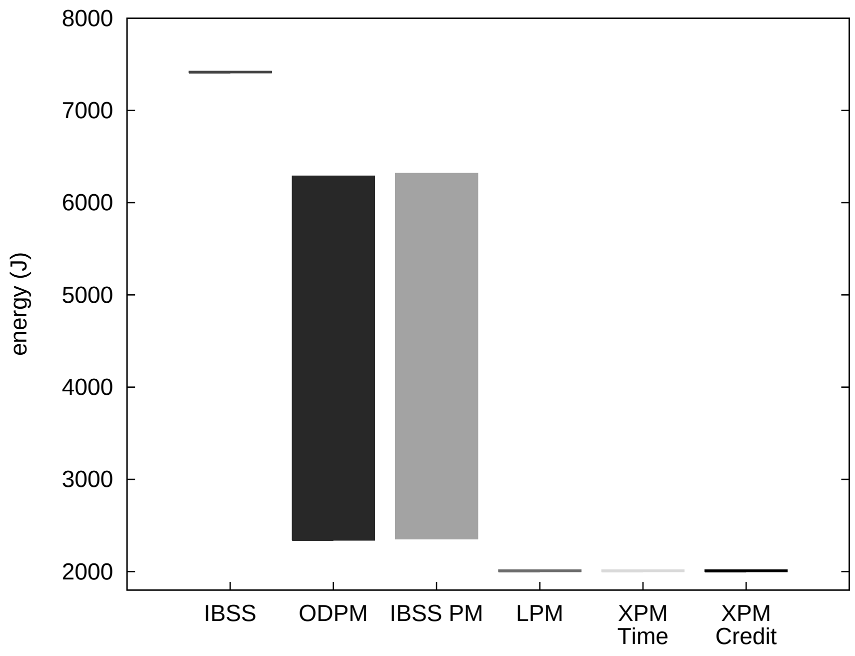


(b) $\frac{1}{2}$ idle power.

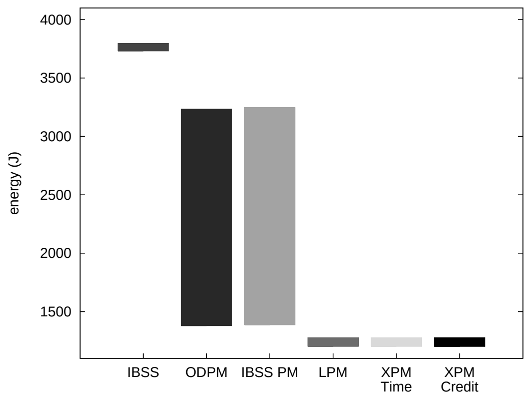


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

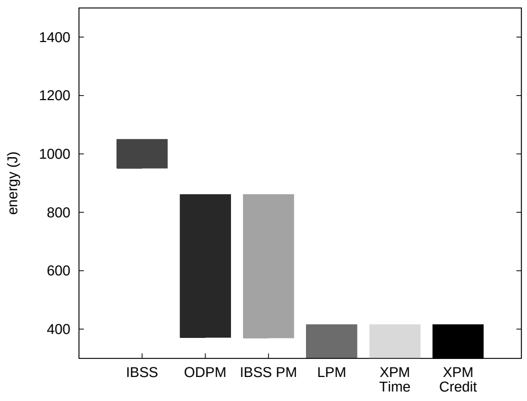
Figure 6.3: Moderately structured energy range for idle traffic load.



(a) Feeney & Nilsson.

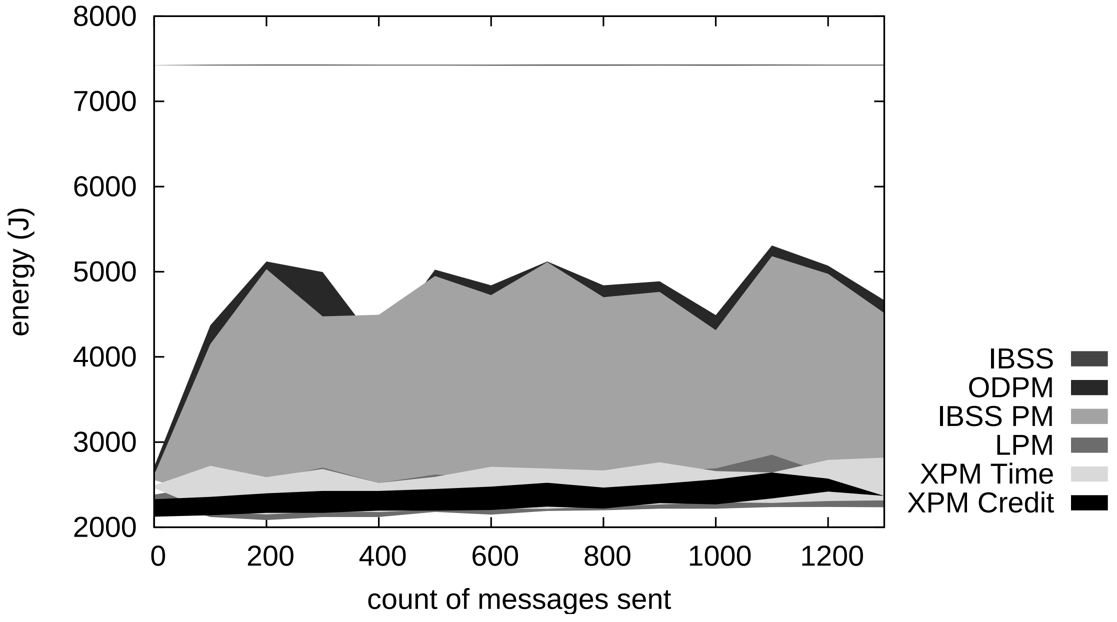


(b) $\frac{1}{2}$ idle power.

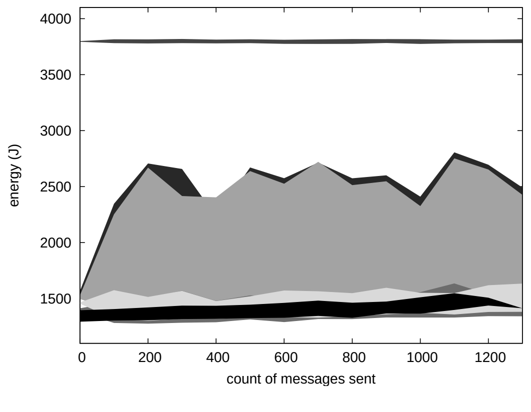


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

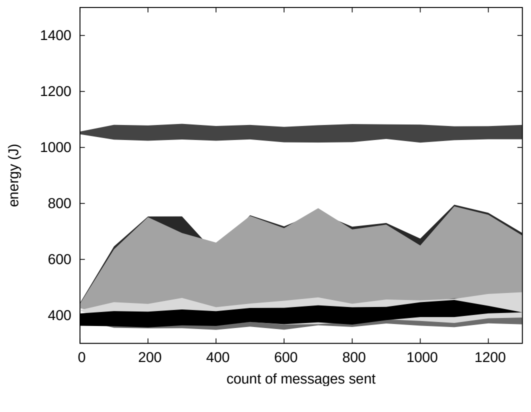
Figure 6.4: Highly structured energy range for idle traffic load.



(a) Feeney & Nilsson.



(b) $\frac{1}{2}$ idle power.



(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

Figure 6.5: Unstructured energy range for 1-flow traffic load.

The traffic loads are generated in such a way that when all messages are successfully **sourced**, the buckets all contain a comparable number of nodes. A node which is prevented from sourcing some of its messages — either because it could not **discover** a route, or because it could not **afford** one — may be moved to a lower bucket. The former affects all five power management modes. The latter is relevant only to XPM with **Credit Balance** pricing, for which unaffordable routes are common.

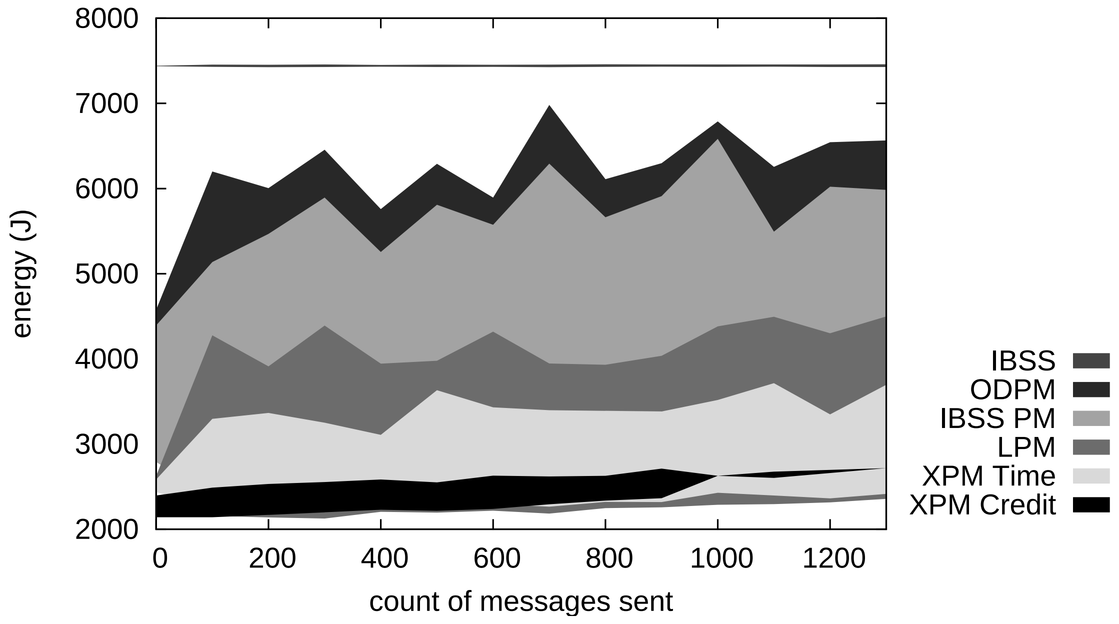
This representation reinforces the fact that a node's energy consumption is not necessarily linked to its own communications activity. For example, Figure 6.5(a) shows that under ODPM or IBSS PM, it is possible for a node that sources 200 messages to consume about 5,000J. It is also possible for a source that sends *six times* as many messages to consume *half* as much energy!

Figure 6.5(a) shows that the range for IBSS again has a very small magnitude, about 10J. The *minimum* energy consumed under IBSS — about 7,420J — is 40% greater than the *maximum* of IBSS PM, and 175% greater than the maximum of LPM. In this and subsequent Figures, the order of entries in the key — top to bottom — generally follows the order of peak energy consumption — greatest to least.

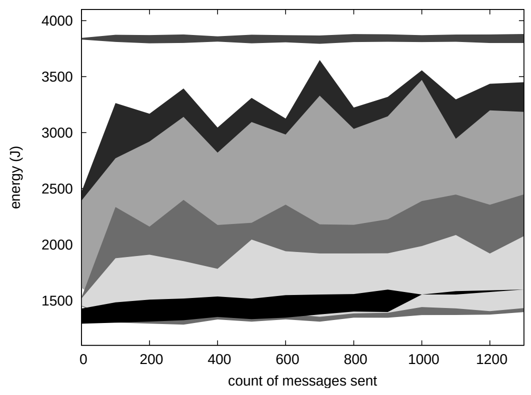
The magnitudes of the ODPM and IBSS PM ranges are generally larger than that of LPM or XPM. We compare ranges by **averaging** the magnitudes over the buckets, and reporting the **factor** by which the average IBSS PM magnitude is larger than the others. Expressed this way, **larger** factors are better, as this means the mode being compared to IBSS PM has lower variability. In Figure 6.5(a), the **magnitude** of the range for IBSS PM is 5.1 times that of LPM, 5.2 times that of XPM Time, and 9.6 times that of XPM Credit. This shows that all of the **multi-hop power management** designs yield much more **predictable** energy behavior than 802.11 IBSS power management. As the degree of **structure** in the mobility model increases, we will show how XPM and its **valuation functions** preserve a small-magnitude range (larger factor), while non-negotiated schemes (*e.g.*, ODPM, IBSS PM, LPM) become less predictable.

The previously-published ODPM design exhibits a range that is similar to IBSS PM. ODPM uses IBSS power management, but uses timers to keep stations "awake" for longer durations following frame activity. Therefore, the ODPM range has a similar shape to the IBSS PM range, with a slightly greater magnitude caused by the longer durations spent in the "awake" state. Because of this similarity to IBSS PM, we will not separately characterize the relationship between ODPM and our LPM and XPM designs in the text.

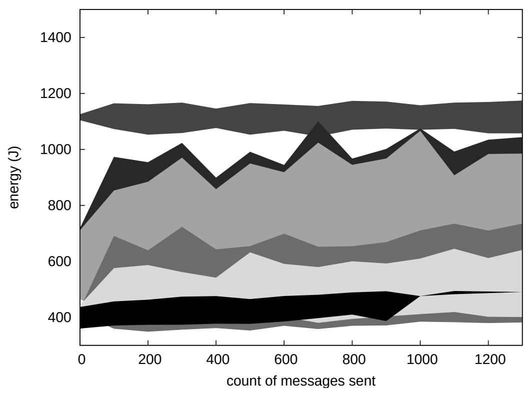
Figure 6.6(a) shows how the energy ranges change as more **structure** is added to the mobility model. Peak energy for IBSS PM, LPM, and XPM Time have all increased visibly. The magnitudes have also increased; IBSS PM has a magnitude almost 50% larger than in the unstructured case. The IBSS PM magnitude is now only 1.8 times that of LPM, and 3.1 times that of XPM Time. However, it is 12.3 times that of XPM Credit, which has kept a tight "cap" on energy consumption,



(a) Feeney & Nilsson.



(b) $\frac{1}{2}$ idle power.



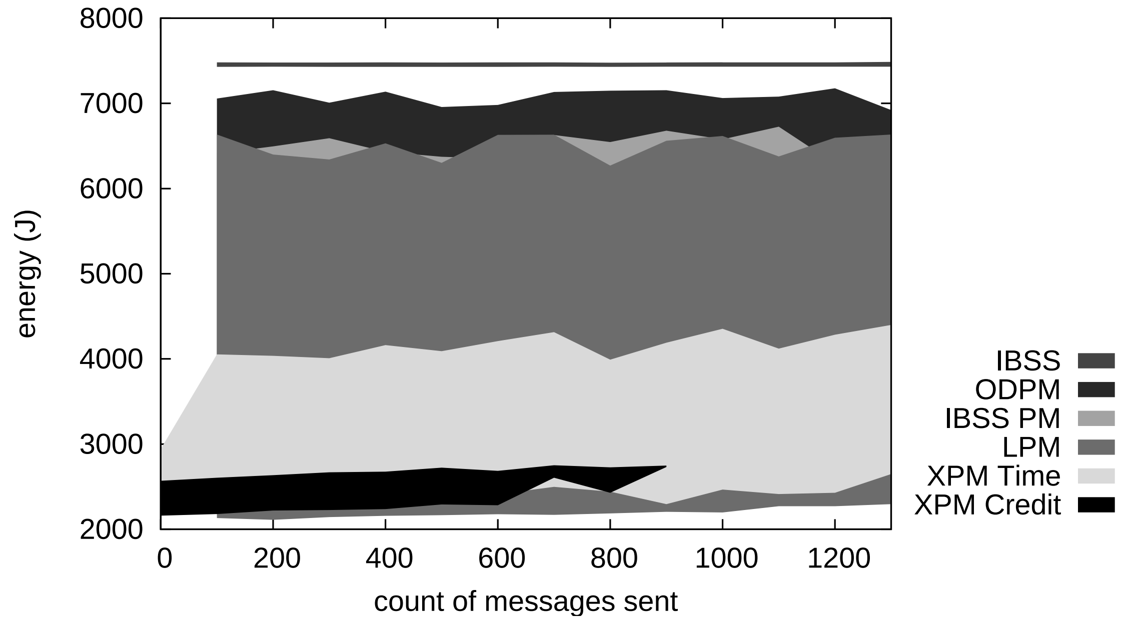
(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

Figure 6.6: Moderately structured energy range for 1-flow traffic load.

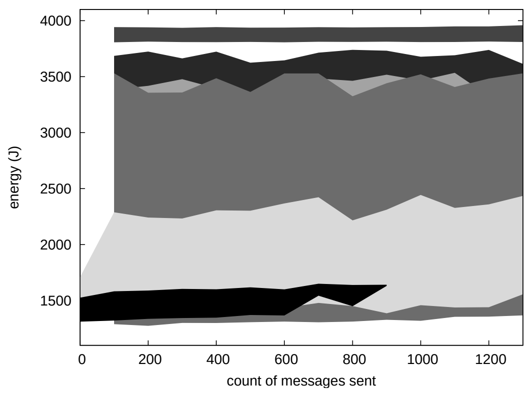
peaking at just 2,720J.

The nodes which experience the high peak energy consumption shown in Figure 6.6 are those in **distinguished positions** with respect to the network topology. In the moderately structured workload, nodes in the center zone occupy distinguished positions. Because these nodes are always near the geographic center of the network, they are more likely to appear on routes that traverse the network. Also, since a route through the center of the network is likely to be **shorter** than a more circuitous route, sources which select **shortest routes** will force these distinguished nodes into service more often. ODPM, IBSS PM and LPM use shortest routes, so we expect those modes to exhibit the worst peak energy consumption. XPM Time takes node **preferences** into account, and tries to route around distinguished nodes, but sometimes has no alternative but to use them. As a result, peak energy consumption for these nodes can still grow, but at a slower rate than the non-negotiated schemes. XPM Credit, through the use of **unaffordable routes**, has a means of avoiding overuse of distinguished nodes. By preventing overuse, peak energy is kept low.

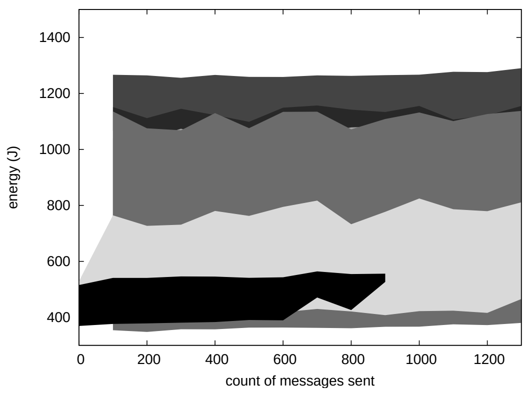
In the highly structured mobility model of Figure 6.7(a), LPM behaves very much like IBSS PM, with the two peaking at 6,634J and 6,726J, respectively. These peaks are within about 10% of IBSS mode for the same workload, meaning that some nodes save almost nothing by using power management! The affected nodes are those in **distinguished positions**, which in the highly structured mobility model



(a) Feeney & Nilsson.



(b) $\frac{1}{2}$ idle power.



(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

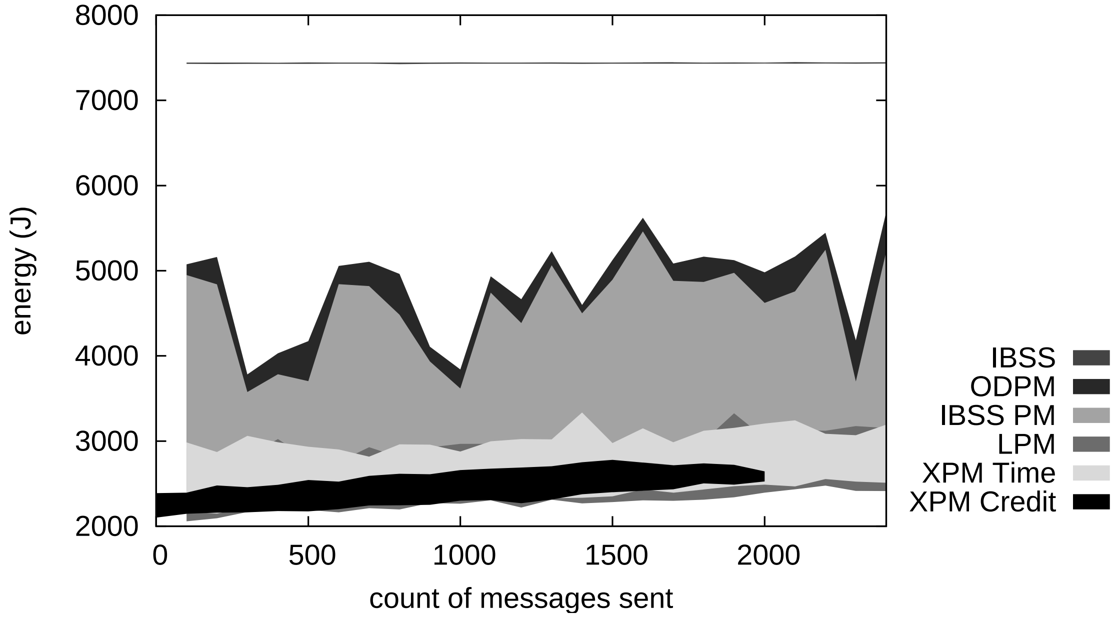
Figure 6.7: Highly structured energy range for 1-flow traffic load.

are those in the center corridor. The magnitude of IBSS PM is also about 11% *smaller* than LPM, because while the *worst* cases are comparable, in the *best* case LPM consumes less energy. IBSS PM has a magnitude that is 2.3 times that of XPM Time, and 11.3 times that of XPM Credit. Again, XPM Credit keeps energy consumption below 2,751J, but as we will later show, this comes at the cost of not being able to deliver some messages. XPM Time, which emphasizes **high delivery ratio**, must tolerate a higher worst-case energy consumption of 4,399J.

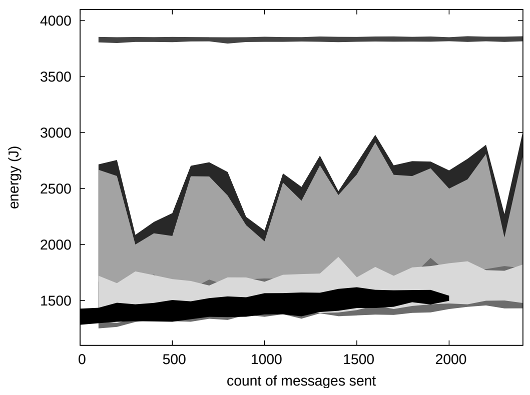
Figure 6.8(a) increases the number of active traffic flows to 2 in the unstructured mobility model. Peak energy has increased by a few hundred Joules for all modes except IBSS. The magnitude of the IBSS PM range is 3 times that of LPM and XPM Time, and 6.5 times that of XPM Credit.

The energy ranges become more volatile in Figure 6.9(a), which depicts the moderately structured mobility model for the 2-flow traffic load. IBSS PM now peaks at 6,982J, at which point the energy savings are 7% relative to IBSS-mode (with no power management). The IBSS PM magnitude is only 1.4 times that of LPM, and 2.5 times that of XPM Time, but 11.5 times that of XPM Credit.

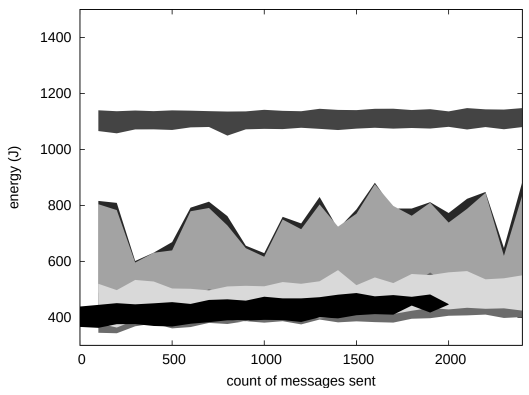
Finally, Figure 6.10(a) shows the 2-flow traffic load under the highly structured mobility model. As with the 1-flow case, LPM and IBSS PM are very similar under these conditions as they approach the upper bound $E_{\max}$. XPM Time is able to contain energy consumption below 4,892J while maintaining a high delivery ratio. XPM Credit offers the more aggressive "cap" of 2,919J with a tradeoff in de-



(a) Feeney & Nilsson.

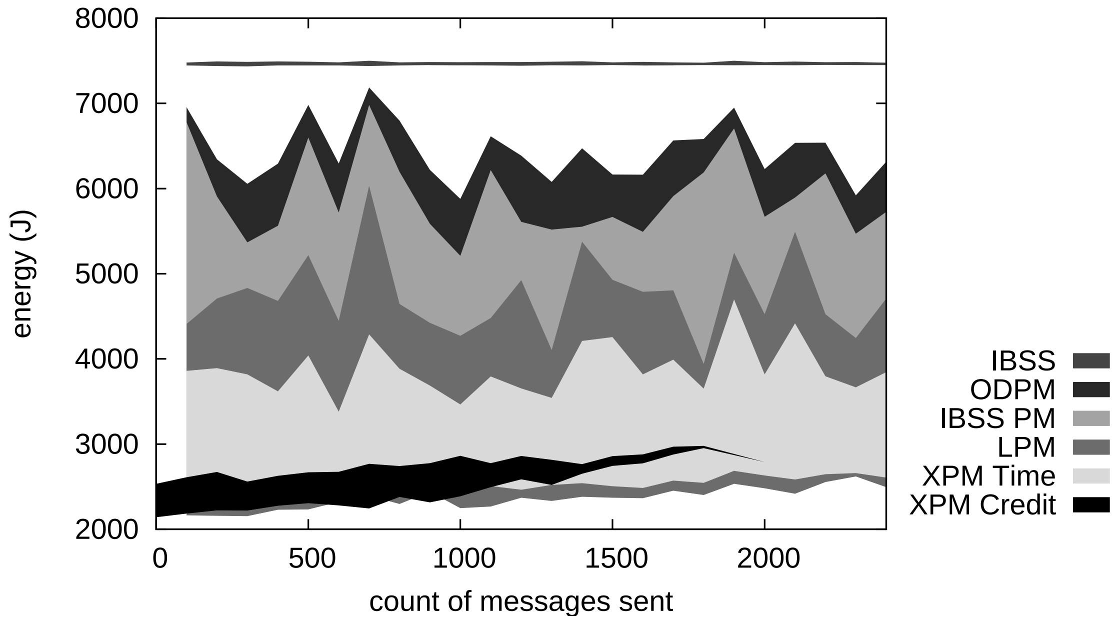


(b) $\frac{1}{2}$ idle power.

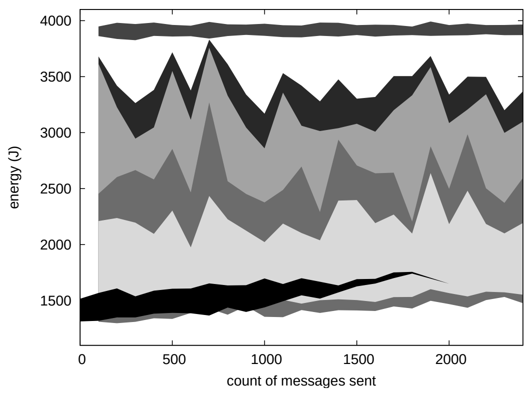


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

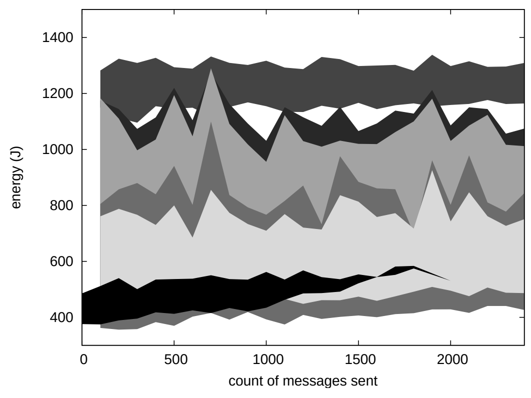
Figure 6.8: Unstructured energy range for 2-flow traffic load.



(a) Feeney & Nilsson.

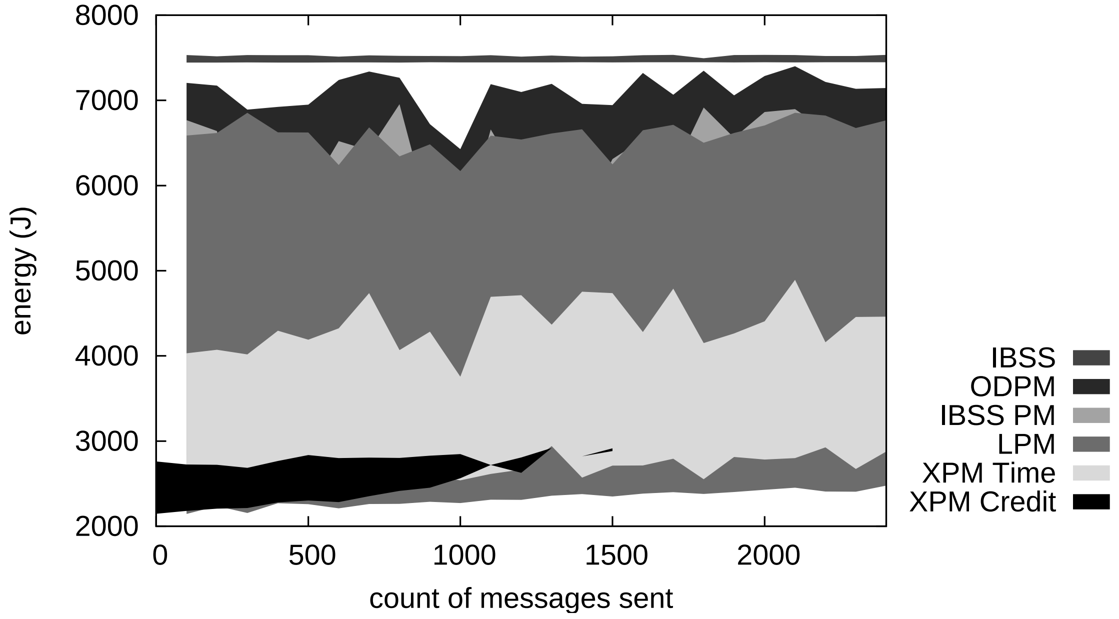


(b) $\frac{1}{2}$ idle power.

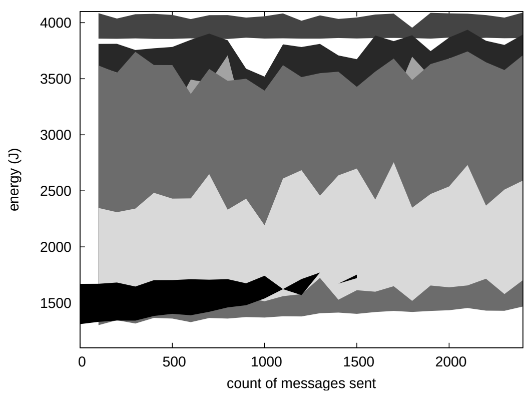


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

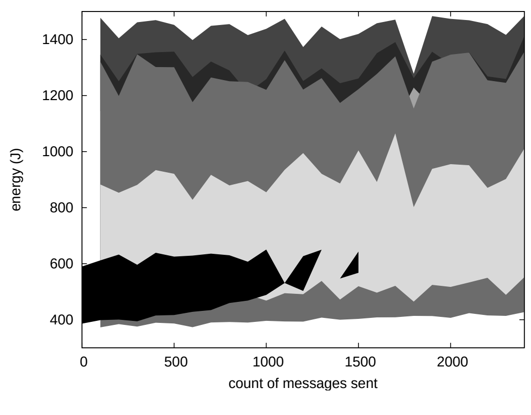
Figure 6.9: Moderately structured energy range for 2-flow traffic load.



(a) Feeney & Nilsson.



(b) $\frac{1}{2}$ idle power.



(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

Figure 6.10: Highly structured energy range for 2-flow traffic load.

livery ratio. Comparing worst cases (peak energy consumption) relative to IBSS-mode, IBSS PM saves about 8% energy, LPM saves 9%, XPM Time saves 35%, and XPM Credit saves 62%. The magnitude of the IBSS PM range is about 17% *less* than that of LPM, but 2 times that of XPM Time, and 10.5 times that of XPM Credit.

We have shown that **Exchange Power Management** delivers a more **compact** range of energy consumption values under a variety of traffic and mobility conditions. Under intensive workloads, timer-based power management schemes such as **Local Power Management** yield the same worst-case energy behavior as 802.11 IBSS power management. By using negotiations that incorporate node **preferences**, XPM is able to limit the worst case in two ways. Using **Time Balance** valuations, relays and sinks that have already provided "enough" service to the network can indicate their relative willingness to supply more. This lets sources route around overused relays while still keeping delivery ratio high. An additional limit available with the **Credit Balance** valuation function is the concept of **unaffordable routes**. A relay or sink that is "too expensive" for a source will *not* be used, and can achieve a tight bound on worst-case energy consumption. When high delivery ratio is required, these results show that the magnitude of the energy consumption range can be reduced by factors of 2–5.2 relative to 802.11 IBSS power management. When **unaffordable routes** are permitted, this reduction improves to factors of 6.5–12.3.

An interesting feature of Exchange Power Management is that its **valuation**

functions shape energy consumption without specifically taking energy as an input. Instead, the functions operate on abstractions such as **credit** or the **time** spent supplying or demanding service in the network. Because of this design, XPM can be applied in heterogeneous environments for which not all nodes have the same energy consumption or capacity characteristics.

No previous on-demand design for power management in *ad hoc* networks has addressed the problem of **worst case** energy consumption, or **variability** in energy consumption. **On-demand Power Management** (§3.1.3.5), an existing design which we simulated, experiences worst case energy consumption that is greater than that of IBSS PM. **Span** (§3.1.3.4) and **GAF** (§3.1.3.3), which are **proactive** protocols, also suffer from the problem of **distinguished nodes**. In both designs, an active relay having no neighbors which could replace it *must* remain awake — even if there are no active traffic flows!

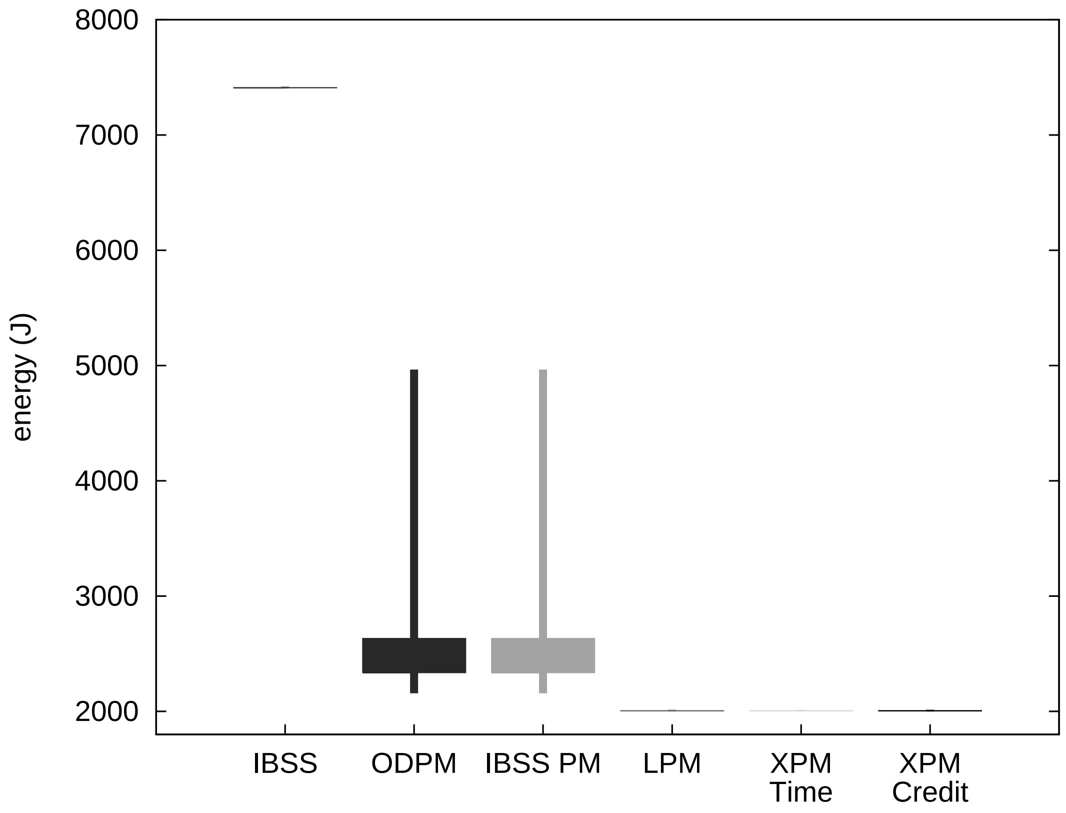
6.1.2 Per-Node Energy Quartiles

The previous Section characterized the **range** of energy consumption values for the six power management modes. This Section provides additional information about the dispersion of the data, in the form of **quartiles**. We present the relationships between the range, the **median**, and **interquartile range** (Jain, 1991).

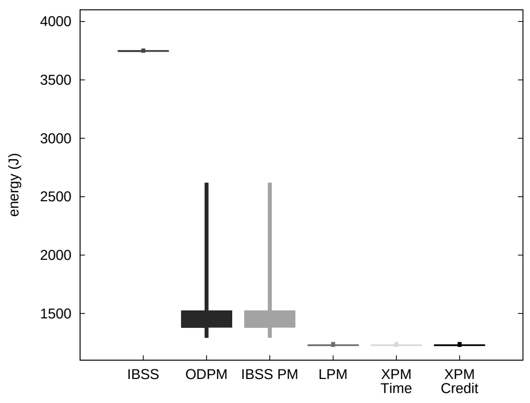
The quartile representation shows that most of the sample data tend to be clustered near the lower end of the range. This is what we expect for our workloads, since the nodes with near-worst case energy consumption will be those “stuck” in **distinguished positions** within the network, and these are in the minority. Our goal in developing XPM was not to improve average case energy consumption, although the data show that XPM performs better in this regard than previous designs. Instead, we focus on controlling the worst case, which we demonstrated in the previous Section.

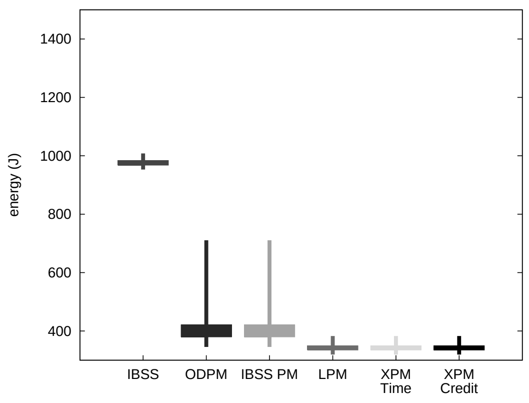
Predictable energy consumption is just as important as **expected** energy consumption in some environments. For instance, consider soldiers in a battlefield scenario. A communications device which experiences unexpectedly high energy consumption may exhaust its battery, and leave its operator unable to contact the rest of the unit. Such a situation may endanger the operator or the entire group. By controlling the worst case energy consumption of the wireless interface, mission planners can better equip soldiers with appropriately-sized batteries.

Figure 6.11 shows the energy consumption quartiles for each of the six power management modes in the unstructured mobility model with no application traffic. The candlestick representation shows, from bottom to top, the minimum, first quartile (Q_1), third quartile (Q_3), and maximum (Q_4). ODPM and IBSS PM have high maxima due to the 802.11 requirement that stations which transmit a beacon must remain awake for the entire beacon interval (§5.1.2.3), as explained in the previous Section. The minority of stations with “fast” timers that tend to win the beacon contention algorithm more often will occupy the high end of the en-



(a) Feeney & Nilsson.





(b) $\frac{1}{2}$ idle power.

(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

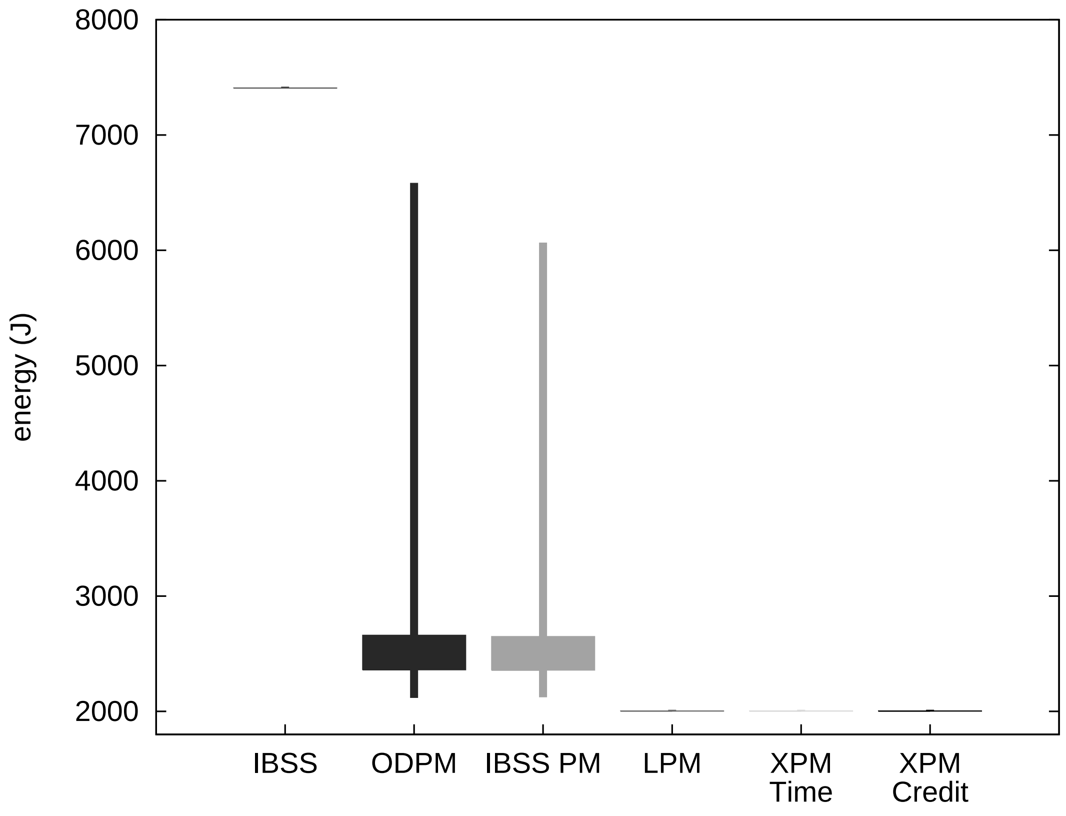
Figure 6.11: Unstructured energy quartiles for idle traffic load.

ergy range, which accounts for the distributional skew towards the low end. Figures 6.12 and 6.13 show quartiles for the moderately structured and highly structured workloads. The form of the energy distributions is similar to that of the unstructured case.

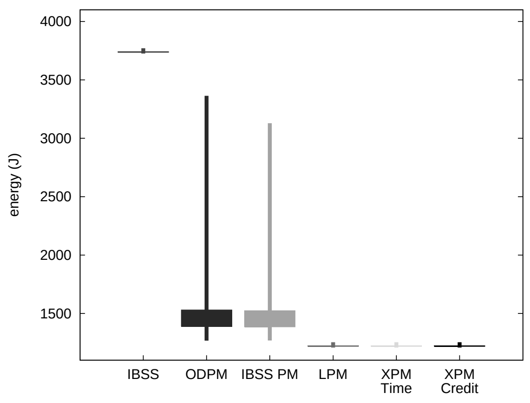
Figure 6.14 shows the quartiles for the unstructured mobility model with the 1-flow traffic load. As with the range plots from Section 6.1.1, the nodes are sorted into buckets based on the number of messages they were able to source during a trial. For ease of presentation, the size of the buckets has been increased to 500 messages. We again plot quartiles as with the idle traffic load cases, but additionally join the **median** (Q_2) data points across buckets for each series.

The quartiles of Figure 6.14 reveal that, relative to IBSS mode with no power management, all of the power managing designs yield median energy savings of about $\frac{2}{3}$. As mentioned earlier, the best case savings from a design based on 802.11 IBSS power management are 73.5% using the parameters adopted in this work. In fact, the median energy consumption (Q_2) for LPM and XPM is lower than that for the previously-published ODPM or IBSS PM. Also, the interquartile range $[Q_1, Q_3]$ is tighter for our designs than for ODPM or IBSS PM. This says that our improvement to the worst case (Q_4) has *not* come at the expense of the common case.

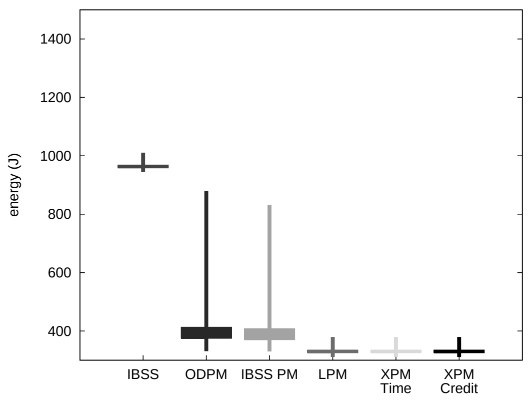
In Figures 6.15 through 6.19, the distributions move upwards. The worst case (Q_4) for LPM begins to look more like that of ODPM and IBSS PM, although the median (Q_2) is still lower. XPM, which consistently provides better worst case be-



(a) Feeney & Nilsson.

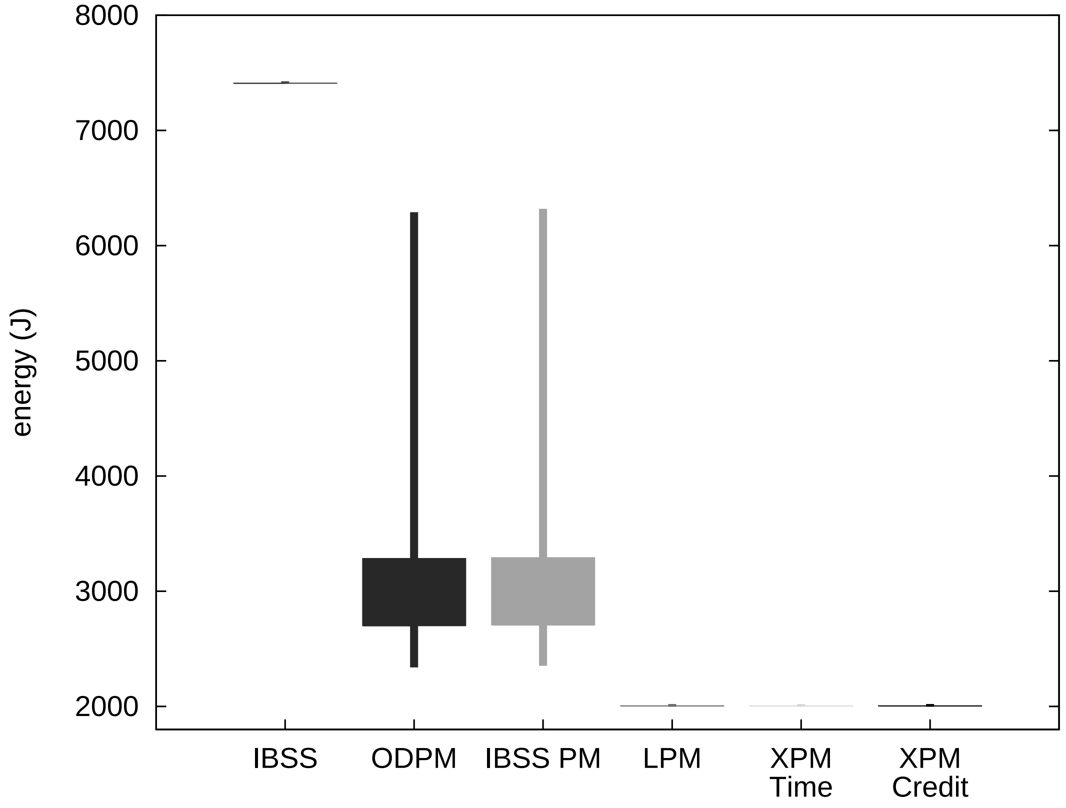


(b) $\frac{1}{2}$ idle power.

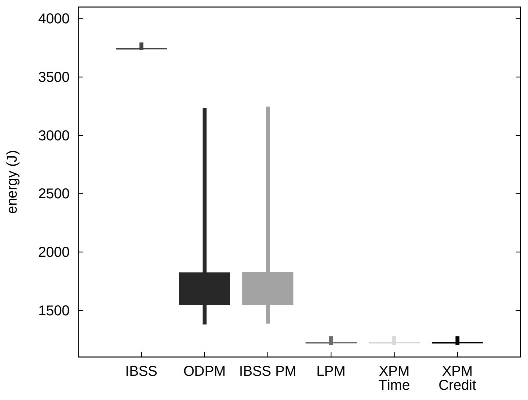


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

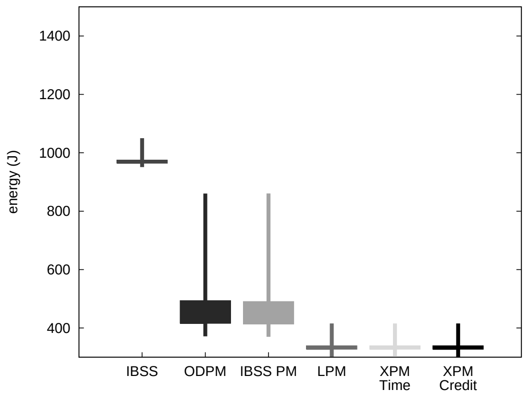
Figure 6.12: Moderately structured energy quartiles for idle traffic load.



(a) Feeney & Nilsson.

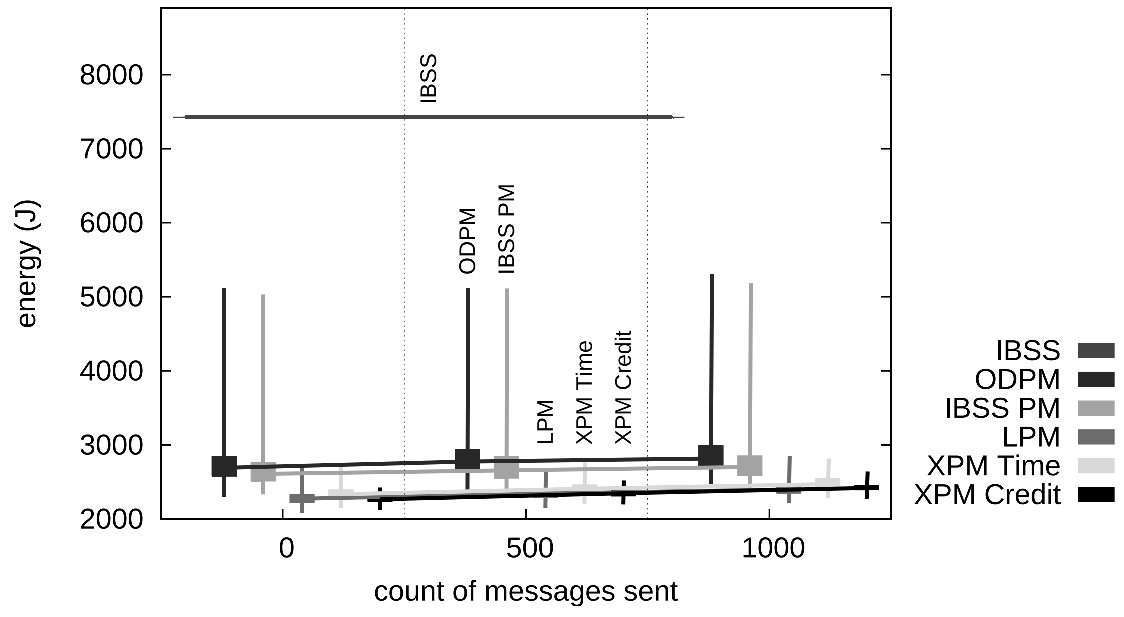


(b) $\frac{1}{2}$ idle power.

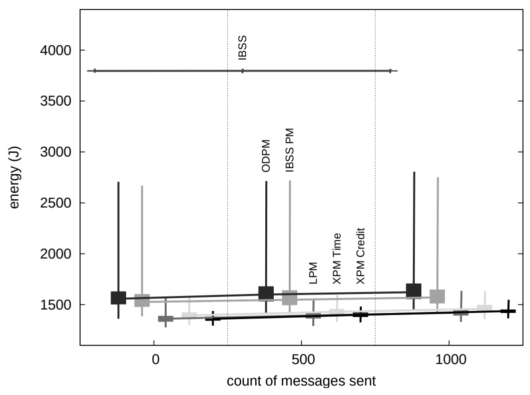


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

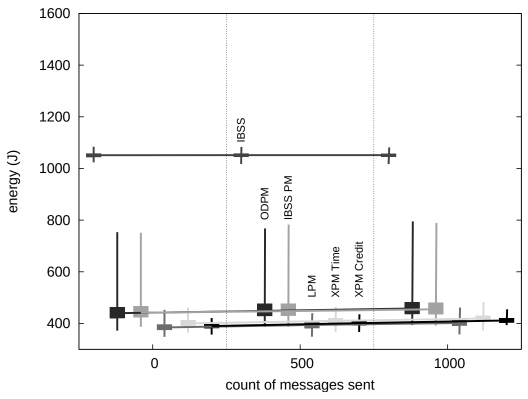
Figure 6.13: Highly structured energy quartiles for idle traffic load.



(a) Feeney & Nilsson.

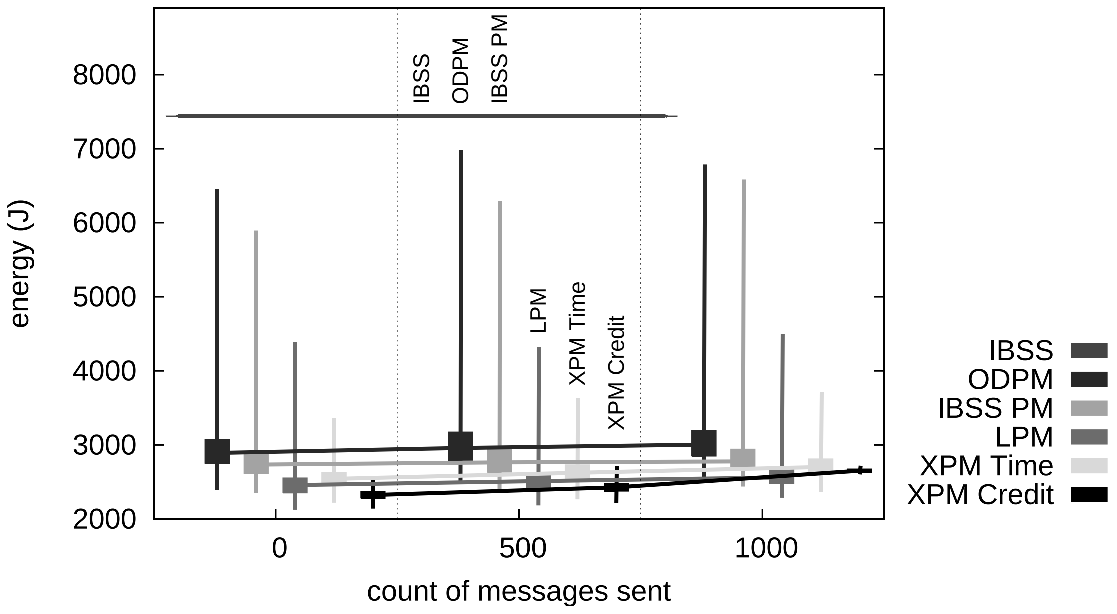


(b) $\frac{1}{2}$ idle power.

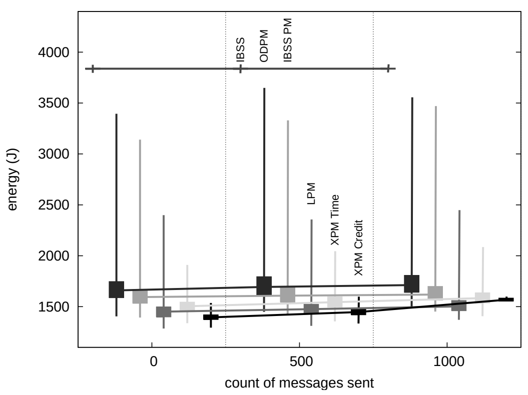


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

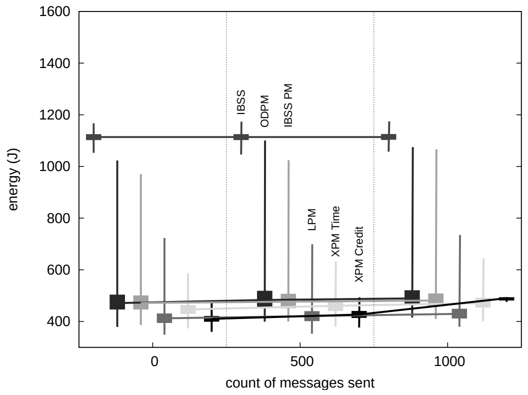
Figure 6.14: Unstructured energy quartiles for 1-flow traffic load.



(a) Feeney & Nilsson.

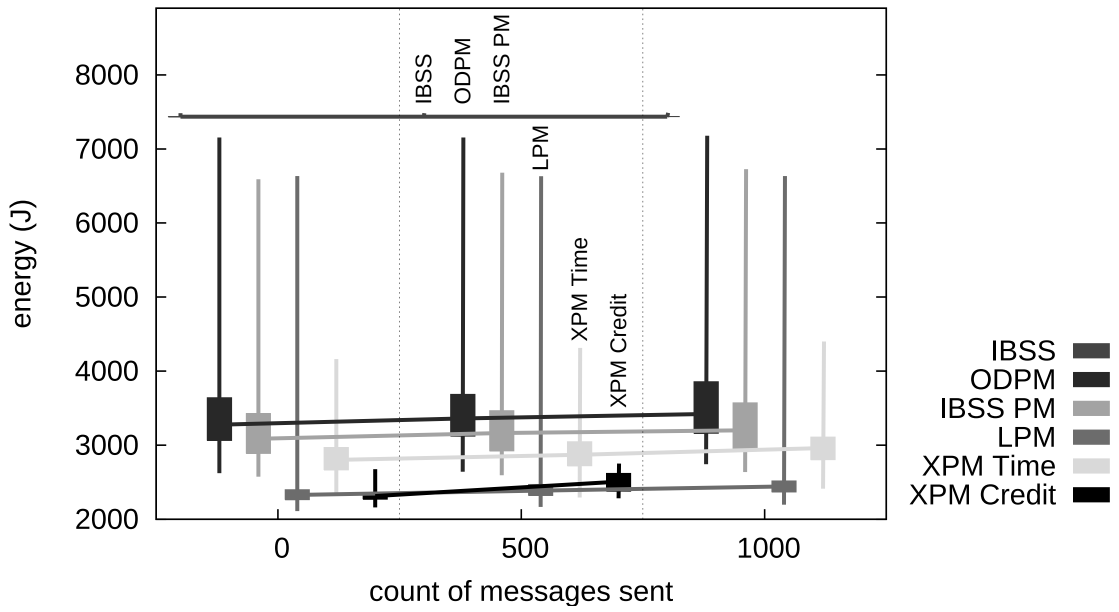


(b) $\frac{1}{2}$ idle power.

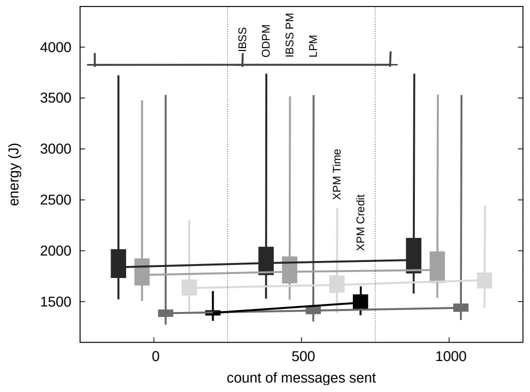


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

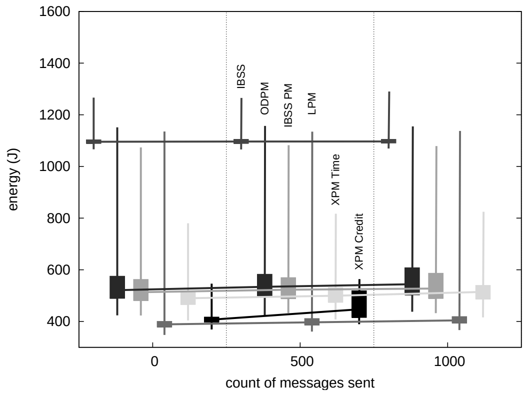
Figure 6.15: Moderately structured energy quartiles for 1-flow traffic load.



(a) Feeney & Nilsson.

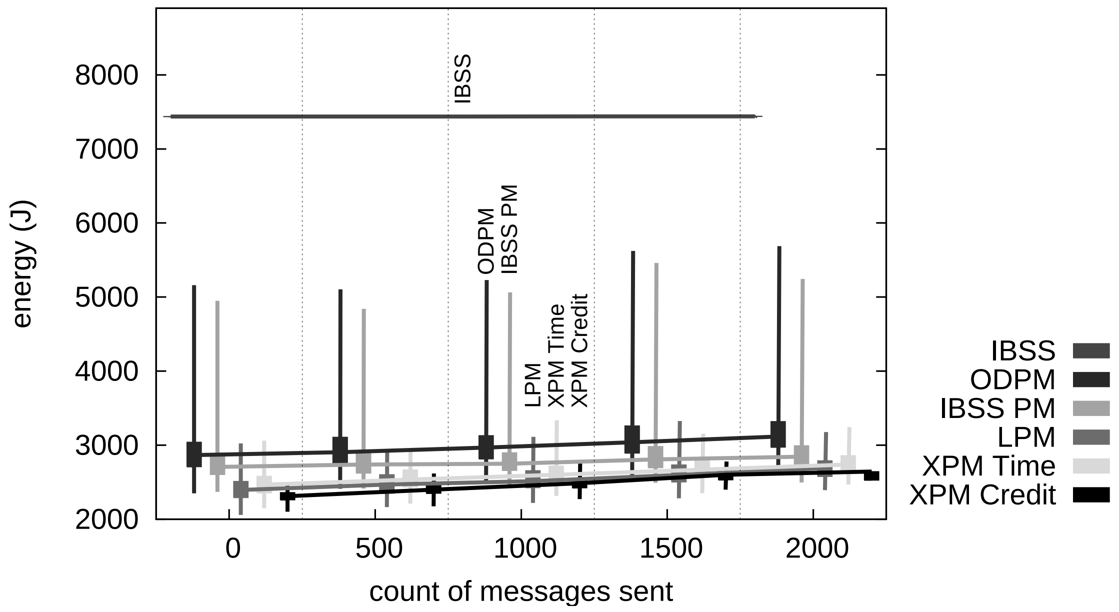


(b) $\frac{1}{2}$ idle power.

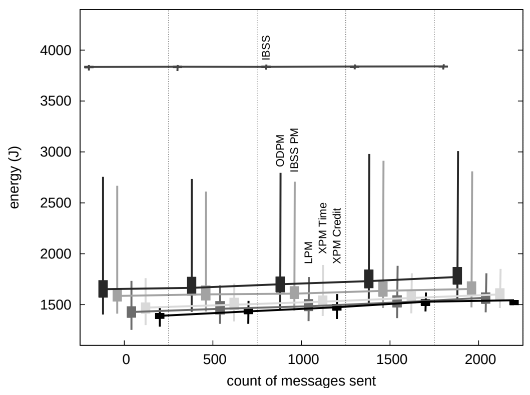


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

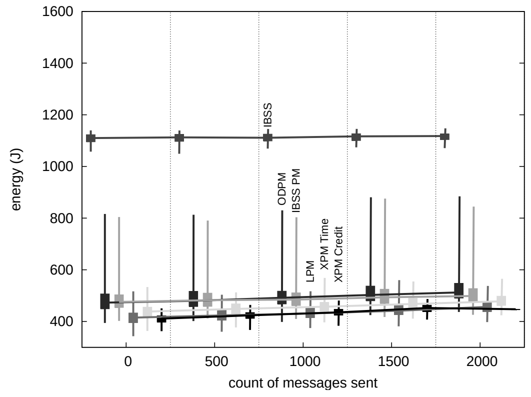
Figure 6.16: Highly-structured energy quartiles for 1-flow traffic load.



(a) Feeney & Nilsson.

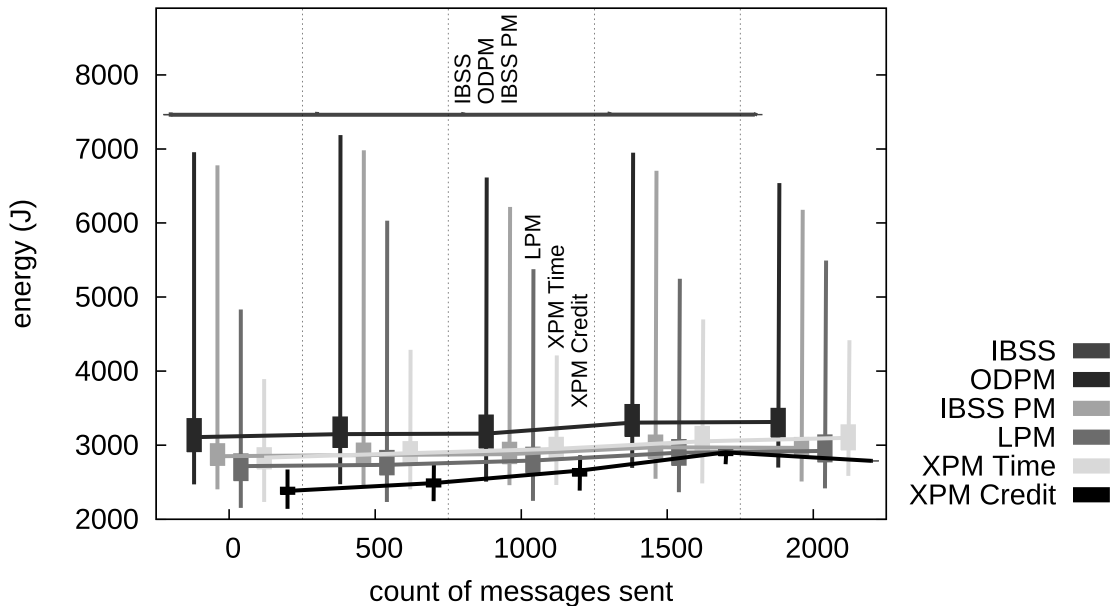


(b) $\frac{1}{2}$ idle power.

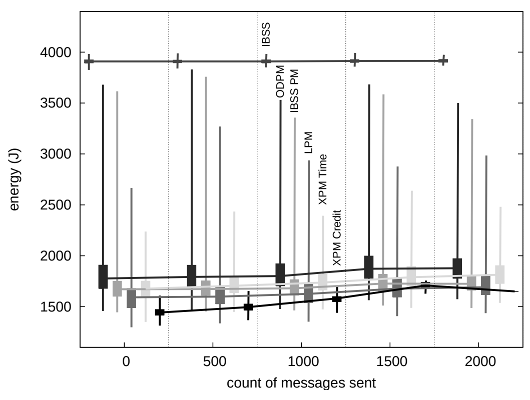


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

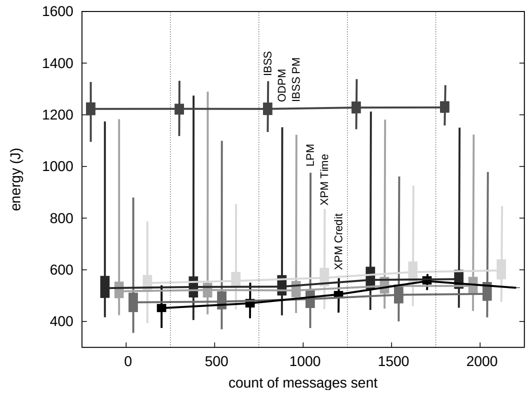
Figure 6.17: Unstructured energy quartiles for 2-flow traffic load.



(a) Feeney & Nilsson.

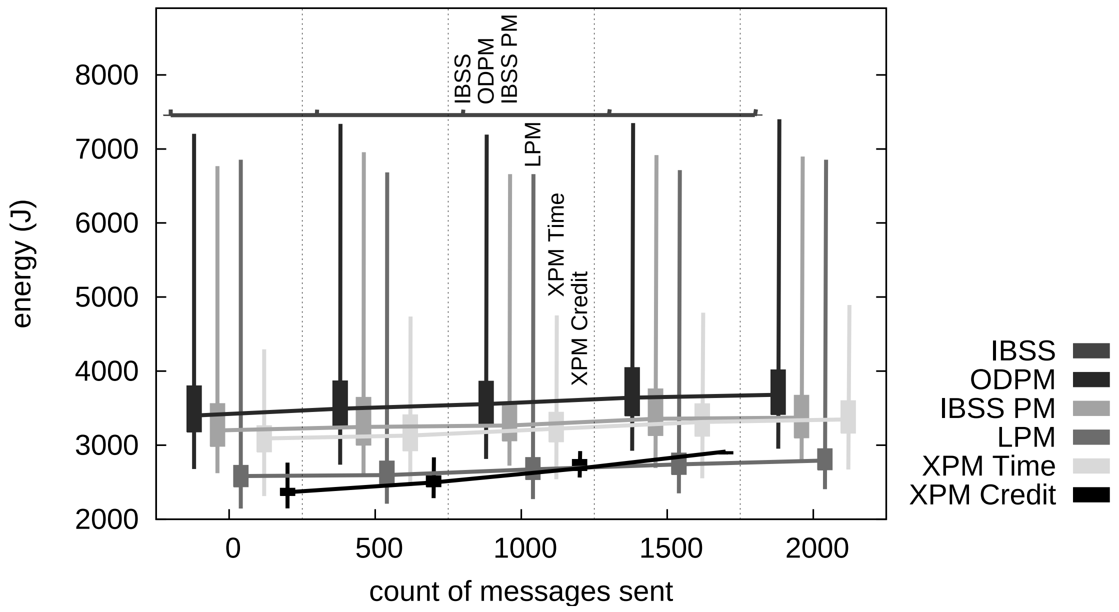


(b) $\frac{1}{2}$ idle power.

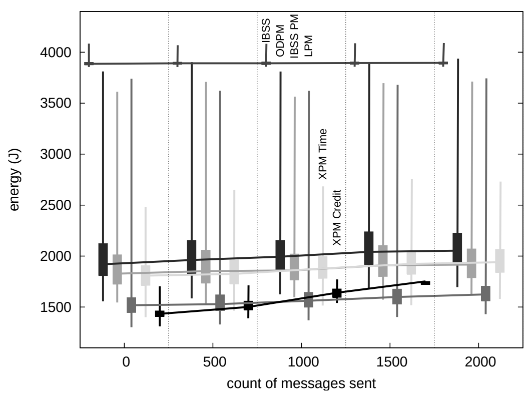


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

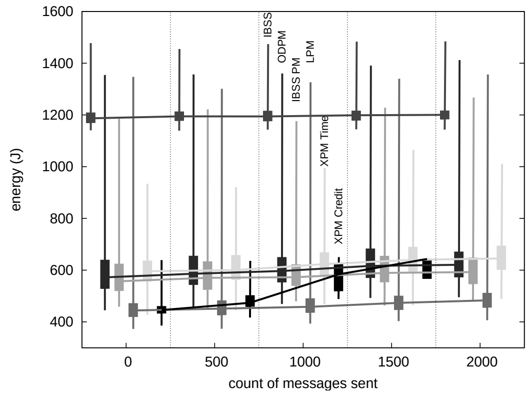
Figure 6.18: Moderately structured energy quartiles for 2-flow traffic load.



(a) Feeney & Nilsson.



(b) $\frac{1}{2}$ idle power.



(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

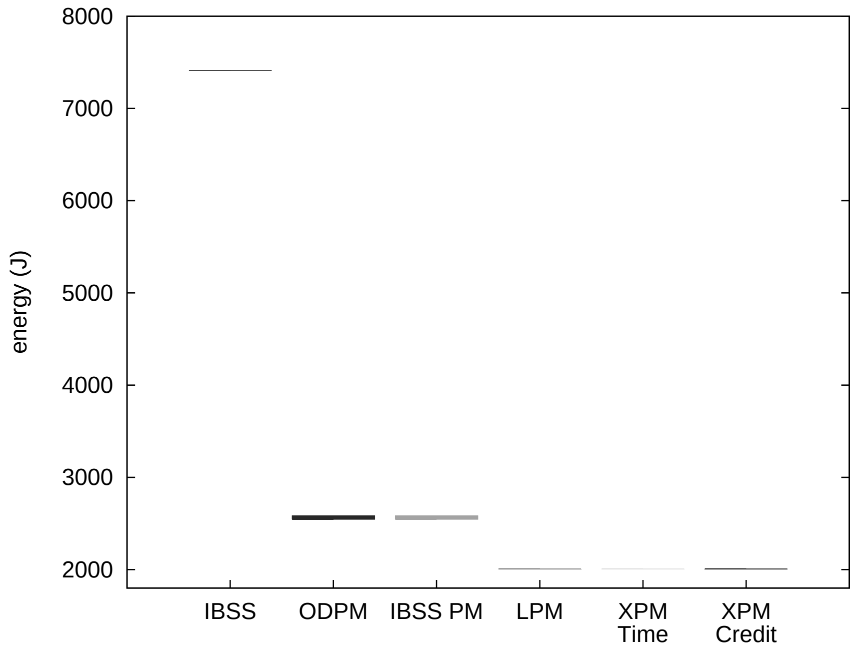
Figure 6.19: Highly-structured energy quartiles for 2-flow traffic load.

havior than ODPM or IBSS PM, also has a lower median. In most cases, XPM with the Credit Balance valuation function has the lowest median energy consumption of all the designs tested.

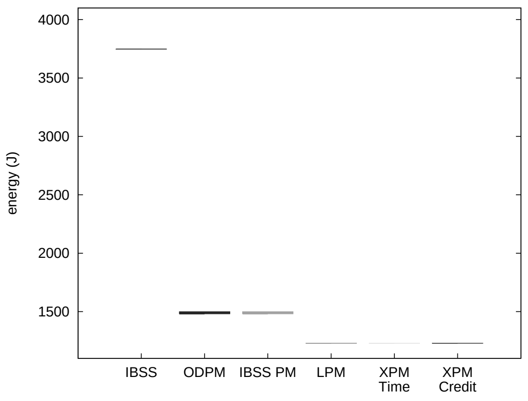
6.1.3 Per-Node Energy Mean

Sections 6.1.1 and 6.1.2 presented the **range** and **distribution** of energy consumption values experienced by individual nodes. Those measures described the **variability**, and importantly, the **worst case** for energy consumption. We now characterize the **average case** energy behavior for the same set of trials. Specifically, we present **95% confidence intervals** for the **sample mean** of per-node energy consumption. Each Figure shows the intervals computed over 150 simulation runs.

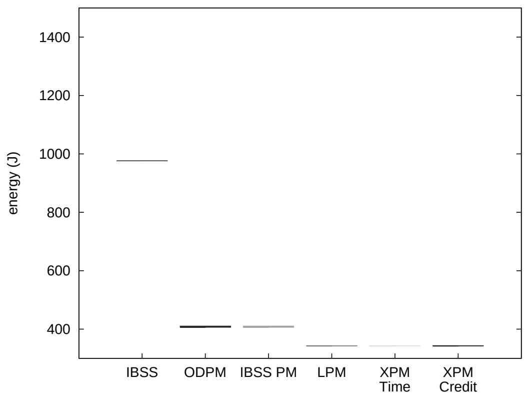
Starting with the **idle traffic load**, Figure 6.20(a) shows the mean energy consumption for the **unstructured mobility model**. The confidence intervals are tight for all five power management modes. Relative to IBSS-mode (no power management), ODPM and IBSS PM save 65% energy, and the **multihop power management** designs — LPM and XPM — save 73%. (The optimal savings, given our choice of ATIM Window and beacon interval, were shown earlier to be 73.5%; the difference is due to beacon activity.) This is a strength of an **on-demand** protocol such as DSR: when the network is idle, node energy consumption approaches $E_{\min}$. A **proactive** protocol which periodically exchanges topology information would need to **wake** nodes during these exchanges, thereby increasing their en-



(a) Feeney & Nilsson.



(b) $\frac{1}{2}$ idle power.



(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

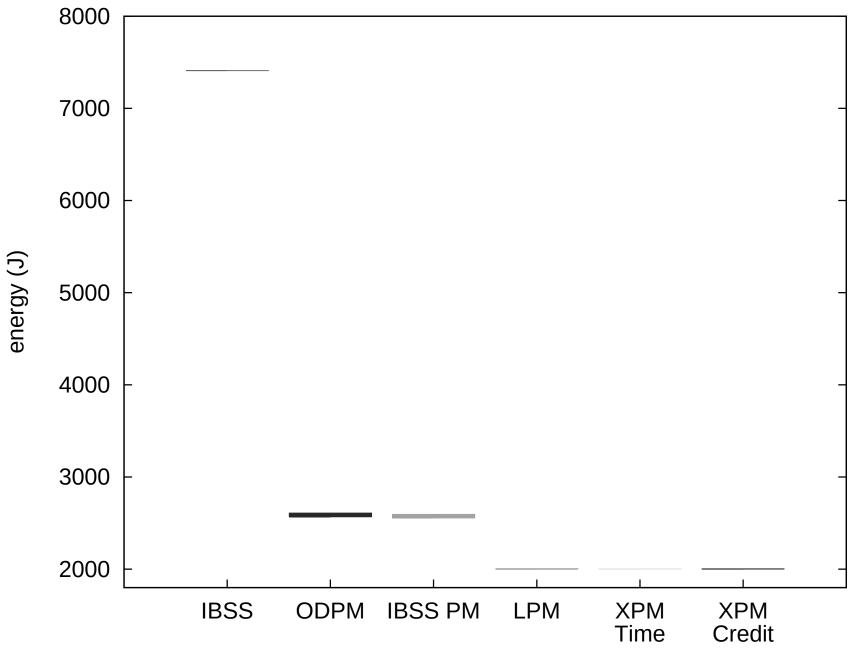
Figure 6.20: Unstructured energy mean for idle traffic load.

ergy consumption. Proactive power management designs, such as **Span** (§3.1.3.4) and **GAF** (§3.1.3.3), must also consume more energy, as a subset of nodes are awake *at all times*.

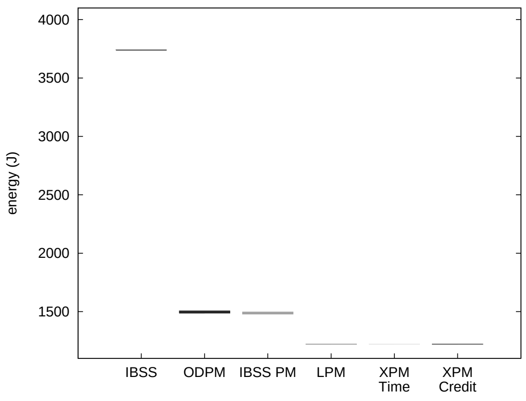
The **moderately structured** and **highly structured** workloads in Figure 6.21 and 6.22, respectively, show mean energy values that are largely unchanged. The exceptions are ODPM and IBSS PM in the highly structured mobility model (Figure 6.22(a)), which only save 58%, a 7% degradation from the less-structured cases. This is due to the **sparseness** of the highly structured model. Stations have fewer competing neighbors in the **beacon generation algorithm**, and are therefore more likely to win, thus increasing the number of **beacon intervals** in which they must stay **awake**. LPM and XPM again save a near-optimal 73%.

The following twelve Figures show confidence intervals for the sample mean in the 1-flow and 2-flow traffic loads. Because the mean for IBSS-mode is much higher than that for the other modes, we have separated the data for each set of trials into two graphs. The first presents only the IBSS-mode and IBSS PM intervals, to provide context for the power management values. The second then **re-scales** and reproduces the IBSS PM intervals alongside those for ODPM, LPM, and XPM.

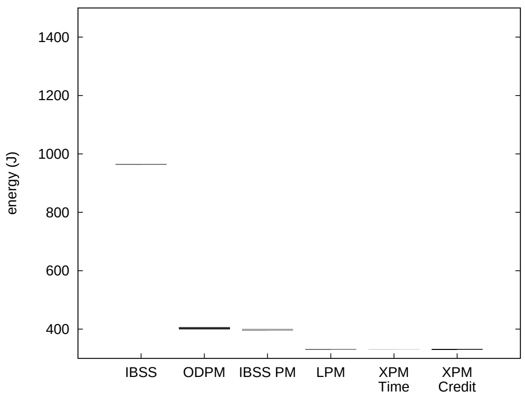
Figure 6.23(a) shows the IBSS and IBSS PM mean energy consumption for the 1-flow traffic load under unstructured mobility. In expectation, IBSS PM saves about 63% energy relative to IBSS-mode. Zooming in and adding the **multihop power management** designs, Figure 6.24(a) shows that LPM and XPM offer consistently



(a) Feeney & Nilsson.

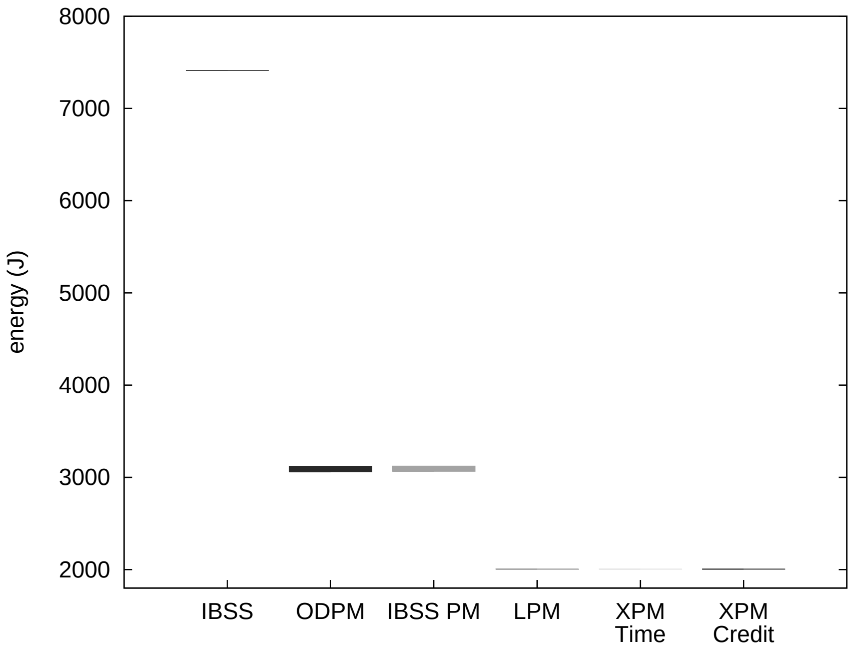


(b) $\frac{1}{2}$ idle power.

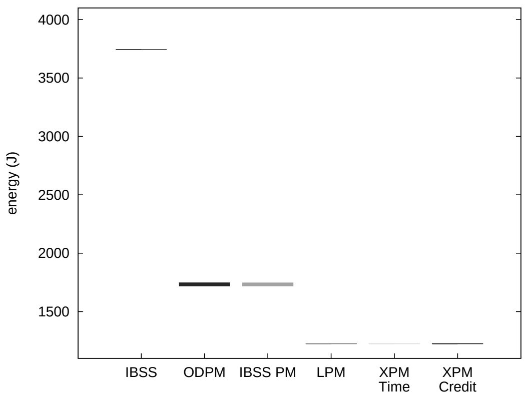


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

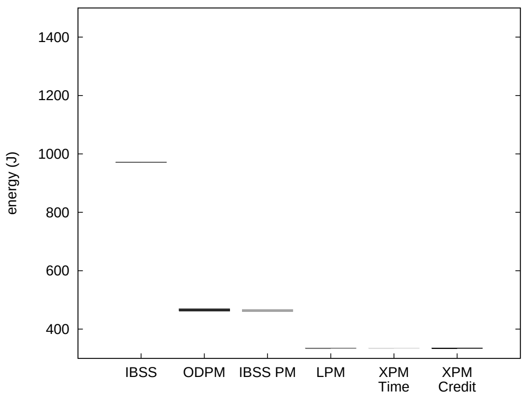
Figure 6.21: Moderately structured energy mean for idle traffic load.



(a) Feeney & Nilsson.

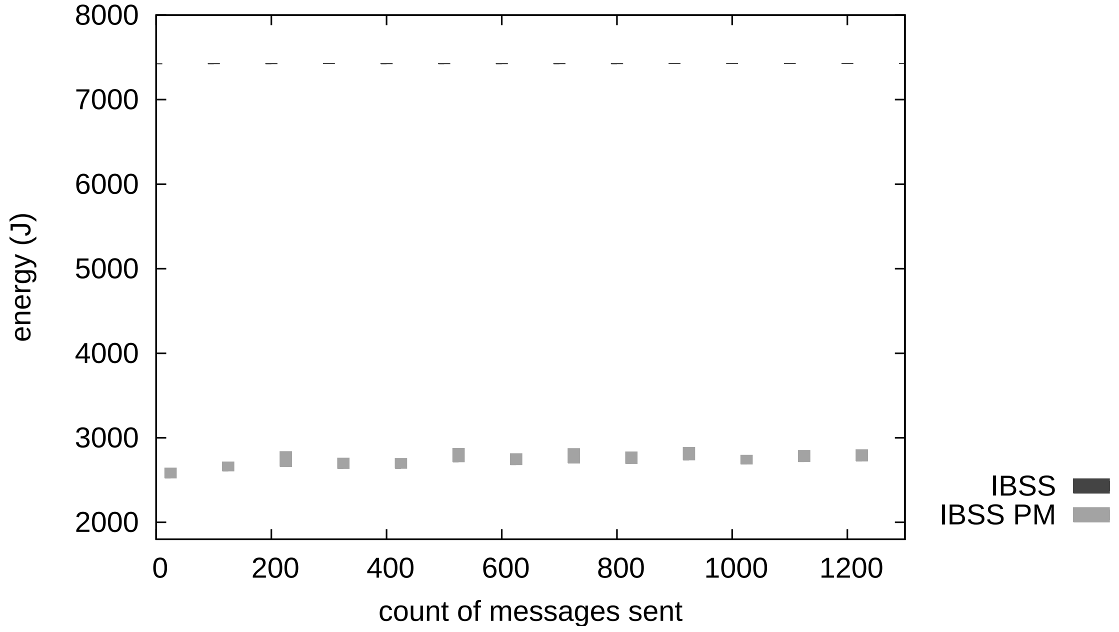


(b) $\frac{1}{2}$ idle power.

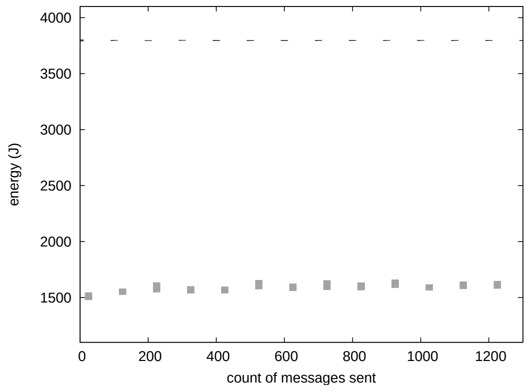


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

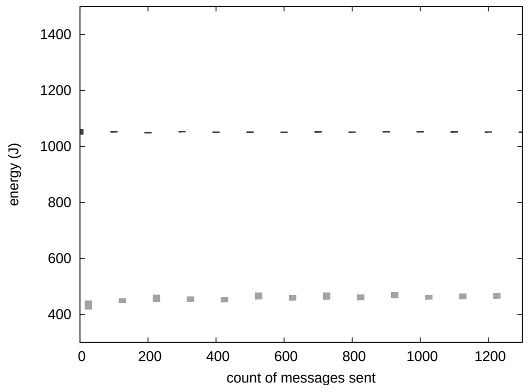
Figure 6.22: Highly structured energy mean for idle traffic load.



(a) Feeney & Nilsson.

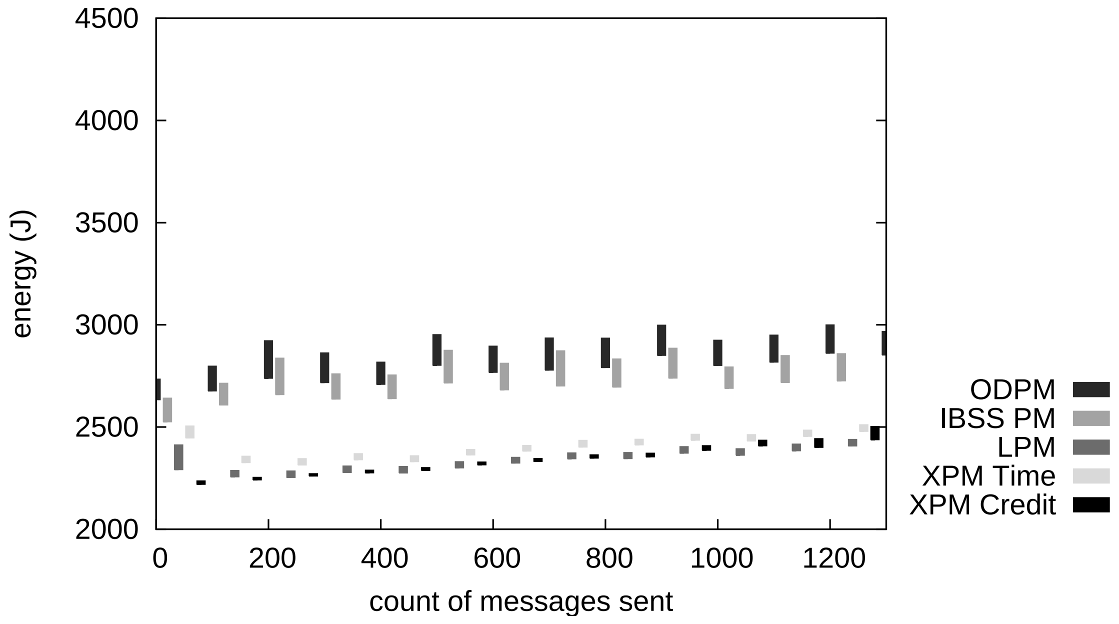


(b) $\frac{1}{2}$ idle power.

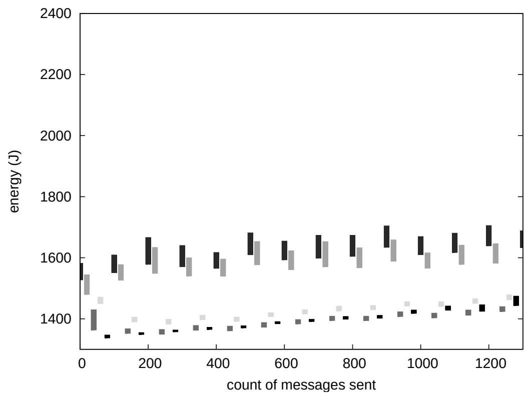


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

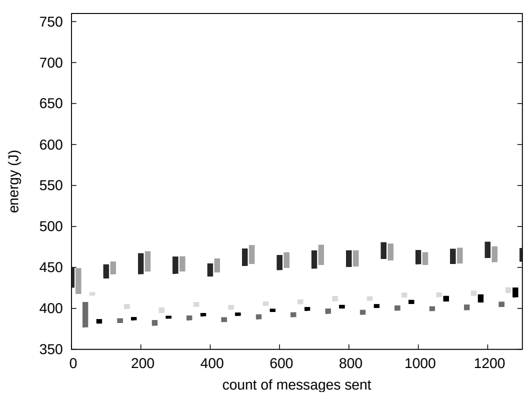
Figure 6.23: Unstructured energy mean for 1-flow traffic load.



(a) Feeney & Nilsson.



(b) $\frac{1}{2}$ idle power.



(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

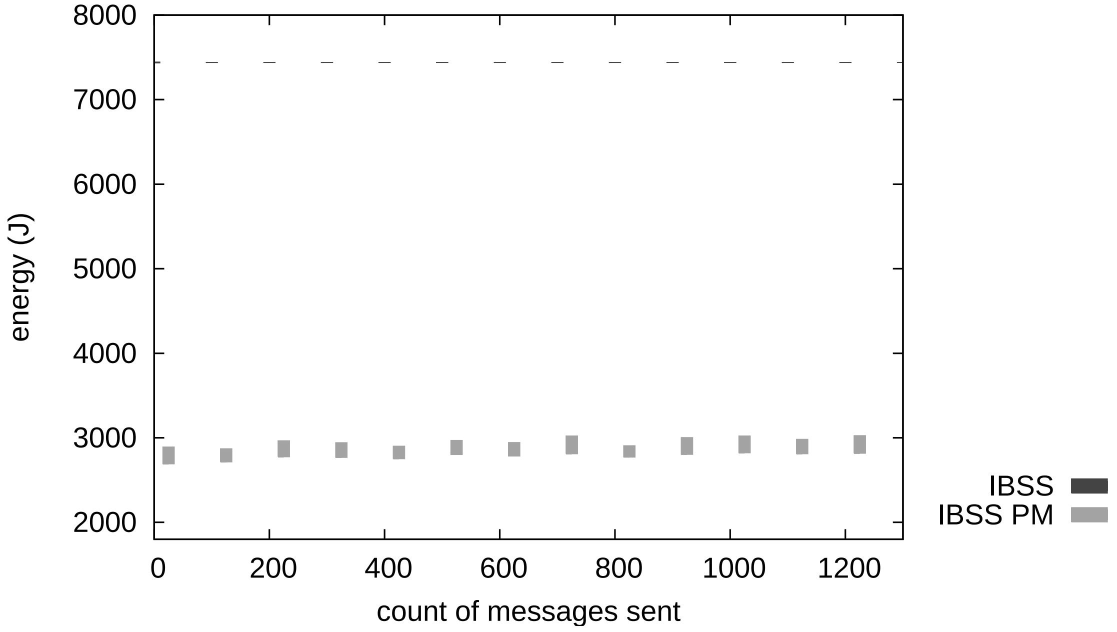
Figure 6.24: Unstructured energy mean for 1-flow traffic load (continued).

better average-case performance than IBSS PM. LPM and XPM typically consume about 400J *less* than IBSS PM. Relative to the case of *no* power management, LPM and XPM Time save 68% energy, while XPM Credit saves 69%.

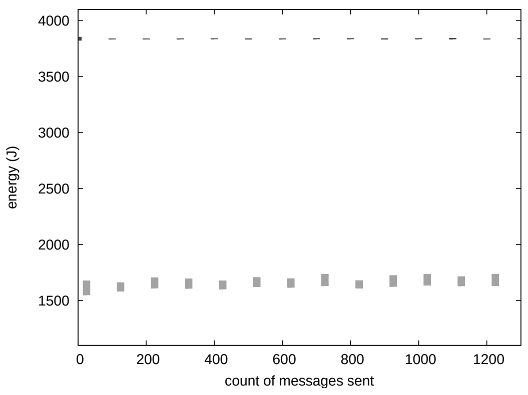
As with the energy range, the average case behavior for ODPM is similar to that of IBSS PM. We observe that the ODPM means have a shape similar to those of IBSS PM, but at a higher magnitude. Again, this is because ODPM uses timers to keep stations “awake” for longer durations following frame activity than IBSS PM would. Because of this similarity, and the fact that ODPM does not provide better average-case energy savings than IBSS PM, we will not separately describe the ODPM data in the text.

Proceeding to the moderately structured mobility model in Figure 6.25(a), the IBSS mean has not changed, while the IBSS PM mean has increased by about 125J. Zooming in, Figure 6.26(a) shows that LPM and XPM still outperform IBSS PM by 240–530J, on average. Compared with IBSS-mode, IBSS PM saves 61% energy, LPM saves 66%, XPM Time saves 64%, and XPM Credit saves 68%.

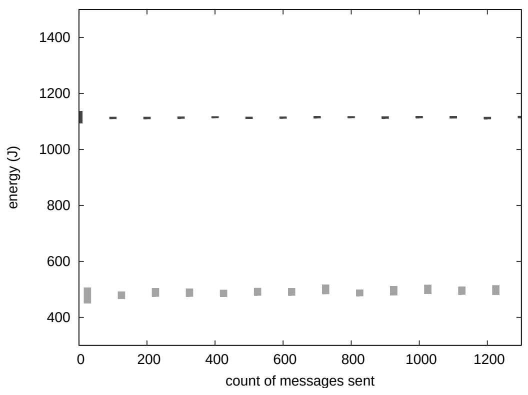
Figure 6.27(a) shows the average case energy for the 1-flow traffic load under the highly structured mobility model. Relative to the moderately structured model, IBSS-mode is unchanged, and IBSS PM has increased by an additional 470J. IBSS PM savings are 55%, on average. The rescaled Figure 6.28(a) shows that, again, both LPM and XPM yield better performance than IBSS PM. XPM Time consumes 412 fewer Joules on average, while XPM Credit saves 1,012J. Compared to



(a) Feeney & Nilsson.

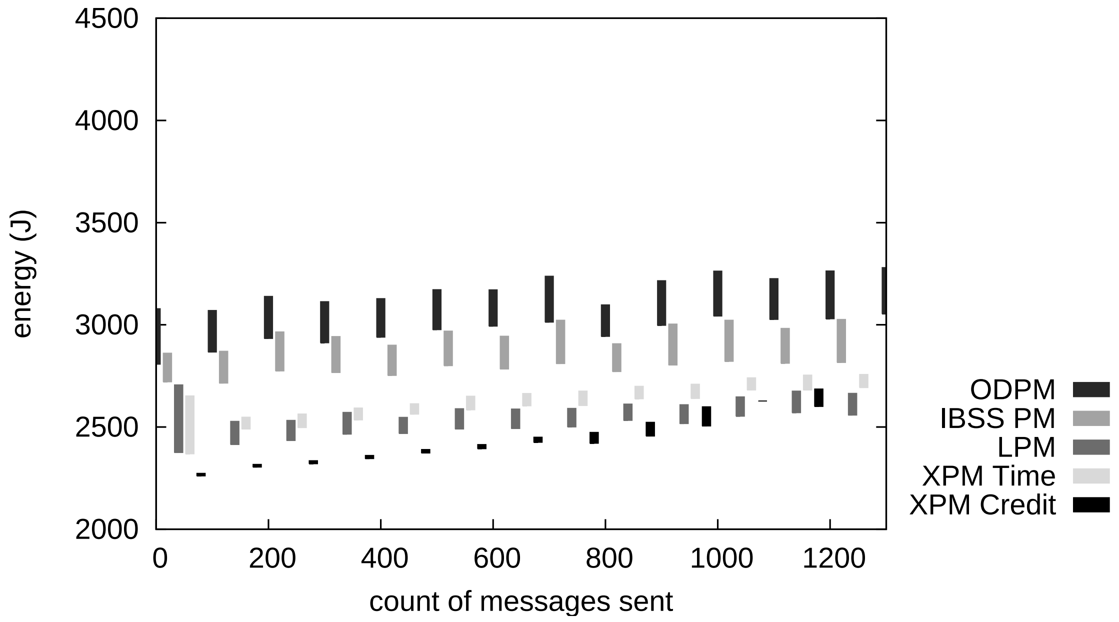


(b) $\frac{1}{2}$ idle power.

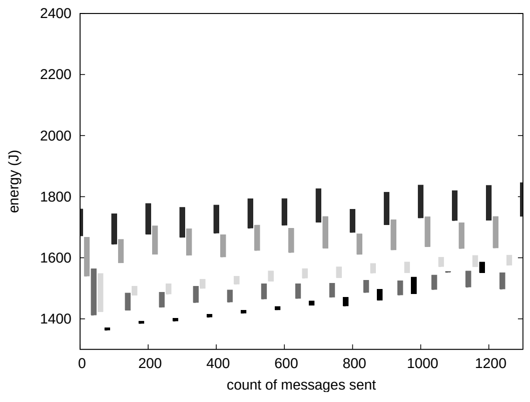


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

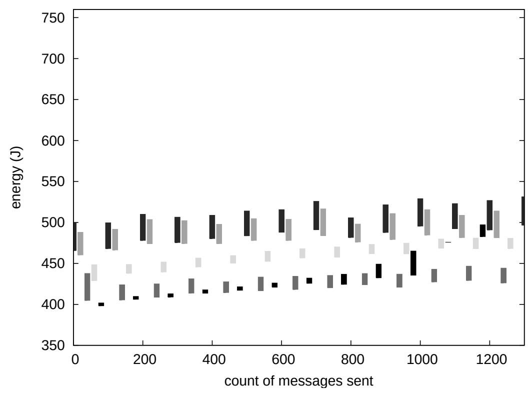
Figure 6.25: Moderately structured energy mean for 1-flow traffic load.



(a) Feeney & Nilsson.

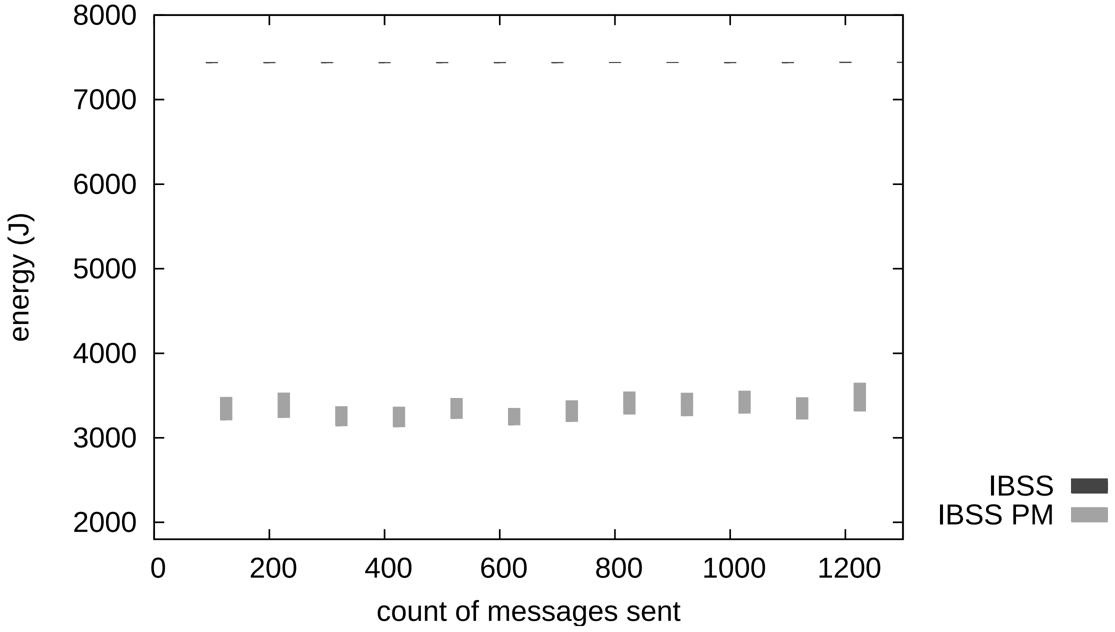


(b) $\frac{1}{2}$ idle power.

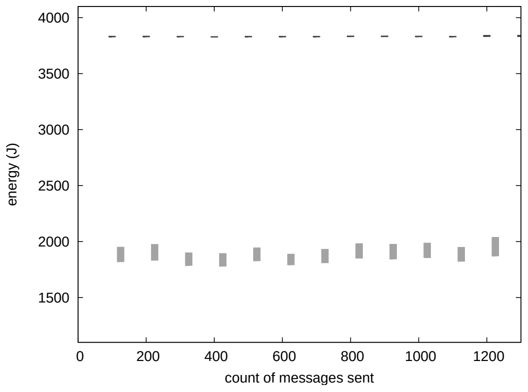


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

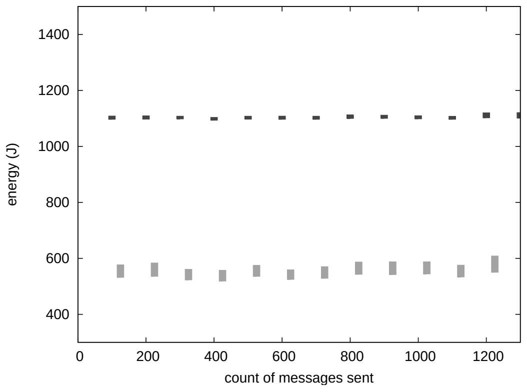
Figure 6.26: Moderately structured energy mean for 1-flow traffic load (continued).



(a) Feeney & Nilsson.

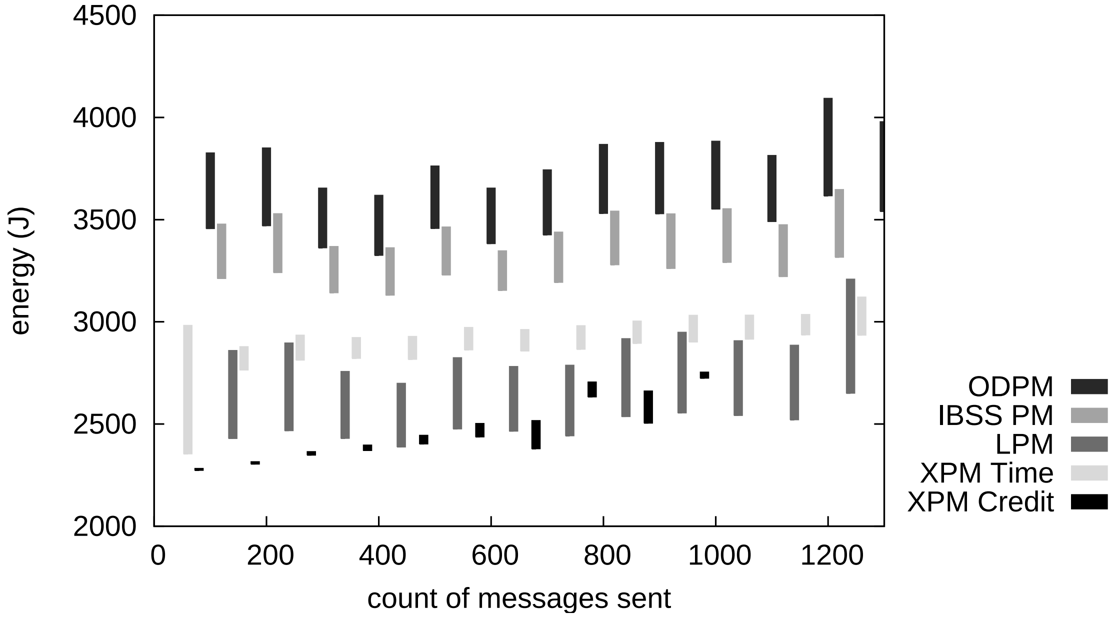


(b) $\frac{1}{2}$ idle power.

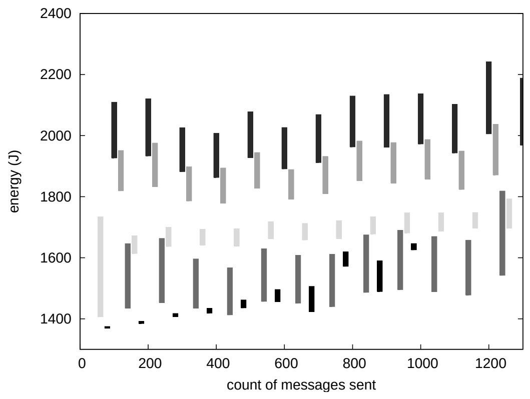


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

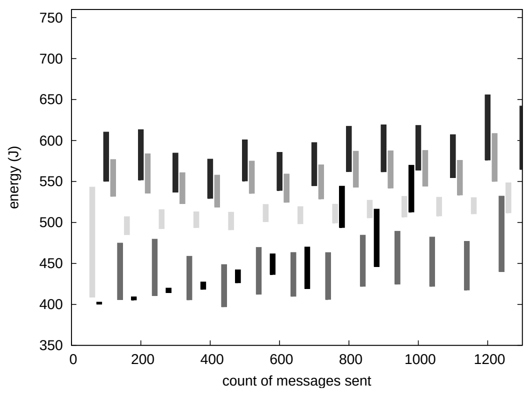
Figure 6.27: Highly structured energy mean for 1-flow traffic load.



(a) Feeney & Nilsson.



(b) $\frac{1}{2}$ idle power.



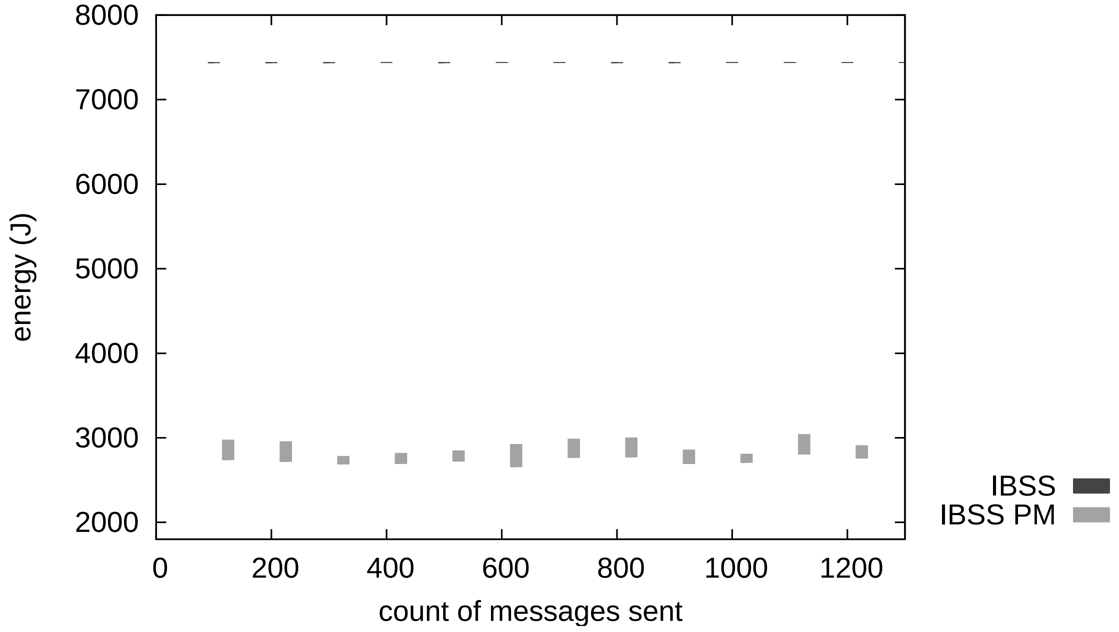
(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

Figure 6.28: Highly structured energy mean for 1-flow traffic load (continued).

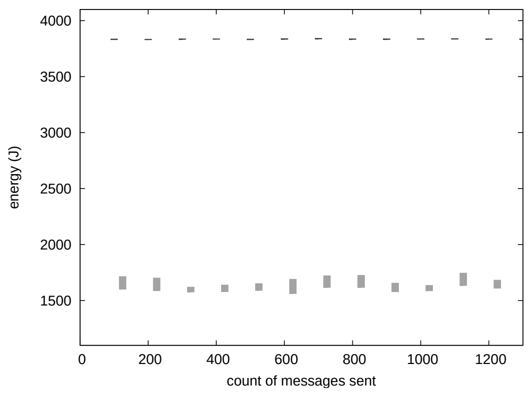
IBSS-mode (no power management) LPM provides a 64% improvement in energy consumption, with improvements of 61% and 69% for XPM Time and XPM Credit. Also, note that the confidence intervals for XPM are visibly **tighter** than those for IBSS PM or even LPM. This is a consequence of the decreased **energy variability** described in Section 6.1.1.

The 2-flow, unstructured mobility scenario in Figure 6.29(a) repeats the pattern: IBSS PM saves 61% energy relative to IBSS-mode, LPM saves 66%, XPM Time saves 65%, and XPM Credit saves 68%. We note that the confidence intervals for XPM Credit increase in size for some of the larger buckets. This is due to the small number of samples in these buckets; for example, the 2,000-message bucket only holds two samples, while most other buckets contain 40–120 samples. When a traffic source cannot **afford** some of its routes, as can happen with XPM Credit, it is less likely to appear in one of the high-count buckets.

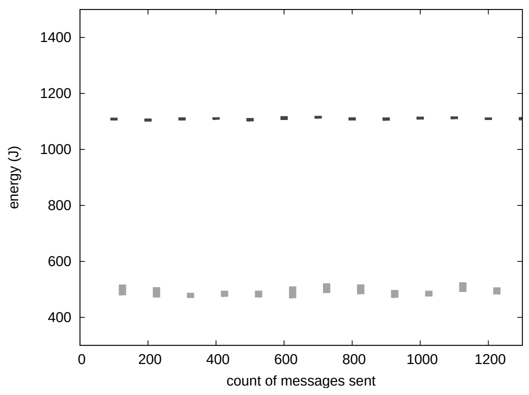
The moderately structured mobility model for the 2-flow traffic load, shown in Figure 6.31(a), shows that IBSS PM provides 60% average savings relative to IBSS PM. Figure 6.32(a) shows that LPM and XPM Time are generally comparable to, but not worse than, IBSS PM, with LPM offering 61% savings and XPM Time 60%. Perhaps most interesting is XPM Credit — averaging 67% savings — which exhibits a clear trend towards increasing **energy** with a node's own **communications activity**. This indicates that **Credit Balance** valuation is able to save energy for nodes that don't source many messages.



(a) Feeney & Nilsson.

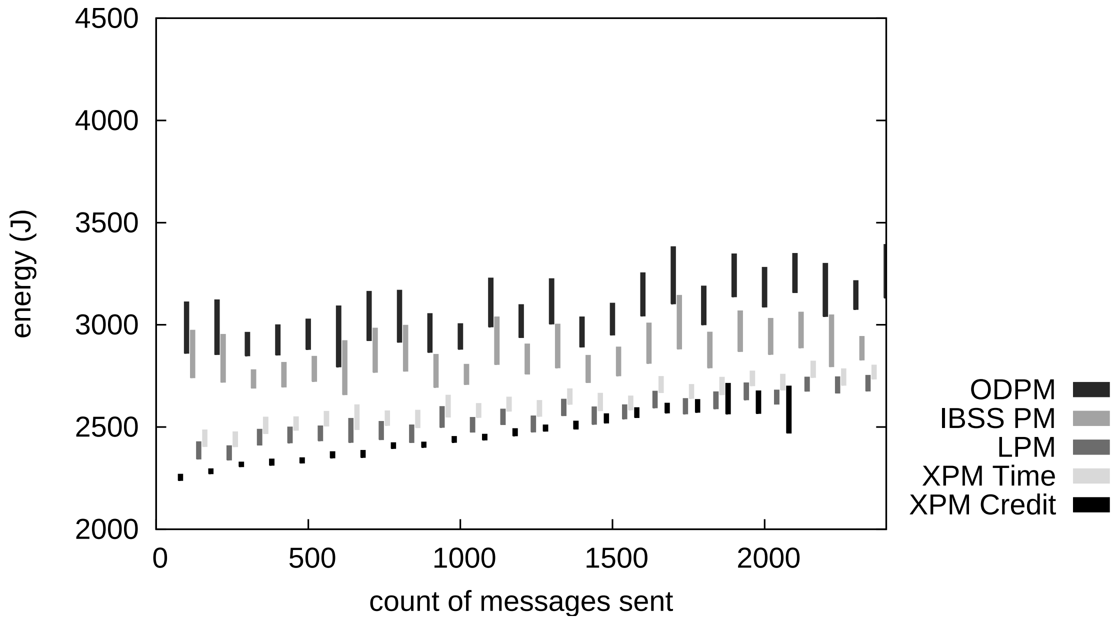


(b) $\frac{1}{2}$ idle power.

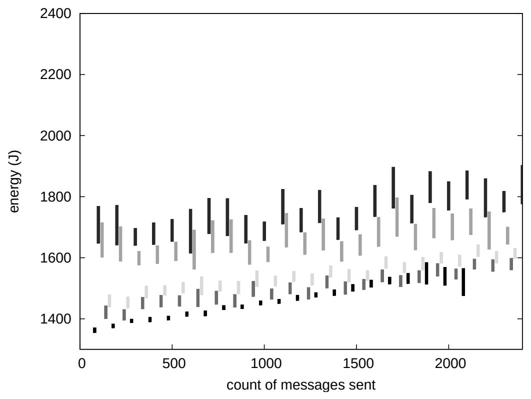


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

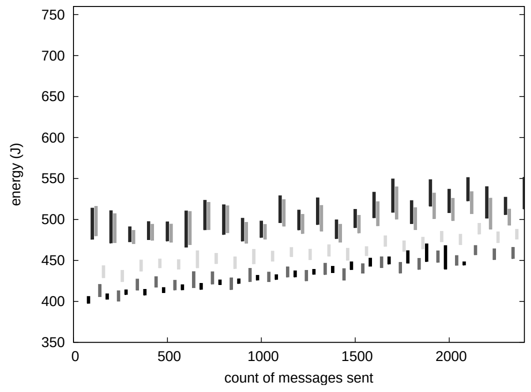
Figure 6.29: Unstructured energy mean for 2-flow traffic load.



(a) Feeney & Nilsson.

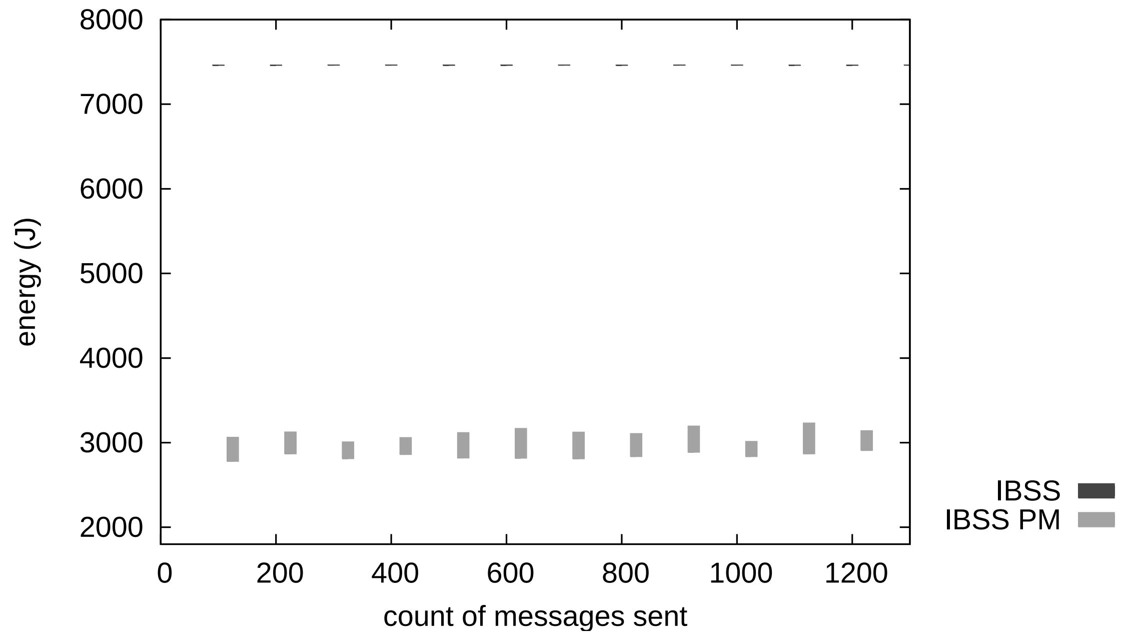


(b) $\frac{1}{2}$ idle power.

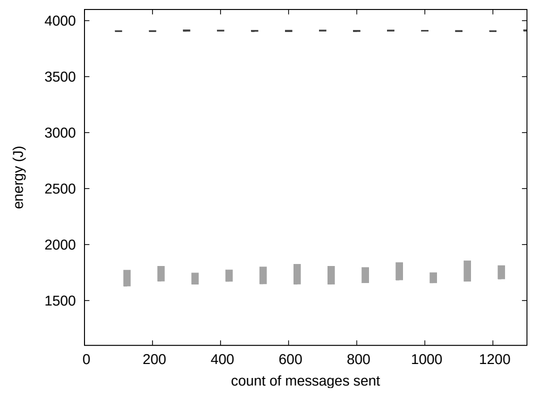


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

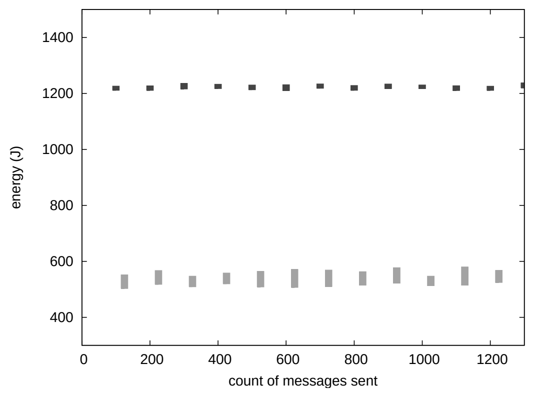
Figure 6.30: Unstructured energy mean for 2-flow traffic load (continued).



(a) Feeney & Nilsson.

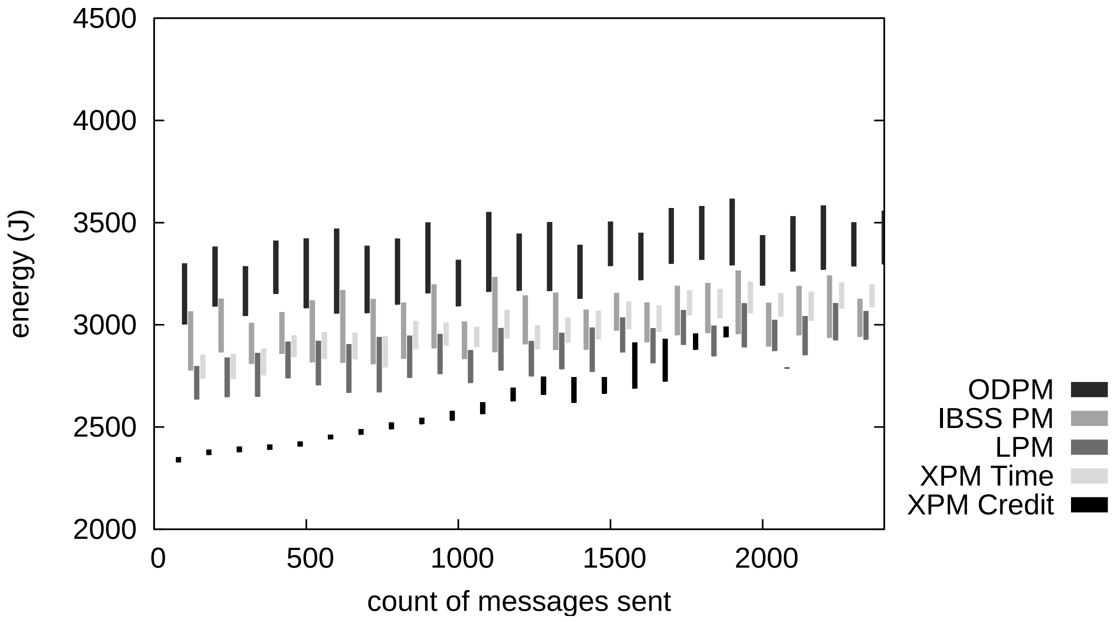


(b) $\frac{1}{2}$ idle power.

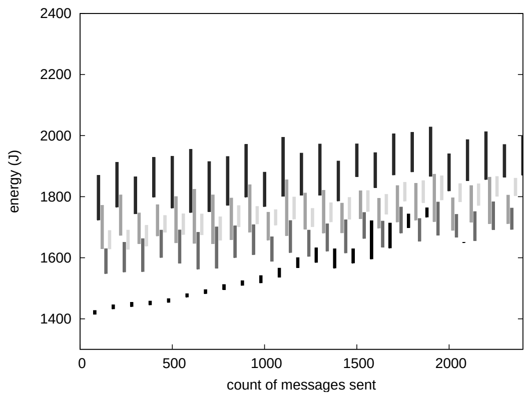


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

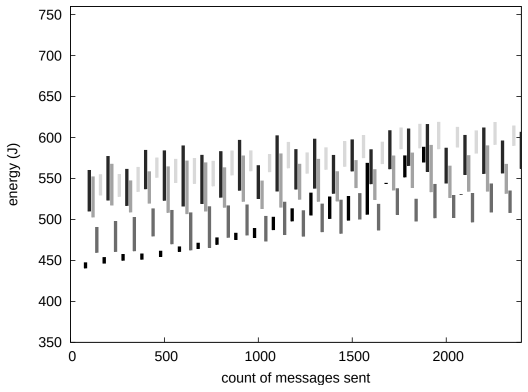
Figure 6.31: Moderately structured energy mean for 2-flow traffic load.



(a) Feeney & Nilsson.

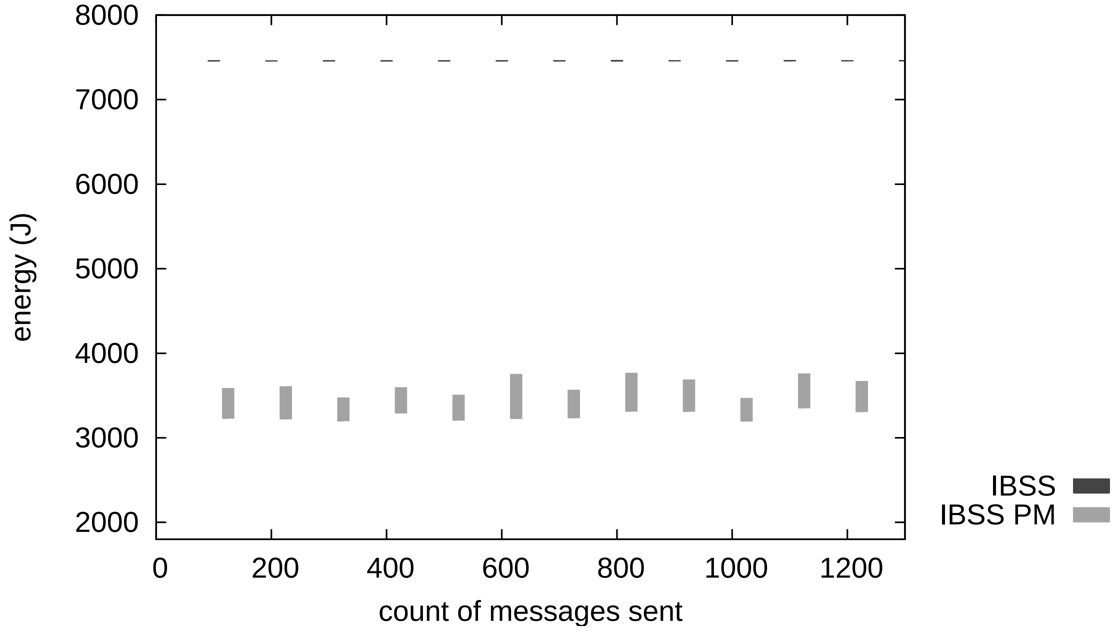


(b) $\frac{1}{2}$ idle power.

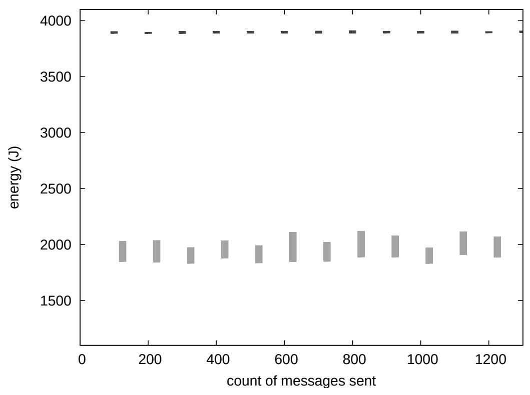


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

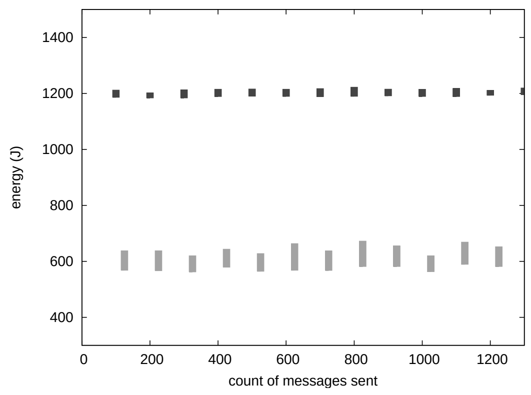
Figure 6.32: Moderately structured energy mean for 2-flow traffic load (continued).



(a) Feeney & Nilsson.

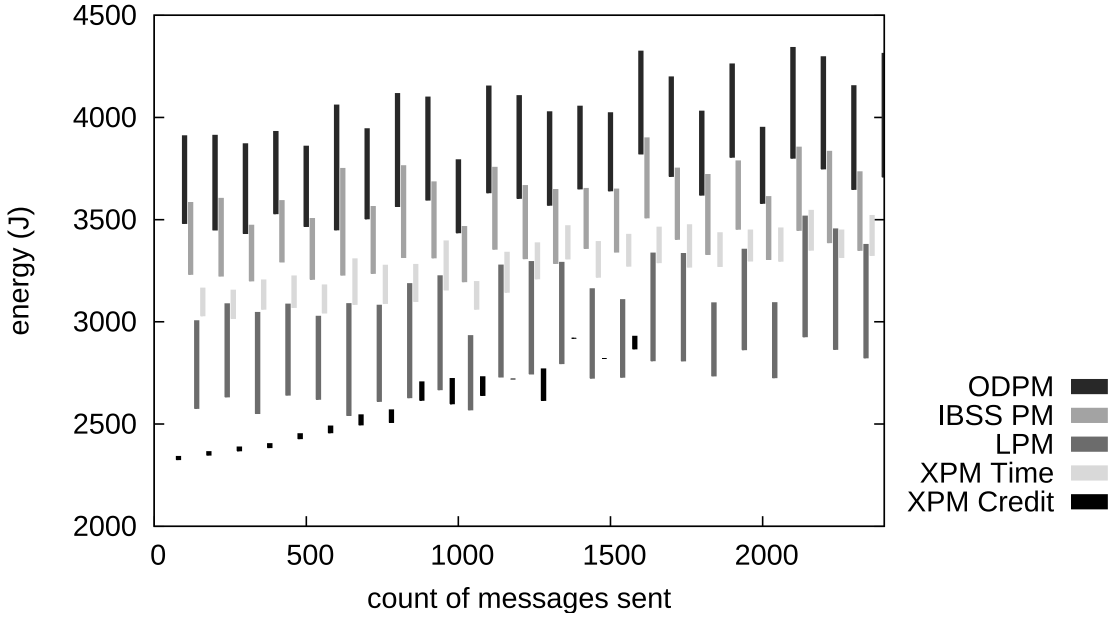


(b) $\frac{1}{2}$ idle power.

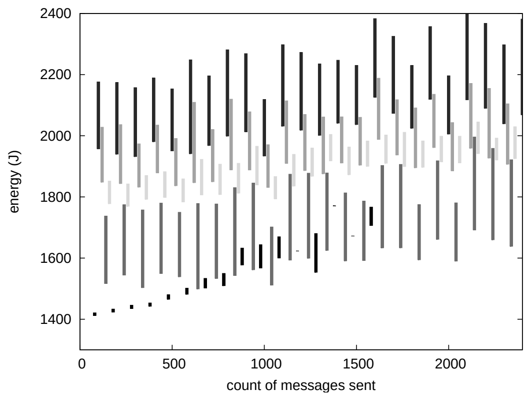


(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

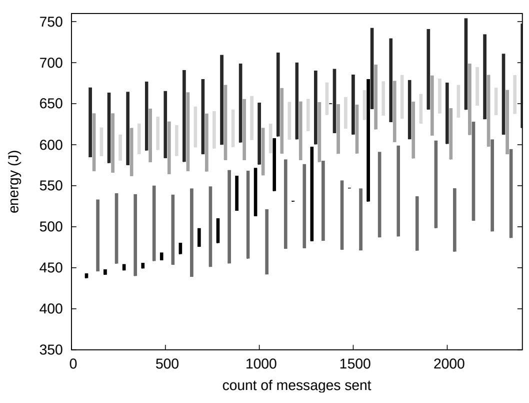
Figure 6.33: Highly structured energy mean for 2-flow traffic load.



(a) Feeney & Nilsson.



(b) $\frac{1}{2}$ idle power.



(c) $\frac{1}{10}$ idle power, $\frac{1}{100}$ doze power.

Figure 6.34: Highly structured energy mean for 2-flow traffic load (continued).

Finally, Figure 6.33(a) shows the 2-flow load with highly structured mobility. IBSS PM provides 53% average energy savings compared with IBSS-mode. In Figure 6.34(a), LPM and XPM are consistently comparable to, or better than, IBSS PM. The average savings are 60%, 56%, and 68% for LPM, XPM Time, and XPM Credit, respectively.

In summary, LPM and XPM deliver **average-case energy consumption** that is at least as good as 802.11 IBSS power management. In many cases, the mean energy consumption provided by LPM or XPM is hundreds of Joules lower than IBSS PM. This is especially significant for XPM, as the ability to **limit** the worst case energy consumption (§6.1.1) does *not* come at the expense of the average case.

Mobility	**ODPM**	**IBSS PM**	**LPM**	**XPM Time**	**XPM Credit**
Unstructured	61%	63%	68%	68%	69%
Moderately Structured	59%	61%	66%	64%	68%
Highly structured	51%	55%	64%	61%	69%

(a) 1-flow traffic.

Mobility	**ODPM**	**IBSS PM**	**LPM**	**XPM Time**	**XPM Credit**
Unstructured	59%	61%	66%	65%	68%
Moderately Structured	55%	60%	61%	60%	67%
Highly structured	49%	53%	60%	56%	68%

(b) 2-flow traffic.

Table 6.2: Average energy savings relative to no power management.

Table 6.2 reproduces the average-case energy savings presented in this Section.

In all cases, LPM and XPM offer greater savings, relative to the case of *no* power management, than 802.11 IBSS power management. We can now amend Table 3.1 from Chapter 3 by adding our claims about energy performance improvement with LPM and XPM. Table 6.3 shows the revised summary.

Name	**Energy Performance**
BECA	35–40% energy savings, 20% network lifetime extension
AFECA	40–45% energy savings, 55% network lifetime extension
GAF	40–60% energy savings, 200–300% network lifetime extension
Span	200–250% network lifetime extension
On-demand	49–61% energy savings (claimed 50%)
LPM	61–68% energy savings
XPM Time	56–68% energy savings, reduced energy range by 2×–5.2×
XPM Credit	67–69% energy savings, reduced energy range by 6.5×–12.3×

Table 6.3: Augmented power management summary: energy performance.

Many differences exist between the evaluation environments used by the authors of the various designs. These differences include — but are not limited to — radio power models, 802.11 power management implementation and parameters, mobility models, and traffic loads. We do not read too much into the differences between our average case savings and those of earlier authors. Instead, we observe that XPM provides **expected energy savings** comparable to existing designs. *In addition*, XPM provides the **energy-shaping** capability described in the previous Sections, and the **low-latency** communications performance presented in the next.

6.2 Latency

Section 2.2.2 presented several problems with the interaction between **on-demand source routing networks** and power management. The **latency** of the **Route Discovery** process in the worst case was shown to be a linear function of the **route length** and the 802.11 **beacon interval**. Once a route was discovered, we showed that the worst case **application message delivery latency** was also linear in the route length and beacon interval. This Section presents measurements of both Route Discovery and delivery latency for the five power management modes.

6.2.1 Route Discovery Latency

The Route Discovery latency problem is an instance of a general design issue for **on-demand**, or **reactive**, network protocols. On-demand designs **defer** work until it is actually needed. This contrasts with **proactive** designs that perform work in anticipation of future need; when this need is unrealized, the work is **wasted**. On-demand designs can be less wasteful, but carry the cost of **setup latency**. For example, when a new route is needed, a proactive protocol might already know it, but a reactive protocol typically must discover it.

The **multihop power management architecture** uses the **fast wakeup** technique (§4.1.2) to improve Route Discovery latency. Fast wakeup accelerates the transition of power managing stations to a low-latency state. The goal is to enable the Route Discovery process to complete in just one or two **beacon intervals**.

A DSR traffic source may generate multiple **Route Request** messages while trying to discover a route. For example, if the first Request does not produce results in some amount of time, a second Request is sent. Our measurements mark the elapsed time from the generation of the *first* Request to the moment the source learns the first discovered route. We chose this method rather than trying to determine which of possibly many Requests triggered the first Route Reply. Also, this measure better reflects what sources care about: from the time a route is first needed, how long does it take to discover one?

Mobility	**IBSS**	**ODPM**	**IBSS PM**	**LPM**	**XPM Time**	**XPM Credit**
Unstructured	45.0ms ±1.78ms	1,278ms ±81.4ms	1,077ms ±31.2ms	163ms ±34.9ms	162ms ±1.96ms	165ms ±2.28ms
Moderately Structured	62.7ms ±19.5ms	1,037ms ±35.1ms	2,171ms ±71.1ms	164ms ±20.6ms	172ms ±16.6ms	184ms ±17.0ms
Highly Structured	71.6ms ±6.29ms	1,748ms ±57.7ms	3,210ms ±85.1ms	206ms ±17.9ms	200ms ±7.11ms	245ms ±15.3ms

Table 6.4: Mean Route Discovery latency for 1-flow traffic load.

Table 6.4 shows the effect of fast wakeup on Route Discovery latency for the three mobility models under the 1-flow traffic load. Each Table cell shows the sample mean over 30 simulation runs, along with the 95% confidence interval. The first feature to note is that the use of power management (IBSS PM) results in a discovery latency that is 24 to 45 *times* that of IBSS-mode.

Even ODPM, which was designed to improve (average case) latency, experiences high Route Discovery latency comparable to IBSS PM. This is because the propagation of DSR Route Request messages occurs in exactly the same manner as in IBSS PM. In some cases, the return of Route Reply messages can be faster than under IBSS PM. This can happen when a node forwards the Reply to a neighbor which is already known to be suspending power management. For example, such a neighbor might have recently been active, supporting some traffic flow, and therefore appears in the forwarding node's **suspending neighbor list**. In such cases, the forwarding node need not wait for the ATIM process, and can send the Route Reply immediately. Although no setup latency statistics for ODPM have been published, Figure 7 of (Zheng and Kravets, 2003) shows a 1.8-second setup latency for a 3-hop route, which is worse than the average case measurements we have collected.

The use of fast wakeup in LPM and XPM reduces discovery latency by an order of magnitude. For the unstructured and moderately structured mobility models, Route Discovery typically requires only a single beacon interval. Relative to IBSS PM, fast wakeup improves discovery latency in these models by factors of 6 and 12–13, respectively. The highly structured mobility model is a more challenging environment, with **longer routes** in a **sparser topology**; it completes discovery in two beacon intervals. Discovery latency improves by factors of 13–16 relative to IBSS PM.

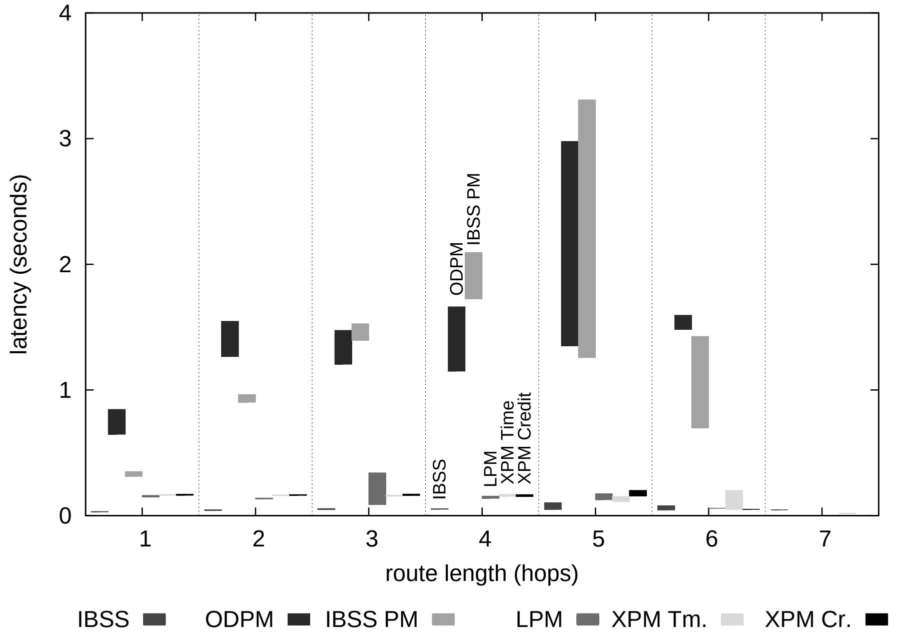
A few of the cells in Table 6.4 show large confidence intervals for the sample mean. LPM in the unstructured mobility model and IBSS-mode under moderately structured mobility are examples. This appears to be caused by high variability in the discovery latency for a specific set of routes. In the unstructured LPM data, routes of length 3 exhibit higher variation, while in the moderately structured IBSS case, the larger spread is seen in routes of length 4. We believe these characteristics are peculiarities of the workloads themselves.

Mobility	**IBSS**	**ODPM**	**IBSS PM**	**LPM**	**XPM Time**	**XPM Credit**
Unstructured	58.8ms ±4.38ms	1,039ms ±64.2ms	1,062ms ±31.1ms	160ms ±4.66ms	220ms ±13.1ms	233ms ±21.5ms
Moderately Structured	91.6ms ±22.8ms	930ms ±28.5ms	2,104ms ±56.3ms	193ms ±24.1ms	230ms ±17.5ms	253ms ±18.9ms
Highly Structured	83.6ms ±5.12ms	1,507ms ±38.6ms	2,807ms ±52.1ms	240ms ±26.2ms	410ms ±34.0ms	513ms ±41.6ms

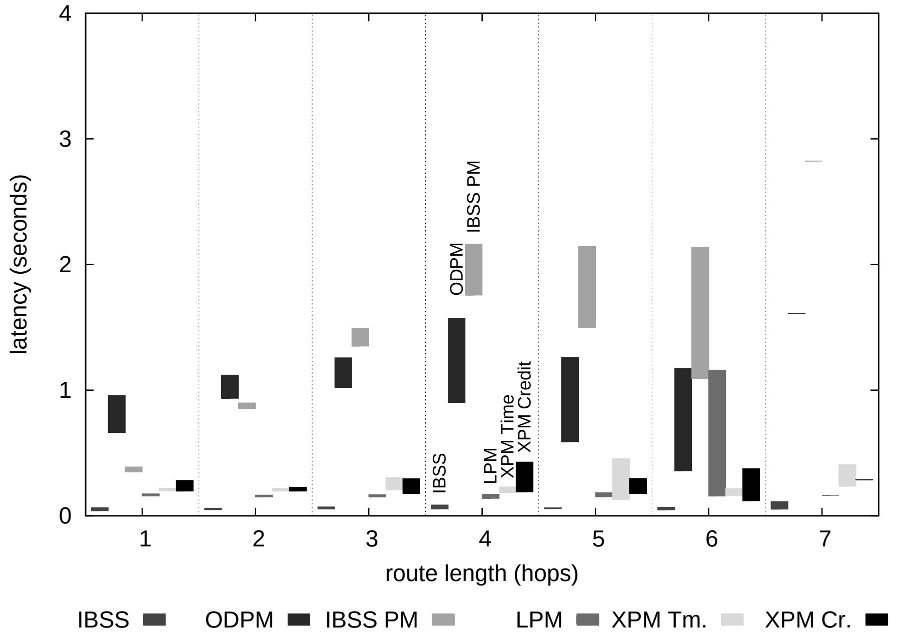
Table 6.5: Mean Route Discovery latency for 2-flow traffic load.

Route Discovery performance for the 2-flow traffic load is shown in Table 6.5. IBSS PM again has much higher latency than IBSS-mode without power management, requiring 18 to 34 times the delay. The additional traffic flow has degraded the performance of **fast wakeup** somewhat, especially for XPM. Still, there is a significant improvement over IBSS PM, by factors of 4–8 in the worst case (XPM Credit), and 7–12 in the best case (LPM).

Figures 6.35, 6.36, and 6.37 show Route Discovery latency as a function of **route length** ($|P| - 1$). The graphs illustrate 95% confidence intervals for the sample mean in each bucket. This representation shows that the latency performance of LPM and XPM is much closer to that of IBSS-mode than IBSS PM. In each graph, we have delineated the buckets using dotted lines, and have individually labeled the data in a single bucket to improve readability.

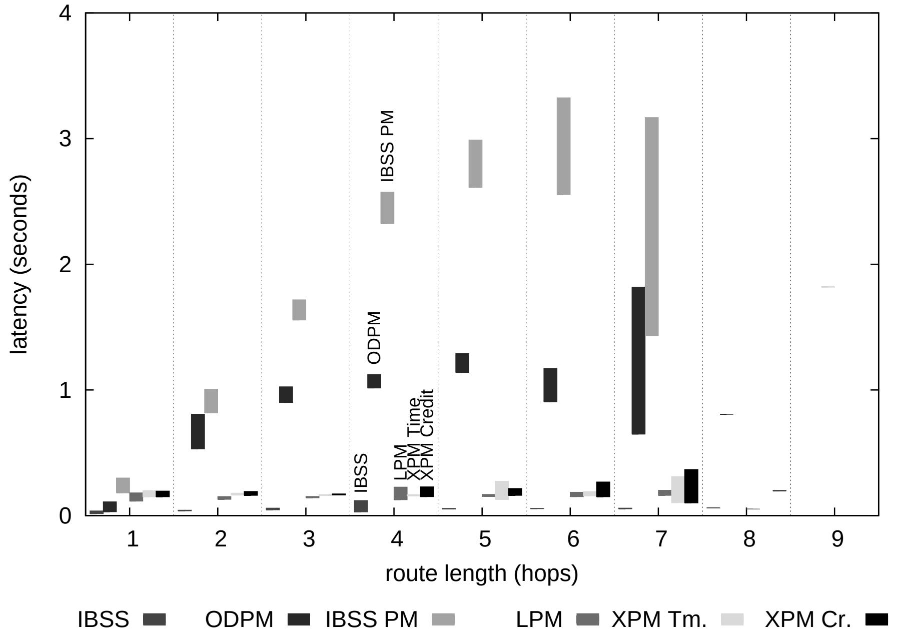


(a) 1-flow traffic load.

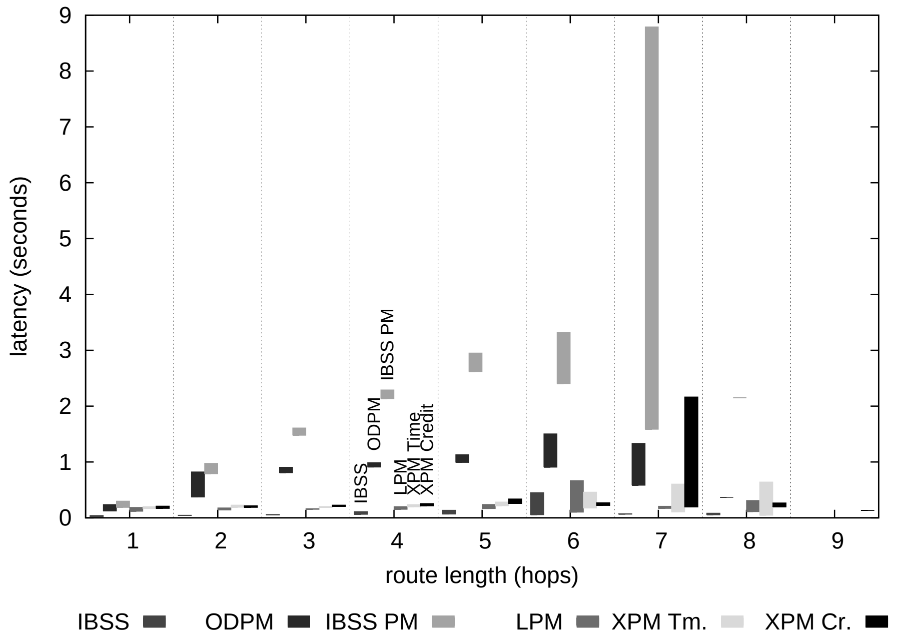


(b) 2-flow traffic load.

Figure 6.35: Unstructured Route Discovery latency means.

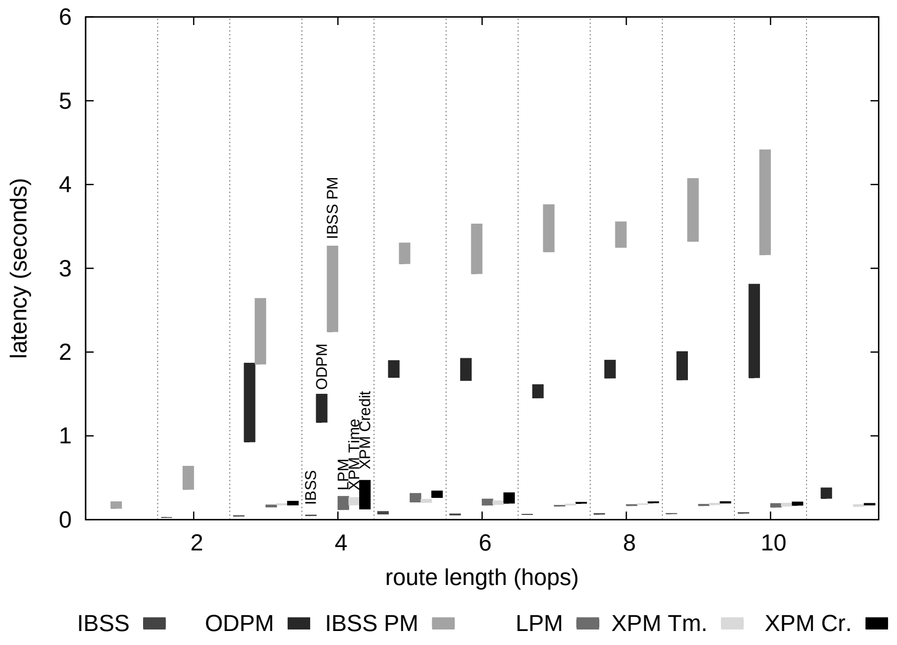


(a) 1-flow traffic load.

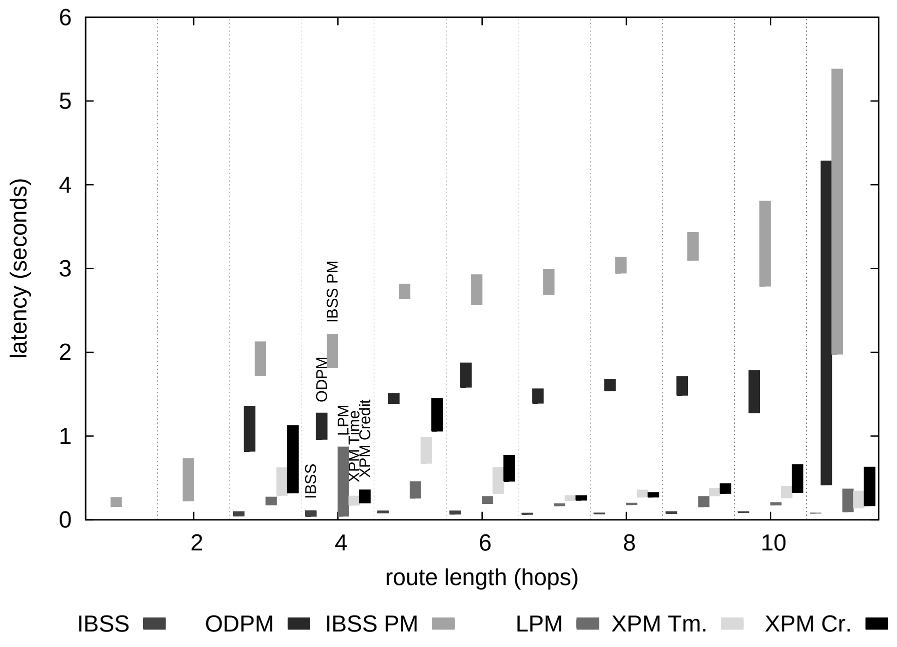


(b) 2-flow traffic load.

Figure 6.36: Moderately structured Route Discovery latency means.



(a) 1-flow traffic load.



(b) 2-flow traffic load.

Figure 6.37: Highly structured Route Discovery latency means.

The **fast wakeup** technique, used by LPM and XPM, provides a clear performance advantage over the 802.11 IBSS power management mode when measuring the latency of DSR Route Discovery. We have shown latency improvements by factors of 4 to 16, resulting in performance much closer to the case of *no* power management than that of IBSS PM. This technique permits DSR networks to enjoy the energy-saving benefits of power management without incurring the worst-case **setup delay**.

6.2.2 Application Message Delivery Latency

The second problem with the interaction between 802.11 power management and DSR concerns multihop **application message delivery latency**. That is, the delay between when a message leaves the application layer at the source and when it reaches the application layer at the sink. The problem is that unless the inter-message period is smaller than the 802.11 **beacon interval**, messages would incur a beacon interval of **delay** at each hop while the **ATIM** process completes (§2.2.2).

For the CBR application, the inter-message period (250 milliseconds on average) is *longer* than the beacon interval (200 milliseconds). As a result, we expect IBSS PM to exhibit poor delivery latency. On the other hand, we expect ODPM, LPM, and XPM, which use **power management suspension** (§4.1.1), to offer latency that is closer to the case of *no* power management.

Message delivery latency is related to the length of the route traversed by that

Mobility	IBSS	ODPM	IBSS PM	LPM	XPM Time	XPM Credit
Unstructured	1.90 ±0.002	1.80 ±0.001	1.82 ±0.002	1.90 ±0.002	1.88 ±0.002	1.82 ±0.002
Moderately Structured	3.94 ±0.002	3.56 ±0.002	3.64 ±0.002	3.79 ±0.002	3.93 ±0.002	3.92 ±0.003
Highly Structured	5.77 ±0.003	5.45 ±0.002	5.53 ±0.002	5.37 ±0.002	6.88 ±0.003	6.70 ±0.007

(a) 1-flow traffic.

Mobility	IBSS	ODPM	IBSS PM	LPM	XPM Time	XPM Credit
Unstructured	1.89 ±0.001	1.81 ±0.001	1.83 ±0.001	1.90 ±0.001	1.88 ±0.001	1.80 ±0.002
Moderately Structured	3.93 ±0.001	3.62 ±0.001	3.67 ±0.001	3.82 ±0.001	4.04 ±0.001	3.96 ±0.002
Highly Structured	5.84 ±0.002	5.45 ±0.002	5.70 ±0.002	5.58 ±0.002	6.99 ±0.002	6.79 ±0.005

(b) 2-flow traffic.

Table 6.6: Mean route length.

message. Table 6.6 shows the mean route lengths, with 95% confidence intervals, observed in our experiments. For our workloads, route length increases with the degree of structure in the mobility model. We expect a similar trend in delivery latency. Of special note are the lengths of routes chosen by XPM in the highly structured cases, which are greater than those chosen by the other power management designs. This occurs because the relays on the shortest routes become more expensive than other nodes, causing more circuitous routes to be assigned winning. We expect these longer routes to affect delivery latency.

Table 6.7 shows delivery latency performance for the 1-flow traffic load. IBSS PM

Mobility	**IBSS**	**ODPM**	**IBSS PM**	**LPM**	**XPM Time**	**XPM Credit**
Unstructured	3.47ms ±22μs	56.7ms ±1.13ms	292ms ±876μs	9.00ms ±79μs	8.83ms ±34μs	9.12ms ±49μs
Moderately Structured	7.98ms ±154μs	42.0ms ±683μs	775ms ±1,993μs	13.8ms ±139μs	14.9ms ±194μs	18.3ms ±459μs
Highly Structured	11.1ms ±84μs	93.3ms ±1.19ms	1,385ms ±2,480μs	18.6ms ±218μs	22.6ms ±166μs	38.0ms ±1,037μs

Table 6.7: Mean delivery latency for 1-flow traffic load.

exhibits substantially higher latency than IBSS, by factors of 84 to 125. The previously-published ODPM design improves this latency by using power management suspension. Its average latency is higher than IBSS by factors of 5.2 to 16; these figures reflect the high route setup latency experienced under ODPM. Our own designs, which reduce setup latency, show a delivery latency that is *at most* 3.4 times that of IBSS-mode. These improve latency relative to IBSS PM by factors of 32 to 74 for LPM, and 32 to 61 for XPM.

Mobility	**IBSS**	**ODPM**	**IBSS PM**	**LPM**	**XPM Time**	**XPM Credit**
Unstructured	3.71ms ±28μs	36.9ms ±641μs	296ms ±656μs	9.22ms ±36μs	11.8ms ±209μs	13.5ms ±294μs
Moderately Structured	10.4ms ±305μs	34.2ms ±469μs	810ms ±1,345μs	16.6ms ±306μs	20.4ms ±325μs	32.7ms ±813μs
Highly Structured	11.5ms ±34μs	60.1ms ±595μs	1,237ms ±1,527μs	20.4ms ±198μs	47.7ms ±672μs	153ms ±2,955μs

Table 6.8: Mean delivery latency for 2-flow traffic load.

The 2-flow traffic load latency is shown in Table 6.8. Again, IBSS PM incurs 78 to 108 times the latency of IBSS. Generally, LPM and XPM offer similar benefits to those seen in the 1-flow data, up to a factor of 61 improvement over IBSS PM.

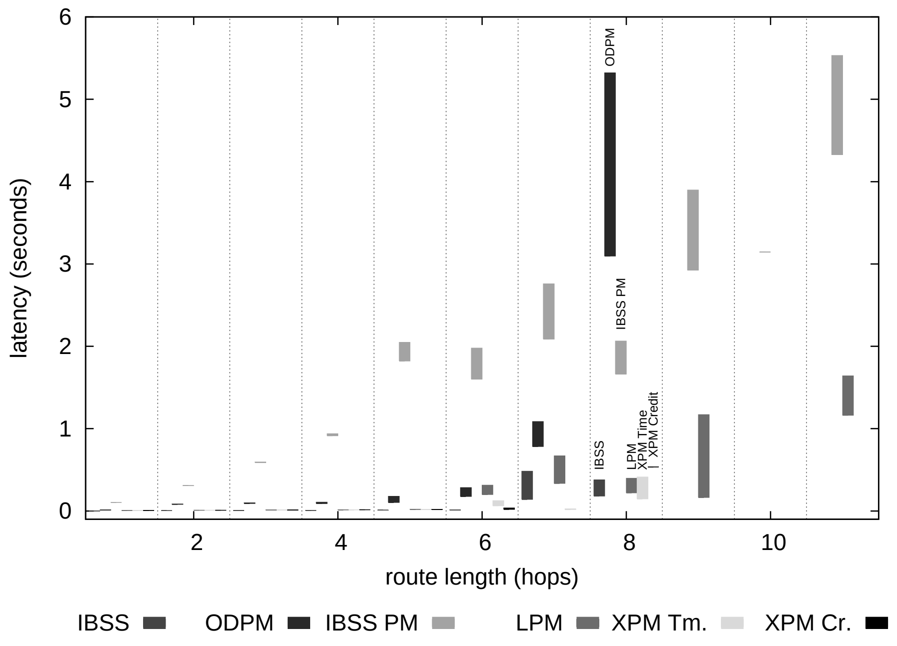
The exception is XPM Credit under the highly structured mobility model, where the improvement is only a factor of 8. We hypothesize that **queueing delays** at the traffic source during **Route Discovery**, which was shown to take a long time in Table 6.5, are responsible for this high average latency.

Figures 6.38, 6.39, and 6.40 again show 95% confidence intervals for mean delivery latency as a function of **route length**. Since many of the sample mean values are close to zero, the bottom edge of each graph has been lowered by 0.1 second to improve readability. Since most routes tend to be 8–10 hops or less, the confidence intervals for longer routes are spread out due to their smaller number of samples. Nevertheless, it is clear that LPM and XPM yield delivery latency that is in some cases several *seconds* faster than IBSS PM.

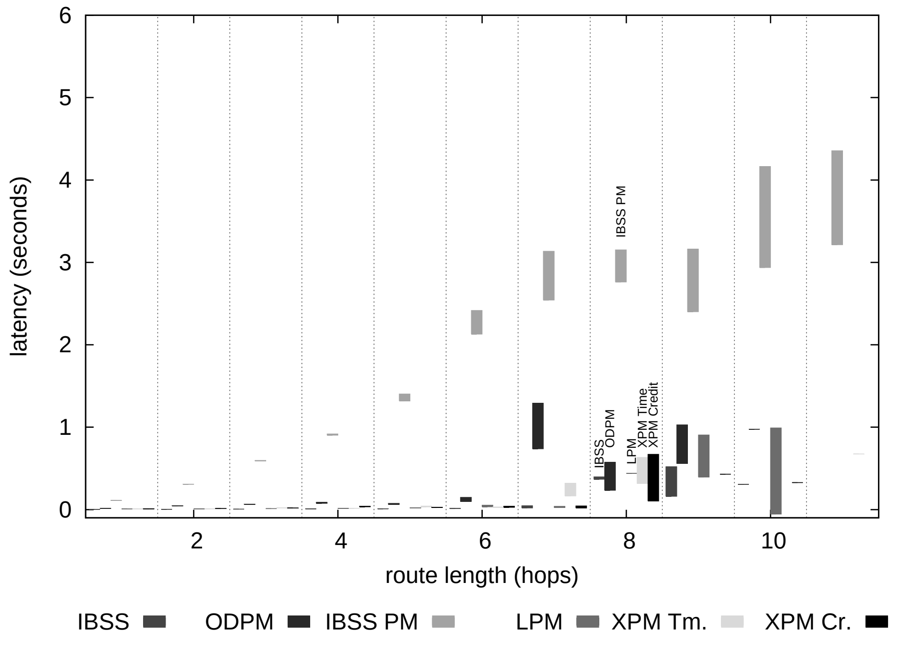
Name	**Latency**
BECA	20× increase in route setup latency
AFECA	30× increase in route setup latency
GAF	2× increase in average latency
Span	3× increase in average latency
On-demand	10×–28× route setup latency, 3×–16× average latency
LPM	2.1×–3.6× route setup latency, 1.6×–2.6× average latency
XPM Time	2.5×–4.9× route setup latency, 1.9×–4.1× average latency
XPM Credit	2.7×–6.1× route setup latency, 2.3×–13.3× average latency

Table 6.9: Augmented power management summary: latency.

Table 6.9 summarizes the the latency performance of LPM and XPM alongside the designs presented in Chapter 3. The measurements are all expressed relative to the latency of a non-power-managing network, which gives the best performance. The worst case latency is given by a network that uses 802.11 **IBSS power man-**

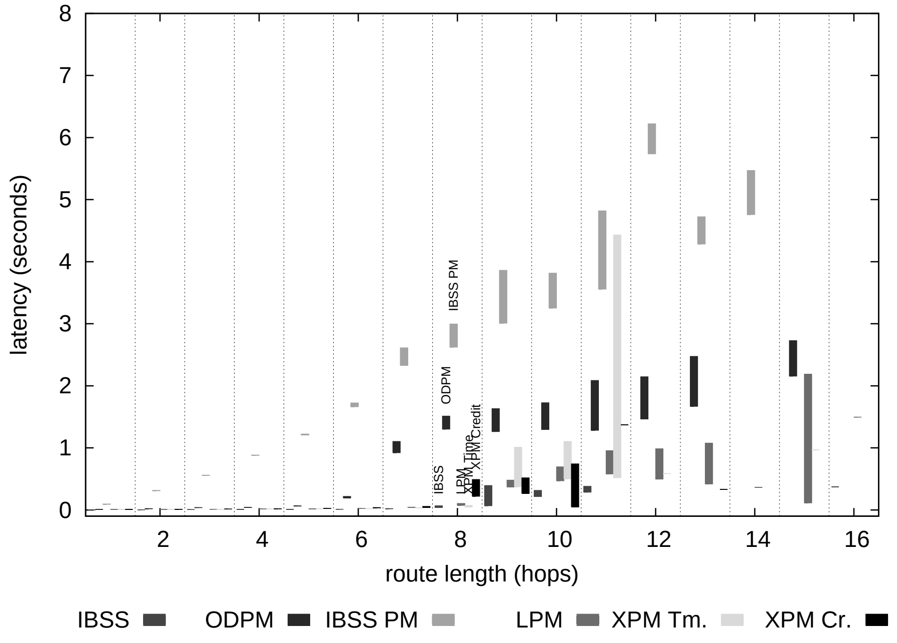


(a) 1-flow traffic load.

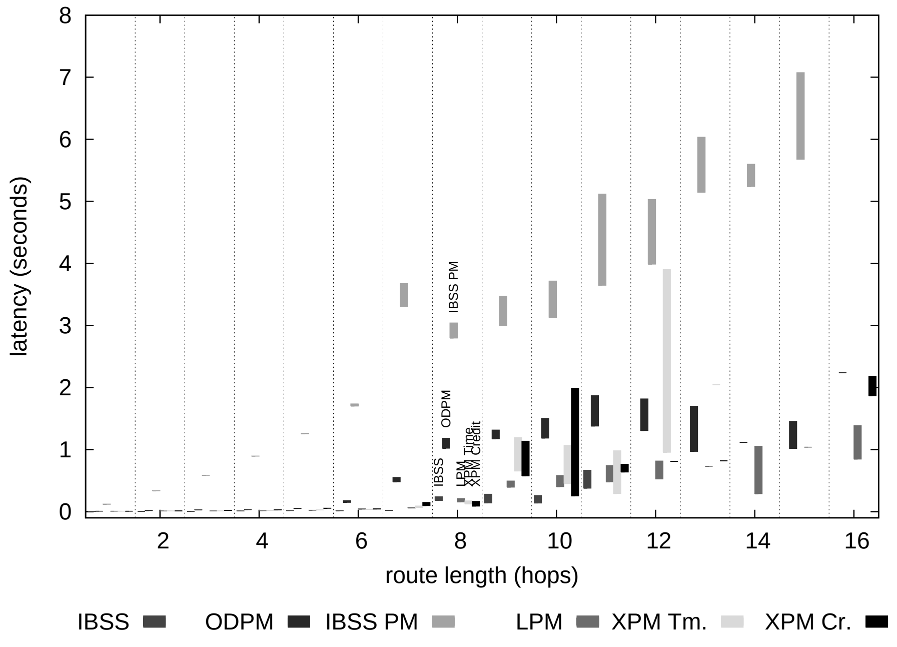


(b) 2-flow traffic load.

Figure 6.38: Unstructured application message delivery latency means.

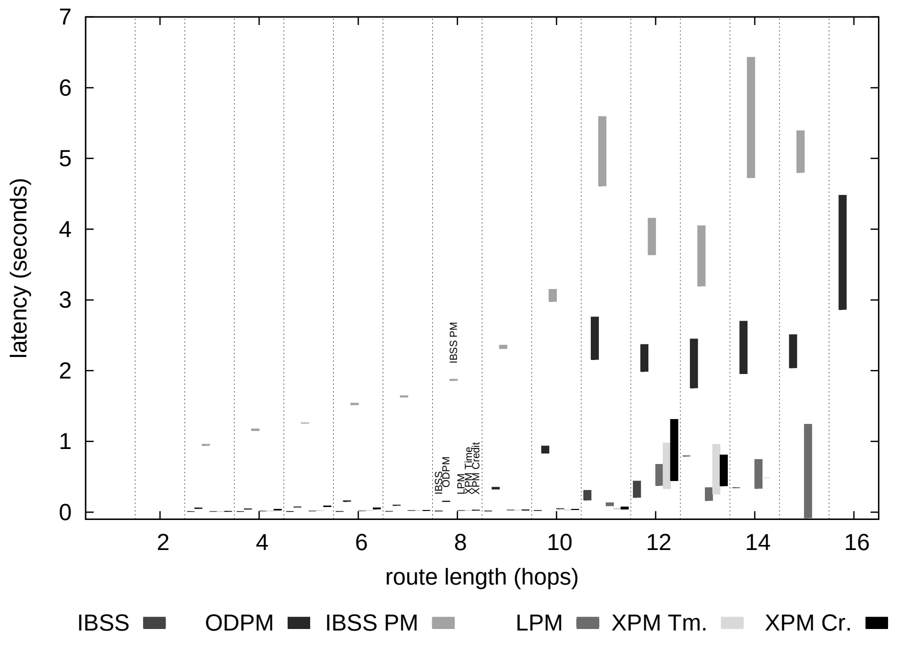


(a) 1-flow traffic load.

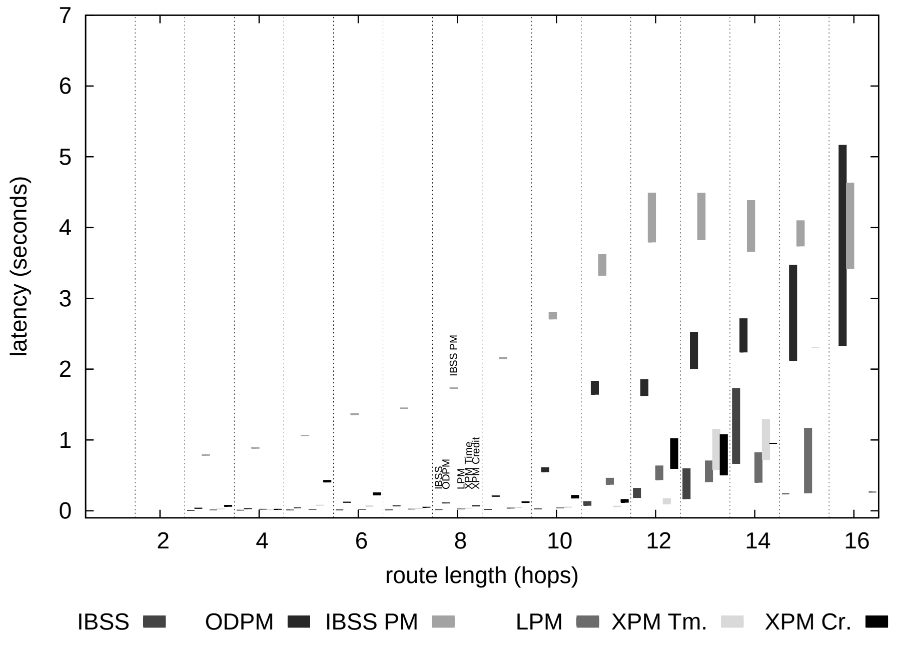


(b) 2-flow traffic load.

Figure 6.39: Moderately structured application message delivery latency means.



(a) 1-flow traffic load.



(b) 2-flow traffic load.

Figure 6.40: Highly structured application message delivery latency means.

agement. The results of this Section show that route setup latency increases by factors of 18 to 45 under IBSS PM. The **BECA**, **AFECA**, and **On-demand Power Management**[2] setup latencies are similar. **GAF** and **Span** are **proactive** designs, and achieve near-best-case setup latency by requiring relays to *always* be active. By comparison, LPM and XPM latency increases are an order of magnitude smaller than other **on-demand** designs, and approach the performance of the proactive designs. Our measurements also show that IBSS PM average (application message) latency is 78 to 125 times that of the non-power-managing IBSS mode. Again, LPM and XPM exhibit average latency that is one or two orders of magnitude less than this, and typically comparable to existing designs.

6.3 Delivery Ratio

A traditional metric used to evaluate routing protocols is the number of messages they successfully deliver. Call the **delivery ratio** the fraction of application messages *received* compared to the number which were *sent*. Modern protocols are expected to deliver nearly 100% of their messages.

There are several reasons why a routing protocol, particularly one operating in a **mobile wireless network**, might not deliver all of its messages. First, the protocol might not be able to **discover** a route for some messages. Second, a route

[2]Summary latency statistics were not previously published for On-demand Power Management. Several plots in (Zheng and Kravets, 2003) illustrate that average latency is near that of a non-power-managing network, while setup latency is much higher.

might **break** during use. This is a real problem for the CBR application running on DSR, since DSR only makes a limited effort to **salvage** messages with broken routes. (If the forwarding node which discovers a broken **next hop** does not know an alternate route to the sink, then the message is discarded.) The CBR application does *not* detect lost messages, and has no **retry** facility. Therefore, we *expect* to lose some number of messages due to **node mobility** and **broken links**.

Exchange Power Management introduces a third reason why some messages might not be delivered. When a source cannot **afford** any of the routes it knows to a sink, it is blocked from sending messages for the remainder of the current negotiation. Such messages in our simulations are simply **discarded** at the source. When using XPM with **Credit Balance** valuation, these **unaffordable messages** count against the delivery ratio.

We must clarify one point regarding the delivery ratio for XPM. A ratio $d \leq 1$ does *not* imply that messages are **randomly** lost at the rate $1 - d$. Rather, it means that a fraction of messages $1 - d$ were generated during negotiation intervals when the source had no affordable routes. Affordability is determined by the **state** of the network — its **geometry**, the **preferences** of nodes on demanded routes — at the time messages become available for transmission. If the application could **reschedule** its communication so that it only sent messages when it could **afford** routes, its delivery ratio would rise. The CBR application in our simulations has not been modified to behave this way, but such adaptation is an interesting subject for future work.

Mobility	**IBSS**	**ODPM**	**IBSS PM**	**LPM**	**XPM Time**	**XPM Credit**
Unstructured	99.9%	99.3%	99.1%	99.9%	99.9%	70.3%
Moderately Structured	99.8%	98.8%	98.0%	99.8%	99.9%	45.9%
Highly Structured	99.8%	98.7%	98.0%	99.9%	99.2%	27.8%

Table 6.10: Application message delivery ratio for 1-flow traffic load.

Table 6.10 shows the observed delivery ratio for the 1-flow traffic load. Each cell contains the cumulative data for 30 simulations. Since DSR was designed for **non-power-managing networks**, IBSS-mode is the "gold standard" for message delivery performance. As expected, IBSS-mode provides nearly 100% delivery under all mobility models. Adding power management, ODPM and IBSS PM are slightly less reliable due in part to the high latency of **Route Discovery**. LPM, which addresses this latency problem, has reliability comparable to IBSS.

XPM, when using **Time Balance** valuation, offers a high delivery ratio comparable to IBSS-mode or LPM. Time Balance **bid pricing** tries to offer a sufficiently high reported valuation so that *all* routes are affordable. The slightly lower ratio observed for the highly structured mobility model is an indication that this valuation function requires additional tuning. In a small number of cases (0.8%), it is encountering **unaffordable routes** due to an insufficiently high bid price. These cases almost certainly involve the overused nodes in the center corridor of the mobility model, which are offering very large **ask prices** to try and avoid service.

The **Credit Balance** valuation function, in contrast with Time Balance valuation, offers a much lower delivery ratio. This is an intended consequence of **unaffordable routes**. For example, in the unstructured mobility model, nearly 30% of messages were generated at times when the state of the network made the routes for those messages too expensive. As suggested earlier, it is entirely possible that if the sources could have rescheduled their transmission attempts for a time when affordable routes were available, this ratio could be improved. Section 6.1.1 showed the advantage of accepting this behavior: nodes can "cap" their **energy consumption** and avoid being overused.

Mobility	**IBSS**	**ODPM**	**IBSS PM**	**LPM**	**XPM Time**	**XPM Credit**
Unstructured	99.9%	99.5%	99.4%	99.9%	100%	62.2%
Moderately Structured	99.8%	99.3%	98.5%	99.8%	99.8%	37.6%
Highly Structured	99.9%	99.3%	98.6%	99.9%	99.6%	24.2%

Table 6.11: Application message delivery ratio for 2-flow traffic load.

Table 6.11 shows the delivery ratio data for the 2-flow traffic load. The results are similar to the 1-flow case. The ratio for XPM Credit has fallen slightly, because the increased network utilization causes ask prices to rise more quickly. As a result, sources encounter unaffordable routes sooner.

To support the claim that the delivery performance exhibited by XPM is mostly attributable to **affordability**, Table 6.12 shows the delivery ratio for affordable messages. These data show that, when messages are sent along affordable routes, the

	1-flow		2-flow	
Mobility	**XPM Time**	**XPM Credit**	**XPM Time**	**XPM Credit**
Unstructured	99.9%	99.9%	100%	99.9%
Moderately Structured	99.9%	99.9%	99.9%	99.8%
Highly Structured	100%	99.9%	99.8%	99.4%

Table 6.12: Affordable message delivery ratio.

delivery ratio approaches 100%. We expect this performance; once a negotiation is in place, message delivery using XPM is essentially the same as with LPM, which also gives a high delivery ratio.

6.4 Negotiation Characteristics

This Section profiles several attributes of the **Exchange Power Management** negotiation procedure. We characterize the amount of work done by XPM under the various workloads. First, we examine how *often* a node participates in a negotiation. For those negotiations that succeed, we measure how many **procedure restarts** were required due to failures (*e.g.*, broken edges in the **spanning tree**). We profile the amount of time required to complete the **Discovery**, **Structure**, and **Negotiation** phases. Next, we show how long the **Implementation** phase typically lasts, accounting for route breakage and other factors. Finally, we present measurements of the message and data overhead of the XPM design.

6.4.1 Active Participant Restart Separation

The first measurement we present is the mean time between negotiation procedures in which a node is an **active participant**. A node is an active participant when it begins the procedure with **active destinations** (§4.3.1), or when it receives an XPM message during the procedure. We measure the time between **restart events** that mark the beginning of negotiations in which a node is active. This metric gives an idea of how often a node will have to do work in a negotiation (as opposed to simply timing out).

	1-flow		2-flow	
Mobility	**XPM Time**	**XPM Credit**	**XPM Time**	**XPM Credit**
Unstructured	242s ±5.04s	242s ±5.07s	119s ±1.89s	125s ±2.00s
Moderately Structured	107s ±1.42s	113s ±1.56s	55.6s ±548ms	62.8s ±645ms
Highly Structured	65.5s ±778ms	71.6s ±897ms	43.5s ±500ms	51.0s ±541ms

Table 6.13: Mean separation between active participant restart events.

Table 6.13 gives the **restart separation** sample means, with 95% confidence intervals, for each combination of mobility model and traffic load. With the greater number of traffic flows, nodes participate in negotiations more frequently. This frequency also increases with the degree of **structure** in the mobility models. XPM tends to use longer routes in the more structured models as it routes around overused relays. Longer routes offer more opportunities for **route breakage**, therefore increasing the frequency of **renegotiation**.

6.4.2 Restarts per Negotiation

Chapter 4 explained that since the XPM negotiation procedure is brief, **faults** are generally handled by **restarting** the procedure. For example, if an edge in the **spanning tree** breaks before it can be used to distribute **Result** messages, the procedure restarts and the tree is formed again "from scratch." We have proposed methods for **tree repair** (§4.6.2) that handle such events more gracefully. It is worth asking, however, how much work results from the current handling.

	1-flow		2-flow	
Mobility	**XPM Time**	**XPM Credit**	**XPM Time**	**XPM Credit**
Unstructured	1.05 ±0.002	1.05 ±0.002	1.07 ±0.002	1.08 ±0.002
Moderately Structured	1.14 ±0.002	1.12 ±0.002	1.44 ±0.005	1.39 ±0.006
Highly Structured	1.22 ±0.002	1.18 ±0.002	1.68 ±0.004	1.62 ±0.004

Table 6.14: Mean number of restarts per negotiation.

Table 6.14 shows the sample mean number of restart events that occur in an execution of the negotiation procedure. In other words, this shows how many tries are typically needed to "get it right." The minimum value for this variable is 1. In most cases, the mean is quite close to the minimum. For the more structured mobility models with multiple traffic flows, larger **tree sizes** offer more opportunities for edge breakage, thus increasing the likelihood of additional restarts. Still, the mean is less than two restarts per run. The multiple-flow data suggest that graceful tree repair techniques will help the **scalability** of XPM in larger and more active networks.

6.4.3 Negotiation Time

It is useful to know how much time elapses from the initial **restart** event until a node receives its **Result** message and enters **Implementation**. This measurement includes the time spent in **Discovery**, **Structure**, and **Negotiation**, and accounts for intermediate restarts resulting from events such as tree edge breakage. In our trials, the **tree root** spends 250 milliseconds of simulated time solving the **VCG mechanism** (§4.4.3) independent of the number of bids. This delay is incorporated into the total time required for the negotiation procedure. Section 6.5 shows that 250 milliseconds may be excessive for most of our simulations, but insufficient for larger networks.

	1-flow		2-flow	
Mobility	**XPM Time**	**XPM Credit**	**XPM Time**	**XPM Credit**
Unstructured	408ms ±1.78ms	409ms ±1.84ms	415ms ±1.29ms	419ms ±1.33ms
Moderately Structured	537ms ±1.40ms	517ms ±1.39ms	710ms ±1.77ms	690ms ±1.92ms
Highly Structured	622ms ±1.09ms	585ms ±1.08ms	836ms ±1.57ms	820ms ±1.65ms

Table 6.15: Mean time required for negotiation process.

Table 6.15 shows the sample mean negotiation times with 95% confidence intervals. For most workloads, the mean is about half a second. With increasing structure and traffic activity, negotiation times rise; this is due to the higher likelihood that such negotiations will involve multiple **restarts**. Every restart adds at least one **beacon interval** — 200 milliseconds in our simulations — to the negotiation time. This is the real cost of intermediate restarts, since nodes must be awake during these wasted intervals, consuming energy at the high **idle power rate**.

These data characterize the **overhead** of the negotiation process. Nominally, this means about a half second of overhead per 20 seconds of **Implementation**. Section 6.4.4 measures the typical length of the Implementation phase.

6.4.4 Negotiation Lifetime

Finally, we measure the **lifetime** of a negotiation; that is, the duration of the **Implementation** phase. Section 4.4.4 explained that once the results of a negotiation are distributed, nodes **implement** those results for a **negotiation interval**, which is 20 seconds in our simulations. Several factors can cause the interval to be shortened. A new traffic flow might become active, or an existing flow might become inactive or break. These events all trigger a **restart**, which ends the current negotiation interval early.

Table 6.16 presents the sample mean negotiation lifetime observed in our experiments. In most cases, the expected lifetime is more than 75% of the **nominal** lifetime. As the degree of structure and the amount of traffic increase, the likelihood of early termination increases. These measurements illustrate the usefulness of the proposed **interim mechanism solutions** (§4.6.1). If nodes did not have to

	1-flow		2-flow	
Mobility	**XPM Time**	**XPM Credit**	**XPM Time**	**XPM Credit**
Unstructured	17.0s ±85.0ms	17.1s ±102ms	15.2s ±78.2ms	16.2s ±94.5ms
Moderately Structured	16.5s ±91.2ms	16.4s ±137ms	14.2s ±80.1ms	15.6s ±130ms
Highly Structured	16.4s ±94.2ms	16.3s ±189ms	14.2s ±82.1ms	15.9s ±173ms

Table 6.16: Mean negotiation lifetime.

abandon their current negotiations every time a new route becomes active, or an existing route changes, then the average negotiation lifetime could approach the target duration. This is desirable for several reasons, including the fact that it gives agents a better sense of how much **service** they are buying or selling.

6.4.5 Overhead

Traditional metrics for network protocol performance include the number of additional **messages** or **bits** of data generated as **overhead**. For example, the 802.11 MAC protocol generates three control frames for every directed data frame it sends, for a message overhead of 3:1. We have measured the message and data overhead for XPM and DSR with respect to the CBR application traffic load. Our methodology recorded every message **sent** by a source or **forwarded** by a relay. So, for example, a CBR or XPM Notify message sent along a route having $|P| - 1$ hops would be counted as $|P| - 1$ messages. A broadcast DSR Route Request counts as a single message.

Table 6.17 shows the overhead for DSR and XPM separately, measured with respect to the number of CBR messages transmitted. For instance, in the unstructured, 1-flow case, using XPM with Time Balance valuation, DSR sends 0.144 messages for every CBR message, and XPM generates an additional 0.098 messages for every CBR message. We have also broken out the XPM overhead by message type. It is worth noting that the greater XPM overhead under Credit Balance valuation is largely due to the smaller number of CBR messages transmitted.

Before presenting the raw data overhead, Table 6.18 shows the mean message sizes, with 95% confidence intervals, observed during the trials. These message sizes reflect the payloads passed to the MAC layer. Note that the messages which are sent along **routes**, as opposed to those sent to immediate neighbors, grow in size with the degree of mobility structure. These include CBR, DSR, and XPM Notify messages. As shown in Section 6.2.2, route length increases with structure in our workloads. The growth in message size reflects the length of the route encoded in these messages.

The XPM Submit and Result messages are particularly interesting. Submit message size is nearly identical under the Time Balance and Credit Balance valuation functions, increasing with the degree of structure. The Result messages are noticeably smaller under Credit Balance, however. This is because fewer winning bids tend to be encoded for distribution to agents under that valuation function.

	Unstructured		Moderately structured		Highly structured	
Message	**XPM Time**	**XPM Credit**	**XPM Time**	**XPM Credit**	**XPM Time**	**XPM Credit**
CBR	1	1	1	1	1	1
DSR	0.144	0.206	0.118	0.227	0.092	0.255
XPM (all)	0.098	0.140	0.199	0.405	0.247	0.808
XPM Notify	0.017	0.026	0.031	0.063	0.033	0.106
XPM Update	0.013	0.018	0.035	0.070	0.048	0.157
XPM Respond	0.030	0.043	0.066	0.132	0.080	0.261
XPM Submit	0.019	0.027	0.034	0.070	0.044	0.144
XPM Revoke	0.002	0.003	0.008	0.016	0.016	0.054
XPM Result	0.017	0.024	0.025	0.053	0.026	0.086

(a) 1-flow traffic.

	Unstructured		Moderately structured		Highly structured	
Message	**XPM Time**	**XPM Credit**	**XPM Time**	**XPM Credit**	**XPM Time**	**XPM Credit**
CBR	1	1	1	1	1	1
DSR	0.136	0.223	0.116	0.250	0.084	0.254
XPM (all)	0.127	0.200	0.293	0.627	0.312	0.967
XPM Notify	0.022	0.035	0.047	0.099	0.047	0.145
XPM Update	0.018	0.028	0.058	0.122	0.063	0.193
XPM Respond	0.040	0.063	0.102	0.215	0.107	0.330
XPM Submit	0.023	0.037	0.045	0.099	0.050	0.156
XPM Revoke	0.004	0.006	0.015	0.033	0.023	0.073
XPM Result	0.019	0.030	0.026	0.059	0.022	0.069

(b) 2-flow traffic.

Table 6.17: Message overhead relative to CBR application traffic.

Message	Unstructured XPM Time	Unstructured XPM Credit	Mod. structured XPM Time	Mod. structured XPM Credit	Highly structured XPM Time	Highly structured XPM Credit
CBR	545 ±0.005	545 ±0.007	553 ±0.004	553 ±0.006	565 ±0.005	565 ±0.011
DSR	47.6 ±0.067	47.5 ±0.067	69.8 ±0.094	68.7 ±0.099	131 ±0.218	126 ±0.239
XPM (all)	69.7 ±0.182	67.0 ±0.171	77.6 ±0.142	72.2 ±0.127	97.2 ±0.169	82.7 ±0.129
XPM Notify	50.3 ±0.049	50.3 ±0.052	58.4 ±0.036	58.1 ±0.034	69.6 ±0.035	69.2 ±0.036
XPM Update	44.0 ±0	44.0 ±0	44.0 ±0	44.0 ±0	44.0 ±0	44.0 ±0
XPM Respond	44.0 ±0	44.0 ±0	44.0 ±0	44.0 ±0	44.0 ±0	44.0 ±0
XPM Submit	104 ±0.394	104 ±0.393	134 ±0.387	134 ±0.387	180 ±0.377	178 ±0.380
XPM Revoke	44.0 ±0	44.0 ±0	44.0 ±0	44.0 ±0	44.0 ±0	44.0 ±0
XPM Result	121 ±0.521	105 ±0.530	171 ±0.589	124 ±0.518	289 ±0.937	152 ±0.675

(a) 1-flow traffic.

Message	Unstructured XPM Time	Unstructured XPM Credit	Mod. structured XPM Time	Mod. structured XPM Credit	Highly structured XPM Time	Highly structured XPM Credit
CBR	545 ±0.004	545 ±0.006	553 ±0.006	553 ±0.008	566 ±0.004	566 ±0.010
DSR	48.8 ±0.055	48.4 ±0.054	73.0 ±0.077	71.4 ±0.081	133 ±0.183	131 ±0.200
XPM (all)	69.6 ±0.137	66.2 ±0.129	76.6 ±0.105	71.4 ±0.096	96.1 ±0.135	83.7 ±0.111
XPM Notify	50.5 ±0.035	50.5 ±0.036	59.5 ±0.029	59.6 ±0.032	72.0 ±0.027	71.1 ±0.029
XPM Update	44.0 ±0	44.0 ±0	44.0 ±0	44.0 ±0	44.0 ±0	44.0 ±0
XPM Respond	44.0 ±0	44.0 ±0	44.0 ±0	44.0 ±0	44.0 ±0	44.0 ±0
XPM Submit	111 ±0.361	111 ±0.376	151 ±0.352	152 ±0.379	211 ±0.388	211 ±0.421
XPM Revoke	44.0 ±0	44.0 ±0	44.0 ±0	44.0 ±0	44.0 ±0	44.0 ±0
XPM Result	126 ±0.460	103 ±0.444	197 ±0.652	128 ±0.492	347 ±1.16	164 ±0.769

(b) 2-flow traffic.

Table 6.18: Mean message size (octets).

	Unstructured		Moderately structured		Highly structured	
Message	**XPM Time**	**XPM Credit**	**XPM Time**	**XPM Credit**	**XPM Time**	**XPM Credit**
CBR	1	1	1	1	1	1
DSR	0.013	0.018	0.015	0.028	0.021	0.057
XPM (all)	0.013	0.017	0.028	0.053	0.043	0.118
XPM Notify	0.002	0.002	0.003	0.007	0.004	0.013
XPM Update	0.001	0.001	0.003	0.012	0.004	0.012
XPM Respond	0.002	0.003	0.005	0.011	0.006	0.020
XPM Submit	0.004	0.005	0.008	0.017	0.014	0.045
XPM Revoke	0.000	0.000	0.001	0.001	0.001	0.004
XPM Result	0.004	0.005	0.008	0.012	0.013	0.023

(a) 1-flow traffic.

	Unstructured		Moderately structured		Highly structured	
Message	**XPM Time**	**XPM Credit**	**XPM Time**	**XPM Credit**	**XPM Time**	**XPM Credit**
CBR	1	1	1	1	1	1
DSR	0.012	0.020	0.015	0.032	0.020	0.059
XPM (all)	0.016	0.024	0.041	0.081	0.053	0.143
XPM Notify	0.002	0.003	0.005	0.011	0.006	0.018
XPM Update	0.001	0.002	0.005	0.010	0.005	0.015
XPM Respond	0.003	0.005	0.008	0.017	0.008	0.026
XPM Submit	0.005	0.007	0.012	0.027	0.018	0.058
XPM Revoke	0.000	0.000	0.001	0.003	0.002	0.006
XPM Result	0.004	0.006	0.009	0.014	0.013	0.020

(b) 2-flow traffic.

Table 6.19: Bit overhead relative to CBR application traffic.

Table 6.19 shows the bit overhead, which is a function of the message overhead and the message size. In almost all scenarios, XPM contributes less than 10% bit overhead. In the worst case, using Credit Balance valuation in the highly structured scenarios, this overhead is less than 15% even though the amount of CBR traffic is small.

6.5 Computational Performance

The last set of measurements we present profile the performance of the **Branch on Bids** implementation, described in Section 4.4.3.2. This algorithm uses **heuristic search** techniques to solve an $\mathcal{NP}$-complete optimization problem. As such, it is critical to the scalability of XPM that the BOB implementation be as fast as possible. Our implementation is based on a design that is several years old, and does not contain all of the **optimizations** published for that algorithm (Sandholm, 2003). More recent commercial implementations of winner determination algorithms are claimed to be orders of magnitude faster than BOB. Other algorithms could replace BOB in an XPM implementation; all we require is that such an algorithm **correctly** and **optimally** solve the winner determination problem for combinatorial exchanges.

6.5.1 Winner Determination

We begin by profiling the BOB algorithm itself, which is used to determine the winning bids and asks during mechanism solution. One interesting result is that the bids and asks generated using **Time Balance** pricing are often more **computationally expensive** for BOB than those generated with **Credit Balance** pricing. Since Time Balance bids have prices that are high enough to afford *any* route, such bids cannot be **preprocessed** out of the input to the heuristic search stage.

Table 6.20 shows the sample mean compute time for BOB as measured during our simulations. The test environment was an Athlon MP 1900+ system running Linux 2.4.19; the implementation was built using the Intel C++ Compiler for Linux, version 7.0. Filesystem logging for the BOB code was disabled during the experiments. The data are only for the **winner determination** step in the **mechanism solution**; they do *not* include runs of BOB used to compute agent **Vickrey discounts**.

	1-flow		2-flow	
Mobility	**XPM Time**	**XPM Credit**	**XPM Time**	**XPM Credit**
Unstructured	374μs $\pm 3\mu$s	357μs $\pm 4\mu$s	444μs $\pm 6\mu$s	440μs $\pm 8\mu$s
Moderately Structured	1.08ms $\pm 11\mu$s	870μs $\pm 12\mu$s	2.85ms $\pm 40\mu$s	2.77ms $\pm 61\mu$s
Highly Structured	2.53ms $\pm 23\mu$s	1.84ms $\pm 33\mu$s	6.47ms $\pm 95\mu$s	6.91ms $\pm 183\mu$s

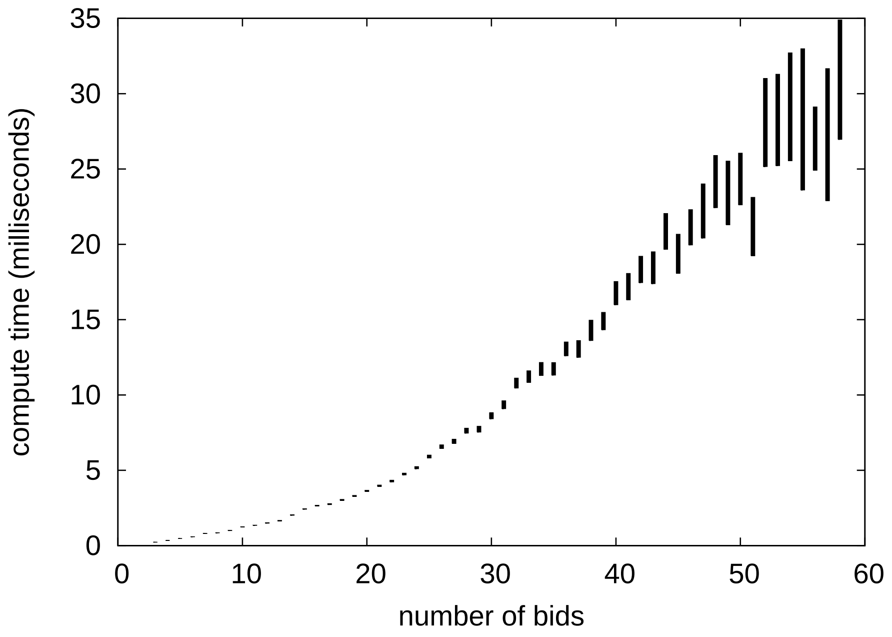
Table 6.20: Mean winner determination time.

The data show that for the problem sizes that occur in our simulations, even a relatively unoptimized BOB is fast in the average case. Solution time increases with the number of traffic flows and the **sparseness** of the network. In the more structured mobility models, individual routes are less likely to have nodes in common, so the number of relays submitting asks increases. Also, as mentioned earlier, the ability to **preprocess** unaffordable bids has a visible impact on solution performance. **Credit Balance** pricing reduces solution time by up to 27% relative to **Time Balance** pricing.

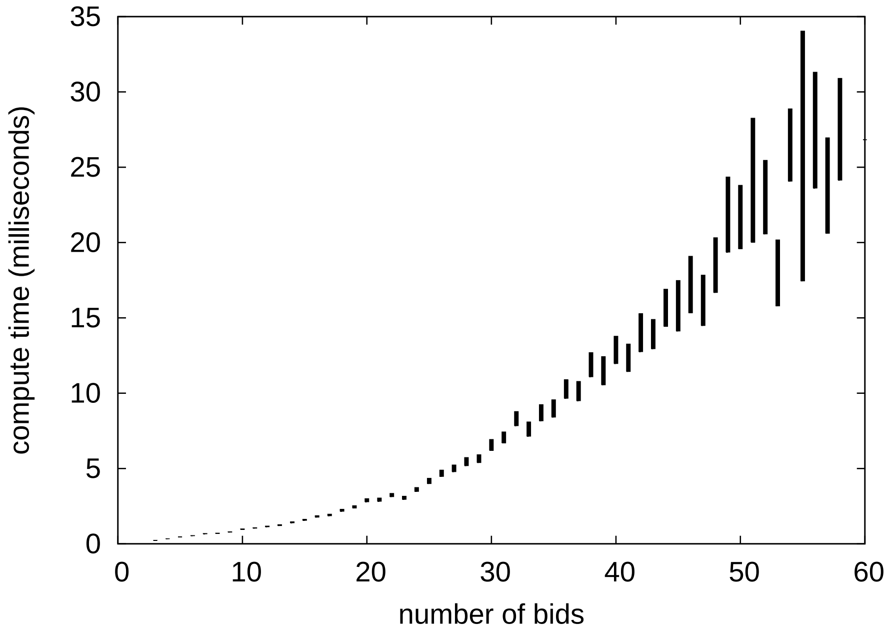
Figure 6.41 plots solution time *vs.* the number of input bids $|\mathcal{B}|$ over *all* runs of the algorithm in the initial winner determination step. The graphs show 95% confidence intervals for the sample mean at each input size. The intervals grow as the input size increases, due to the decreased number of samples for larger problems. Under both valuation functions, problem sizes of 30 bids or greater account for 5% of the sample data.

6.5.2 Vickrey Clarke Groves Mechanism Solution

The solution of the **Vickrey Clarke Groves mechanism** requires the solution of the **winner determination problem** once for all the bids $\mathcal{B}$, then again for each set $\mathcal{B}_{-i}$ with winning agent i removed. The smaller problems are solved to compute each agent's **Vickrey discount**, which determines the agent's **payment** to the mechanism. Bid **preprocessing**, described in Section 4.4.3.2, is helpful in containing the computational costs of these additional solutions.



(a) Time Balance valuation.



(b) Credit Balance valuation.

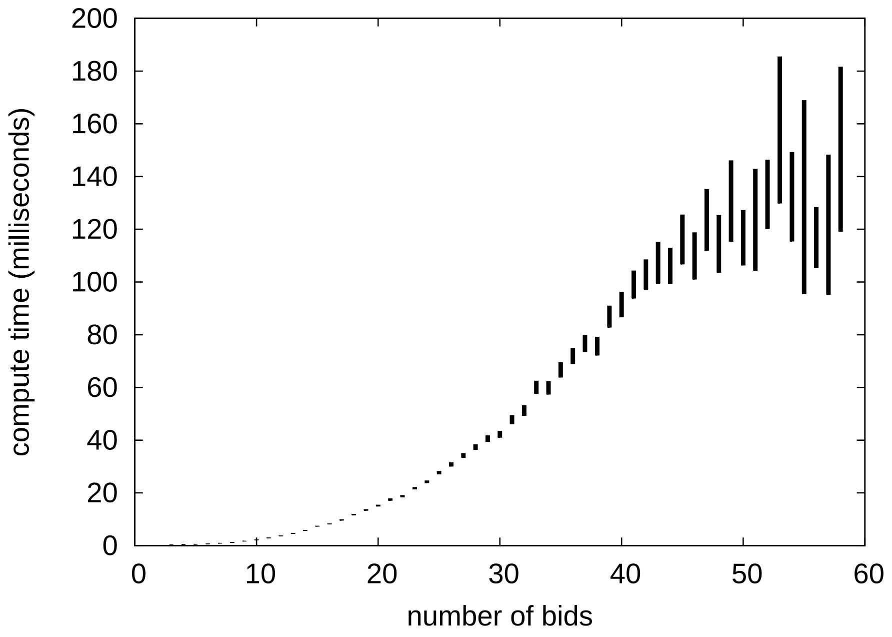
Figure 6.41: Winner determination time.

	1-flow		2-flow	
Mobility	**XPM Time**	**XPM Credit**	**XPM Time**	**XPM Credit**
Unstructured	500μs $\pm 9\mu$s	466μs $\pm 10\mu$s	745μs $\pm 19\mu$s	685μs $\pm 24\mu$s
Moderately Structured	2.64ms $\pm 40\mu$s	1.73ms $\pm 45\mu$s	11.6ms $\pm 202\mu$s	8.19ms $\pm 225\mu$s
Highly Structured	8.59ms $\pm 121\mu$s	4.95ms $\pm 172\mu$s	31.6ms $\pm 549\mu$s	24.7ms $\pm 863\mu$s

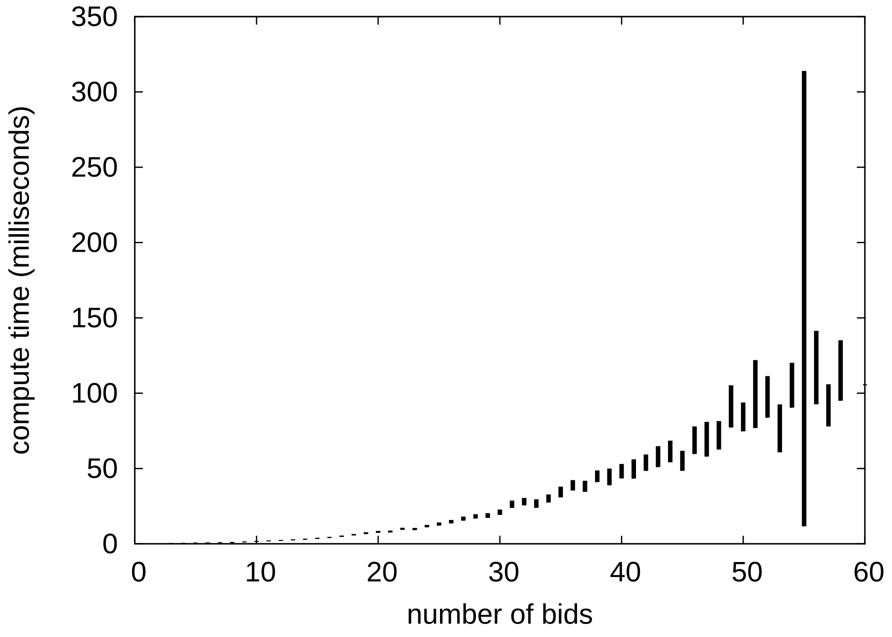
Table 6.21: Mean VCG mechanism solution time.

Table 6.21 shows the total solution time for the VCG mechanism, including winner determination and Vickrey payment computation. Compared to the costs of **winner determination** presented in Table 6.20, full mechanism solution typically requires no more than 3.5 times as much compute time. Also, the greatest mean solution time, for **Time Balance** valuation in the highly structured mobility model with 2-flow traffic, is still 87% less than the **simulated time** we allocated for mechanism solution in our experiments. In many cases, we could have reduced the simulated time budgeted for this activity, and improved both the **time** and **energy** required for the negotiation process. Also, Section 4.5.1 explained that since buyers using Time Balance valuation are **insensitive** to their payment amounts, the use of **Vickrey payments** provides no incentives. We could eliminate Vickrey payment computation, and reduce the amount of work to just that of winner determination (Table 6.20).

Finally, Figure 6.42 shows the mechanism solution time as a function of input size. (Note that the magnitude of the vertical axis differs between Figures 6.42(a)



(a) Time Balance valuation.



(b) Credit Balance valuation.

Figure 6.42: VCG mechanism solution time.

and 6.42(b).) This representation shows why a more efficient winner determination algorithm is important to the future scalability of XPM. For **Time Balance** valuation, mechanism solution requires *one fifth of a second* for the largest problem sizes encountered in our simulations, and will require more time as the number of flows increases.

6.5.3 Effects of Optimizations

The **Branch-on-Bids** heuristic search algorithm (§4.4.3.2) employs a number of optimizations that estimate when the search tree may be pruned, and determine the order in which bids are explored in the tree. The major optimizations are:

- ***h* upper revenue bound** (Procedure 11, line 4). Estimates revenue achievable from bids that have not been assigned winning or losing.
- ***L* lower revenue bound** (Procedure 11, line 7). Finds a single bid that can be feasibly added to the current allocation.
- ***H* lower revenue decrease bound** (Procedure 11, line 15). When the current allocation is infeasible, estimates the least decrease in revenue needed to make the currently-infeasible item allocations feasible.
- **Overlap branching bids** (Procedure 11, line 19). When the current allocation is infeasible, branch on a bid that demands additional units of a currently-infeasible item, thus forcing the search of multi-unit asks early.

- **Min branching bids** (Procedure 11, line 21). When the current allocation is infeasible, branch on a bid that most reduces the magnitude of the net demand for some item which is currently infeasible.

In order to measure the relative impact of these optimizations, we ran the 30 trials for the 2-flow highly structured scenario using Credit Balance valuation. We then measured the time required to complete VCG mechanism solution, as in Section 6.5.2, with each optimization disabled in turn. In the case of the three pruning heuristics, these were disabled simply by not computing their value, and never executing the respective pruning step. For the two bid ordering techniques, these were disabled by replacing them with the **descending-price** bid selection, which uses a bid order computed in advance. This is exactly the same bid ordering used on line 10 of Procedure 11.

All On	***h* Off**	***L* Off**	***H* Off**	**Overlap Branching Off**	**Min Branching Off**	**Min and Overlap Branching Off**
213ms $\pm$14.1ms	233ms $\pm$15.6ms	245ms $\pm$20.7ms	91.0s $\pm$7.81s	24.7ms $\pm 863\mu$s	99.1ms $\pm$6.54ms	77.3ms $\pm$4.77ms

Table 6.22: Mean VCG solution times with individual optimizations disabled.

Table 6.22 shows the mean VCG solution times, with 95% confidence intervals, for each disabled optimization. We also measured performance with all optimizations enabled, and with *both* bid ordering techniques disabled. As expected, h, L, and H improve performance. Surprisingly, our bid ordering optimizations were not as helpful. In particular, by disabling the **overlap branching bid** technique, we were able to improve solution performance by more than a factor of 8. As a result of these measurements, the data presented in Sections 6.5.1 and 6.5.2 were collected with the overlap branching bid technique disabled.

6.6 Summary

We have shown that **Exchange Power Management** can be an effective tool for limiting the **worst case** energy consumption of nodes in a mobile ad hoc network. When maximal message delivery is required, XPM reduces the **magnitude** of the energy consumption **range** by up to a factor of 5 compared with **802.11 power management**. If **unaffordable routes** are permitted, the energy shaping capabilities of XPM improve, and can reduce this magnitude by up to a factor of 12. These energy shaping capabilities give XPM an advantage over **non-negotiated** schemes such as **Local Power Management**, which was shown to have poor **worst case** energy consumption in highly structured environments. Energy shaping is a contribution of XPM which distinguishes it from prior work in the area of **cross-layer power management** for *ad hoc* networks.

The improved worst case behavior does not come at the expense of **average** energy consumption. XPM energy savings are comparable to, or better than, IBSS power management mode. Relative to the case of *no* power management, XPM yields energy savings of 56%–69%, compared with 53%–63% for 802.11 power management. XPM is competitive with earlier cross-layer power management designs in the average case. When the network is **idle**, XPM nearly achieves the optimal energy savings, something **proactive** protocols cannot do.

Using techniques such as **fast wakeup** and **power management suspension**, we showed that XPM provides better **latency** than IBSS power management. **DSR Route Discovery** latency improved by up to a factor of 16, giving better performance than previous on-demand designs. Application message **delivery** latency improved upon IBSS power management by up to a factor of 61. Table 6.23 highlights these experimental results.

Feature	**Description**
Energy Range	Reduced magnitude by up to factor of 5 relative to IBSS PM, by up to factor of 12 using unaffordable routes.
Energy Mean	Savings of 56%–69% relative to no power management, competitive with previous work.
Route Discovery Latency	Improved by up to factor of 16 relative to IBSS PM.
Message Delivery Latency	Improved by up to factor of 61 relative to IBSS PM.

Table 6.23: Exchange Power Management results summary.

When using **Time Balance** valuation, XPM was shown to deliver a proportion of application messages comparable to the "gold standard," IBSS-mode. **Credit Balance** pricing trades off **delivery ratio** to limit the energy consumption of nodes. We explained how applications that are aware of route **route affordability** might in the future **reschedule** their communications for times when affordable routes are available.

We profiled the XPM negotiation procedure, and found that the practice of

restarting the procedure "from scratch" in the event of a **tree edge break** does not harm performance too much. For most workloads, the average number of restarts per run of the procedure is close to the minimum, and in all cases averages fewer than two restarts per run. Also, for most workloads, the average time required to complete the negotiation procedure is about half a second. Negotiations typically last for at least 75% of their **nominal** lifetime before events such as traffic flow changes or route breakage **interrupt** them.

Finally, we presented measurements of our **Vickrey Clarke Groves mechanism** implementation. Our version of the **Branch on Bids** algorithm provided satisfactory performance for the problem sizes encountered in our simulations. We indicated that additional optimization would be necessary in order for XPM to **scale** to larger or more active networks.

Chapter 7

Conclusion

The use of economic tools such as **game theory** in the analysis and operation of networks has recently become a field of interest for research. Multihop *ad hoc* networks, in which nodes **relay** messages for other nodes, are an important class of networks characterized by a conflict between individual **energy performance** and system-wide **communications performance**. Understanding the energy costs experienced by network agents, we can design a negotiation framework that enables agents to report their **preferences** over network configurations. If a **credit system** is available to account for the **value** exchanged in this environment, these negotiations can be made **incentive compatible**.

This thesis represents the first application of game theory and **mechanism design** to the problem of **power management** in realistic *ad hoc* networks. Taking a systems view, we have shown that the energy costs are dominated by the **idle** power state in wireless interfaces such as the popular 802.11 radios. In this context, the **marginal energy costs** of relaying additional similar traffic flows, given that a relay is active at all, are small. We showed that an incentive-compatible mechanism for this environment must account for marginal costs by considering the **overlap** between different **source routes**. A mechanism based on the **combinatorial exchange** — which incorporates the reported preferences of overlapping sources, relays, and sinks — solves a network configuration that maximizes **surplus** over the agent valuations. Only an optimal solution to the combinatorial exchange winner determination problem incents the agents to truthfully report their value, or loss of value, from network participation.

We have developed a practical method called **Exchange Power Management** to implement exchange-based negotiation in on-demand **source routing** networks. Our design augments existing protocols such as **Dynamic Source Routing** and 802.11 by using higher-level information to direct the use of low-level **power management**. XPM is an instance of a general four-phase framework for negotiations in *ad hoc* networks. We described specific algorithms and actions for each phase, but noted that the phases are somewhat orthogonal and that the algorithms could be replaced. The first phase uses DSR **Route Discovery** to identify a set of **demand routes** for which a traffic source will submit **bids**. In the second phase, a **spanning tree overlay** is constructed in on-demand fashion across the overlapping routes. Next, a **wave algorithm** is used to collect bids, the mechanism is solved using

heuristic search techniques, and the results are distributed to the agents. Finally, the agents implement the results by participating in active routes, or by entering a low-power state.

Using a realistic simulation environment, we demonstrated the practical performance of XPM under a range of **mobility** and **communications** workloads. Quantitative measurements based on the power profile of a well-known 802.11 interface show that XPM reduces the **variability** of node energy consumption. The greatest improvement occurs when relays and sinks are allowed to become **unaffordable** to traffic sources. This enables XPM to reduce the energy consumption **range** by an order of magnitude, at the cost of making some messages undeliverable. Future applications could adapt to this characteristic by **rescheduling** their communications for a time when affordable routes become available. We also showed that the degree of **structure** in the topology of the network has a significant impact on the worst-case energy consumption for nodes in **distinguished positions**.

This improvement in variability does not come at the cost of average-case energy performance. Our measurements showed that XPM saves at least as much energy as 802.11 power management, and that it is comparable to **timer-based** designs such as **Local Power Management**. We also showed that the techniques of **fast wakeup** and **power management suspension** dramatically improve communications latency in power-managing networks. Compared with 802.11 power management, we improved the latency of DSR Route Discovery by up to a factor of 16, and improved application message delivery latency by up to a factor of 61.

This thesis has identified many interesting areas for future research on the application of mechanism design to power managing networks, including:

- More sophisticated **agent valuation functions** could yield even better constraints on energy variability.
- Application awareness of the underlying negotiation system may, through rescheduling, allow a high message delivery ratio without sacrificing tight energy bounds.
- The use of **interim mechanism solutions** may reduce the overhead of negotiation by localizing the response to topology or traffic flow changes.
- An improved spanning tree formation algorithm will increase scalability and reduce the communications and energy costs of negotiation.
- Optimizations to the **Branch on Bids** heuristic search algorithm — or a new algorithm entirely — will allow the mechanism itself to scale to larger and more active networks.

The use of game theory in the design of networked systems is still young, and the ultimate rôle of these techniques is not yet clear. What is already clear is that these approaches will add to the complexity of conventional protocols. This thesis has contributed a new result: game theoretic techniques can yield **practical** benefits to system performance. We have demonstrated the kinds of energy variability improvements that future designs might expect to realize. Now that the concepts have been introduced, the long road of engineering optimization begins.

Bibliography

Anderegg, L. and Eidenbenz, S. (2003). Ad hoc-VCG: A Truthful and Cost-Efficient Routing Protocol for Mobile Ad hoc Networks with Selfish Agents. In *Proceedings of the Ninth Annual International Conference on Mobile Computing and Networking (MobiCom '03)*, pages 245–259, San Diego, California.

Awerbuch, B. (1987). Optimal Distributed Algorithms for Minimum Weight Spanning Tree, Counting, Leader Election and related problems. In *Proceedings of the Nineteenth Annual ACM Conference on Theory of Computing*, pages 230–240, New York, New York.

Axelrod, R. M. (1984). *The Evolution of Cooperation*. Basic Books, Inc., New York.

Banerjee, S. and Misra, A. (2002). Minimum Energy Paths for Reliable Communication in Multi-hop Wireless Networks). In *Proceedings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2002)*, pages 146–156, Lausanne, Switzerland.

Barfield, W. and Caudell, T. (2001). *Fundamentals of Wearable Computers and Augmented Reality*. Lawrence Erlbaum Associates.

Broch, J., Maltz, D. A., Johnson, D. B., Hu, Y.-C., and Jetcheva, J. (1998). A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '98)*, pages 85–97, Dallas, Texas.

Buchegger, S. and Boudec, J.-Y. L. (2002). Performance Analysis of the CONFIDANT Protocol (Cooperation of Nodes: Fairness in Dynamic Ad-hoc NeTworks). In *Proceedings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2002)*, pages 226–236, Lausanne, Switzerland.

Buttyán, L. and Hubaux, J.-P. (2000). Enforcing Service Availability in Mobile Ad-Hoc WANs. In *Proceedings of the First Annual Workshop on Mobile and Ad Hoc Networking and Computing (MobiHOC 2000)*, pages 87–96, Boston, Massachusetts.

Buttyán, L. and Hubaux, J.-P. (2001a). Nuglets: a Virtual Currency to Stimulate Cooperation in Self-Organized Mobile Ad Hoc Networks. Technical Report DSC/2001/001, Institute for Computer Communications and Applications, Department of Communications Systems, Swiss Federal Institute of Technology — Lausanne.

Buttyán, L. and Hubaux, J.-P. (2001b). Stimulating Cooperation in Self-Organizing Mobile Ad Hoc Networks. Technical Report DSC/2001/046, Institute for Computer Communications and Applications, Department of Communications Systems, Swiss Federal Institute of Technology — Lausanne.

Čagalj, M., Hubaux, J.-P., and Enz, C. (2002). Minimum-Energy Broadcast in All-Wireless Networks: NP-Completeness and Distribution Issues). In *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking (MobiCom 2002)*, pages 172–182, Atlanta, Georgia.

Chen, B., Jamieson, K., Balakrishnan, H., and Morris, R. (2001). Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks. In *Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking (MobiCom '01)*, pages 85–96, Rome, Italy.

Chipcon (2004). *CC1100 Single Chip Very Low Power RF Transceiver Data Sheet*. Chipcon AG. Rev. 2.2.

Clarke, E. H. (1971). Multipart pricing of public goods. *Public Choice*, 11:17–33.

Clausen, T. and Jacquet, P. (2003). *Optimized Link State Routing Protocol (OLSR)*. IETF Mobile Ad Hoc Networking Working Group. IETF Request for Comments 3626, `draft-ietf-manet-olsr-11.txt`.

Cruz, R. L. and Santhanam, A. V. (2003). Optimal Routing, Link Scheduling and Power Control in Multi-hop Wireless Networks. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom 2003)*, pages 702–711, San Francisco, California.

Dorsey, J. G. (2001). *Game-Theoretic Resource Allocation in Mobile Ad Hoc Networks*. PhD thesis prospectus, Carnegie Mellon University.

Dorsey, J. G. (2004). The Spot Wearable Computer: Hardware Revision 3. Technical report, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Dorsey, J. G. and Siewiorek, D. P. (2002). Online Power Monitoring for Wearable Systems. In *Proceedings of the Sixth International Symposium on Wearable Computers*, pages 137–138, Seattle, Washington.

Ebert, J.-P., Burns, B., and Wolisz, A. (2002). A trace-based approach for determining the energy consumption of a WLAN network interface. In *Proceedings of European Wireless 2002*, Florence, Italy.

Eidenbenz, S., Kumar, V. S. A., and Zust, S. (2003). Equilibria in Topology Control Games for Ad Hoc Networks. In *Proceedings of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing (DIALM-POMC'03)*, pages 2–11, San Diego, California.

Fall, K. and Varadhan, K. (1998). ns *Notes and Documentation*. The VINT Project.

Faloutsos, M. and Molle, M. (1995). Optimal Distributed Algorithm for Minimum Spanning Trees Revisited. In *Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 231–237, Ottowa, Ontario, Canada.

Feeney, L. M. and Nilsson, M. (2001). Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment. In *Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2001)*, pages 1548–1557, Anchorage, Alaska.

Feigenbaum, J., Papadimitriou, C., Sami, R., and Shenker, S. (2002). A BGP-based Mechanism for Lowest-Cost Routing. In *Proceedings of the Twenty-First ACM Symposium on Principles of Distributed Computing (PODC 2002)*, pages 173–182, Monterey, California.

Feigenbaum, J. and Shenker, S. (2002). Distributed Algorithmic Mechanism Design: Recent Results and Future Directions. In *Proceedings of the Sixth International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 1–13, Atlanta, Georgia.

Ferguson, N. (1993). Single Term Off-Line Coins. Technical Report CS-R9318, CWI (Centre for Mathematics and Computer Science).

Ferguson, N. (1994). Extensions of Single-term Coins. In Stinson, D. R., editor, *Advances in Cryptology — CRYPTO '93*, pages 292–301. Springer-Verlag.

Gafni, E. (1985). Improvements in the Time Complexity of Two Message-Optimal Election Algorithms. In *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing*, pages 175–185, Minaki, Ontario, Canada.

Gallager, R. G., Humblet, P. A., and Spira, P. M. (1983). A Distributed Algorithm for Minimum-Weight Spanning Trees. *ACM Transactions on Programming Languages and Systems*, 5:66–77.

Groves, T. (1973). Incentives in teams. *Econometrica*, 41:617–631.

Heindl, A. and German, R. (2001). Performance modeling of IEEE 802.11 wireless LANs with stochastic Petri nets. *Performance Evaluation*, 44:139–164.

Hershberger, J. and Suri, S. (2001). Vickrey Prices and Shortest Paths: What is an edge worth? In *Proceedings of the 42nd IEEE symposium on Foundations of Computer Science (FOCS'01)*, pages 252–259, Las Vegas, Nevada.

Hu, Y.-C. and Johnson, D. B. (2000). Caching Strategies in On-Demand Routing Protocols for Wireless Ad Hoc Networks. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (MobiCom 2000)*, pages 231–242, Boston, Massachusetts.

Huang, L. and Lai, T.-H. (2002). On the Scalability of IEEE 802.11 Ad Hoc Networks. In *Proceedings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2002)*, pages 173–182, Lausanne, Switzerland.

IEEE (1997). *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications (IEEE Std. 802.11-1997)*. LAN MAN Standards Committee of the IEEE Computer Society.

IEEE (1999). *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher-Speed Physical Layer Extension in the 2.4GHz Band (IEEE Std. 802.11b-1999)*. LAN MAN Standards Committee of the IEEE Computer Society.

IEEE (2003). *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, Amendment 4: Further Higher Data Rate Extension in the 2.4GHz Band (IEEE Std. 802.11g-2003)*. LAN MAN Standards Committee of the IEEE Computer Society.

Jain, R. (1991). *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc.

Jetcheva, J. G. (2004). *Adaptive Demand-Driven Multicast Routing in Multi-Hop Wireless Ad Hoc Networks*. PhD thesis, Carnegie Mellon University School of Computer Science, Pittsburgh, Pennsylvania.

Johnson, D. B. and Maltz, D. A. (1996). Dynamic Source Routing in Ad Hoc Wireless Networks. In Imielinski, T. and Korth, H., editors, *Mobile Computing*, chapter 5, pages 153–181. Kluwer Academic Publishers.

Johnson, D. B., Maltz, D. A., and Hu, Y.-C. (2003). *The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)*. IETF MANET Working Group. Internet-Draft, `draft-ietf-manet-dsr-09.txt`.

Jung, E.-S. and Vaidya, N. H. (2002). An Energy Efficient MAC Protocol for Wireless LANs. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom 2002)*, pages 1756–1764, New York, New York.

Kamerman, A. and Monteban, L. (1997). WaveLAN-II: A high-performance wireless LAN for the unlicensed band. *Bell Labs Technical Journal*, 2.

Kastner, R., Hsieh, C., Potkonjak, M., and Sarrafzadeh, M. (2002). On the Sensitivity of Incremental Algorithms for Combinatorial Auctions. In *Proceedings of the Fourth IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS'02)*, pages 81–88, Newport Beach, California.

Kawadia, V. and Kumar, P. R. (2003). Power Control and Clustering in Ad Hoc Networks. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom 2003)*, pages 459–469, San Francisco, California.

Kraut, R. E., Sunder, S., Morris, J., Telang, R., Filer, D., and Cronin, M. (2002). Markets for Attention: Will Postage for Email Help? In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW 2002)*, pages 206–215, New Orleans, Louisiana.

Lehmann, D., O'Callaghan, L. I., and Shoham, Y. (1999). Truth Revelation in Approximately Efficient Combinatorial Auctions. In *Proceedings of the First ACM Conference on Electronic Commerce (EC-99)*, pages 96–102, Denver, Colorado.

Lloyd, E. L., Liu, R., Marathe, M. V., Ramanathan, R., and Ravi, S. S. (2002). Algorithmic Aspects of Topology Control Problems for Ad hoc Networks). In *Proceedings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2002)*, pages 123–134, Lausanne, Switzerland.

Lucent (1998). *IEEE 802.11 WaveLAN PC Card User's Guide*. Lucent Technologies.

Maltz, D. A., Broch, J., Jetcheva, J., and Johnson, D. B. (1999). The Effects of On-Demand Behavior in Routing Protocols for Multi-Hop Wireless Ad Hoc Networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1439–1453.

Maltz, D. A., Broch, J., and Johnson, D. B. (2000). Quantitative Lessons From a Full-Scale Multi-Hop Wireless Ad Hoc Network Testbed. In *Proceedings of the Second IEEE Wireless Communications and Networking Conference (WCNC 2000)*.

Martin, T. L. (1999). *Balancing Batteries, Power, and Performance: System Issues in CPU Speed-Setting for Mobile Computing*. PhD thesis, Carnegie Mellon University Department of Electrical and Computer Engineering, Pittsburgh, Pennsylvania.

Mas-Collel, A., Whinston, M. D., and Green, J. R. (1995). *Microeconomic Theory*. Oxford University Press.

Métivier, Y., Mosbah, M., Wacrenier, P.-A., and Gruner, S. (2001). A Distributed Algorithm for Computing a Spanning Tree in Anonymous T-prime Graphs. In *Proceedings of the International Conference on Principles of Distributed Systems (OPODIS 2001)*, pages 141–158, Manzanillo, Mexico.

Monarch (1999). *The CMU Monarch Project's Wireless and Mobility Extensions to* ns. Computer Science Department, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Myerson, R. B. (1983). Mechanism design by an informed principal. *Econometrica*, 51:1767–1797.

Nisan, N. and Ronen, A. (2000). Computationally Feasible VCG Mechanisms. In *Proceedings of the Second ACM Conference on Electronic Commerce (EC-00)*, pages 242–252, Minneapolis, Minnesota.

Osborne, M. J. and Rubinstein, A. (1994). *A Course in Game Theory*. The MIT Press.

Park, V. D. and Corson, M. S. (1997). A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In *Proceedings of the Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '97)*, pages 1405–1413.

Parkes, D. C. (2001). *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*. PhD thesis, University of Pennsylvania.

Parkes, D. C., Kalagnanam, J., and Eso, M. (2001). Achieving Budget-Balance with Vickrey-Based Payment Schemes in Combinatorial Exchanges. Technical Report RC 22218 W0110-065, IBM Research, Yorktown Heights, New York.

Perkins, C. E. (2001). *Ad Hoc Networking*. Addison-Wesley.

Perkins, C. E., Belding-Royer, E. M., and Chakeres, I. (2003). *Ad Hoc On Demand Distance Vector (AODV) Routing*. IETF Mobile Ad Hoc Networking Working Group. Internet-Draft, `draft-perkins-manet-aodvbis-00.txt`.

Perkins, C. E. and Bhagwat, P. (1994). Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *Proceedings of the ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244.

Perkins, C. E. and Royer, E. M. (1999). Ad-hoc On-Demand Distance Vector Routing. In *Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '99)*, pages 90–100.

Perlman, R. (1985). An Algorithm for Distributed Computation of a Spanning Tree in an Extended LAN. In *Proceedings of the Ninth Symposium on Data Communications*, pages 44–53, Whistler Moutain, British Columbia, Canada.

Plummer, D. C. (1982). *An Ethernet Address Resolution Protocol – or – Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware*. IETF Network Working Group. IETF Request for Comments 826.

Przytycka, T. and Higham, L. (1996). Optimal Cost-Sensitive Distributed Minimum Spanning Tree Algorithm. In *Proceedings of the Fifth Scandanavian Workshop on Algorithm Theory*, Reykjavík, Iceland.

Qiu, Y. and Marbach, P. (2003). Bandwidth Allocation in Ad Hoc Networks: A Price-Based Approach. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom 2003)*, San Francisco, California.

Rappaport, T. S. (1996). *Wireless Communications: Principles and Practice*. Prentice Hall PTR.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Sandholm, T. (2002). Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54.

Sandholm, T. (2003). BOB: Improved winner determination in combinatorial auctions and generalizations. *Artificial Intelligence*, 145(33–58).

Sandholm, T., Suri, S., Gilpin, A., and Levine, D. (2002). Winner Determination in Combinatorial Auction Generalizations. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 69–76, Bologna, Italy.

Schurgers, C., Tsiatsis, V., Ganeriwal, S., and Srivastava, M. (2002). Topology Management for Sensor Networks: Exploiting Latency and Density). In *Proceedings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2002)*, pages 135–145, Lausanne, Switzerland.

Shaffer, J. (2003). Mobility on a Large Scale Wireless Network. Unpublished technical report, Carnegie Mellon University.

Shaffer, J. and Siewiorek, D. P. (2003). Locator@CMU a Wireless Location System for a Large Scale 802.11b Network. In *Proceedings of the International Conference on Wireless Networks (ICWN '03)*, pages 61–65, Las Vegas, Nevada.

Shih, E., Bahl, P., and Sinclair, M. J. (2002). Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices). In *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking (MobiCom 2002)*, pages 160–171, Atlanta, Georgia.

Shneidman, J. and Parkes, D. C. (2003). Overcoming Rational Manipulation in Mechanism Implementations. Unpublished work, Harvard University.

Smith, A., Balakrishnan, H., Goraczko, M., and Priyantha, N. B. (2004). Tracking Moving Devices with the Cricket Location System. In *Proceedings of the Second International Conference on Mobile Systems, Applications, and Services (MobiSys 2004)*, pages 190–202, Boston, Massachusetts.

Srinivasan, V., Nuggehalli, P., Chiasserini, C. F., and Rao, R. R. (2002). Energy Efficiency of Ad Hoc Wireless Networks with Selfish Users. In *Proceedings of European Wireless 2002*, Florence, Italy.

Srinivasan, V., Nuggehalli, P., Chiasserini, C. F., and Rao, R. R. (2003). Cooperation in Wireless Ad Hoc Networks. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom 2003)*, San Francisco, California.

Stemm, M. and Katz, R. H. (1997). Measuring and Reducing Energy Consumption of Network Interfaces in Hand-Held Devices. *IEICE Transactions on Communications*, E80-B(8):1125–1131.

Tanenbaum, A. S. (1996). *Computer Networks*. Prentice Hall PTR.

Tel, G. (2000). *Introduction to Distributed Algorithms*. Cambridge University Press.

Tseng, Y.-C., Hsu, C.-S., and Hsieh, T.-Y. (2002). Power-Saving Protocols for IEEE 802.11-Based Multi-Hop Ad Hoc Networks. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom 2002)*, pages 200–209, New York, New York.

Vickrey, W. (1961). Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37.

Xu, Y., Heidemann, J., and Estrin, D. (2000). Adaptive Energy-Conserving Routing for Multihop Ad Hoc Networks. Technical Report USC/ISI TR-2000-527, University of Southern California Information Sciences Institute, Marina del Rey, California.

Xu, Y., Heidemann, J., and Estrin, D. (2001). Geography-informed Energy Conservation for Ad Hoc Routing. In *Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking (MobiCom '01)*, pages 70–84, Rome, Italy.

Zheng, R. and Kravets, R. (2003). On-demand Power Management for Ad Hoc Networks. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom 2003)*, San Francisco, California.

Zhong, S., Chen, J., and Yang, Y. R. (2003). Sprite: A Simple, Cheat-Proof, Credit-Based System for for Mobile Ad-Hoc Networks. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom 2003)*, San Francisco, California.

Zussman, G. and Segall, A. (2003). Energy Efficient Routing in Ad Hoc Disaster Recovery Networks. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom 2003)*, San Francisco, California.

Index