UNIVERSITY OF CALIFORNIA,
IRVINE

MINA: A Multinetwork INformation Architecture in Cyber Physical Systems/Internet of Things Environments

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Networked Systems

by

Zhijing Qin

Dissertation Committee:
Professor Nalini Venkatasubramanian, Chair
Professor Marco Levorato
Professor Sharad Mehrotra
Professor Amelia C Regan

2015

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to thank my advisor, professor Nalini Venkatasubramanian, for her advice and guidance during my Ph.D. study at University of California, Irvine. She made invaluable contributions not just to my PhD research and future career, but also personality development. Her broad vision and direction guided and pushed me to always attempt my best for discovering research topics that are novel, realistic, and useful in real life. She helped me to participate in internship programs to exploit different directions of my thesis. She always taught me to practice more on oral and written communication skills, which are valuable treasures that I can get benefits even after I graduate. I was also encouraged to attend International conferences and events to present our work, enhance the visibility of our work and build the social network. All those are critical things for me to achieve career successes in future.

I would also like to thank Dr. Grit Denker at SRI International for her guidance and helps during my internships. In the first two stays at SRI, I started to study formal language and use Maude to analyse the multinetwork information. During the third stay, I developed a SDN based flow scheduling algorithms. Both works are the key contributions in my PhD theis. Grit put a lot of effort on helping me develop ideas, correct proposed systems and algorithms, and clean up writing in any single paper during my stay.

I appreciate support from my committee members Professor Sharad Mehrotra, Amelia C Regan, Marco Levorato. They gave me valuable comments and feedbacks during each meeting and my topic defense exam. Those comments and feedbacks are very important for me to generate a high quality PhD dissertation.

I would like to thank Luca Iannario from University of Bologna. Luca implemented part of MINA prototype during his visiting time in our lab. Without him, I would spend much more time on the MINA system implementation. A portion of the content in implementation chapter refers to his master thesis which is advised by me and Nalini.

I also would give my thanks to my lab mates and colleagues, Ngoc Do, Ye Zhao, Reza Rahimi, Kyle Bension and Ranga Raj. I obtained helps nearly from every steps during my research: we study together how to use simulator; we discuss ideas and algorithms; we polish paper for others, etc. I really enjoyed the collaboration and atmosphere in this lab, and believe this contributed significantly to my research's achievements.

Finally, I would like to especially thank my family for giving me continuous love and care. I could not have done it without them and their supports.

CURRICULUM VITAE

Zhijing Qin

EDUCATION

Doctor of Philosophy in Computer Science **2009–2015**
University of California, Irvine *Irvine, CA*

Master of Software Engineering **2007–2009**
Peking University *Beijing, China*

Bachelor of Software Engineering **2003–2007**
Beihang University *Beijing, China*

WORKING EXPERIENCE

Graduate Research Assistant **2009–2014**
University of California, Irvine *Irvine, California*

Software Engineer Intern **2014.6-2014.9**
Vmware, Inc *Palo Alto, California*

Student Associate **2013.7–2013.9**
SRI International *Menlo Park, California*

Student Associate **2012.7–2012.9**
SRI International *Menlo Park, California*

Student Associate **2011.6–2011.9**
SRI International *Menlo Park, California*

PUBLICATIONS

The art of Wireless Sensor Networks **2014**
Book Chapter, Springer Berlin Heidelberg

A Software Defined Networking Architecture for the Internet-of-Things **2014**
IEEE/IFIP NOMS

MINA: A Multinetwork INformation Architecture for Managing Heterogeneous Pervasive Networks **2014**
IEEE/IFIP NOMS

Software-Defined Cyber-Physical Multinetworks **2014**
IEEE ICNC

Smart Communications via a Tree-based Overlay for Multiple and Heterogeneous Spontaneous Networks **2013**
IEEE SaCoNet

Modeling Path Duration Time in Dynamic Convergecast Network **2013**
IEEE WCNC

Achieving Resilience of Heterogeneous Networks Through Predictive, Formal Analysis **2013**
ACM HiCoNS

SOFTWARE

MINA
Multinetwork Management Software Suits JavaAndroid

ABSTRACT OF THE DISSERTATION

MINA: A Multinetwork INformation Architecture in Cyber Physical Systems/Internet of Things Environments

By

Zhijing Qin

Doctor of Philosophy in Networked Systems

University of California, Irvine, 2015

Professor Nalini Venkatasubramanian, Chair

Recent advances in embedded computing, networking, and related information technologies have made it feasible to create Instrumented Cyber Physical Spaces (ICPSs) that are technologically cutting-edge and exemplary of highly instrumented spaces of the future. Such ICPSs integrate a variety of sensing devices to create a digital representation of the evolving physical world that can then be exploited by applications for a variety of purposes. The networking landscape of today, especially in Instrumented Cyber Physical Spaces, is characterized by diverse access technologies including cellular, WiFi, Ethernet, MANETs, and ZigBee, and properly managing this heterogeneous networking infrastructure is a key challenge to take full advantage of its many opportunities.

In this thesis, we propose MINA (Multinetwork INformation Architecture), a reflective (self-observing and adapting) middleware approach to realize and manage dynamic and heterogeneous multi-networks in CPS/IoT environments. A novel aspect of MINA is that it embodies an Observe-Analyze-Adapt (OAA) loop to a) achieve a reasonably accurate, centralized global view of the multi-network through the design of novel techniques for overlay structuring, network state collection, b) employ formal methods to perform network state analysis and answer questions in network management domain such as: which nodes are more

important and should be protected? and c) take advantage of the global view for adapting multi-network structure by reallocating application flows across networks and proactively planning and deploying additional network resources. For each step in the OAA loop, we have proposed, implemented and evaluated several sophisticated methodologies to achieve certain objectives on the networking simulator platform. To demonstrate the validation of MINA, we designed, implemented and thoroughly evaluated a set of software over multiple networks and platforms. The results shows that MINA actually improves the exploitation of multi-network capabilities.

Chapter 1

Introduction

In this thesis, we study several research problems to enable the management of multiple networks in Cyber Physical System/Internet of Things (CPS/IoT) environments. Technology advances have created a wide range of computing devices equipped with different sensing, computing and networking capabilities. Today, consumers can use smart phones to capture different types of sensor data communicated to the cloud via different network interfaces; cameras can provide surveillance coverage in a Cyber Physical Space; vehicles can share road trip information via multiple hop short range communications, etc. Such kind of heterogeneity poses significant challenges on efficient and effective network management. This dissertation tries to identify those challenges and overcomes them accordingly via different methodologies. In this chapter, we begin by presenting the motivation behind our research in Sec. 1.1. We then state our thesis problem, research challenges and lay out the scope of work within the thesis in Sec. 1.2. Finally, we highlight our contributions in Sec. 1.3.

1.1 Motivation

Technology advances in sensing, architecture and communication in Cyber Physical Spaces are creating a rich and complex networking scenario characterized by an increasing number of pervasive devices equipped with notable computational hardware, multiple communication interfaces, and diverse sensing capabilities. For instance, today's personal devices (smartphones, tablets) are multifunctional sensing, storage, computation, and communication platforms. They provide multiple connectivities (e.g. UMTS/HSDPA/LTE,IEEE 802.11X, BlueTooth) to support trade-offs between connectivity, bandwidth and economic costs. They balance connectivity, bandwidth and economic costs by exploiting multiple connectivities (e.g. UMTS/HSDPA/LTE, IEEE 802.11a/b/g/n, BlueTooth).Other examples include: fixed surveillance cameras or mote devices for environmental sensing; personalized body-area sensing platforms for healthcare;smart vehicles that sense and communicate with other vehicles and roadside devices; and general-purpose mobile devices for participatory sensing. Such devices utilize heterogeneous, often intermittent networks at the edge (ZigBee, BlueTooth, PANs, MANETs, IEEE DSRC, mesh) that interface with higher capacity, relatively fixed backbone networks (e.g wired IP, 3G/4G, WLAN, WiMax, satellite). Today's devices can also provide connectivity in a peer-to-peer unplanned manner to create mobile hotspots and share connectivity with nearby devices. Such diverse capabilities enable connectivity and communication on a broader scale than ever before, including crowdsensing [42], mobile social networks [72], peer-oriented media sharing [87], lifelogs [56] and so on [11, 76, 9].

Such multi-network scenarios represent several notable advantages for end consumers, network providers, and network administrators. For instance, users may exploit multiple networking opportunities to reliably access remote services and seamlessly share resources. More pervasive services can be designed and provided generally regardless of the specific network environment at run time. From a provider perspective, the ability to deploy multiple diverse

networks simultaneously increases capacity and facilitates the satisfaction of a larger number of traffic requests without deployment of additional equipment. Network administrators can get a unified view above the multiple physical networks and can also better manage the network through the cooperation of multiple networks.

However, pushing the communication envelope toward the exploitation of any and all available networks has its challenges, primarily due to the diverse nature of traffic and the distributed, dynamic nature of the multiple connectivities. First of all, changes, such as those induced by mobility or newly sensed events, are frequent and can reduce communication reliability and information quality in traditional network architectures. Second, Failures, such as those triggered by a natural disaster, can cause significant loss of connectivities at critical moments. Third, network resources are provisioned in an isolated (for a single network) and mission-oriented manner, with little or no visibility for the whole network topology or state.

Our past experiences dealing with heterogeneous networks (within the Irvine Sensorium infrastructure at UC Irvine [52], disaster response drills with local California agencies [83, 81] and the RAMP spontaneous networking platform developed at U. of Bologna [12]) have indicated that changes, such as those induced by mobility or newly sensed events, are frequent and can reduce communication reliability and information quality in traditional network architectures. Failures, such as those triggered by a natural disaster, can cause significant loss of connectivities [28] at critical moments. Often, network resources are provisioned in an isolated (for a single network) and mission-oriented manner, with little or no visibility for the whole network topology or state. The exploitation of multi-network resources was often uncoordinated; static offline network planning strategies are not designed for dynamic deployment of connectivities when and where they are needed. All of this points towards a need for a structured approach to multinetwork management.

1.2 Thesis Problems, Challenges, and Scope

1.2.1 Problems

In this thesis, we propose MINA (Multi-network INformation Architecture), a middleware approach that realizes and manages dynamic and heterogeneous multi-networks in pervasive environments. MINA has a centralized server that collects network information from each client device and sends hints/commands back after some intelligent analysis to maintain the effectiveness and robustness of the whole network. More specifically, we aim to solve three major problems in MINA:

Efficient Topology Management with Heterogenous Networks and Devices through Network Monitoring. CPS/IoT spaces consist of heterogeneous devices characterized by mobility and dynamicity. Devices move around incurring potential communication handover and resulting in changes to routes/paths to the moving nodes. THe MINA server collects network state information from many stationary and mobile devices using a Convergecast communication pattern - this requires changes to existing topology management and network monitoring mechanisms.

Efficient Modeling and Analysis of Multinetwork State Information. Once the MINA server has network state information from a huge number of devices, one may pose the question of what can be derived from this collected data? Given the dynamicity of the networking environments, how quickly should we derive this knowledge? From a scalability perspective, what does the analysis look like when the scale of the network grows?

Flexible Control and Adaptation on Multinetworks. The ultimate goal of MINA is to be able to control and adapt the multinetwork cooperatively based onthe specifics of knowledge that can be derived, to facilitate a broad range of applications executing over multinetworks. Our final problem is the design of a flexible framework that is able to meet the

individual goals of heterogeneous applications given the heterogeneous nature of underlying infrastructure.

1.2.2 Challenges

There are challenges in each problem mentioned above.

Challenges in Topology Management and Network Monitoring:

- *Heterogeneity*: Devices in Multinetwork Environments have different computing, sensing and networking capabilities. Some devices have plug-in power supply and huge storage and high CPU frequency, while others have battery supply and limited storage and computation capabilities. Similarly some devices have reliable and high speed Ethernet connections, while some have lossy and unreliable wireless connections. Topology management and network monitoring design policies should take this heterogeneity into account, by pushing more computation workloads and network traffic aggregation points to the nodes having stronger capabilities. For less-capable devices such as sensors and smart phones, energy is a critical issue. Proper energy saving policies should be designed for those kinds of devices.

- *Mobility*: Mobility of sensors and smart phones cause communication links to go up and come down between devices and networking infrastructures (e.g. smart phone and wifi hotspot), devices and devices (e.g. bluetooth connections between two smart phones). Many mobility management mechanisms exist in infrastructure networks, such as cellular networks, to reduce the negative effect of link disconnection. In mobile ad hoc networks, self-organized protocols are designed so that a node can always find another one via message changes over neighbors. However, those solutions do not work well in multinetwork environments given both infrastructure and ad hoc connections often

coexist. Different topology management mechanisms may be deployed based on the availability of underlying networks accordingly. Further, to enable centralized network information analysis, topology management should be handled by a centralized server. In this case, additional to be considered include scalability and traffic aggregation.

Challenges in Modeling and Analysis of Information

- *Dynamicity*: The Multinetwork setting is highly dynamic. Constant movement of nodes results in frequent disconnections/reconnections of the communication network. Devices may not be accessible due to battery power depletion or malfunction, and new devices might be added in by new users or deployed by network administrators. Application flows start and end at arbitrary times. This level of dynamicity requires a highly available backend that is capable of quickly capturing and communicating Multinetwork state so that accurate analysis can be performed, especially from the temporal perspective. Another challenge here is the determination of which information is necessary and which is redundant or can be derived/predicted from existing information.

- *Scalability requirements*: Modeling and analysis require resources in terms of CPU, memory and bandwidth. MINA's centralized server csn accommodate network state information from a small size of devices. However, with the increasing number of devices in Multinetwork, both the changes of state and network information volume increase as well. It is fatal if there is no specific techniques dealing with scalability issue. Traditional solutions usually employ distributed clusters to scale the system. However, this will impose more complexity on both design and implementation.

Challenges in Mutinetwork Control and Adaptation

- *Application traffic heterogeneity*: Applications running over mutinetworks in CPS/IoT

environments vary - they include control information from power networks with small size data (a dozen of bytes) to images captured by road camera networks(megabytes), from voice data generated by fire alerts with a bursty pattern to video surveillance data with a flat rate. Application packet sizes and flow patterns have significant impacts on networking infrastructure, including wireless interference levels, loss rate, congestions, and inter flow contentions, etc. Taking this heterogeneity awareness into account allows the MINA server better profile the whole system and make reasonable adaptations back to the applications.

- *Various performance requirements and metrics*: Different applications have different performance requirements or metrics. File transfer applications require higher throughput; Tele audio applications require less delay; Video streaming applications require less jitter; Alert massages require low loss rate. Having a good understanding of network infrastructure and application profile, it is critical to take individual application requirements into account in the control and adaptation step.

1.2.3 Scope

To lend focus, we scope multinetwork management based on the following characteristics in this dissertation:

Tree-Based Overlay Construction and Path duration modeling: To deal heterogeneity and mobility of the underlying nodes, we proposed a Tier based architecture in which more stable and stationary nodes are placed on the upper tier while mobile nodes reside on the lower tiers. We further attempt to construct and maintain a tree based overlay with the centralized server as root, to manage the topology without incurring much network traffic overhead. In addition, we also studied a path duration model so that a mobile node can predict when a path towards to server is down and up so that it can control its neighbour

discovery heat beat messages to reduce network traffic overhead further without degrade the topology management effectiveness.

Formal Method Based Network State Information Analysis: We employed a light weighted, high level formal language to specify, execute, and reasoning our multinetwork, to deal with dynamicity and scalability issues. The high level features of the formal method enable us to analyze multinetwork very quickly regardless of networking cross layer communication details. More important, the increase of the network scale would not bring more complexity into the analysis.

Software Defined Networking Based Flow Scheduling: We adopted the novel Software Defined Networking (SDN) paradigm into Multinetwork control and adaptation. Specifically, we have a novel SDN controller designed for CPS/IoT environment. It acts like a bridge: it reads heterogeneous flow patterns from upper layer applications (north bound), matches them with the heterogeneous network links, it then reroutes the flows (south bound) via proper network links so that the individual flow requirements can be met.

1.3 Thesis Contributions and Organization

In this dissertation, we address the challenges in Multinetwork Management in CPS/IoT environments.

A key aspect of MINA is that it implements an Observe-Analyze-Adapt (OAA) loop to guide the configurations, state management and coordination of the multi-network (Section 3). The overall OAA flow empowers the multi-network to make communication decisions locally or higher up in the hierarchy using available knowledge of network status while accounting for tolerance parameters (timing, accuracy, reliability). In the **Observe** step (Section 4), we create and maintain a *hierarchical overlay structure* that captures the underlying networks'

dynamic and heterogeneous nature and employ a *path duration modelling* methodology that effectively predicts the path duration time in data collection scenarios. The **Analyze** step (Section 5) is augmented with a *formal methods driven approach* that supports both proactive and semantics-driven "what-if" analysis of the collected network state information. In the Adapt step, we borrowed the Software Defined Networking philosophy from Data Center Network to schedule heterogeneous application flows over heterogeneous networks in order to satisfy each individual flow's QoS requirements.

We point out that MINA is effectively a reflective middleware system for multi-network management; the OAA approach inherently embodies computational reflection principles [59]. The reflective feature in MINA is realized through the interaction between the MINA middleware (the meta-level) and the underlying multi-network environment (the base-level). MINA observes network state (implementing reification), analyses it to determine what is adapted and implements adaptations based on application context and observed network state (implementing reflection). The multi-network state is stored in a DB, as a meta-level representation of the underlying system state. The reflective approach is also a natural fit for the growing Software Defined Networking paradigm that aims to manage heterogeneous networks in an abstract, high-level, and logically centralized way. To the best of our knowledge, MINA is the first system to implement a reflective middleware approach and utilize on-the-fly, lightweight formal methods in the context of multi-network management.

The dissertation is organized as follows. In Chapter 2, we present existing work on network management systems and their limitations. We illustrated MINA design philosophy and architecure in Chapter 3. Chapter 4 is dedicated to presenting our overlay algorithms and path duration modelling techniques in observe step. In Chapter 5, we describe how we employ formal methods to perform analysis on multinetwork state information and how we use it to adapt the network. Chapter 6 shows how the Software Defined Networking techniques can be utilized in MINA to obtain better performance in terms of individual flow QoS fulfilment.

We described the details of MINA implementations in Chapter 7. Finally, in Chapter 8, we conclude the thesis and discuss some interesting future research directions.

Chapter 2

Related Work

There are bunch of related work in individual network management in the past several years.

Single Network Management The most famous work, which became a mature standard on Internet management, is Simple Network Management Protocol (SNMP). It is an application layer protocol that facilitates the exchange of management information between network devices. It denes a protocol for communication of queries and responses, and also set an agent for sending responses within a network [1]. SNMP operates on top of the UDP or TCP transport protocol and relies on a functioning IP layer to route information. In the Internet, the Internet Control Message Protocol (ICMP) provides a mechanism to send error messages when services on routers and hosts are not available.

LNMP [2] is a management architecture for IPv6 based low power wireless personal area networks [2]. It has an operational architecture that defines three kinds of nodes: end device, coordinator, and gateway. The coordinator collects network state information from end device and sends it to the gateway. Then the gateway will forward the information to the database. It also has information architecture which defines the state information structure for four layers: physical layer, mac layer, adaptation layer and network layer.

MeshMan[3] is a network layer agnostic, low overhead solution to network management to cope with unreliable wireless channels, link and network dynamics in wireless mesh networks. It combines the concepts of source routing with hierarchical addressing, and provides a native efficient query interface. Its core parts include: hierarchical addressing, adoption protocol, and management traffic routing.

SNSP [4] is a sensor network management system designed to be simple and have minimal impact on memory and network traffic, while remaining open and flexible. It is an application cooperative management system with two core services: a query system to enable rapid, user-initiated acquisition of network health and performance data; and a logging system to enable recording and retrieval of system-generated events.

Hybrid Network Communications: There are also some works focuses on communication on hybrid networks. MMHC [5] uses mobility/throughput/energy context to manage connectivity opportunities effectively on wireless and blue tooth hybrid networks. The key ideas are: i) of exploiting context data to reduce the space of potential candidates for selected connectivity opportunities and ii) of splitting management operations into a local phase and a global phase.

RAMP[6] is actually a service framework based on MMHC. It provides simple and wide-accepted communication abstractions for general purpose application need. Also it enables a set of services dynamically and temporarily offered by spontaneous network(BT, Wifi) peers.

UCAN[7] is Hybrid network of 3G and wireless ad hoc network for data delivery. The idea is 3G station forwards messages for nodes with poor link to proxy nodes with good link, and the proxy nodes then use ad hoc network to forward the data to the appropriate node.

Industrial Products on Network Management: Recent industrial efforts (e.g., Cisco Prime Infrastructure [23], HP OpenView [74], and SpiderCloud [88]) aim to address the

challenges of multi-networking and the integrated synergistic management of heterogeneous wireless networks. To date, these efforts have focused on exploiting lower-layer features specific to an access network or network-layer contexts, typically via new Layer 3 protocols [98, 92, 39, 65].

Limitations: For those work on individual network management, the limitation is they only focus on single network and cannot be applied in multinetwork environment. For those work on multinetwork communication, they do not provide management functions for the hybrid network they use. Even in MMHC which claims it has connectivity management, there are also no concrete functionalities for network management. Industrial products focused on low-layer features specific to access networks and vendor provided devices. Lack of analysis and control of the network from global perspective.

For industrial product, these solutions still manage networks in a low level, distributed, and vendor-specific manner which is error-prone and inefficient. In other words, State-of-the-art related approaches have not addressed effectively yet the need for effective scalability over large deployment environments, especially when dealing with global network management optimization goals in a lightweight and dynamic way.

Chapter 3

MINA Architecture and OAA Design Philosophy

In this section, we will describe the motivation and rationale behind the overall management architecture and system design of the MINA framework. While MINA realizes management functionalities similar to those of current network management platforms, it specifically focuses on addressing issues arising from heterogeneous networks in a pervasive computing environment. For instance, key tasks include performance management, configuration management, fault analysis and recovery, and network operations and security management. MINA's fundamental difference lies in its ability to perform the above tasks while handling the diversity and dynamic nature of the constituent network platforms.

Tier-Based Architecture and OAA Approach: The underlying systems managed by MINA are **dynamic, heterogeneous** and **large**. To support scalability to large number of diverse nodes, MINA is designed as a tiered, hierarchical architecture (see Fig. 3.1(a)). The higher levels of the tiered architecture are more stable (i.e. stationary) and resource-rich, aggregating information from other lower, more mobile and less stable nodes. At the

heart of the MINA system (Tier 1) is a centralized server that collects and analyzes the network state information from each device. Stationary, resource-capable nodes (e.g. those with Ethernet connections like routers, access points, and stationary PCs) are designated as Tier 2 nodes. Tier 3 typically consists of mobile nodes (e.g. smartphones, laptops, and tablets) that connect to Tier 2 nodes, either directly or by intermediate relay nodes, via multiple kinds of wireless radio networks. The MINA multi-tier architecture exploits the diverse capabilities of the network nodes, more effectively supporting node mobility. In fact, upper tier nodes assume a larger fraction of the computation and communication workload, while less capable (and more dynamic) mobile nodes are relegated to lower levels of the tiered architecture.

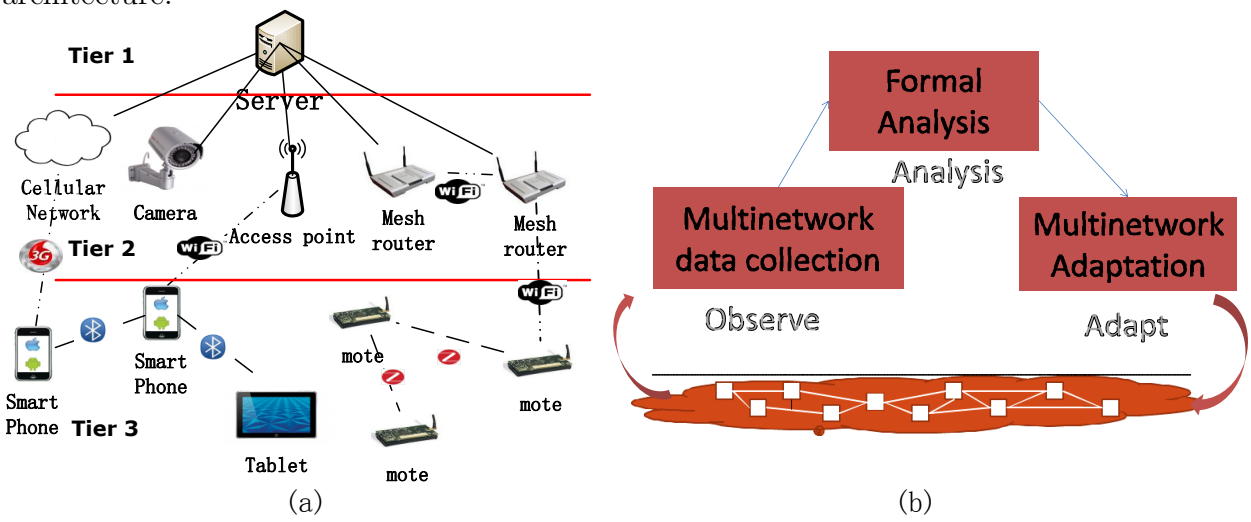


Figure 3.1: Tier-based Architecture and OAA Paradigm

To capture and support dynamicity, MINA's design is based on an Observe-Analyze-Adapt (OAA) approach (see Fig. 3.1(b)). A self-observing, introspecting system monitors the dynamically changing multi-network state, analyzes state information streams, and adapts the multi-network usage and configuration to ensure reliable communication functionality for the end applications. MINA enforces interactions between nodes in the tier to realize the OAA management approach.

The **Observe** step (see Chapter 4 for more details) collects network state information from

nodes in Tiers 2 and 3. The key objective here is to balance trade-offs between state accuracy (at the server) and state capture overhead. Accurate state capture requires reliable communication of state information despite dynamic connectivities. We employ reliable overlay construction mechanisms that exploit the relatively stable nature of Tier 2 nodes for this purpose. To reduce overhead for state capture, we devise quality-aware data collection protocols that only communicate state information when required and at granularities dictated by the applications at hand (e.g. QoS performance of flows). Collected state information is stored in a state database at the server.

The **Analyze** step (see Chapter 5) processes state information for the management tasks at hand (e.g. fault and performance management). While current systems largely rely on query-driven data analytics executed on the information in the network state DB, MINA introduces a novel approach based on *lightweight formal methods* to this step. MINA captures a formal executable specification of the multi-network system and instantiates the formal model using dynamic state information from the network state DB to conduct a semi-online reasoning of the current multi-network state for varied analyses. The formal methods approach captures and represents multi-network information at higher levels of abstraction to realize tremendous benefits. It simplifies analysis and allows us to conduct "what-if" analysis to suggest to network operators options which improve multi-network execution. It also allows us to embed higher level context and semantics in a simple manner, creating new possibilities for adaptations at levels outside of the network. And finally, it allows us to embed higher level context and semantics in a simple manner, creating new possibilities for adaptations at levels outside of the network. We will illustrate examples for achievement of the tasks above in our evaluation via real world case studies (see Chapter 5).

The **Adapt** step (see Chapter 6) leverages information from the analysis step to adapt the structure of the multi-network, flow of content, and augment the network with new devices and components through operator intervention. More specifically, MINA adopted the

Software Defined Networking paradigm from the Data Center Networks and a layered SDN controller is installed in MINA server, which is tailored for computing in Cyber Physical System. The primary objective in this step is to schedule flows with various QoS requirements above heterogeneous networks so that each individual flow's QoS requirements can be fulfilled.

The three steps described above are implemented through simple modules at the MINA client side (local state capture, local control for adaptations) and more powerful capabilities (global state assimilation, storage, analysis, logical adaptation etc.) at the MINA server. Delving into finer details, the **MINA workflow** can be summarized as follows. MINA's centralized server collects, monitors and analyzes state information from each device in the network. Typically, the centralized server resides on a device managed by a network administrator, eventually minimizing human operator involvement through formal methods and autonomous software components. One of the server's primary tasks is initiation and coordination of the construction of the overlay network that realizes the tiered/hierarchical architecture described above. Once the overlay is set up, the MINA server maintains the overlay and continuously collects network state information in order to actively monitor the managed environment; every client collaborates sending local state information to the server along the overlay links. Additionally, Tier 2 nodes (with more computing resources) may aggregate state information from Tier 3 nodes to reduce messaging overhead and bandwidth usage during state collection. Collected state persists in a **Network state Information Base (NIB)** for later analysis, e.g. to efficiently provision networks for various tasks, detect possible faults and unexpected behaviors, and apply formal reasoning methods for fault detection. Based on the results of the analysis, commands and hints for management of traffic and requests from nearby other devices may be sent to clients. Every client receives configuration hints or commands in order to (try to) recover from faults or to improve the overall network performance.

In the following Chapters, we will discuss the algorithms deployed in each step in details. And in Chapter 7, we will demonstrate the implementation of MINA Server, Client and Network Information Base.

Chapter 4

Observe:Topology Management and Network Monitoring

In this chapter, we study how MINA manages the multinetwork topology given network heterogeneity and mobility. More specifically, the MINA server constructs and maintains a tree-based overlay network following the tier-based approach (Fig. 3.1(a)) by dynamically evaluating nodes and placing them at the tier best fitting their capabilities. The overall goal is to ease the topology management and data collection in multiple heterogeneous networks.

4.1 Problem Description and Motivation:

Our past experiences dealing with heterogeneous networks in controlled instrumented environments (within the Irvine Sensorium infrastructure at UC Irvine, disaster response drills with local California agencies, and the RAMP spontaneous networking platform at U. of Bologna [13]) have yielded several observations [28]: a) Failures are likely to happen frequently in multi-networks, most of the time without the capability to diagnose the problem

on the spot, reconfigure computers, swap/recharge batteries, or change cables. b) Current network deployments (WiFi infrastructures, cellular, etc.) are sensitive to noise. Even limited network noise can cause a significant drop in information quality especially for rich media data; in such cases exploiting alternative ad-hoc and spontaneous communication can be beneficial. c) Changes to the underlying network topology, especially those changes induced by mobility, further reduce reliability. To address these challenges, it is of primary importance to efficiently and promptly spread state information among nodes in different networks.

Specifically, this chapter proposes the Tree-based Overlay over Multinetwork, whose main purpose is to ease the topology management of multiple heterogeneous networks. The MINA server creates and manages a tree-based overlay to proactively spread management information among nodes in inter networking and heterogeneous spontaneous networks; in this way, it can be exploited to efficiently achieve a global view of network conditions. This allows MINA to make more effective network management decisions based on the full knowledge of available nodes and their current connectivity capabilities. As one simple example, MINA can observe novel communication links or link failures; remote endpoints can take advantage of this kind of awareness to exploit alternative and more powerful paths as soon as they become available or avoid sending packets along paths with broken links.

Many research activities have already investigated overlay construction in both wired [58, 19] and wireless networks [77, 21]. On the one hand, unstructured overlays [58, 45, 57] do not impose a rigid relation between the overlay topology and where resources or their indices are stored. In this manner, overlay networks are easier to implement even in dynamic environments and at the cost of limited scalability. On the other hand, structured overlay networks [19, 18] impose a structure on the overlay topology by setting routing table entries to fit certain criteria depending on the respective Distributed Hash Tables (DHTs), which bound the looking up complexity as O(log n). However, the MINA overlay structure requires new

features: first of all, packet transmission is always sent to/from the central server, without any pure peer-to-peer connection between any two arbitrary mobile nodes. Moreover, overlay network construction mechanisms should be lightweight to limit resource consumption on mobile devices. Finally, mobile awareness should be taken into account to promptly and appropriately reconfigure the overlay network.

In anticipation of a few notable aspects of the proposed solution, we stress that our tree-based overlay network takes advantage of a hierarchical information architecture and of an implemented middleware for managing communication in heterogeneous multi-networks. The overlay network allows nodes to make communication decisions locally (or higher up in the hierarchy), using available knowledge of network state and taking into account tolerance parameters (timing, accuracy, reliability). The advantages of the proposed approach and its strong originality if compared with the state-of-the-art primarily relate to the efficiency over large deployment environments, especially when dealing with global network management optimization goals in a lightweight and dynamic way, as better illustrated in the following sections.

4.2 Tree-based Overlay Guidelines

Creating an overlay on the underlying heterogeneous topology of nodes/links allows us to maintain the collection topology at low cost, especially when node mobility comes into play. Note that the overlay approach is based on the RAMP middleware [12], which eases the task of integrating multiple networks regardless of the underlying heterogeneous link layer technologies and enables MINA to naturally use RAMP Node UIDs rather than IP addresses to organize nodes.

Our solution is based on three design criteria/goals for the overlay construction protocol that determines where nodes and links are positioned in the overlay structure. Lower overhead

is the first design criteria. Due to the limitation of mobile devices, it is important to make sure adding new nodes to the overlay incurs a limited overhead by involving only a limited number of nodes already in the overlay. In fact, the performance of the overlay should not dramatically change when the number of nodes in the network grows. Secondly, promptness. The overlay should rapidly react to changes in the underlying network topology (mobile nodes moving around causes great dynamicity) so that it does not incur large end-to-end delays on applications running above the overlay. Third, mobility awareness. The created overlay must accommodate the node mobility and intermittent connectivity that is characteristic of mobile nodes. We observe that the tree-based overlay structure fits well with mobility management in the tiered architecture, since it pushes the more dynamic nodes towards the leaves.

The above three criteria potentially conflict with each other since there is an inherent tradeoff between these goals. In traditional wireless networks, nodes implement a periodic heartbeat mechanism to discover changes in network topology, i.e., joining and leaving of neighbors. Frequent heartbeats make easier the accurate discovery of paths and decrease the end-to-end delay of data collection, but obviously with higher costs in terms of network resources. Prior work has explored a broad range of techniques (theoretical analysis [54], machine learning [30, 32], mobility prediction [64]) to adaptively tune the performance under such tradeoffs in ad-hoc sensor networks. However, there are several new challenges in supporting such tradeoffs in the MINA overlay: a) The convergecast pattern for data collection indicates nodes primarily care about maintaining a path to their parents. In contrast, solutions developed for MANET routing are intended for communication between two random nodes and hence are not suitable here. b) In heterogeneous networks, some nodes are stationary and some are mobile; as a result some links are inherently more stable than others. c) Nodes in Tier 3 are mobile and resource-limited: to avoid frequent overlay network modifications and limit their power consumption, they should not perform management tasks.

Prior to presenting the tree-based overlay construction protocol, we first define notations, data structures, and message types used in MINA overlay construction. The following functions are defined on a given node with unique identifier UID. *Neighbor* represents a node within a one-hop distance from a node UID. *IsParentCand* is true if the neighbor has offered to be a parent for a node UID (parent candidate). *IsParent* is true if the neighbor is currently the parent of this node, whereas *IsChild* is true if the neighbor is currently a child of this node. Furthermore, we store a *DescendantList* where each entry represents the set of descendants for a specific child of this node. *Generation* is an integer that is assigned to each node of the overlay, which is incremented by one from parent to child. The source node (root) has a generation number 0.

To better understand how the algorithm works, we also list and explain a set of important messages used in MINA between two possible A and B nodes. *Parent Claim Broadcast*: Once A finds a new parent it will broadcast this message to claim itself as a potential parent of its neighbors. B marks A as a parent candidate when B receives this message. The source node performs the broadcast by default. *Parent Request Unicast*: B selects the best parent (based on some criteria, e.g., lowest generation number as default) among parent candidates, and sends a parent request message to this selected best parent. *Parent Confirm Unicast*: When A receives the parent request message from B, if there is enough space to admit a new child, A will send a parent confirm message to B. Otherwise A will send a "parent refuse" message to B. *Parent Request Broadcast*: If B loses connectivity with its parent and currently there are no available parent candidates, it will broadcast a parent request message to find a new parent candidate. *Parent Accept Unicast*: when A receives a parent request broadcast message, if there is enough space to admit a new child, A will send parent accept messages to B and B will mark A as parent candidate. *Descendant Update Unicast*: when B admits a new child, it will send the ID of the new child to its parent; therefore, each node knows all its descendants.

During overlay construction, appropriate runtime data structures are initialized and the overlay construction procedure is initiated. Overlay maintenance executes mechanisms to handle dynamic changes to the existing overlay, e.g., node mobility or node failure that may trigger parent re-selection.

4.3 Overlay initialization and maintenance

Initialization is structured into two phases. In the first phase, nodes are iteratively added into the overlay from source node (root) to leaf nodes. The procedure begins with the source node broadcasting parent claim messages. Nodes that receive parent claim messages from parent candidates will respond with parent request messages to the "'best parent candidate"; when the parent confirmation message is received, the parent-child relationship is established and the node is added into the overlay. This process is repeated until propagated to the leafs of the tree. The second phase starts when a node has already chosen a parent and advertises itself to accept new children. When a node admits a new child, it will send a Descendant Update message to its parent; the parent then adds a record to its DescendantList and forwards the message to the upper level parent, until the root is reached. Note that a node only knows the UID of its children and all the descendants of each child; it does not need to know the exact topology of its descendants in the subtree. Hence, our proposal does not consume too much memory and bandwidth to maintain a complete topology map in each overlay node. For example, in Fig. 4.1 Node S only knows that nodes 3, 4, and 5 are below Node 2 (its children) but does not know in which order. To send a message to Node 5, Node S only needs to know that the next hop node is Node 2, while the latter only needs to know that it should forward the message to Node 3, and so on. Moreover, solid lines in Fig. 4.1 indicate handshake procedures including Parent Claim, Parent Request, and Parent Confirm messages, while dashed lines indicate only Parent Claim messages.

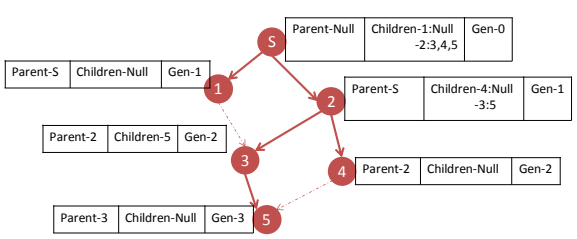


Figure 4.1: Overlay Initialization.

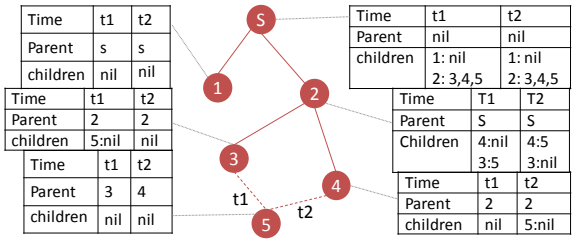


Figure 4.2: Overlay Maintenance.

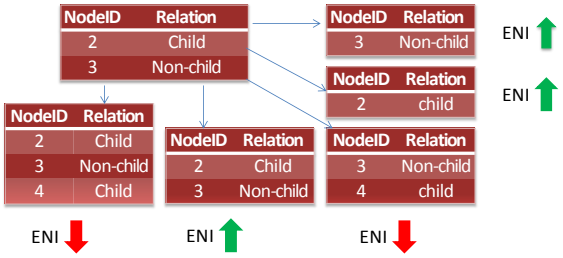


Figure 4.3: ENI Policy.

Once overlay construction is completed, data collection is underway. Because of possible node mobility and failures, additional mechanisms are required at runtime to dynamically react to changes in the underlying topology (overlay maintenance). Two key events to address are parent loss and child join. Every node periodically broadcasts a heartbeat message to its children. If a node does not receive a heartbeat message from its parent within a timeout, it will infer that its parent is not reachable anymore and will delete it from the neighborlist. If the node has other parent candidates, it will send Unicast Parent Request message to the best candidate; otherwise it will broadcast a parent request until a new parent is determined. Based on the generation number of the new parent, this node will decide to keep the parent-child relation with its current children or not (via explicit notice to children). By doing so, we can avoid the so called routing loop problem.

Child joining/leaving events are handled as follows. When a new child joins the overlay, the parent will add it into its children list immediately and propagate this information upwards. To limit the number of upward propagated messages, we exploit the following mechanism: when a node leaves the old parent and joins a new one, the Descendant Update message is

propagated only until it reaches the most recent ancestor of the new and the old parents. When a node receives the Descendant Update message, it will check whether this newly joined descendant already existed in other DescendantLists or not. If yes, this parent is the most recent ancestor and stops reporting this information further. Fig. 4.2 depicts the procedure: at time t1 Node 5 initially has a link to Node 3; when later (at t2) it joins Node 4, it updates its parent from 3 to 4; the descendant update is propagated only until Node 2.

4.4 Optimization

4.4.1 Expected Next Interval Messages

The basic MINA overlay construction and maintenance mechanisms ensure that the source node can collect information about all nodes and that each node has an accurate image of its descendants at runtime. In this section, we present our enhancements to the basic MINA overlay construction process in order to achieve the appropriate delay/overhead tradeoff in state collection. Recall that parent nodes use heartbeat messages to inform children of their existence; child nodes use a periodic neighborlist flush function to delete stale parent connections. On the sender side, increasing the rate of heartbeat messages will reduce end-to-end delays in data collection; however, the increased rate introduces additional messaging overhead. At the receiver end, a high neighborlist flush frequency will get rid of old routes; however, valid routes with longer lifetimes may also be deleted, which can result in increased collection delays.

To achieve the above goals, we have introduced a dedicated field to the heartbeat message, *Expected Next Interval* (ENI), that can provide hints to the recipient on when to expect the next heartbeat message. Upon receiving a heartbeat, a recipient uses the ENI as a timeout factor to determine when to flush the neighbour entry. The ENI is incremented/decremented

at the sender side based on link dynamicity; the rate of increase/decrease can be tuned, also dynamically, to meet application requirements (possibly changing at provisioning time). For example, to support low end-to-end delays as compared to lower overheads, we employ slow increase and fast decrease functions as follows:

$$Increase: \quad ENI_{current} = ENI_{current} + Delta;$$
$$Decrease: \quad ENI_{current} = Max\{ENI_{current}/2, ENI_{def}\};$$

where the initial value of $ENI_{current}$ is ENI_{def}. ENI changes are triggered when there are changes to the set of children (of a parent node). Specifically, we decrease ENI when the child set changes and increase ENI when the set of children of a parent node is relatively stable. The rationale is as follows. When a new child joins a parent, there is a reasonable likelihood that the child is mobile and will hence leave soon, triggering more changes. When a current child leaves, it is likely that the sender (parent) is mobile and, if so, other children are likely to leave as well. Fig. 4.3 depicts a simple example where the receiver side uses the ENI information in the heartbeat to determine "delete" or "keep" actions for entries in the neighborlist.

4.4.2 MINA Overlay Evaluations

We have conducted extensive experiments to evaluate our tree-based construction protocol, with and without ENI-based enhancements. By using QualNET[79] as the simulation platform, we compare the performance of our overlay with more traditional AODV[77] and DYMO[21] approaches, well recognized and widespread in MANET scenarios. In all our simulations we have used one fixed node that acts as source (server), 4 fixed nodes that act as Tier 2 nodes, and several mobile nodes scaling from 8 to 24 as Tier 3 nodes that move

around by following the random waypoint mobility model with speed ranges [1m/s,2m/s] and [10m/s,20m/s]. Each mobile node is configured to generate Constant Bit Rate traffic (periodically sending a 500 byte message to the server), thus emulating network state collection. Each simulation round has a 250s duration. We evaluate our techniques using three metrics: application end-to-end delay, message overhead, and delivery ratio. For ENI, we set $ENI_{def} = 7s$ and $Delta = 0.75s/1s$; motivations of this parameter settings are in the following sections.

Results are collected from the simulations under the random way point mobility model with speed range $[10, 20]$. Fig. 4.4 shows the heartbeat message overhead and end-to-end delay comparison between different ENI $Delta$ values. With ENI enabled, the heartbeat message overheads are greatly reduced (from 70% to 42%) as compared to the basic version of the overlay protocol; note that the end-to-end delay (with an average deviation of 1.2%) and delivery ratio (max. deviation of 1.6%) are hardly impacted by ENI modifications, as shown in Fig. 4.6. If we compare the end-to-end delay between ENI with $Delta = 0.75s$ and AODV/DYMO, our ENI-based algorithm reduces the message overhead a lot as compared to both AODV (from 60% to 26%) and DYMO (from 68% to 20%), as shown in Fig. 4.5. Similar results have been obtained under the same mobility model with lower speed range $[1, 2]$. In Fig. 4.7, enabling ENI with $Delta = 0.75s$ can reduce the message overhead by a range from 53% to 37%, and the range is even higher (78% to 44%) with $Delta = 1s$. The end-to-end delay is not changed much (with an average deviation of 5%) and the experienced delivery ratio is almost the same (with a maximum deviation of 2%) as shown in Fig. 4.9. Also if we use ENI with $Delta = 1s$, our overlay construction protocol generates many fewer messages than AODV and DYMO, as shown in Fig. 4.8. Note that although the MINA overlay techniques do not explicitly focus on reliable state collection, our solution inherently achieves high delivery ratios of over 96.5%.

As a general consideration, whether there is data to exchange among nodes or not, MINA

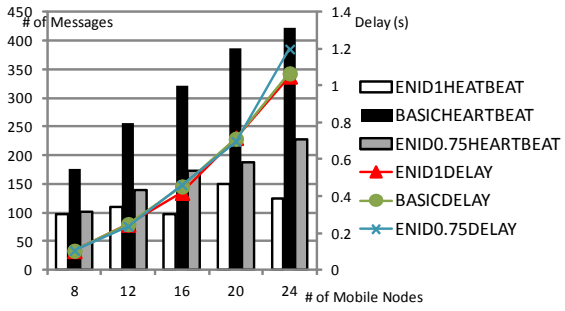


Figure 4.4: Overhead and Delay with Different ENIs in High Speed

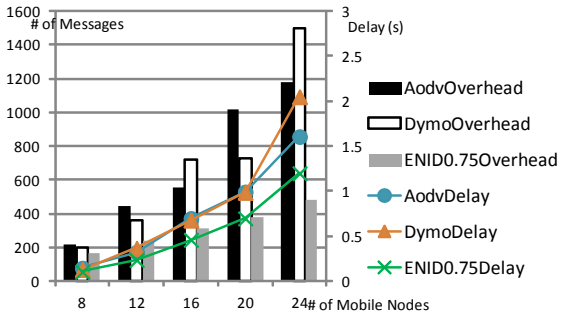


Figure 4.5: Comparison with AODV and DYMO in High Speed

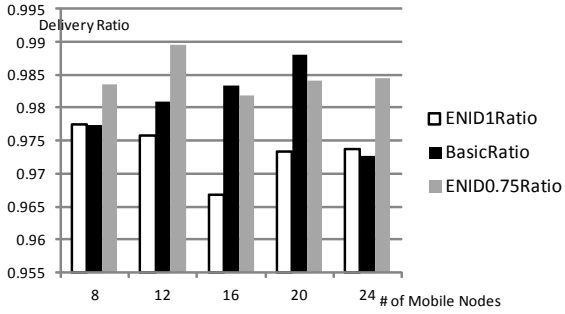


Figure 4.6: Delivery Ratio with different ENIs in High Speed

always initializes and maintains the overlay for multiple purposes. So, the queue delay (which is a major part of the end-to-end delay) is unlikely to happen. More specifically, the broadcast Parent Request message and the ENI-based enhancements can improve the end-to-end delay further. Fig. 4.5 and Fig. 4.8 show that MINA can provide a very small end-to-end delay (around 65%) compared to AODV and DYMO (which are on-demand), and with much less message overhead. We can find that our overlay, without and with ENI ($Delta = 0.75s$), has proportional heartbeat message overhead to the number of mobile devices, shown in Fig. 4.4 and Fig. 4.7. More interestingly, we also find that, when enabling ENI with $Delta = 1s$, the message overhead is almost constant: the reason is that when $Delta$ exceeds a threshold, the message sending interval is big enough that, when the node is about to send the next message, it misses some children changes, and then the ENI continues to increase (which may lead to inaccurate route and packet loss). Let us conclude this subsection by stating

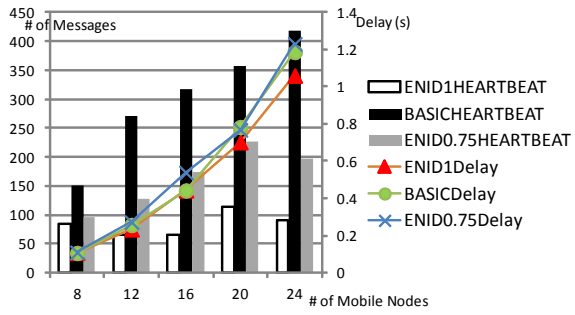


Figure 4.7: Overhead and Delay with Different ENIs in Low Speed

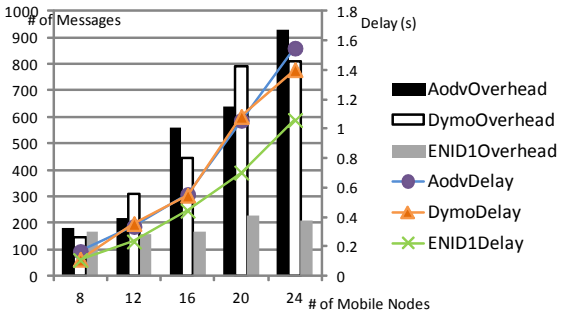


Figure 4.8: Comparison with AODV and DYMO in Low Speed

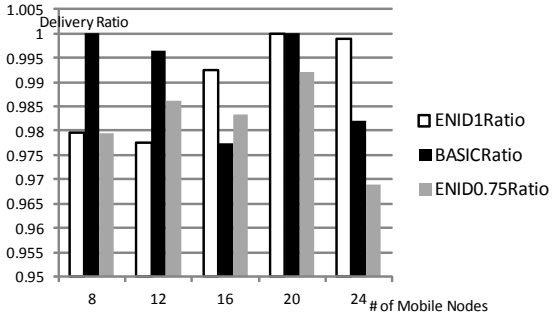


Figure 4.9: Delivery Ratio with different ENIs in Low Speed

that the choice of ENI_{def} and *Delta* cannot be arbitrary, since it determines the default heartbeat rate in the baseline MINA protocol. These values depend on average *path duration time* in the multi-network, which is a function of the number of nodes, mobility model, and transmission ranges. In next section, we will look deeper into this function. Here we estimate path duration times based on input from simulation scenarios: our measurements yield a default ENI value of 7s. The *Delta* factor can serve to further tune the delay/overhead tradeoff. To compare the overhead of ENI techniques with the basic MINA overlay protocol, we have tuned the *Delta* and let it achieve a similar end-to-end delay/delivery performance with the basic one. This is the methodology with which we have chosen 0.75s and 1s as the two different *Delta* values in our experiments. The results from Fig. 4.4 and Fig. 4.7 illustrate these points further.

4.5 Accurate Modeling on path duration in Convergecast Networks

Estimating the path duration time through a simulation scenario is not satisfied in a realistic scenario. When a node starts to connect to the server, it should be able to estimate when the current path towards the server will be broken in real time, given the environment settings including its speed, wireless radio transmission range, node density, etc. Accordingly it will send the next heart beat message after this estimated time and piggy back this information as ENI in the current heart beat message.

Specifically, we focus on the many-to-one communication pattern, called convergecast, where mobile sensors report their sensed values to one or more servers (data sinks) in a periodic or queried manner. This is exactly how MINA collects network information from each device via the overlay. In convergecast, either stationary or mobile sensors report their sensing value to one or more sources in a periodic or queried manner. According to QoS requirements, most data collection applications set delay as an important evaluation metric, especially when mobility is considered.

4.5.1 Related Work

The use of convergecast pattern for data collection has increased significantly over the past years[104][93] and there exists a number of protocols and systems that enable various kinds of information collection - quality aware raw data collection[48], surveillance video data collection[94], health monitoring via low power networking[22], etc. These protocols and systems always set end-to-end delay as one of the most important performance metrics, especially in cases where the real time and time sensitive applications are running above them [95]. To better understand how delay occurred and how the delay scales with the number

of sensor nodes and source nodes, plenty of work has been done both from experimental observation and theoretical analysis. More specifically, theoretical analysis on network delay includes mainly three aspects: capacity and delay analysis, path availability modeling and TDMA-based convergecast scheduling.

Firstly, in the pioneering work [78], Gupta and Kumar discovered that the capacity of a pure static ad hoc network will be limited as the number of nodes increased. Conducting power control shown in [6], can achieve a throughput of $\Theta(\sqrt{n})$. Adding a relatively small number base stations[14, 31] or mobile relay nodes [73] to a static ad hoc network can enhance capacity. Second, if we examine the delay from the routing layer, path availability is a critical parameter that affects the performance of the protocol in terms of delay. Generally speaking, the end-to-end delay of a routing protocol in a mobile network mainly consists of route discovery time, data transmission time, route failure detection and recovery time, which are all related to the path duration time. Also as a pioneering work, [7] first derived a link and path availability model for MANET with a random walk-based mobility. In [103], the authors provided insight into the link and path available time and other availability properties in a general ad hoc network. Samar and Wicker [84] added the link dynamicity into the analytical evaluation process. [99] used a two-state Markov model that takes the node speed and direction changes into consideration. A more recent work [47] used the link availability model to predict link and route duration time, and derived their link availability based routing protocol. Third, in a convergecast network people always concern the conflicts of two children sending data concurrently to their parent. Hence theoretical work usually focuses on the time slot scheduling to avoid collisions in the link layer and achieve optimal goals. Due to the ability to provide time bounds, most work used TDMA-based scheduling algorithms to enable fast and timely delivery of data with the goals of minimizing the time to complete convergecast, i.e. minimizing the latency [82][41]. In[43], algorithms are proposed to enable quick convergecast operations with minimum latency while complying with the ZigBee standard.

Since theoretical analysis and link layer approaches are not realistic from system implementation perspective, in this section we try to understand delay from the routing path availability fold. Previous work considered the path duration probability as the product of each hop's duration probability, which is not true in convergecast networks, and they did not exploit how network scales can affect the path duration. We will now present a novel model to overcome these limitations.

4.5.2 Models and Assumptions

Network Model

The network consists of N mobile sensor nodes and M static data sources all lying in a 2D unit square area (of side length 1). The location of the static data sources are fixed, and uniformly distributed at random over the unit square area. The sensors periodically send data to a unique sink via stationary nodes. We assume the links between the sink and stationary nodes are stable. So in the following, we only consider path duration among stationary nodes and mobile nodes. We will use data source and stationary node interchangeably. The mobile sensor nodes are distributed uniformly at random in the unit square area at time t=0. At later times their position and velocities are given by the mobility model described below.

Mobility model

Here we adopt random walk mobility model, which has been proved to be able to maintain the uniform distribution property[103]. Based on this model, each node's movement consists of a sequence of random length intervals called mobility epochs, during which a node moves at a randomly chosen velocity. A velocity is a vector with two elements: speed and direction. The speed is a random variable v, which is distributed uniformly between V_{min} and V_{max}.

The direction is a random variable θ, which is distributed uniformly between 0 and 2π. We define the probability density function(PDF) of velocity as:

$$f_v(v,\theta) = f_v(v) \times f_v(\theta) =$$

$$\begin{cases} \frac{1}{2\pi \times (V_{max} - V_{min})} & \text{if } v \in [V_{min}, V_{max}] \text{ and } \theta \in [0, 2\pi] \ , \\ 0 & \text{Otherwise.} \end{cases} \tag{4.1}$$

Link duration time

If two nodes are within a constant communication range R (all nodes have same the R) of each other, we assume a bidirectional link exists between them. For simplicity we refer to bidirectional links as links for rest of this paper. The link between nodes MN_1 and MN_2 is on when the distance between nodes MN_1 and MN_2 is smaller than R and is down when this distance is bigger than R. The link duration is the interval between two successive on and down transitions.

Other assumptions

Transmission time scale is much smaller than moving time scale A data collection system usually generates small data. Both the packet size and transmission time are small. The speed of mobile nodes in our scenario is low. So In this paper we focus on the link availability time inside only one epoch denoted as T_E. In other words,we assume the speed of the mobile nodes will not change once initialed. **Small volume of data** Since we focus on a small amount of data such as network state information, we ignore the queueing delay in the routing node. This means that the End-to-End delay mainly consists of route discovery delay, route failure and recovery delay. Through this way we can obtain a direct correlation between End-to-End delay and path availability.

4.5.3 Theoretical analysis

In this section we derive analytical expressions for the path duration time of our convergecast network scenario. First we will analyze the one hop path availability towards the data source, which serves as the basis for our further multi-hop cases in the second part.

One hop path duration time

As mentioned above, there are M static data sources uniformly distributed in the unit square area. Each data source dominates a sub square whose side length is $\Theta(\frac{1}{\sqrt{M}})$. We choose one such sub square to analyze the one hop path duration time. In this sub square, all mobile nodes send data to the dominator(static data source). So when a mobile sensor node MN is about to enter a dominated area(on the border), the distance between MN and its dominator S is r. Based on r, the transmission range R, and the velocity we can analyze the one hop link duration time of this mobile node. We assume the distance that MN traveled during the link duration period is l, as in Fig. 4.10. We have:

$$l(r,\theta) = 2\sqrt{R^2 - r^2\sin^2\theta} \tag{4.2}$$

$$r = \sqrt{(D/2)^2 + (D/2 - x)^2} \tag{4.3}$$

where D is the side length of the sub square and x is the initial position of MN on the border. We assume that x is uniformly distributed in $[0, D]$, which means the node can be at any point of the dominated area border with the same chance.

When MN moves to MN', the link is up; while it moves to MN'' the link down. So we obtain the CDF (Cumulative Distribution Function) of the link duration time T, i.e P(T

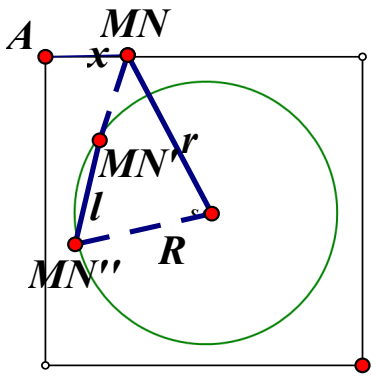


Figure 4.10: One hop duration time

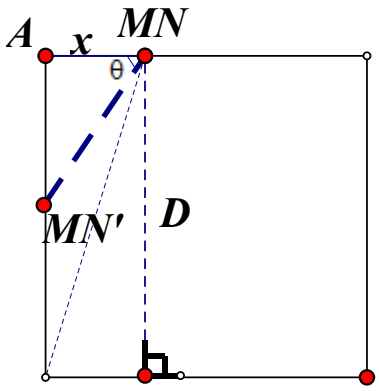


Figure 4.11: square area duration time

$\leq t$):

$$F_t^{[1]}(t) = \int_0^D \int_{-\arcsin(R/r)}^{arcsin(R/r)} \int_{V'_{min}}^{V_{max}} \frac{f_v(v,\theta)}{D}\, dv\, d\theta\, dx \tag{4.4}$$

where V'_{min} is the minimum required velocity from MN' to MN'' within time t:

$$V'_{min} = \frac{l(r,\theta)}{t} \tag{4.5}$$

So the PDF (Probability density function) of T can be obtained:

$$f_t^{[1]}(t) = \frac{\partial F x_(r,\varphi,)x}{\partial t} \tag{4.6}$$

So the expectation of the one hop link duration time in one dominated area is:

$$E_t^{[1]}(t) = \int f_t^{[1]}(t) t dt \tag{4.7}$$

To evaluate the one hop link duration time under different numbers of stationary nodes (different dominated area size), we need to analyze the average time that a mobile node move crosses a dominated area whose side length is D. As show in Fig. 4.11,when $0 < \theta <$

$\arctan(D/x)$,

$$k_1(x,\theta) = \frac{x}{\cos\theta} \tag{4.8}$$

while $\arctan(D/x) < \theta < \frac{\pi}{2}$

$$k_2(x,\theta) = \frac{D}{\cos(\frac{\pi}{2} - x)} \tag{4.9}$$

So assuming Ts as the time that a mobile node spends to cross a square area, the CDF (Cumulative Distribution Function) of Ts, i.e P(Ts $\leq t$):

$$F_{ts}(t) = \begin{cases} \int_0^D \int_0^{arcsin(D/x)} \int_{\frac{k_1}{t}}^{V_{max}} \frac{f_v(v,\theta)}{D}\, dv\, d\theta\, dx \\ \int_0^D \int_{arcsin(D/x}^{\frac{\pi}{2}} \int_{\frac{k_2}{t}}^{V_{max}} \frac{f_v(v,\theta)}{D}\, dv\, d\theta\, dx \end{cases} \tag{4.10}$$

For $\frac{\pi}{2} < \theta < \pi$ case, it is the same as we described above. Accordingly,$f_{ts}(t)$ and $E_{ts}(t)$ can be found. Note D is determined by the number of stationary nodes M. The bigger M is, the more frequently a mobile node crosses a dominated area and the more likely a mobile node has a one hop link. So given length of the whole epoch, T_E, the average one hop duration time can be got by $\frac{E_t^{[1]}(t) * T_E}{E_{ts}(t)}$

Two hop path duration time

A two hop path contains a one hop link from the stationary node to the relay node, as we discussed in the previous subsection, and a one hop link from the relay node to the mobile node. To analyze the link duration between two mobile nodes, relative velocity should be given first.

relative velocity of mobile nodes

As shown in Fig. 4.12, similar with the concept in [47], the nodes' movement is centrosym-

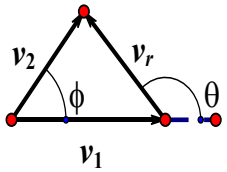


Figure 4.12: Relative velocity

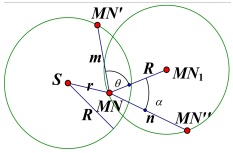


Figure 4.13: two hop path duration time

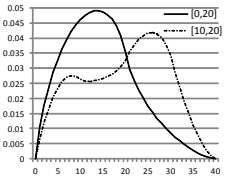


Figure 4.14: Probability Density

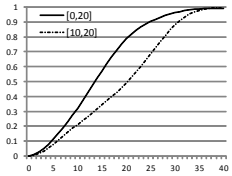


Figure 4.15: Cumulative Distribution

metric. We can assume v_1 is parallel to the X-axis without lose of generality. Assume mobile node MN_1 has a velocity$(v_1, 0)$, MN_2 has a velocity (v_2, ϕ) and their relative velocity is (v_r, θ). Note the angle ϕ between v_1 and v_2 is uniformly distributed in $[0, \pi]$. v_r has a angle θ, which is uniformly distributed in $[0, 2\pi]$. According to the cosine rule, we have:

$$v_r^2 = v_1^2 + v_2^2 - 2v_1 v_2 \cos\phi \tag{4.11}$$

since θ, v_1,v_2 are independent, the PDF of the joint function is:

$$\begin{aligned} f_{v_1,v_2,\phi}(v_1, v_2, \phi) &= f_v(v_1) f_v(v_2) f_\phi(\phi) \\ &= \frac{1}{\pi(v_{max} - v_{min})^2} \end{aligned} \tag{4.12}$$

According to the Jacobian transform, we have

$$f_{v_1,v_2,v_r}(v_1, v_2, v_r) = \frac{\partial\phi}{\partial v_r} f_{v_1,v_2,\phi}(v_1, v_2, \phi) \tag{4.13}$$

where

$$\frac{\partial\phi}{\partial v_r} = \frac{2v_r}{\sqrt{2v_1^2v_r^2 + 2v_2^2v_r^2 + 2v_1^2v_2^2 - v_r^4 - v_1^4 - v_2^4}} \tag{4.14}$$

Hence we get the PDF of the magnitude of the relative velocity:

$$f_{v_r}(v_r) = \int_{v_{min}}^{v_{max}} \int_{v_{min}}^{v_{max}} f_{v_1,v_2,v_r}(v_1, v_2, v_r)\, dv_1\, dv_2 \tag{4.15}$$

Fig. 4.14 and Fig. 4.15 show the PDF and CDF of the relative velocity of $f_v(v, \theta)$ whose speeds are in the range of $[10, 20]$ and $[0, 20]$, respectively.

Analytical expressions for two hop path duration time In contrast to pure ad hoc wireless networks, in convergecast the path is established from the source(root) to the mobile node, like a tree. In the data collection phase, a node always reports its data to its parent in the tree. So when a two-hop path is about to be established, the distance between the relay node and the mobile node are exactly the transmission range R, while the distance between the source node and the relay node may be shorter than R, which means this one hop link has been on for some time. As shown in Fig. 4.13, when the path S-MN-MN_1 is established, the distance between MN and MN_1 must be R, while the distance between S and MN is $r < R$. Since we assume the mobile nodes are uniformly distributed in the experimental area, $f_r(r) = \frac{2r}{R^2}$. We use T,T_1,T_2 to denote the duration time of S-MN-MN_1, S-MN, and MN-MN_1, respectively. T is bigger than t if and only if T_1 and T_2 are bigger than t. So we have:

$$P\{T \leq t\} = 1 - P\{T_1 > t\} \times P\{T_2 > t\} \tag{4.16}$$

$$P\{T \leq t\} = 1 - (1 - P\{T_1 \leq t\}) \times (1 - P\{T_2 \leq t\}) \tag{4.17}$$

We use $F^{[2]}(t)$, $F_1(t)$ and $F_2(t)$ as the Cumulative Distribution Function of T, T_1 and T_2.

MN moves to MN' as a velocity of (v, θ); total distance is m. So we can get:

$$F_1(t) = \int_0^R \int_0^{2\pi} \int_{\frac{m}{t}}^{V_{max}} f_v(v,\theta) f_r(r)\, dv\, d\theta\, dr \tag{4.18}$$

where

$$m(r,\theta) = \sqrt{R^2 - r^2 \sin^2\theta} - r\cos\theta \tag{4.19}$$

so the PDF of T_1 is:

$$f_1(t) = \frac{\partial F_1(t)}{\partial t} \tag{4.20}$$

The CDF of T_2 is similar with T_1. However, because MN and MN_1 are mobile nodes, we need to use relative velocity here. In addition, the start distance between MN and MN_1 is exactly R. So we have:

$$F_2(t) = \int_0^{2\pi} \int_{\frac{n}{t}}^{V_{max}} f_{rv}(v) \frac{1}{2\pi}\, dv\, d\alpha \tag{4.21}$$

where

$$n(\alpha) = 2R\cos\alpha \tag{4.22}$$

so the PDF of T_2 is:

$$f_2(t) = \frac{\partial F_2(t)}{\partial t} \tag{4.23}$$

So the CDF of the two-hop path duration time T can be expressed as:

$$F_t^{[2]}(t) = 1 - (1 - F_1(t)) \times (1 - F_2(t)) \tag{4.24}$$

The PDF is:

$$f_t^{[2]}(t) = f_1(t)(1 - F_2(t)) + f_2(t)(1 - F_1(t)) \tag{4.25}$$

The expectation of two hop path duration time is:

$$\bar{E}_t^{[2]}(t) = \int f_t^{[2]}(t) t dt \tag{4.26}$$

Note there is a big difference between $E_t^{[1]}(t)$ and $\bar{E}_t^{[2]}(t)$. When we calculate $E_t^{[1]}(t)$, the start position of the mobile node is on the border of the dominated area, while in $\bar{E}_t^{[2]}(t)$ the start position is where a mobile node establishes a path towards the source. The reason is that for one hop path, the link is always on right after the mobile nodes entering the transmission zone of the stationary node(ignoring the control message exchange time). But for two hop paths, this is not the case. As shown in Fig. 4.16, there are two main prerequisites that a

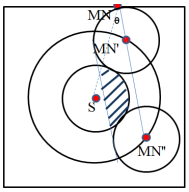


Figure 4.16: two hops path in square

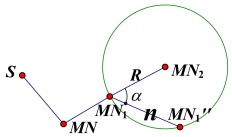


Figure 4.17: three hops path

mobile node has a two hop path towards to the stationary node: (a). This node crosses the ring area, $[R, 2R]$ (all nodes have the same transmission range). Note we only consider the ring area rather than the whole circle area with radius $2R$, because we also assume that if a node moves into the transmission range of the stationary node, it will change its two-hop path to a one hop path immediately. (b). There is at least one other mobile node in the shadow area serving as a relay node. We can simply get the probability of a mobile node crossing the ring area, say P_a, using a similar idea with the one hop case. Now we will show how to calculate the expectation of the probability that at least one relay node exists in the shadow area. Let S be the size of the shadow area and P{at least one node existing in S} $= 1 - (\frac{1-S}{1})^{N-1}$ (all nodes are lying in an unit square area), where N is the total number

of mobile nodes. Note we assume the whole environment area is a unit square area. S is a function of θ, as depicted in Fig. 4.16:

$$S(\theta) = \frac{2\arccos(\frac{2R\sin\theta - R}{R})}{2\pi}\pi R^2 = R^2\arccos(\frac{2R\sin\theta - R}{R}) \tag{4.27}$$

So given a variable s,$P\{S \leq s\} = p\{Theta > g(s)\}$, where $g(s)$ is the inverse function of $S(\theta)$. So:

$$F_s(s) = 1 - F_\theta(g(s)) = 1 - \int_{\frac{\pi}{6}}^{g(s)} \frac{1}{\pi}\, d\theta \tag{4.28}$$

$$f_s(s) = \frac{\partial F_s(s)}{\partial s} \tag{4.29}$$

So the expectation of the probability that at least one node in shadow region,

$$E' = \int_0^{\frac{\pi * R^2}{2}} [1 - (\frac{1-S}{1})^{N-1}] f(s)\, ds \tag{4.30}$$

So the expectation of the two-hop path duration time is

$$E^{[2]} = P_a E' \bar{E}_t^{[2]}(t) \tag{4.31}$$

Multihop path duration time

The expectation of n-hop path duration time can be got iteratively from the two-hop path case. Considering the characteristics of convergecast, a tree-like topology is always built from the root down to the leaves. In other words, when a node is about to join the tree, its parent already has a path towards to the root. So at the start of the path availability, each

relay node has a distance $r(\leq R)$ away from its parent, while the leaf mobile node has a distance R from its parent (R is the transmission range). Fig. 4.17 shows the case of a three hop path, and we will briefly describe how a three-hop path duration time can be calculated. Here we use $T^{[2]}$ and $T^{[3]}$ to denote the duration time of path $S-MN-MN_1$ and path $S-MN-MN_1-MN_2$. Also we use T_3 to denote the duration time of link MN_1-MN_2. We use $F^{[3]}(t)$ and $F_3(t)$ as the Cumulative Distribution Function of $T^{[3]}$ and T_3. MN_1 moves to MN_1'' as a velocity of (v,α) and the total distance is n. Similar with (4.21) we have:

$$F_3(t) = \int_0^{2\pi} \int_{\frac{n}{t}}^{V_{max}} f_{rv}(v) \frac{1}{2\pi}\, dv\, d\theta\, dr \tag{4.32}$$

where

$$n(r,\alpha) = 2R\cos\alpha \tag{4.33}$$

so the PDF of T_3 is:

$$f_3(t) = \frac{\partial F_3(t)}{\partial t} \tag{4.34}$$

So the CDF of the three hop path duration time $T^{[3]}$ can be expressed as:

$$F_t^{[3]}(t) = 1 - (1 - F_t^{[2]}(t)) \times (1 - F_3(t)) \tag{4.35}$$

The PDF is:

$$f_t^{[3]}(t) = f_3(t)(1 - F_t^{[2]}(t)) + f_t^{[2]}(t)(1 - F_3(t)) \tag{4.36}$$

The expectation of two hop path duration time is:

$$\bar{E}_t^{[3]}(t) = \int f_t^{[3]}(t) t dt \tag{4.37}$$

Usually the path hop counts depend on the number of stationary nodes. If stationary nodes are enough, the dominated area will be covered by the transmission range of the stationary node and there will be only one hop path. In addition, the path availability will be dramatically decreased when the hop counts increase in our convergecast scenario. In our experiment, we only consider paths consisting of at most three hops.

4.5.4 Simulation and Verification

In order to verify the correctness of our model, we compare the results of our theoretical model described above with the actual simulation results using Qualnet. There is one stationary central sink having Ethernet connections with M static data sources. N mobile nodes move around with random way point mobility model (we set the pause time to 0 so that it is equal to our mobility model). Both mobile and stationary nodes are equipped with 802.11 interfaces. Each mobile node periodically sends data to the central sink directly via any of the stationary nodes or through possible relay nodes, according to its position. Due to the high availability of the Ethernet connections, we only focus on the M stationary sources and N mobile nodes, which is the exact same scenario we assumed in our theoretical model. In Qualnet, we modified the built-in AODV routing protocol so that the path duration time can be calculated with the life time of routing entries in mobile nodes. The average value will be calculated on all mobile nodes. The simulation terrain is a two-dimensional space(3000,3000), which represents a square area of 3000m $\times$ 3000m. M stationary nodes and N mobile nodes are randomly deployed in this area. The number of the mobile nodes N scales from 4 to 40, and the number of stationary nodes M is from 2 to 12, depending on the simulation scenario. The magnitude of velocity is uniformly distributed in [10m/s,20m/s] and [1m/s,11m/s], and the direction is uniformly distributed in $[0,2\pi]$. In Qualnet, the value of a random variable only depends on the seed number. For each simulation run we used 25 different seeds and got 25 different results. The average is considered as the result of each

run. The transmission range of all nodes is 400m. We used the CBR application protocol to simulate the data collection process: each packet is 500 bytes long and one packet per second. The simulation period (T_E)is 250 seconds.

One Hop Path Duration Time

Setting $N = 1$ means there is only one mobile node hence only one hop path duration time is considered. Fig. 4.18 shows that the expectation time from our model is quite similar with the path duration time from the simulation. Here M (i.e. the numbers of stationary node) is up to 12, in which case if all stationary nodes are uniformly distributed. There is little overlap and the total coverage area will be almost the same with the whole experiment area. In other words, if we have more stationary nodes, the whole experiment area might be fully covered by the stationary nodes. Hence no matter where the mobile node is, it can always be connected with a static source node. As a result, the one hop duration time of each mobile node will be 250 second, which is meaningless for us. In Fig. 4.18 $N = 1$ (i.e. the number of mobile node is 1) and the X-axis is the number of the stationary nodes (M) and Y-axis is the path duration time. We use two different velocity distribution ranges($[10, 20], [1, 11]$) and the mean error rates between the model value and the experiment value are 9% and 7%, respectively. The reason is in Qualnet, the transmission range is not an exact circle. Instead they adopt a physical model, which means a node can send radio frames successfully as long as its signal strength sensed by the destination exceeds the threshold, to simulate wireless communication. In addition, from the result we can find that nodes with higher velocity will have shorter path duration time. In other words, dynamicity will incur poor connectivity.

Two Hop Path Duration Time

We deployed more than one mobile node which means possible two-hop connections are also considered. Note that in this set of simulations we only consider M up to 6, whose dominated

area is almost the same with the two hop coverage area.The reason is similar with the one hop case. The two hop coverage area should be bounded by the dominated area. Although there are existing three-hop paths, we argue that it can only take place when the three hop coverage area is smaller than the dominated area. So we only choose $M = 4, 5, 6$, in which case the the hop counts are unlikely more than 2. In Fig. 4.19,4.20,4.21, the X-axis is the number of mobile nodes (N) and Y-axis is the duration time. The results show that the simulation results are quite consistent with the model results, the average difference rate is around 5%. An interesting observation from the results is that when $M = 4$ and $M = 5$, if the number of the mobile nodes exceed a threshold, say $N = 30$, the duration time from the simulation becomes higher than the one from our model. This is mainly because when N becomes larger, the opportunity that a node has a multiple-hop (more than two) path increases, especially when it is out of the two-hop coverage area (the circle with radius of $2R$). Because we do not take the multiple-hop paths into consideration when $M > 4$, the duration time calculated by our model is less than the simulated one. The multiple-hop path opportunity indeed exists, but it is very small as we argued above. The situation changed when $M = 6$, the mobile nodes are almost covered by the two-hop coverage area. In this case there is less chance that a node finds a multiple-hop (more than two) path. However, in simulations more stationary nodes will incur more handover and hence incur less path duration time, which is why the results derived from our model are slightly bigger than the simulated one in Fig.4.21.

4.5.5 Three Hop Path Duration Time

We further decreased the number of stationary nodes (M) to 2. Under this scenario, the dominated area is slightly larger than the circle with radius of $3R$ and possible three-hop paths are considered. As described in 4.5.3, the duration time of three-hop paths depends on the first hop link and the second hop link. While in our experiment during the 250

second simulation time, a node may be connected directly with the stationary node (one hop path) or two/three-hop path, according to its position. We argue that although there may be four/five-hop path existing in $3R$ circle, its probability is extremely low and hence can be negligible. Fig. 4.22 shows the results of the path duration time with maximum hop number of three. The average difference rate is below around 10%. The main reason is that the three-hop path analysis is more complicated than the two-hop case. In our model, we assume that the three-hop path can only be established when the third node falls in the ring area whose radius is between $[2R, 3R]$. In fact if a node falls into the ring of $[R, 2R]$, it can also have the opportunity of establishing a three-hop path, which is hard to model.

End to End Delay and the Duration Time

The relation between end-to-end delay and the path duration can be viewed from two parts: a). In the ideal case, whenever the application on a mobile device sends data, there is always an available path already established by the routing layer towards the sink. However, the mobile device usually needs to wait to send application data before the path is established, which will incur delays. b). Since handover needs time, more stable links are preferred. For example, two different paths with duration time 5s each will incur bigger delay than a single path with duration time 10s, due to the handover. So the less dynamic the paths are, the smaller delay will be achieved. Fig. 4.23 shows our results revealing the correlation between delay and the path duration time in a convergecast network. The x-axis is the number of mobile nodes, scaling from 4 to 40. The y-axis on the left is the average End-to-End Delay of the CBR application while the y-axis on the right is the average path duration time. We tested three scenarios with different numbers of stationary nodes ($M = 4, 5, 6$).

As discussed above, more stationary nodes will provide larger coverage areas and network access opportunities. The results show that the path duration time of $M = 6$ is longer than $M = 5$ and $M = 4$ and hence leads to the smallest delay. With more mobile nodes, we

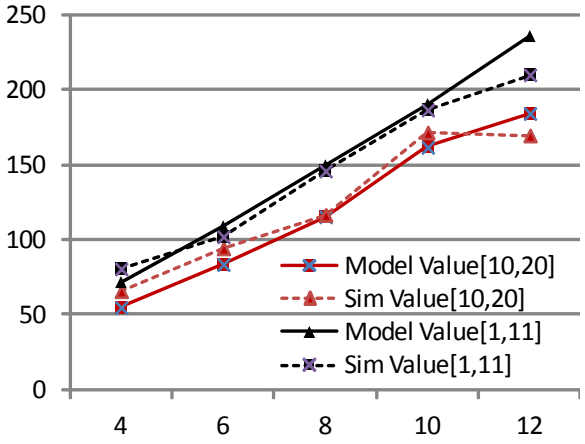


Figure 4.18: One Hop duration time

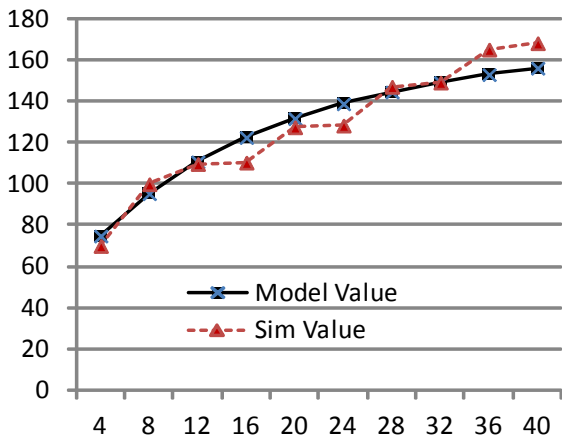


Figure 4.19: Two Hops M=4

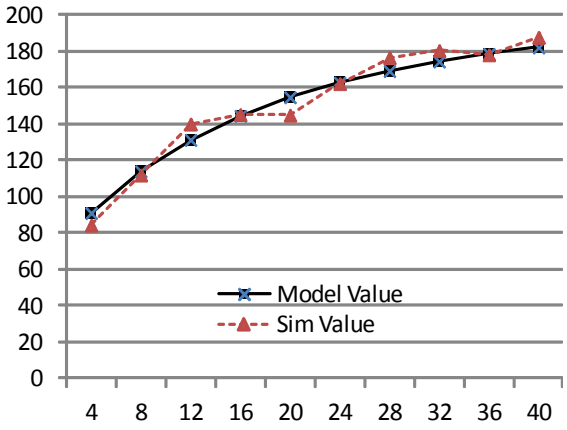


Figure 4.20: Two Hops M=5

find the path duration time also increased. However, the distribution of end to end delay is bimodal. With an increasing number of mobile nodes, the delay first increases from $N = 4$ to $N = 12$ and then decreases, and the second local maximum point is reached around $N = 32$. The reason is that when the scale of the network is small, the path duration time is quite short, and with more mobile nodes added, one needs to spend more time on handover and the newly obtained path availability can not compensate the handover time consumption. So the end to end delay increases. If mobile nodes continue being added, there are more network access opportunities which make the handover time be compensated. Thus the delay decreases. When the network scales to some extent, traffic congestion will happen, which

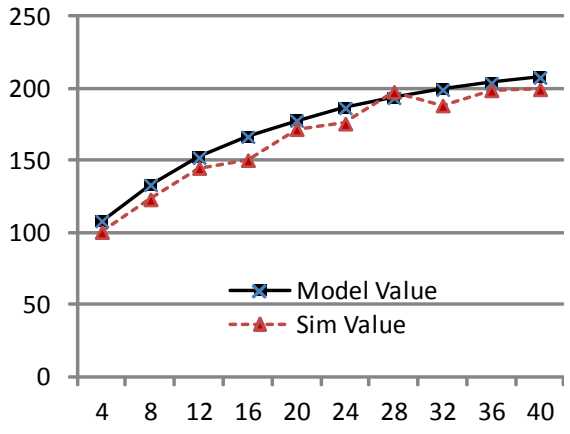


Figure 4.21: Two Hops m=6

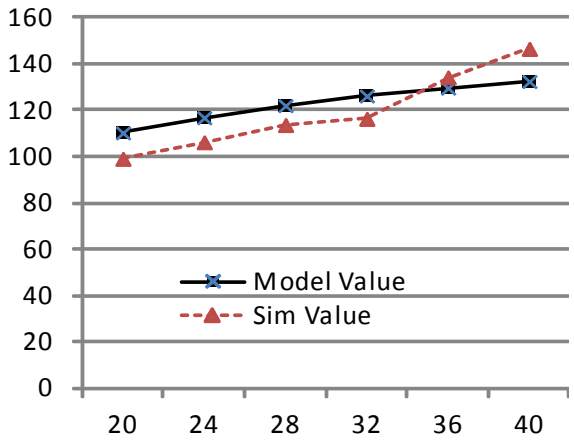


Figure 4.22: Three Hops

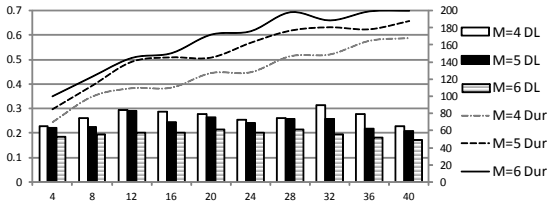


Figure 4.23: Two Hops M=5

incurs data transmission time. As we can observe, the delay increases again around $N = 32$. After that the enhanced path availability again dominates the end to end delay. As shown in Fig. 4.23the delay decreases when $N > 32$.

From the experiments, one inference is that adding more devices, either stationary or mobile, can improve the link availability in the network while only adding stationary nodes can decrease the delay. This makes sense since the stationary nodes provide the first hop connections, which are more stable. In reality, the network administrators usually deploy more stationary routers and access points to improve delay. On the other hand, carefully planning the scale of mobile nodes can also improve the end to end delay, which requires a more comprehensive understanding of delay, congestion and link availability. We will investigate that in the future.

4.6 Chapter Conclusion and Future Work

This chapter first presents the design, extensions/enhancements, and simulation-based evaluation of the MINA overlay solution. A key aspect is the development of a novel, dynamically constructed and mobility-aware tree-based overlay structure that can effectively balance end-to-end data collection delay and overhead.

The encouraging results achieved are stimulating our further research activities along the path duration modeling direction. We proposed a novel path duration time model for data collection in convergecast networks. We claim that the probability of the multi-hop path duration time is not merely a product of each link. Instead, the n-hop path duration time is always based on its previous $n-1$ hop path. We show that network density affects the path duration time. The results demonstrate that our model can accurately reflect the path duration time in simulation. We also present analysis on the correlations between end-to-end delay and the path duration time, which will help to understand the relationship between delay, path duration time and nodes density in convergecast network. This work can serve as a guidance of link availability based routing protocol design and link quality aware systems.

Chapter 5

Analyze: Formal Method Based Multinetwork Analysis

5.1 Problem Description and Motivations

Network analysis approaches roughly divide into *reactive* and *proactive*. *Reactive analysis* approaches involve real-time observation of network conditions and adaptation procedures to account for changes. Reactive analysis is often implemented in various parts of the system and scattered throughout its components. While this type of analysis may be very optimized for goals that must be achieved in millisecond or less time, it also often has limitations because it is tuned to a certain analysis aspect or constrained to a certain part of the system, and thus, mostly concerned with local network aspects.

Proactive analysis approaches focus on analyzing and optimizing global network conditions in a predictive manner, anticipating and preventing major negative impacts of changes on the network. This approach requires a slightly larger timeframe than reactive analysis but provides an opportunity for adaption in the context of the multi-network. Proactive analysis

determines the most beneficial changes to the network from a global view on the multi-network and informs the dynamic adaptation step of the OAA cycle. Because proactive analysis is done in a separate, dedicated module that accounts for a snapshot multi-network state, it can also use other contextual information as part of the analysis. For example, parameters like application-specific requirements, known patterns of network load, or user-defined goals can inform proactive analysis for managing the multi-network.

The objective of MINA analysis is to ensure that quality-of-service (QoS) requirements of network flows are satisfied. A network flow is traffic flowing from a source node to one or more target nodes. For example, a flow may describe imagery flowing from a camera installed on campus to the campus' security control center. Or a flow may describe the traffic that flows from the control center to all cameras in a building to disable power-safe modes on the cameras. The concept of network flow abstracts from the specific route the packets for this flow will take. To check satisfaction of flow QoS, we need network state information as collected in MINA. We employ formal analysis methods to check QoS satisfaction.

The analysis techniques are decoupled with MINA implementation. One or more analysis tools can be plugged into MINA analysis module, but in this chapter we focus on a formal method based approach. The proposed analysis methodology performs various "what if" analyses asking questions such as

- What happens if a node fails? The objective of this analysis is to determine how critical a node is, not just to one network flow, but to all flows.
- What should we do if we have additional network resource such as mobile routers? Where should we deploy them most effectively? The objective of the analysis is to determine where additional nodes would positively impact QoS of all flows.
- What happens if the load of selected flows changes and how can we best mitigate the effects?
- What happens if the link quality changes due to congestion or interference and how

can we best mitigate the effects of degraded link quality?

Answers to these questions help to decide how to best utilize network resources or reconfigure the network to achieve better overall QoS requirements satisfaction. In turn, this will lead to better network resilience and dependability.

Existing work on network analysis and prediction can be divided into static analysis and dynamic analysis. Static analysis only exams a snapshot of the the network. In [101], the authors determine the full network state including Internetwork Operating System bugs of devices, configuration errors, static/dynamic routing, and so on. They calculate all possible virtual paths according to the full network state and then compare the virtual paths with the available physical paths to check reachability. However, due to the large search state space, this approach is not very efficient. The same problem exists in [67]. The authors pre-compute routing tables for each state and employ formal methods to model every possible behavior of the network. [29] provides an effective way to reduce the state space. Both [67] and [101] only focus on the one time link failure and the network will converge to another stable state. In dynamic analysis a failure may cause other failures in future states. For example, a failing node does not only affect the flows going through that node, but also other flows due to flow rerouting. This is called cascading failures. In [40] the authors consider statistics of link failures and limited number of cascading failures. Different failures combinations can lead to loss of connectivity within a network or to severe congestion, as shown in [71]. They proposed a framework to analyze link availability in the context of link failures, changes of user behavior and routing. However, most of the cited approaches do not consider heterogeneous networks and do not study how the flows are affected by failures. In this chapter, we provide an approach to handle (1) changing heterogeneous network topology (i.e. node failure, adding backup router, or network reconfiguration), while (2) taking into consideration how the flows' QoS performance are affected due to flow redistribution triggered by changes in the network topology. This approach is based on formal methods. Formal methods have been used to do network analysis [2], protocol proof [91] and network model

checking [69]. To our best knowledge, there exist no tool that uses formal methods for the flow oriented proactive analysis of multinetworks.

To overcome these limitations, we propose to use **formal methods for proactive analysis**. Formal methods are a good match for proactive analysis because they apply abstraction to ensure capture of the system's dynamic while keeping it abstract enough to allow automated tools to check critical conditions. For the proactive analysis of multi-networks we use the formal method Maude [25], an executable specification language. Maude models system states through user-defined data types and system dynamics through rules. Moreover, not only the Maude interpreter is very efficient at simulating complex systems, but also provides efficient built-in search and model checking capabilities (see *http://maude.cs.uiuc.edu*).

5.2 Formal Method-Based Analysis Methodology

The objective of our analysis is to answer questions such as "what effect does a failing node have on all network flows" or "if one had more resources to deploy in the network, what would be the best location for an additional node." It turns out that underlying all of the questions listed above is the concept of how critical a node is to the satisfaction of QoS requirements. We introduce the notion of "Node Criticality Index (NCI)." Nodes with the highest criticality index have the most negative effect on the overall QoS of flows when they fail. Thus, these nodes are the first targets when it comes to deploying backup nodes or relieving a node of high traffic loads. We distinguish our concept of NCI from conventional node importance measures used in existing work. The simplest node importance measure is node degree, which is defined as the number of neighbors a node has (see [15]). Closeness-based measure is another node importance measure that finds the distance center or the median of a graph. It has application in facility location [46], package delivery [16] and operations research problems. It is computed by summing up the distances from the current

node to all remaining nodes. Betweenness is one of the most prominent node importance measures. It measures the influence of a node over the connection of other nodes by summing up the fraction of shortest paths between the other nodes that pass through it [17, 35]. However, none of these measures take into account how redistributed flows due to network changes influence overall QoS. The analysis methodology proposed here addresses that issue.

We use Maude [24] as our underlying formal method tool. Maude is a multiparadigm executable specification language encompassing both equational logic and rewriting logic. Maude allows modeling system states through user-defined data types and system dynamics through rewrite rules. The Maude interpreter is very efficient, allowing prototyping of quite complex test cases. Maude also provides efficient built-in search and model checking capabilities. Maude sources, executables for several platforms, the manual, a primer, cases studies, and papers are available from the Maude website `http://maude.cs.uiuc.edu`.

We formalize the analysis objectives as Maude models and rules. For example, if the analysis objective is to determine the most critical network node, then we provide a Maude specification and a set of rules that simulate in a network that a node fails, reroute affected flows and simulate QoS of the new network. Iterating this process over all nodes and comparing the resulting QoS of all flows allows us to determine the most critical node. If the analysis objective is to choose the best new access point for a wireless node when its access point fails, then we have a Maude specification and rules that simulates the overall performance for the different alternate access points. Thus, we generate a Maude specification according to the reasoning objective (step 3 in Figure 5.1). Information about the specific network topology and state and the flows is pulled from the MINA database (step 2 in Figure 5.1). We use the Maude engine to execute the specific analysis (step 4) and store any relevant analysis results (e.g., node criticality index or hints where new resources should be deployed) in the database (step 5). This information can be used by network administrators to devise new network reconfigurations and issue accordingly commands to the multinetwork (step 6).

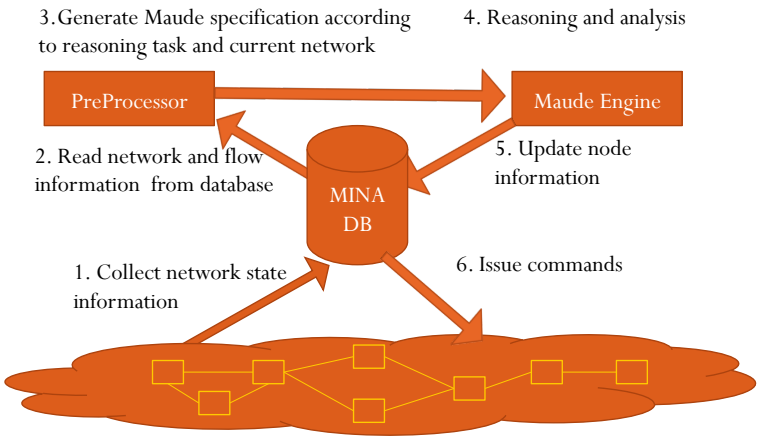


Figure 5.1: Workflow in the context of MINA

In the following we present some details about how we model networks and flows and what the rewriting rules to process network flows and determine QoS look like.

5.2.1 Network and Flow Specification

Information about networks and flows is stored in the MINA database as follows. A network node is represented by a 3-tuple containing *NodeId*, *neighbor list* and *weight list*. *neighbor list* contains all nodes directly linked to the node *NodeID*. The ith element in *weight list* represents the capacity of the link between *NodeId* and the ith element in the *neighbor list*. A flow is represented by an 8-tuple: *source*, *route*, *destination*, *flowID*, *type*, *throughput*, *packetlength* and λ. *route* is the sequence of nodes on the path of the flow from the source to the destination, *type* specifies the kind of traffic (e.g., ftp, audio, video, and so on), *packetlength* corresponds to the packet length when it is an ethernet packet or the length of the item sent on a different network, and λ is the mean packet arrival rate of this flow. Note the product of λ and *packetlength* is the throughput. In the following, we will show how we use the information about networks and flows that is stored in the MINA database to generate a Maude model.

We consider the nodes and links through which a flow goes as tandem queues (see Figure 5.2).

The tandem queues refer to an arrangement of queues in which the queues are lined up one after the other: the outgoing traffic of the current queue is the incoming traffic of the next queue. A node consists of the queue and the network interface (NIC; there is possibly more than one NIC on a node). Currently we do not consider queue length limitations, or in other words, we assume there no packets are lost due to limited queues.

Since link propagation delay is extremely small, we only consider delay introduced by queueing and service time in the NIC. Thus, in our abstract model we assume the outgoing packet of one node to be the incoming packet of the subsequent node. This way, we divide an end-to-end flow into several sub flows, one for each hop along the path of the flow (see Fig 5.2). By doing so, we can iteratively propagate the QoS parameter computed in each node and accumulate the end-to-end QoS for a flow from its source to its destination. In this chapter, we use delay as an example QoS parameter to illustrate our analysis methodology.

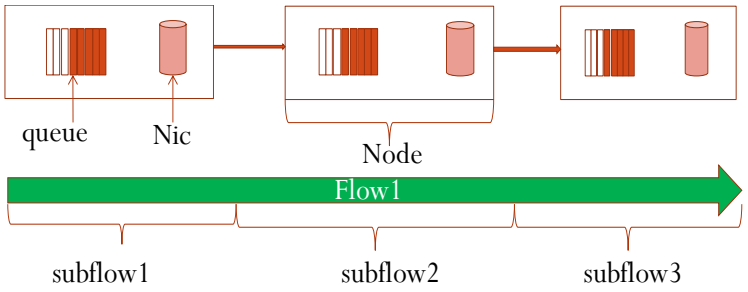
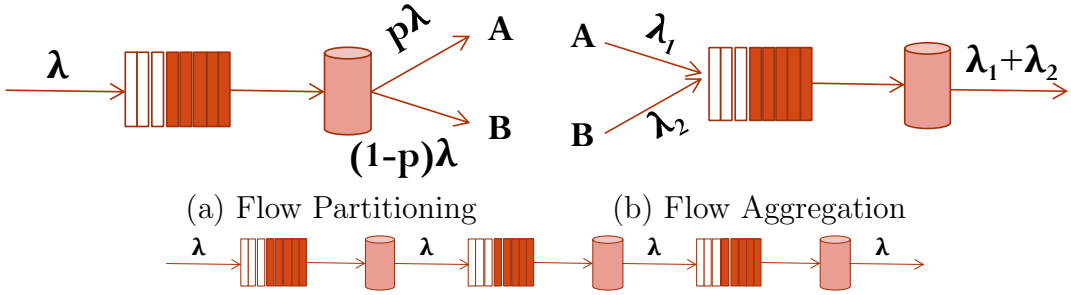


Figure 5.2: Network Flow Model

The flow model enables us to get the end-to-end delay by aggregating and propagating the delay incurred by each node. We use a simple M/M/1 model [89] to calculate the delay incurred by a single node. The M/M/1 represents the queue in a system having a single server, where arrivals are determined by a Poisson process and job service times have an exponential distribution. We denote the Poisson Arrival Distribution with (λ, n), where λ is the packet mean arrival rate and n is the packet length. The server's (network interface) mean service time is denoted by Ts. We use Poisson distribution to describe the flow pattern mainly because it is widely used to simulate network flow [89] and the Poisson distribution

has perfect aggregation and partitioning attributes. For example, as shown in Fig 5.3(c), the Poisson pattern is not changed when a flow goes through a node with exponential service time. In Fig 5.3(b), flow A and B with mean arrival rate λ_1 and λ_2 are both the incoming Poisson flows at one node. After processing, the new aggregated flow is still a Poisson flow with a mean arrival rate $\lambda_1 + \lambda_2$. In Fig 5.3(a), a Poisson flow with mean arrival rate λ can be partitioned into two Poisson flows whose mean arrival rate's sum is still λ.



(a) Flow Partitioning (b) Flow Aggregation

(c) Tandem Queue with Poisson Flow

Figure 5.3: Key Attributes of Flows with Poisson Arrival

These important features enable us to apply the Possion-based delay calculation in each processor, regardless of how the flows are distributed in the multinetwork:

$$Delay = n/(Ts - n)$$

5.2.2 Rewriting Rules for Analysis

We view the network as a set of nodes and flows. Each flow is a sequence of subflows, one subflow for each hop along the flow's path. Nodes and (sub)flows are basic concepts in our Maude model. We model the network as nodes N and sub flows SF. We also model the QoS parameters such as $d$elay, throughput, jitter, packet loss and so on. So a network is modeled in Maude as a structure with the following elements: `N(node attributes), SF(subflow attributes), QoS attributes.`

Nodes N are modeled as a collection of attributes. Each attribute is represented as label(value)-pair. Variables (using capital letters) are used as a place holder for values in structures or rules. Attributes of nodes are the identity of the node (`Nid(N1)`), type of NIC (`NIC(T)`)[1], transmission rate `Tx(X)`), Poison parameters (`PoiPara(`λ`,n)`), the number of flows expected to route through this node (`Tot(S)`), and the flows that have already arrived (`Cur(C)`). Thus a node is represented as `N(Nid(N1),NIC(T),Tx(X),PoiPara(`λ`,n), Tot(S),Cur(C))`.

Each subflow is modeled as a collection of attributes: identity of the parent flow `Par(F1)`, the identity of the subflow `Sub(S1)`, the identify of the source node of this subflow `Src(N1)` and its NIC type (`SrcNIC(T)`), and identity of the destination node of this subflow `Dest(N2)`, the poison parameters `PoiPara(`λ_{in}`,n)` for this subflow. and the incoming and outgoing QoS characteristics of this subflow. Since the examples in this chapter only explore delay, we will only model incoming and outgoing delay (`Delayin(DIN)` and `Delayout(DOUT)`). Finally, we add a boolean flag to indicate whether the subflow was already processed by a rule (`Proc(B)`). Thus a subflow is represented as `SF(Par(F1), Sub(SUB1),Src(N1), SrcNIC(T), Dest(N2), PoiPara(`λ_{in}`,`n_{in}`), Proc(B), Delayin(DIN), DelayOut(DOUT))`.

We have three main Maude rules that describe propagation of QoS characteristics of flows in the multinetwork: a) flow accumulation, b) flow processing, and c) flow propagation.

- *flow accumulation:* This rule is collecting all the sublows at one processor. The idea is that when multiple Poisson flows come into a processor, we aggregate the λ and n value for all flows to calculate the overall delay they have suffered given the service

[1]For simplicity and because of space limitation, we present our rules with one NIC per node.

capacity of this processor (the capacity/transmission rate of the network interface).

crl `N(Nid(N1), NIC(T), PoiPara(`λ`,`n`), Cur(C),...)`

`SF(Src(N1), SrcNIC(T), PoiPara(`λ_{in}`,`n_{in}`),`

`Proc(false), ...)`

`=> N(Nid(N1), NIC(T), PoiPara(`$\lambda^{'}$`,`$n^{'}$`), Cur(C1),...)`

`SF(Src(N1), SrcNIC(T), PoiPara(`λ_{in}`,`n_{in}`),`

`Proc(true),...)`

if `C1 := C + 1`

$\lambda^{'}$ `:=` λ_{in} `+` λ

$n^{'}$ `:= (` λ_{in} `*`n_{in} `+` λ `*` n`)/(`λ_{in} `+` λ`).`

Once the number of the accumulated flows reaches a predefined amount (indicated in processor by the value of attribute *total*), the *flow processing* rule will be triggered.

- *flow processing:* Once all incoming flows are accumulated, the node has a complete set of values of λ and n, and hence is able to calculate the new delay it incurred via the following rule. The new delay is added to the existing delay. Although we only focus on delay, we could similarly calculate other QoS attributes (e.g. jitter, packet loss,

etc).

```
crl  N(Nid(N1), Tx(X), PoiPara(λ,n),...)
     SF(PoiPara(λin,nin), Delayin(DIN)
        DelayOut(UNKNOWN),...)
=>   N(Nid(N1), Tx(X), PoiPara(λ,n),...)
     SF(PoiPara(λin,nin), Delayin(DIN)
        DelayOut(DOUT),...)
if   ( X - λ * n ) > 0
     DOUT := nin / ( X - λ * n ) + DIN
```

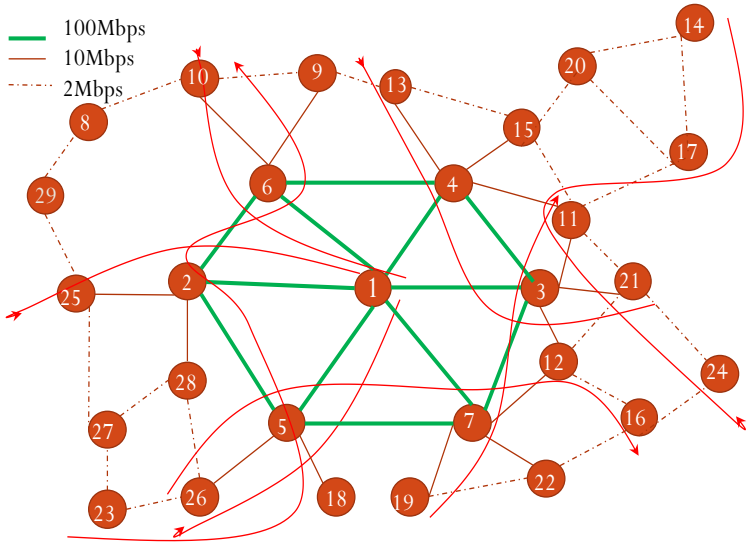
- *flow propagation* Once the outgoing flow is generated by the node via the *flow processing* rule, it will be passed into the next node as an incoming flow. This way, the delay value can be propagated from the source to the destination.

```
rl   SF(Par(F1), Sub(S1), Src(N1), Dest(N2),
        Delayout(DOUT),...)
     SF(Par(F1), Sub(S2), Src(N2), Dest(N3),
        Delayin(UNKNOWN),...)
=>   SF(Par(F1), Sub(S1), Src(N1), Dest(N2),
        Delayout(DOUT), ...)
     SF(Par(F1), Sub(S2), Src(N2), Dest(N3),
        Delayin(DOUT), ...)
```

5.2.3 Preliminary Validation

We performed a preliminary experiment to validate our Maude model. For this, we used the example network and flows shown in Fig 5.4(a) and Fig 5.4(b) as our test scenario. The

example network has 30 nodes connected via links with different capacities (i.e. Ethernet 100Mbps, WiFi 10Mbps, BlueTooth 2Mbps), and 8 flows are specified with different arrival rate and packet lengths. Flow has two types: web/ftp (type 1) and audio/video (type 2). We compare end-to-end delays of flows calculated by Maude with those computed by Qualnet simulator where Qualnet models the same network and flows. In Qualnet, we use point-to-point links to simulate the links in Fig 5.4(a) and we use constant bit rate applications to simulate the flows in Fig 5.4(b). The results of simulating end-to-end delay



(a) network

route	flowid	type	TP	Item length (Mbits)	Arrival num (/s)
1,2,25	1	1	0.8	0.0016	500
1,5,26	2	1	0.8	0.002	400
19,7,3,11	3	1	0.8	0.0016	500
21,3,4,13	4	1	0.48	0.0016	300
1,6,10	5	2	0.48	0.0016	300
24,21,11,17,14	6	2	0.4	0.002	200
23,26,5,2,6,10	7	2	0.4	0.002	200
26,5,7,12,16	8	2	0.24	0.0008	300

(b) Flows

Figure 5.4: Preliminary Experimental Settings

(see Fig 5.6) show that the delays obtained for each flow (x-axis of Fig 5.6) by Maude are

quite consistent with those determined by Qualnet, with an average error of less than 9%. Note that the objective of our formal model is **not** to have the exact same results as Qualnet, but rather achieve enough accuracy so that our prediction results will be viable, while at the same time our analysis method will be more efficient due to its abstraction level. We intend to determine with our formal methods analysis the trends of the delay performance in heterogeneous multinetworks and various kinds of flows. The preliminary experiment has validated our methodology. Next we further show that our methodology can provide solutions for three different case studies.

5.3 Evaluations:

We present three use cases to evaluate our what-if analysis tool. Each use case differs in its objective to adapt a multinetwork in the face of network failures. All use cases have application QoS requirements that must be satisfied. Different "what if" questions will provide guidance on how to best achieve QoS requirements. To compare the effectiveness of our formal method-based analysis approach with other approaches, we apply conventional strategies in each of the three use cases. We see that our analysis results suggest different adaptations than conventional analysis methods. We compare the suggested adaptation strategies from our analysis and conventional strategies by simulating the resulting networks to assess how well the adaptation strategy satisfies QoS requirements. We see that our analysis methodology provide superior guidance compared with conventional strategies.

5.3.1 Node Criticality Index

Node Criticality Index (NCI) is an indicator of how important a node is. Generally the importance of a node can be measured by the impact of a node's failure on other nodes or

network traffic. In this case study, we measure how end-to-end delays of flows are affected given a node failure. The node whose failure will cause the biggest end-to-end delay is the most important node. Traditional Node Critical Index approaches only focus on static attributes, e.g. node degree [15], centrality [46, 16], or workload/betweeness [17, 35]. These traditional measures are not taking into account the affect that a node failure will have on the flows that go through the failing node, and thus, the impact a failing node has on other flows due to contention caused by the redistribution of flows.

Our "what-if" analysis methodology and tool assumes that each node fails, one at a time, and generates a new network and new paths for the flows (using the Dijkstra algorithm to redistribute the flow, which is widely employed in routing protocols). We restrict failing of nodes to those nodes that are not source or destination of a flow. The reason is that if we would let those nodes fail, then we would not only change the paths of flows, we would also change the set of flows. And this would not allow us to compare the end-to-end delay results we generate for one node failing at a time. There are also nodes that are on the critical path of flow, meaning if these nodes fail, there is no longer a path between source and destination of at least one flow. Again, we assume these nodes do not fail so that we can consider the impact of a node failing under the same traffic load or set of flows. Thus, in summary, our analysis lets one node at a time fail—for those nodes that are not source, destination or critical path nodes—and compares the impact of a node's failure on all flows to compute which one has the most negative influence when failing. That node is deemed to be the most critical node.

In the following we refer to nodes that we do not let fail as *critical nodes* and those that we let fail as part of the analysis as *measurable nodes*. We are using the network and flows from Figure 5.4. The left table in Figure 5.5 below summarizes the average end-to-end delays on all eight flows in terms of the delay increase/decrease multiplication factor. The criticality rating is derived from the average multiplication factor, rating the most critical node to

be the one with the biggest positive multiplication factor. We only show the top six most important nodes as determined by our Maude analysis (left table) or as determined using traditional approaches based on degree and workload (right table). For example, when node

Failed Node	Ave	Rank
Node 2	4.2	2
Node 3	4.4	1
Node 4	1.4	5
Node 5	1.1	6
Node 6	1.6	4
Node 7	1.9	3

Importance Ranking by Degree and load			
node	degree	load	Rank
5	6	1.44	1
3	6	1.28	2
7	6	1.04	3
4	6	0.48	4
2	5	1.2	5
6	5	0.88	6

Figure 5.5: Node Importance Ranking: Maude Based Results (left) vs. Degree/Workload Based Results (right)

2 fails, the average end-to-end delay on all flows will be increased by 4.2 times, and when node 3 fails, then the multiplication factor is 4.4. In comparison, we get a different ranking with the traditional approach as shown in the right table of Figure 5.5.

To determine which analysis method delivers a better quality answer to the question "which node is the most important node?", we devised another experiment. The two most important nodes according to the "what-if" Maude analysis are node 3 and node 2 (labelled Maude Group in Fig 5.7), while traditional degree and workload-based analysis (labelled MaxDegree Group in Fig. 5.7) determine node 5 and node 3 to be the most critical ones. The next experiment assumes that the network administrator followed either the Maude analysis results of the MaxDegree analysis results and backed up those two most important nodes. Thus, it is very unlikely for those nodes to fail, but the remaining 16 nodes can still fail. For each group we model the networks in Qualnet and let the 16 nodes each fail for 300s. This means, there is a 4800s long experiment in Qualnet of each group (Maude Group and MaxDegree Group). The two 4800s experiments have the same original flow configuration.

In addition, we run another experiment of 4800s without any node failures. We compare the end-to-end delay of the Maude Group and MaxDegree Group with the original network and flows (without any failure), to see how much increase in flow delay we get for either group. Fig 5.7 shows that if we protect the two modes selected by our "what-if" analysis tool, the average delay increase percentage is smaller (i.e., 25%) than if we protected the nodes chosen by a MaxDegree-based analysis (i.e., 34%)(Although some flows in MaxDegree group do not incur extra delay, i.e. 2, 6, 8). Thus, our "what-if" analysis is able to provide a more accurate NCI for this scenario.

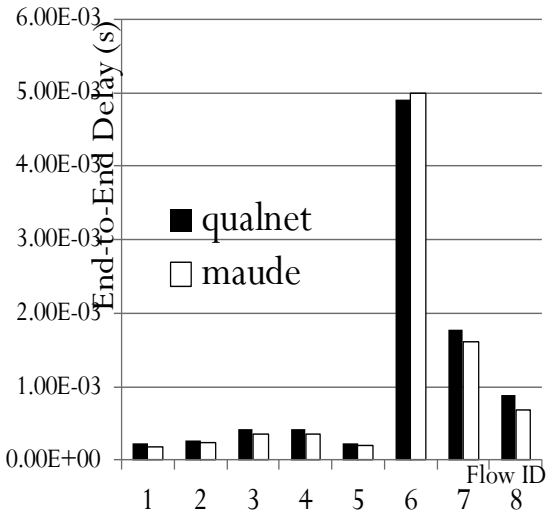


Figure 5.6: Preliminary Experimental Results

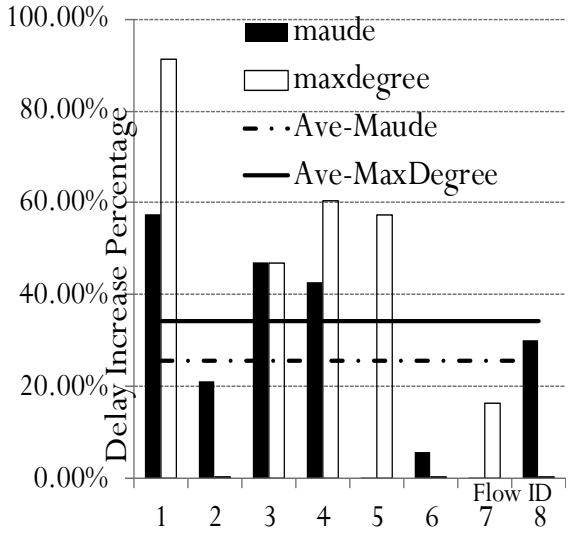


Figure 5.7: Delay increase (in %) for the two analysis approaches

5.3.2 Adding an Additional Router

If one has limited resources available, it becomes a challenge to place them in the network in a way that ensures best use of the resource's capabilities. This is a common problem for network planning and administration. For example, if we have only one extra node which happens to be a router, where should we put it so that the overall end-to-end delay for all flows will be improved most? Traditional approaches usually put the extra router next to the

node with the biggest workload. However, adding a new router will cause flow redistribution which will in turn impact the end-to-end delay of all flows. This aspect is usually not taken into consideration in traditional approaches. In our MINA architecture, wireless routers are in Tier 2 (Tier 1 is the centralized server and Tier 3 contains the mobile nodes), hence we determine that there are six tier two nodes (nodes 2-7) in Fig 5.4(a) that can be further supported by an additional router. Placing an additional router next to a node we assume that this router will have the exact same neighbors and links as that node. However, adding such extra router into the network means that the network changes and thus, flow routes will change too. Our "what-if" analysis tool examines end-to-end delay of each flow for possible position of the additional router and compares the results to determine the optimal placing of the additional router. We use again the network in Fig 5.4(a) and the flows in Fig 5.4(b) with exception of the differences indicated in Fig 5.10.

Flow ID	Ave Delay Inc.	Rank
2	98.71%	1
3	99.25%	2
5	99.61%	4
6	99.50%	3
7	99.99%	5

Figure 5.8: Best Position for additional router: Ranking by Maude analysis.

Node	Contention link	Flow ID	Load	Load sum	Rank
2	2-25	1	0.8	1.2	2
	2-28	9	0.4		
3	3-11	3	0.8	1.28	1
	3-21	21	0.48		
5	5-26	2	0.8	1.04	3
	5-26	8	0.24		
6	6-10	5	0.48	0.88	5
	6-10	7	0.4		
7	7-3	3	0.8	1.04	3
	7-5	8	0.24		

Figure 5.9: Best Position for additional router: Ranking by Workload-based analysis.

In Fig 5.8 we can see that when adding the extra router next to node 2, then the average end-to-end delay on all flows is 98.71% of the one without extra router. Thus this position has the best end-to-end delay improvement, according to our "what-if" analysis. In comparison, if traditional workload-based approach is used, the best position is node 3, as shown in Fig 5.9. Note: Node 4 is not in the Fig 5.9, because in order to do load balancing, nodes need to have at least two flows, and Node 4 only has one.

route	flowid	type	TP	Itemlength	Arrival num
13,4,3,21	4	1	0.48	0.0016	300
16,12,7,5,26	8	2	0.24	0.0008	300
1,2,28	9	2	0.4	0.0008	500

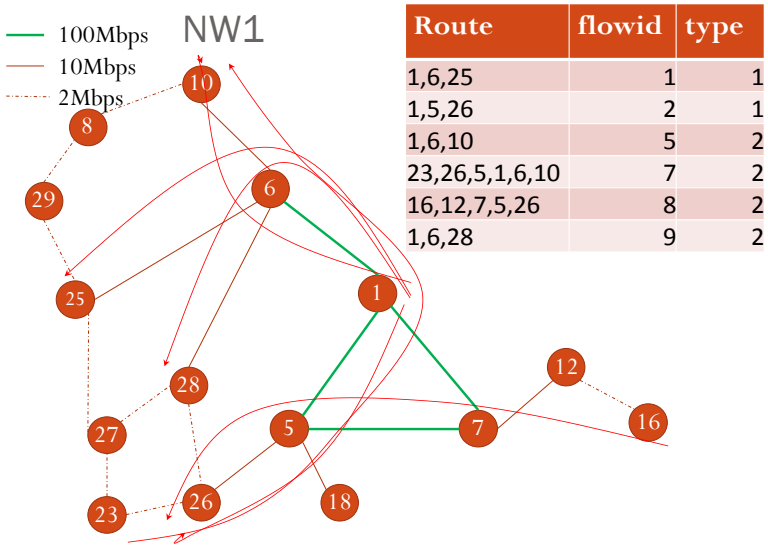
Figure 5.10: Flows in Case 2

Next we deploy the extra router next to nodes 2 and 3, respectively, and use Qualnet to test end-to-end delay improvement. The results show that the average delay decrease rate over all flows is 1.43% if we position the extra router next to node 2, while it is 0.01% if we position it next to node 3. Hence we can conclude that if we only have one extra router, node 2 is the better position to place, which is consistent with Maude "what-if" analysis.

5.3.3 Network Reconfiguration

Mobile devices usually have more than one WiFi hotspot with which to associate. Properly selecting an access point is good for network resource utilization and application performance. We use again the network and flows as in Fig 5.4. If node 2 is down, node 25 and node 28 need to re-associate with either node 6 or node 5. Hence we have four network reconfiguration plans as determined by the possible combinations for node 25 and 28 re-associating with node 5 or 6. One of the four network reconfigurations and the new routes for flows are shown in Fig 5.11. We refer to NW1 where both node 25 and node 28 re-associate with node 6 (25-6;28-6). NW2 is the network where node 25 re-associates with node 6 and node 28 re-associates with node 5 (25-6;28-5). NW3 is (25-5;28-5) and NW4 is (25-5;28-6).

The "what-if" analysis tool calculated the end-to-end delay of flows for all four reconfigurations (see Fig 5.12). The results of the Maude analysis show that NW2 and NW4 will have less delay degradation compared with the one in original network (node 2 is on). However, if we use traditional load balance-based approach (called *LoadBalance*), NW3 and NW4 are



Route	flowid	type
1,6,25	1	1
1,5,26	2	1
1,6,10	5	2
23,26,5,1,6,10	7	2
16,12,7,5,26	8	2
1,6,28	9	2

Figure 5.11: One Possible Network Reconfiguration when Node 2 goes down

the better choice, as shown in fig 5.13.

Network Config	Avg Delay Increase	Rank
NW1	54.0502%	3
NW2	27.6939%	1
NW3	65.6767%	4
NW4	28.5294%	2

Figure 5.12: Best Reconfiguration Ranking using Maude

Network config.	Node6 load:	Node5 load:	Delta	Rank
NW1	2.26	1.04	1.22	4
NW2	2.16	1.14	1.02	3
NW3	1.36	1.94	0.58	2
NW4	1.46	1.84	0.38	1

Figure 5.13: Best Reconfiguration Ranking by Workload

We then put these four resulting configurations into Qualnet to get the average end-to-end delay over all flows. As Fig 5.14 shows, Maude results are quite consistent with Qualnet results. I.e., Qualnet would prefer NW2 and NW4 over NW1 and NW 3, so does Maude. LoadBalance would prefer NW 4 and NW 3 over NW 1 and NW 2. Note the end-to-end delay generated by Qualnet is not exactly the same as the one generated by Maude "what-if" analysis tool. However, the trends and the relationship among different network configurations are quite similar.

Note we use Qualnet as benchmark because it is a well known network simulator that can

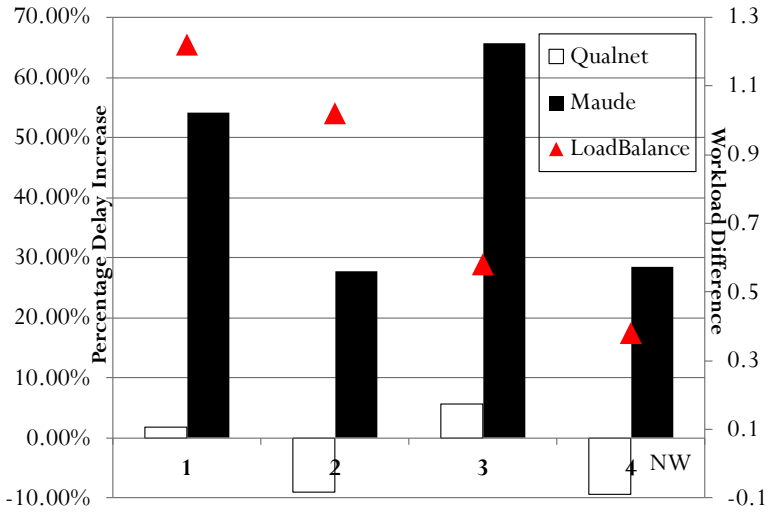


Figure 5.14: Verification by Qualnet

accurately describe the network behaviors across multiple layers. However, we cannot use Qualnet to replace "what-if" analysis tools. Maude is a high-level language providing conceptual abstractions. The three rules in our tools are the basic elements derived from the flow and queue models. Once these basic elements are in place, it is easy to set up and carry out analysis on various network without any further configuration. Network simulators usually need to simulate the packet/frame behaviors in all layers, which is more time-consuming. Moreover, having a human setting up these networks in Qualnet requires time and introduces the potential of misconfiguration by the human. In addition, increasing packet number means more calculation and time while our formal-method based tools scales up nicely due to abstraction. In Fig 5.15, we take the "No additional routers" scenario from case 2 and the "No failure happened" scenario from case 3 as examples to illustrate the efficiency and scalability of our Maude based "what-if" analysis tool. Maude only use 27ms and 11ms in two scenarios, respectively. Qualnet uses much more time in our experiments. Even if we only look at the time consumption in one second-this is the time used to simulate the same data amount with Maude, Qualnet still consumes more time. When we increased the number of packets by a factor of 10, then the time consumed by Maude is constant while it increases in Qualnet. Hence we argue that our formal language based "what-if" analysis

tools have better scalability.

	Maude	**Qualnet**	**Qualnet with 10 times packets**
Case2: bp0	rewrites: 481 in (27ms real)	20.4160s/300s =68ms/1s	24.2847s/300s
Case3: nw0	rewrites: 295 in (11ms real)	12.9815s/300s =43ms/1s	14.8934s/300s

Figure 5.15: Time consumed by Maude and Qualnet

5.4 Chapter Conclusion and Future Work

Traditional network analysis approaches mainly focus on faults and security problems in wired, homogeneous networks where nodes have enough capabilities and links are stable. In addition, those techniques do not take into consideration how changes to the network trigger flow redistribution that potentially impacts QoS of all flows.

In this chapter, we proposed and evaluated a novel what-if analysis tool based on the formal language Maude. We modeled heterogeneous networks in Maude using a general flow and queue model and provided rules that compute how end-to-end delay in the network. Using these models, we can evaluate various kinds of network changes (i.e. failures, backup and reconfigurations) and determine improved network configurations. For example, we can determine what is the best position for additional resources or which nodes are most critical when it comes to failures. Experiments from three case studies have shown that our tool outperforms traditional approaches. The what-if analysis tool could be extended to adopt more QoS example parameters and heterogeneous applications in the future. Another area of future work is to investigate flow models for other types of traffic and formalize those in Maude. This would allow us to handle network traffic more comprehensively.

Chapter 6

Adapt: Software Defined Networking Based Flow Scheduling

In this chapter we study how MINA control and adapt the heterogeneous applications over IoT Multinetworks, according to the network state information collected and processed in the last two steps. Given the heterogeneity of IoT Multinetworks, it is challenging to coordinate and optimize the use of the heterogeneous resources with the goal of satisfying as many tasks as possible.

6.1 Problem Description and Motivations:

The MINA multi-networks are fundamentally heterogeneous;they are often derived from the integration of already independently deployed CPS/IoT sub-networks, characterized by very heterogeneous devices and connectivity capabilities.

This heterogeneity poses novel challenging issues for both academic and industrial researchers, especially in order to synergically exploit the heterogeneous network resources dynamically

available in an open CPS/IoT deployment scenario. The heterogeneous network and device resources create opportunities for a wide range of applications (semantic tasks) with varying service requirements to execute concurrently. The envisioned classes of tasks may include:

1. simple point-to-point client-server applications that require real-time, dependable, and high quality message exchange - e.g., real time information about the road/vehicle status from end devices (highway camera or vehicle) to the data center, e.g., "locate yellow sedan in I-5 highway" or "determine poor road conditions along my path of transit". Such applications require low latencies and reliable delivery of information;

2. monitoring applications that collect data periodically from a multitude of data sources, such as in the case of recharging site, monitoring for global state awareness and optimization. A sample query might be "get availability of recharging sites and traffic statistics on vehicles that have been charged there". In this case, there is no strict requirement on latency (at least within one polling period) and on message loss, but a relatively significant number of updates from traffic, often generated in a very asymmetric way;

3. opportunistic exchange of local monitoring/personal data, especially between moving vehicles or between vehicles and Internet access points on the the way, e.g., "audio chat among cars in a fleet". In this case, due to the interactions between multiple parties, a lower jitter is required, while throughput might be less important.

While opportunities for new classes of applications are created in this heterogeneous setting, new challenges are introduced. The first issue involves shared provisioning of network and sensor resources across applications for efficiency. In the heterogeneous IoT setting, different user-defined tasks may run simultaneously – given the shared space they operate in, they often share the same sensing/networking resources, with differentiated quality requirements in terms of reliability (packet loss), latency, jitter, and bandwidth. Given the randomized

nature of which IoT tasks are required, these applications are often developed, deployed, and triggered in an uncoordinated manner. Optimizing sharing of sensing and communication resources and coordinating messaging in this context is challenging.

The above challenges push for another module in MINA enabling effective resource provisioning in the **IoT Multinetworks** environment, to accomplish heterogeneous IoT tasks with various requirements. In particular, MINA achieves a reasonably accurate, centralized global view of the currently available multi-network environment and takes advantage of this global view for adapting it, e.g., by reallocating application flows across paths. More importantly, MINA adopts state-of-the-art Software-Defined Networking (SDN) technologies to achieve flexible resource matching and efficient flow control in industrial deployment environments. To this purpose, we propose a novel IoT multinetwork controller, based on a layered architecture, that makes easier to flexibly and dynamically exploit IoT networking capabilities for different IoT tasks described by abstract semantics. Moreover, we modify and exploit the Network Calculus to model the available IoT multi-network and we propose a genetic algorithm to optimize its exploitation through differentiated dynamic management of heterogeneous application flows.

The benefits of employing SDN techniques in IoT environments is becoming recognized in multiple domains beyond the smart transportation setting discussed earlier by both researchers and industry practitioners. For example, [90] developed a robust control and communication platform using SDNs in a smart grid setting. Similar efforts have been explored in the smart home domain where IoT devices are extremely heterogeneous, ranging from traditional smartphones and tablets, to home equipment and appliances with enhanced capabilities. Recent efforts include a home network slicing mechanism [102] to enable multiple service providers to share a common infrastructure, and supporting verifying policies and business models for cost sharing in the smart home environment. At a lower device level, [66] employs SDN techniques to support policies to manage Wireless Sensor Networks. In sum-

mary, while there is significant interest in managing IoT environments, many of the efforts in this direction are isolated to specific domains, or a specific system layer. The proposed work employs a layered SDN methodology to bridge the semantic gap between abstract IoT task descriptions and low level network/device specifications.

In this chapter, we will first show the key differences between SDN techniques in traditional Data Center Networks (DCNs) and in IoT environments, and give our vision of a layered SDN controller in IoT settings (Section 6.2). In Section 6.3 and 6.4, we illustrate how to utilize the layered view to match proper resources with low level specification to tasks with high level semantics. We introduce and modify the Network Calculus technique to accurately estimate the flow QoS performance under heterogeneous links. A novel multiple-QoS-constraints flow scheduling algorithm is proposed in Section 6.5, and we have verified it in Section 6.6. Conclusive remarks are given in Section 6.7.

6.2 Controller Architecture

Given the heterogeneity of IoT Multinetworks, it is challenging to coordinate and optimize the use of the heterogeneous resources with the goal of satisfying as many tasks as possible. We conjecture that the SDN paradigm is a good candidate to solve the resource management needs of IoT environments for multiple reasons:

- SDN allows for a clear separation of concerns between services in the control plane (that makes decisions about how traffic is managed) and the data plane (actual mechanisms for forwarding traffic to desired destinations). The decoupling encourages abstractions of low-level network functionalities into higher level services and consequently simplifies the task of network administrators;
- SDN mechanisms aim to provide a balance between the degree of centralized con-

trol/coordination through the presence of an explicit SDN controller and decentralized operations through flow-based routing and rescheduling within the network components; this balance is realized via interactions between controllers and controlled devices.

However, the current realization of SDN technologies are still far from addressing the heterogeneous and dynamic needs of IoT Multinetworks. The popular use of SDN technologies today is in DCNs [27][8], where the focus is on the collection of specific network statistics (e.g., bandwidth consumption) from nodes networked via fast interconnections within the datacenter. In contrast, a typical IoT Multinetworks setting gathers state information from devices distributed over a more loosely coupled (and possibly wide area) network. Second, performance metrics of interest in IoT Multinetworks go beyond bandwidth consumption; with more heterogeneous and time-sensitive traffic as it is the case in IoT Multinetworks, it is equally important to reduce the collection overhead and to keep the effectiveness of the overall data needs. Unlike the case of DCNs, whose network requirements primarily revolve around link utilization and throughput, IoT Multinetworks settings present additional timing related needs - such as delay, jitter, packet loss, and throughput. Third, unlike the situation in a DCN, link and node capabilities in IoT Multinetworks are very heterogeneous and the application requirements are also different. This implies that the single objective optimization techniques in DCN flow scheduling, such as bin packing [27] and simulated annealing [8], are not directly applicable in IoT Multinetworks. Finally, the nature of interactions in current realizations of SDN (e.g., OpenFlow [68]) is limited to *south-bound*, i.e., lower layer interactions between controller and devices such as switches. The so-called *north-bound* interactions between applications/service and controller have received much less attention and are not standardized [70]. Although there are proposals [96, 49] that advocate the use of a network configuration language to express policies such as "ban a device if its usage over the last five days exceeds 10 GB", these policies still focus on lower layer parameters of the

network stack.

More recently, SDN techniques are being applied to wireless networks. OpenRadio [10] suggests the idea of decoupling the control plane from the data plane to support ease of migration for users from one type of network to another easily, in PHY and MAC layers. CellSDN [63] enables policies for cellular applications that are dictated by subscriber needs, instead of physical locations - providing finer control of network flows than previously possible. The OpenWireless [85] prototype supports seamless handover between WiFi and WiMax networks when video data is streamed, using OpenFlow controllers. The wireless SDN solution provides the necessary building blocks for managing IoT Multinetworks, but they are not sufficient. The south-bound approach retains its focus on connecting to a specific lower-level access network; its application to IoT Multinetworks must support mechanisms that abstract out the network heterogeneity. Furthermore, the framework must support north-bound, higher layer interactions, i.e., to the heterogeneous applications and their requirements.

In this chapter, we propose a novel IoT Multinetworks controller architecture to overcome these limitations. As shown in Fig. 6.1, the data collection component collects network/device information from the IoT Multinetworks environment and stores it into databases. This information is then utilized by the layered components in the left side. The controller also exposes the Admin/Analyst APIs, which enable the control processes to be governed not only by the controller itself but also by humans or external programs. Note that while the controller is logically centralized, to improve scalability it can be instantiated multiple times in different locations, e.g., in a per-domain per-service fashion.

We argued that the concept of an abstraction level is fundamental to our vision of IoT Multinetworks since it allows to make use of the heterogeneous multinetwork resources in a flexible manner. As shown in Fig. 6.2, tasks are the highest level of abstractions in IoT Multinetworks that define what is required; this leaves open the choice of what applications/services,

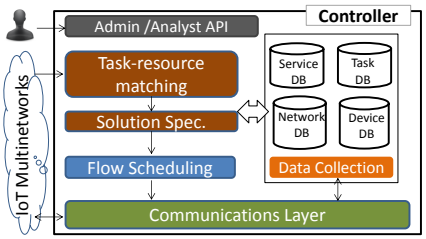


Figure 6.1: IoT controller Architecture

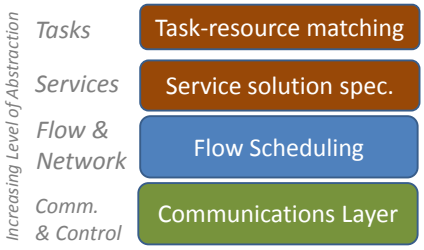


Figure 6.2: Layering in the IoT controller

devices and communication networks should be exploited to accomplish the required task. A simple example might be to determine how many vehicles currently there are in a recharging station. Services are concrete software/hardware entities that help in the realization of a task. A task may be realized by a single service (capture video from recharging station) or a workflow of services that together realize the task (capture video and count vehicles). A task/service mapping specifies which devices and applications should be used to complete the task. The lower level Flow and Network layers decide which networks should be used for application flows and how application flows should be routed across the network. These decisions will be sent out to the corresponding devices via the communication and control layer.

Such a layered view has benefits since it hides the details of lower layers (network/devices) so that tasks can be accomplished in a more flexible way. Furthermore, the separate abstraction levels allow dedicated algorithms to be designated to a certain layer for improved performance. For example, consider a specific instance of a smart space IoT setting (as described earlier). Example tasks here might be "Locate Cab 001 in I-5 free way section 107" or "Alert all vehicles about accident in I-5 free way section 107". Once such a task is submitted to the controller from a requesting node, the controller components process it through a series of steps:

- The task-resource matching component of the controller maps the task request onto the existing resources in the multinetwork. For example, for the first task ("Locate

Cab 001 in I-5 free way"), this component will determine a set of available resources in that location ("I-5 free way section 107"). Then it will filter out resources from this set by checking whether they have the capability of locating and tracking vehicles. The information about the various capabilities of resources and what services they provide is stored in the device and service DB. For example, some resources such as cameras and mesh routers might qualify because they are adverstised to have such capabilities. The result of the task-resource matching component is a set resource solutions, where each resource solution is a set of resources whose combined capabilities could solve the task at hand. The task-resource matching component then further refines each of the resource solution. In our example of locating and tracking vehicles, for the solution that consists of a road camera and a server for image processing, the refinement yields that the video stream coming from the raod camera is sent to the server and it also determines the image processing techniques that will be employed at the server side. These instantiated resource solution, or solution for short, can be filtered by automated policies at the controller or via a human in the loop (i.e., a network operator) - this will decide which solution the controller will adopt and further optimize it (Section 6.3).

- Once a solution is selected, the service solution specification component of the controller maps the characteristics of the devices and services involved in that solution to specific requirements for devices, networks, and application constraints (e.g., minimum throughput). For example, the solution that uses a road camera to locate and track vehicles will imply certain data rate and delay requirements of the video surveillance service, given the video frame resolution, codec, and receiver's buffer (Section 6.4).

- The Flow Scheduling component takes these requirements and schedules flows that satisfy them. Scheduling and coordination of the resources in IoT Multinetworks are complex due to the heterogeneity of the networks and various QoS requirements of flows. We propose to use a logically centralized management and coordination compo-

nent (the flow scheduling algorithm is described in details in Section 6.5).

- Finally the controller triggers the necessary communications in the IoT Multinetworks, e.g., a command like "routing the video data sent from Camera 001 via Ethernet" will be sent to the devices along the path.

6.3 Resourcing Matching

As discussed in the previous section, assigning heterogeneous network or device resources to heterogeneous IoT tasks is challenging. The major reason is that IoT tasks are usually depicted in an abstract manner and they are independent of the underlying network and device resources specifications. Thus a bridge between high level task descriptions and low level resource specifications is needed.

We employ a semantic modelling approach to provide such bridge. We use semantic technology (ontologies and rules) to describe (a) characteristics and capabilities of network and device resources as well as services and (b) IoT tasks as hierarchical semantic tasks descriptions where high-level tasks are refined through sequences or alternatives of low-level tasks. For example, mesh routers are captures as resources that have the cabability to locate, match, and track. Similarly, wireless devices in cars can be identified by mesh routers and than tracked. By way of example, here is a generic task description for "LocateAndTrack(Vehicle,Location)" defined through the following task plan template:

```
LocateAndTrack(Vehicle,Location)=
FindLocationResources(in:Location,neededCap:LocationCapability,
                        out:res:SetOf(LocationResources));
Match(in:Vehicle, SetOf(LocationResources), neededCap:Tracking,
                        out:SetOf(LocationAndMatchingResources))
For all Res in SetOf(LocationAndTrackingResources) Do
   If Res = Camera then
      TrackUsingVisualMeans(in:Res,neededCap:  CameraTracking,
                                        out:TrackingData)
   ElseIf Res = MeshRouter then
      TrackUsingDigitalMeans(in:Res,neededCap:  DigitalTracking,
                                        out:TrackingData)
```

Semantic task descriptions are hierarchical and task parameters are fully typed in terms of ontological concepts. At the lowest level of the task hierarchy are so-called primitive tasks. Primitive tasks are not decomposed any further, rather they are described in terms of a capability that is needed in order to perform the primitive task. In the above example, the locate and track task is decomposed into three subtasks: find location resource, match, ad tracking using either visual or digital means. Each of the subtasks has certain capabilities that a resource must have in order to be considered a possible solution. Through these capabilities, the high-level IoT task description is connected to the lower-level devices, networks and services. The requirements on the capabilities are hard constraints that must be satisfied. Semantic task descriptions can also have additional soft constraints. For exmple, there might be a desire to have high resolution cameras to yield better identification, but a medium resolution camera might also suffice.

Using the database of semantic descriptions of networks, devices and services, our analyzer can match resources to a given task. It is important to note that the analyzer does not depend on the capabilities needed or provided: it is agnostic to the specific domain to which it is applied. The analyzer only assumes that there is a task plan structure with complex tasks refined to primitive tasks specifying required capabilities, while for resources it is assumed that they specify what capabilities they provide.

We have used this approach successfully in other projects [34, 33]. These projects had the

focus to determine the interoperability of various live or simulated military training systems, from F18 fighters to complex simulation systems that are employed in large, joined military trainings [38, 37, 36].

Task plan templates are stored in the task knowledge base (task KB) and resource descriptions are stored in the resource knowledge base (resource KB). Having those two knowledge bases, users or controllers can match the task onto the appropriate taskplan template and submit it to the analyzer. The analyzer imports knowledge from both KBs and tries to find resources that have the required capabilities for the tasks. If such resources exist, the analyzer returns one or more solution taskplans with resources assigned to tasks, ranking solutions according to constraint satisfaction.

6.4 Service Solution Specification

Once a solution is selected, the service solution specification component maps the characteristics of the devices and services involved in that solution to specific requirements for devices, networks, and application constraints. As an instance, Use video surveillance is selected to accomplish task Locate Cab 001 in I-5 highway - detailed parameters such as video resolution(640*800), Frame rate (30fps), Codec(H.264), Client Buffer(100kbytes)are specified. These service requirements are then translated into network and resource requirements: Data Rate of at least 0.7Mbps, Delay less than 1s and Loss Rate less than 5%. The information needed to determine whether the desired data rates and delays are possible is obtained from a Network Information Base (or Network DB) that contains the state of the networks in the space.

6.5 Flow Scheduling

The centralized flow scheduling component in our layered controller can access the network state information of each link and node that is provided by the MINA global network state information view. In addition, all flows are registered in the controller hence the specifications of the flows such as QoS requirements, packet size, and packet rate are known a-priori by the flow scheduling component. One of the key modules in flow scheduling components is the network model, which takes the network state information and flow specification as input and calculates analytical results of the end-to-end performance of each flow before they are actually admitted into the networks. Finding paths for a flow with even two constraints is NP-complete [97], hence here we propose a heuristic algorithm to solve this problem. Every time the algorithm picks up a heuristic solution, it calls the network model to verify if the solution is feasible or not (i.e., if QoS requirements of flows are fulfilled or not). If not, the heuristic algorithm continues to the next iteration until it finds one feasible solution or a predefined iteration time is reached.

6.5.1 Network Calculus-Based Model

Generally, there are two methodologies in analysing QoS in communication networks, one is Queueing Theory [44] and the other is Network Calculus [61]. Queueing Theory is the general mathematical study of queues; it models communication requests or packets as discrete items which could be buffered in a queue and wait for services provided by the server. It has played a fundamental role in modeling, analyzing, and dimensioning communication networks [55]. Initially Queueing Theory was derived from modeling the telephone network. However, unique customer and service characteristics and requirements in such packet-switch networks often make its adoption difficult [55]. Hence more analytical techniques are developed in packet-switched networks. Network Calculus is a technique dealing with queueing

type problems encountered in modern packet-switched computer networks. Its focus is on performance guarantees, modeling the arrival traffic, service capability, and departure traffic as curves. The curves in Fig. 6.3 represent the data volume that arrived ($A(t)$), was served ($S(t)$) and departed ($D(t)$) on a node (system) in a time interval [0,t).

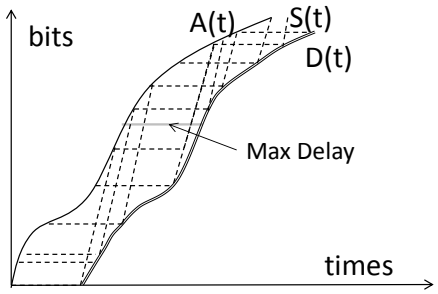


Figure 6.3: System with Service $S(t) = R[t - T]^+$, arrive curve $A(t)$ and departure curve $D(t)$

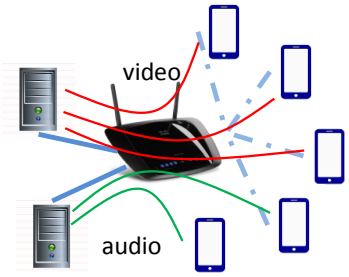


Figure 6.4: Initial Validation Scenario

In this paper we assume that each node has a constant capacity R and can provide a service curve $S(t) = R[t - T]^+$, as shown in Fig. 6.3, where R is the capacity (transmission rate), $[x]^+ = max\{0, x\}$, and T is the transmission delay, which is the time between the first bit of the packet entering the queue and the last bit getting out of the transmitter. Obviously, T depends on R, the length of this packet, and the amount of data in front of this packet when it hits the queue. The core part of this technique is the use of min-plus convolution on arrival and service curves, to generate a departure curve:

$$D(t) = A(t) \otimes S(t) \tag{6.1}$$

which means:

$$D(t) \geq \inf_{s \leq t}(A(s) + S(t - s)) \tag{6.2}$$

There are two properties enabling Network Calculus to model multiple flows in complex networks:

a) if there is more than one flow going through a node, all flows share the same transmission service. Here we assume each intermediate node has a FIFO scheduler, in which packets are served in a sequence as they arrived. Flow i will have a leftover service curve:

$$S_i = \frac{\theta^i}{\Sigma_{j \neq i} \theta^j} R[t - T]^+ \tag{6.3}$$

where R is the capacity of the downlink of this node (transmission rate), and θ is the weight of each flow (i.e., data rate= rl);

b) in a multi-hop path, the departure curve of current hop is the arrival curve of the next hop as shown in Fig. 6.5, and a combination service curve along the path $S(t)$ can be obtained by iteratively adding each node's service curve using the associative operation in min-plus convolution.

$$S(t) = S_1 \otimes S_2 \otimes ... \otimes S_n \tag{6.4}$$

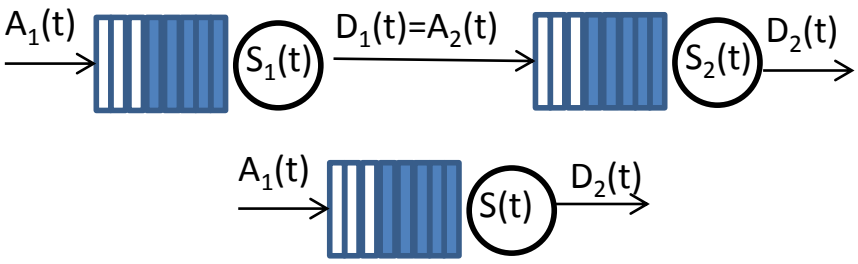


Figure 6.5: Association of service curve

However, Network Calculus originally can only provide an upper bound on the delay for the whole time scale: it is not possible to dig into the fine grained characteristics of the traffic, such as jitter. Hence we slightly modified classic Network Calculus by examining the traffic as a set of discrete points, rather than a curve, where each point represents a packet. It assumes that the profile of each flow (e.g., packet length and sending time) is known at each sender, and each packet is served by the service curve $S(t)$ with a constant capacity

R and a delay T. At the time of a packet arrives, we examine the current queue state in terms of how many packets are there in the queue and what are the lengths. The delay T is the transmission time of all packets that are already in the queue. Hence the total delay of a packet consists two parts: one is T and the other is the transmission (service) time of the packet itself. In this way, we can get an approximate end-to-end delay for each packet. To be consistent with our experiment platform (Qualnet), we defined the jitter as the difference between two successive packet arrival intervals, as specified in [20]. In this paper, we examine three QoS parameters: delay, throughput, and jitter. For each flow, we profile it with correct set of points at the sender side to plot the curve. Once we get the arrival curve $D(t)$ of flow i at the destination node by the modified Network Calculus model, we compare it with flow i's initial arrival curve $A(t)$. Each point (packet) suffers from a delay and have a final arrival time. The average delay, average jitter, and total throughput for each flow can be calculated accordingly. In our initial validation scenario, we have a two-hop network consisting of one video server and one audio server, one router and 5 clients. Each server connects to the router via a 100Mbps Ethernet link while each client connects to the router via a 2Mbps 802.11b wireless link. Each server provides either a video streaming service or a Skype voice service to one of the clients, as shown in Fig. 6.4. The video streaming and Skype voice services are based on real traces collected by the Arizona University [5] and the Polytechnic University of Turin [3]. We tested this initial validation scenario via Qualnet simulator and found the results are consistent with our Network Calculus based model, as shown in Fig. 6.6: the average error rate of the delay, jitter, and throughput are 0.05, 0.08, and 0.03 respectively.

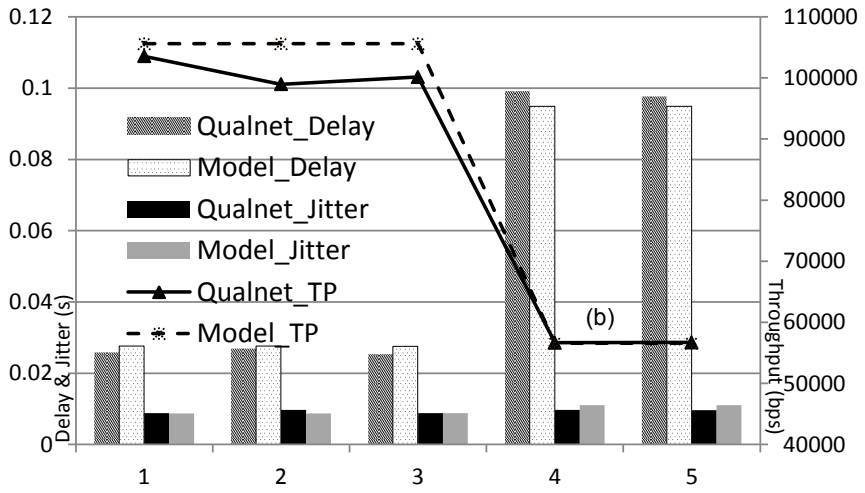


Figure 6.6: Initial Validation Results

6.5.2 Genetic Algorithm-based Multi Constraints Flow Scheduling

Multiple constraints often make the routing problem intractable [62]. For example, finding a feasible path with two independent path constraints is NP-complete [97]. Traditional flow schedulers in DCNs employs heuristic algorithms such as bin packing [27] and simulated annealing [8]; however, these algorithms have good performance when the constraints are only the link utilization and cost in wired networks, but cannot work well under multiple QoS constraints in heterogeneous networks.

Genetic Algorithms (GAs) are adaptive heuristic search algorithms based on the evolutionary ideas of natural selection and genetics. In searching a large state-space, multi-modal state-space, or n-dimensional surface, a genetic algorithm may offer significant benefits over typical search of optimization techniques, e.g., linear programming, depth-first, and breath-first. In particular, a communication path in a network perfectly matches with the chromosome concept in GAs: nodes are the genes, mutation and crossover can be done by replacing a sub-path and exchanging sub-paths between two paths, and the fitness value is the QoS

performance of the flow going through this path.

Many GA-based routing protocols with multiple QoS constraints have been proposed in the past decade, e.g., [62, 100, 60]. However, we argue that our approaches have made the following key contributions in the IoT settings: a) existing approaches only examined single flow performance, while multiple flows with different QoS requirements coexist in an IoT environment. Since the inter flow interference can greatly affect the end-to-end flow performance, our approach takes this effect into consideration; b) heterogeneous network capacity is one of the key characteristics in IoT environments, and thus our approach schedule the flows over links with difference capacity.

Problem statement

Given a directed graph $G < V, E >$, where V is the set of nodes and E is the set of links, each link $(u, v) \in E$ has a capacity $R_{u,v}$, which is equivalent with the transmission rate of node u. F is the set of flows and each flow $f_i \in F$ has several parameters: *source* s, *destination* d, *start time* t_0 and *arrival curve* $A_i(t)$. In IoT settings, each flow has a QoS requirements vector $Q_i =< w_1, w_2, w_m >$, where each element indicates one QoS parameter requirement. In this paper we use the vector $< w_d, w_j, w_t >$, which states the requirements for delay, jitter, and throughput respectively. The problem is to find a path p from source node to destination node for each flow, such that:

$$X_i(p) \vdash Q_i, \text{for each flow} f_i \iff \tag{6.5}$$

$$x_d \leq w_d \text{ and } x_j \leq w_j \text{ and } x_t \geq w_t, \text{for each flow} f_i \tag{6.6}$$

where $X_i(p) =< x_d, x_j, x_t >$ is a vector in which each element represents the end-to-end delay, jitter, and throughput of flow f_i when using path p respectively.

Data structures and Procedures

Chromosome Structure and Initialization. A chromosome represents a path, which is a list containing nodes (genes) from source s to destination d. Each flow f_i eventually has one chromosome. No duplicated genes are allowed in a single chromosome which means no loops in the path. Two initial chromosomes for each flow are set by using Dijkstra's algorithm and the second shortest path between source s and destination d.

Fitness Value. We use the following equation to calculate the fitness value for each chromosome (path):

$$[\alpha\frac{x_d - w_d}{w_d} + \beta\frac{x_j - w_j}{w_j} + \gamma\frac{w_t - x_t}{w_t}]^+ \tag{6.7}$$

where $< x_d, x_j, x_t >= X_i(p)$ is the flow end-to-end performance on delay, jitter, and throughput by using path p respectively. We employed the techniques described in Section 6.5.1 to get the fitness value. α, β, γ are the weight factors of the QoS parameter, which only depends on the flow. Here $[x]^+ = max\{0, x\}$. Apparently, a path with fitness value 0 is a feasible path. We rank individuals by fitness value, the smaller the fitness value, the higher it is ranked.

Crossover. For each flow, we choose the most two top ranked chromosomes (i.e., the shortest and the second shortest paths on the first iteration) with common genes as the parents (if they do not have common genes, we skip crossover in this iteration). A single point crossover at the common genes are performed to generate new offspring. For example, we use path s, a, b, c, d and s, e, b, d to generate two children: s, a, b, d and s, e, b, c, d by performing the crossover at the common gene b. Those four chromosomes are served as input of Mutation procedure.

Mutation. Given a path s, a, b, c, d, we choose a bottleneck node, say node b. Among the

neighbours of its last hop, node a, we randomly choose another node x which can reach the destination node d. Hence we can get a mutation path $s, a, x, .., d$. Here we determine the bottleneck node as the one incurring the largest delay along the path. For each flow, the mutation procedure takes four chromosomes as input and generates four new chromosomes. Those eight chromosomes will be ranked based on their fitness values.

Acceptance and Replacement. The outputs of mutation are eight ranked chromosomes (paths) for each flow, and the top two chromosomes will replace the current two chromosome parents of the current round of iteration. The new chromosome parents will be the input of the Crossover procedure for the next iteration.

Termination. The algorithm will iteratively run until each flow has a feasible path (with fitness value 0) or the predefined generation size is achieved. In our experiments, we set the generation size to 10.

6.6 Customized Simulation Platform and Evaluation

We have implemented a prototype of the proposed controller on top of the Qualnet simulation platform [79]. Qualnet provides a comprehensive environment for designing network protocols, and it enables creating and animating different network scenarios, under which the performance of the protocols can be analysed. We customized Qualnet with SDN features by injecting a OpenFlow-like protocol in IP layer. In every network scenario, there is only one node serving as the controller and the remaining nodes are all controlled devices. While achieved performance results already demonstrate the feasibility of the proposed approach, in the future we intend to further investigate our solution by extending the simulated environment considering the case of an actual vehicular scenario served by multiple controllers.

In Fig. 6.7, we illustrate the operation flow of how this protocol works in a software defined

manner:

1. service or application requirements, network topology, and device properties are registered to the controller and stored in the database;

2. the controller translates service requirements into network QoS requirements. Preprocessing and analysis is performed if necessary;

3. the controller exploits the algorithm described in Section 6.5.2 to schedule flows, in order to fulfill QoS requirements;

4. the controller sends flow entries to controlled devices in charge of routing flows. A flow entry contains information such as source/destination IP address/port, IP address of next hop, and the new destination IP address;

5. controlled devices receive flow entries from the controller;

6. controlled devices identify each flow going through (by source/destination IP address/port), and check whether there is an entry for this flow, then do actions determined by IP address of next hop and the new destination IP address.

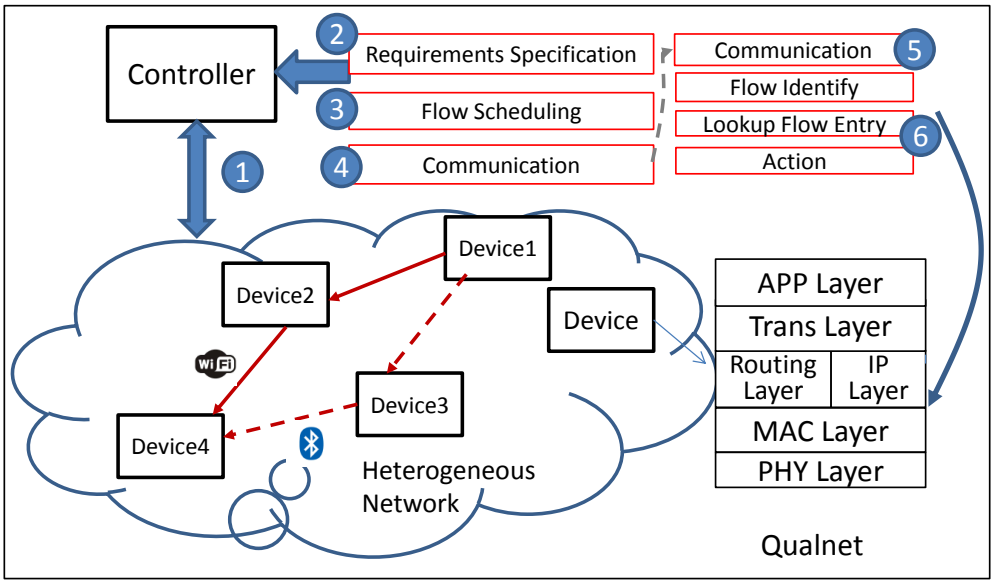


Figure 6.7: Operational Flow Diagram

Note that one of the important differences between SDN in IoT environments and in DCNs is that in IoT end devices usually have multiple network interfaces and horizontal/vertical handovers often happen. Hence, once the intermediate device reroutes a flow, it should not only change the next hop, but also the destination IP address. For example, in Fig. 6.7 when Device 1 reroutes a flow destined to Device 4 from 1-2-4 to 1-3-4, it should not only change the next hop from Device 2 to Device 3, but also change the destination IP address from WiFi interface address to Bluetooth interface address of Device 4. Of course, this procedure requires a more secure mechanism operated in the intermediate device, which is one of our future work directions.

In this section, we evaluate our GA-based flow scheduling methodology and compare it with other two common scheduling algorithms used in SDN world: bin packing and load balance. The former tries to maximize the link utilization, which means it tries to accommodate as many flows as possible into a single link. Instead, the latter assigns flows into a link so that the total amount of the flows are proportional to the capacity of the link. In order to have reasonable results, we exploit a real deployed smart campus network topology [4]. This topology is quite similar to actual application cases where vehicles with wireless connectivity exploit either LTE or WiFi road side units to receive data from servers. The topology consists of 3 data servers, 3 edge switches (each server has a 1Gbps Ethernet link to one single edge switch), 2 core routers(each edge switch has one 10Gbps Ethernet link to every core router), and 15 access points (each access point has one 100Mbps Ethernet Link to every core router) with 45 end devices. There are three types of access points: WiFi, Femtocell and Bluetooth, with data rates 10Mbps, 2Mbps, and 1Mbps respectively (end devices have direct connection with access points). A SDN controller is connected to the network with the layered functionalities. Each device has three network interfaces, at each time instance only one interface can be used; however, vertical handover could be performed if necessary. Each of the three data servers provides either file sharing, tele audio, or video streaming services. We assign each of the 45 end devices a service, randomly chosen from 16 file

sharing services, 11 tele audio services, and 7 video streaming services. File sharing flows are modeled by sending Constant Bit Rate with packet length uniformly distributed in [100, 1000] bytes with period T, the latter uniformly distributed in [0.01, 0.1] seconds. Tele audio and video streaming flows are from real traffic traces [5, 3].

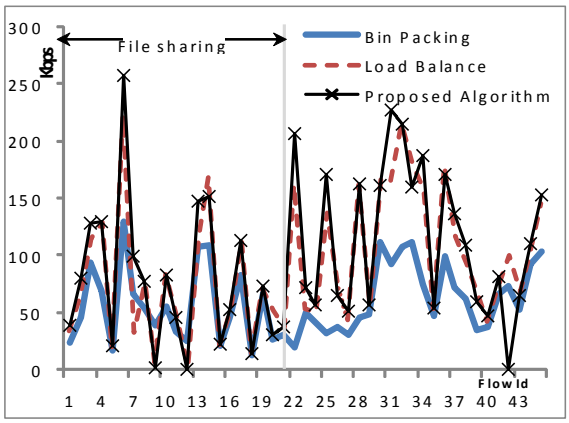


Figure 6.8: End-to-End Throughput

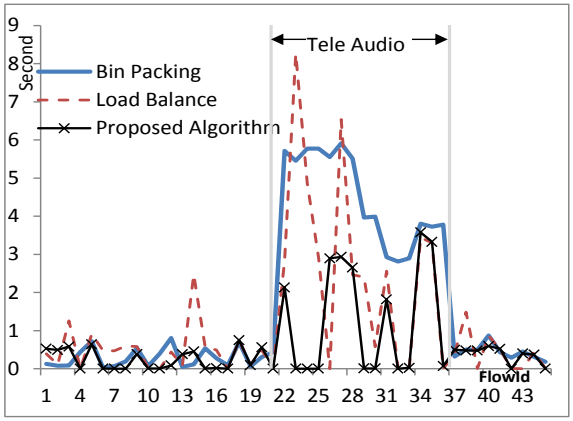


Figure 6.9: End-to-End Delay

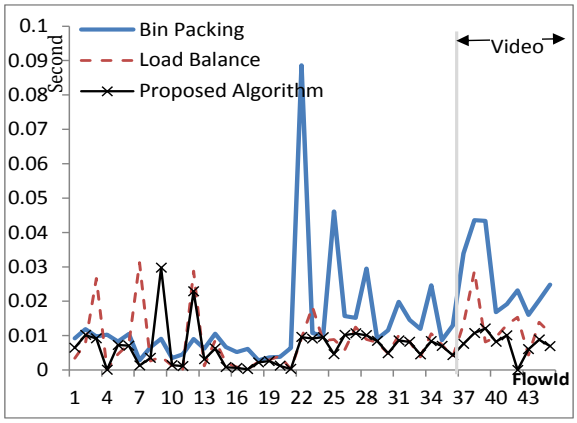


Figure 6.10: End-to-End Jitter

In our GA-based flow scheduling algorithm, we initially choose two paths for each flow as parents. Under this specific network topology, we choose the path generated by load balance algorithm as one of the parents; then, we determine the other parent by exchanging the current core route with the alternative one (we have two core routers). We argue that the file sharing service requires large throughput, the tele audio service requires low delay, while the video streaming service requires low jitter. Since QoS requirements w_d, w_j, w_t mentioned

in Section 6.5.2 highly depend on the user experience, audio/video codec and buffer size in the end devices, etc, we do not set any particular QoS requirement in this simulation based experiment. Instead, we try to optimize the QoS performance (maximize throughput, minimize delay and jitter) in a predefined amount of generations (we set 10 generations here). Hence we slightly change the fitness value in equation (6.7) with $\alpha x_d + \beta x_j + \gamma(10000/x_t)$: for file sharing flows $(\alpha, \beta, \gamma) = (0, 0, 1)$, for tele audio flows $(\alpha, \beta, \gamma) = (1, 0, 0)$, and for video streaming flows $(\alpha, \beta, \gamma) = (0, 1, 0)$.

We have totally 45 flows (each of 45 end devices has one flow): flows 1-21 are file sharing, flows 22-36 are tele audio, and flows 37-45 are video streaming. Fig. 6.8 shows the Flow Throughput comparison. For file sharing flows, the load balance algorithm outperforms the bin packing algorithm, while our proposed algorithm has an average 8% throughput increase if compared with the load balance algorithm. The reason is in wireless links when link utilization exceeds a threshold, the packet drop rate increases dramatically, as indicated in [80]. Fig. 6.9 shows that for tele audio flows, our proposed algorithm can improve the end-to-end delay performance by 51% and 71%, compared to load balance and bin packing algorithm respectively. However, the other two types of flows suffer approximately the same delay experience under these three algorithms. We argue the reason is tele audio flows have bursty traffic patterns; it might not have big data volume, but if two flows are scheduled with similar busty pattern in the same link, a large delay occurs. That is why tele audio flows have poor delay performance under bin packing and load balance algorithms. Fig. 6.10 shows that video streaming flows have an average 32% and 67% less jitter with our proposed algorithm than the other two algorithms. Two observations can be obtained here: a) video streaming flows have a better overall jitter performance than tele audio ones; b) our proposed algorithm has almost the same throughput and delay performance on video streaming flows, compared with the other two algorithms. The reason is video streaming flows have variable packet length, but almost constant inter packet interval. Hence if the interfered flows also have a stable inter packet interval, the jitter should be low. In fact, our proposed algorithm schedules

more video streaming flows with flow sharing flows (more stable inter packet interval) than tele audio flows (variable inter packet interval).

Extra flow entry messages overhead exists in the beginning of the experiments. Since we assume that we perform a one time flow scheduling and flows are stable once they are initialized, we do not examine how the extra message overhead affects the network performance. However, enabling online scheduling with dynamic flow admission is also one of our future work directions.

6.7 Chapter Conclusion and Future Work

In this chapter, we have presented an original SDN controller design in IoT Multinetworks whose central, novel feature is the layered architecture that enable flexible, effective, and efficient management on task, flow, network, and resources. We gave a novel vision on tasks and resources in IoT environments, and illustrated how we bridge the gap between abstract high level tasks and specific low level network/device resources. A variant of Network Calculus model is developed to accurately estimate the end-to-end flow performance in IoT Multinetworks, which is further serving as fundamentals of a novel multi-constraints flow scheduling algorithm under heterogeneous traffic pattern and network links. Simulation based validations have shown that our proposed flow scheduling algorithm has better performance when compared with existing ones. We are currently in the process of integrating this layered controller design with our MINA software stack, in a large IoT electrical vehicular network testbed [1] and developing more secure, sophisticated tools to assist on-the-fly resource provisioning and network control.

What we have realized is that the layered controller design is critical to the management of heterogeneous IoT Multinetworks. Techniques applied at each layer could be different - in

our design, the semantic modeling approach performs resource matching and the GA-based algorithm schedules flows. Those techniques can be viewed as plug-ins and can be adjusted or replaced in different IoT scenarios. We strongly believe that our novel layered controller architecture that inherently supports heterogeneity and flexibility is of primary importance to efficiently manage IoT Multinetworks.

Chapter 7

System Implementation and Evaluation

In this Chapter we will illustrate how we design and implement the prototype of our Multi-network management system. We have a server software stack on Linux OS while a client software stack on both Linux OS and Android OS. In the following subsections we focus more on the most important modules inside server and client stack, as mentioned in Chapter 3. MINA is built on top of the RAMP middleware ([12] [13]) to support low level communication that MINA needs.And MINA applied the OAA loop to collect network state information from managed devices to a centralized data base where analysis will be performed. adaptation and configurations then can be made according to the analysis results.

A network management system typically contains a Management Information Base (MIB) that describes the data structures and schemas that the network state information relies on. The network state information is converted to specific format and record after inserted into the data base and served as the results of queries issued by a management station. In the following Sections we first introduce our information model used to maintain the state

information to the Server.

7.1 Network state Information Base

In MINA, we have designed an Object-oriented Network state Information Base (NIB) [53]that is used by both the Server and Client software stack. Each managed device gathers local network information and report it own instance of the Network state Information Base in order to answer possible queries from the Server.

The most relevant Classes involved in our NIB, also depicted in Figure 7.1 as a UML Class diagram, are the following:

- *Node*
- *NetworkInterfaceCard*
- *Network*
- *Neighbor*
- *Link*

Node is the entry-point Class of our NIB. It models a managed device and it is used as starting point for navigating through the entire graph. Every Node is distinguished by a `UID` (Unique IDentifier), an Integer that is generated locally by the RAMP Dispatcher as the `hashCode` of the MAC address of the Node itself. Considering, however, that each node may have more than one NetworkInterfaceCard (see below) - and, thus, more than one MAC address - Dispatcher chooses one of the available MACs in a deterministic way. A MAC is a 6-byte address that is unique for each Network Interface Card (NIC) by factory and, thus,

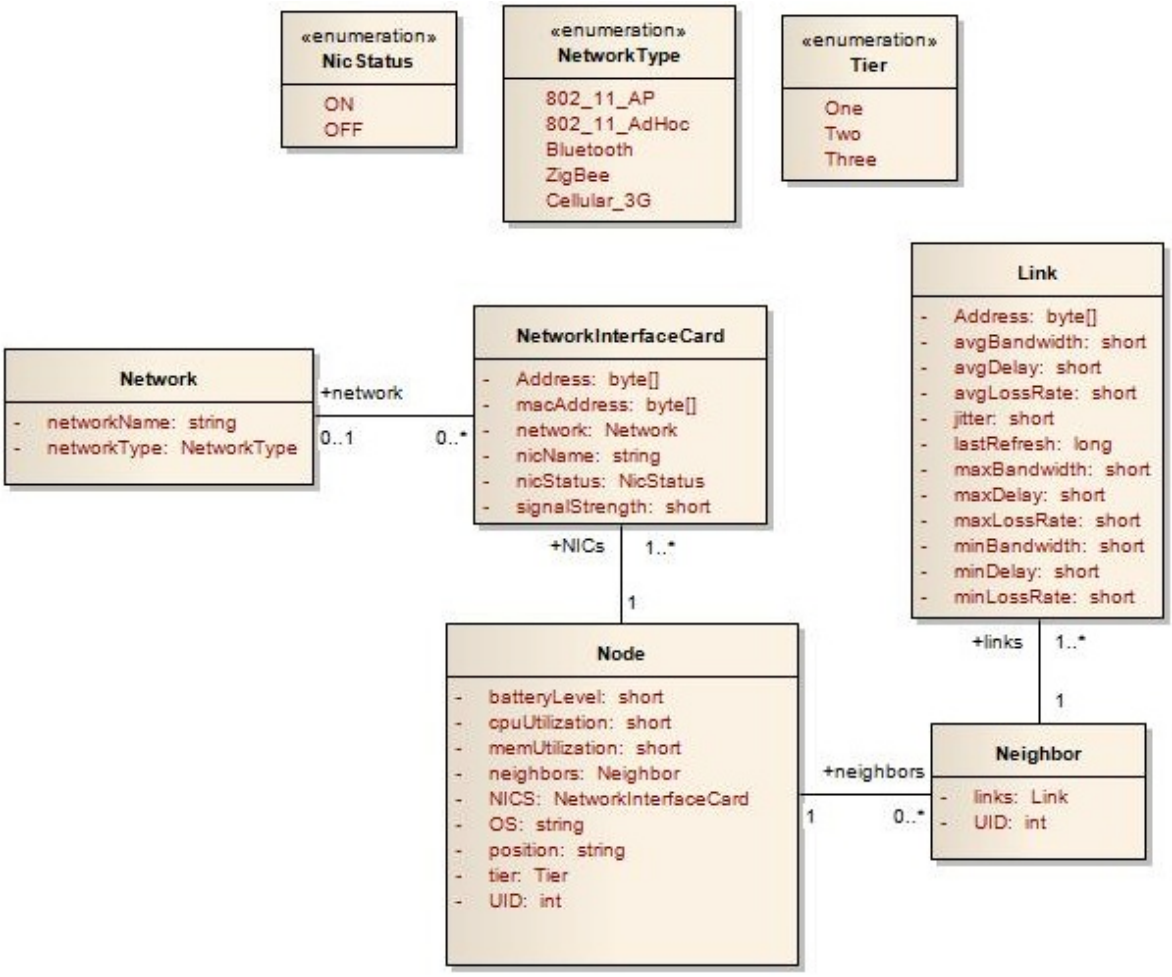


Figure 7.1: Network state Information Base (NIB)

guarantees that each Node has it own unique identifier. The enumeration `Tier` describes which Tier (in the overlay network) the Node belongs to, whereas `position` gives a textual representation of the Node's position (either symbolic or physical). For the monitoring purposes, `batteryLevel`, `cpuUtilization`, and `memUtilization` represent the percentage of battery available (important especially for mobile devices), and the percentage of CPU and RAM utilized by running processes, respectively. Each Node has a 1-to-N association with *Neighbor* and *NetworkInterfaceCard* (see below).

NetworkInterfaceCard models one network interface card available locally for this Node. It is characterized by an `address` (IP address), a `macAddress`, a `nicName`, a `nicStatus`, and `signalStrength`. The enumeration `nicStatus` refers to the actual status of the network interface (it can be either ON or OFF), whereas `signalStrength` shows the percentage quality of the wireless link (as reported by the Ubuntu NetworkManager [26] or BlueZ Bluetooth protocol stack [50]). The meaning of this attribute strictly depends upon the network type the interface is connected to. For example, it does not apply when considering an 802.11 WiFi Ad-hoc network. Each NetworkInterfaceCard is associated (if `nicStatus` is ON) with a *Network* (see below).

Network represents a network available in the managed Multinetwork environment. Each Network has a `networkName` and a `networkType`. For example, in case of an 802.11 WiFi network, `networkName` is the Extended Service Set Identification (ESSID) of the network itself, whereas in case of a Bluetooth Piconet it is the MAC address of the Group ad-hoc Network (GN) device. The enumeration `networkType` defines the network type of the modeled network.

Neighbor is a one-hop distance device reachable from this Node. `UID` is the identifier of the Neighbor (that can be used as key to find the relative Node instance). Each neighbor has a 1-to-N association with *Link* (see below).

Link models a physical link with a Neighbor in order to collect network information. Each Link is characterized by an `address` (IP address) and some metrics that model the channel from the network perspective (e.g., `bandwidth`, `delay`, and `lossRate`).

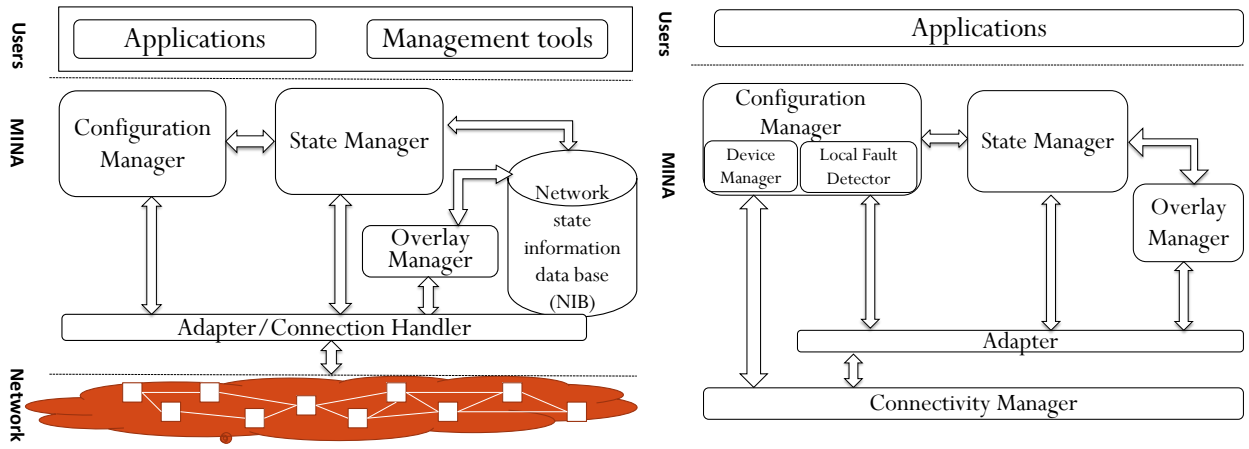


Figure 7.2: Software Stack in MINA

7.2 Server-side Software Stack

We describe the finer details about the Server side software stack [53] of our Multinetwork management system. The description will follow the general architecture that we showed in Chapter 3. Fig. 7.2(a) shows the overall architecture of the server-side software stack. In the following paragraphs we provide detailed descriptions on each module.

7.2.1 Connection Handler

Connection Handler is the lowest layer component in our architecture. It creates an abstraction layer that hides TCP/IP mechanism details to the upper layers and handles incoming connections from the lower tiers nodes. At the same time, it allows the Server to communicate with client nodes. In a nutshell, this component provides some communication APIs

to the upper layer components, simplifying the implementation of the application protocols for the state collection and query dissemination.

7.2.2 Overlay Manager

Overlay manager belongs to the second layer of the Server architecture and is responsible for creating (*Overlay creation*) and maintaining (*Overlay maintenance*) the overlay network that enables devices and Server to communicate with each other. The main goal of this component is to maintain a global picture of the Multinetwork topology. One Thread continuously listens on port 7001 UDP for incoming `OverlayMessage` (or subclasses) messages that allow to coordinate the Server with Tier 2 nodes in order to initially construct and maintain the overlay network at runtime. Each incoming packet is processed by the `handleOverlayMessage(OverlayMessage overlayMessage)` method.

7.2.3 Adapter

Adapter enables connection multiplexing and demultiplexing, allowing upper layer components to send and receive packets possibly sharing the same physical connection (*connection pooling*). As the Overlay manager, it is a second layer component and cooperates with it in order to correctly route messages in the overlay network. This component has two tasks to accomplish: on the one hand, it receives messages from the upper layer components, adapts them to the dissemination protocol, and route them towards the destination component at the client nodes.

On the other hand, *Message receiver* listens for incoming packets from the Connection handler and dispatches them to the local destination component depending on the message type. One Thread continuously listens on port 7002 UDP for incoming `ManagementMessage`

(or subclasses) messages and dispatches them to the upper layer components accordingly. The dispatching mechanism has been implemented following to the Observer pattern. The Adapter uses a Hashtable, `dispatchTable`, that allows to register a message consumer for each class type that extends `ManagementMessage`. In this way, each message consumer will receive only the type of messages which it is registered for. Every time a `ManagementMessage` is received, this method searches into the `dispatchTable` for a registered consumer and, if exists, invokes its callback method `consumeMessage`. This technique allows to create a dynamic and extensible dispatching mechanism that is loosely-coupled with the upper layer components and, thus, allows to create and integrate other components in the future without modifying the Adapter.

7.2.4 State Manager

State manager is one of the core components of the entire system. *State receiver* collects state information from the Tier 2 and Tier 3 devices, and stores them in the database. It registers itself at the Adapter as a message consumer for the `NodeStateUpdate` (a subclass of `ManagementMessage`) messages that are sent from every managed devices in order to update the Network state Information Base (NIB). As previously discussed, one of the techniques that we apply in order to reduce the overall overhead introduced by the state collection process is the message aggregation. In this specific case we refer to an aggregated message as a `NodeStateUpdate` that carries information sent from several nodes as a unique payload. Each `NodeStateUpdate` message may contain several `PartialInfo` objects that represent a partial state information regarding a specific Node in the overlay network. It would be, in fact, too much expensive (in terms of resource consumption) if each managed node sent a complete image of its local NIB every time that it needs to update the Server. Instead, only the information that has actually changed is forwarded to the Server, according to the policies that we are going to describe in Section 7.3.2. The boolean flag `aggregatedUpdate` of

`NodeStateUpdate` allows to distinguish at runtime if a message contains aggregated updates or not. If yes, upon reception, the `infoSourceNodeUID` field of each `PartialInfo` is used in order for the Server to understand which Node the partial information is referred to and to update its database accordingly. Otherwise, we can assume that the partial information is referred to the message sender, that is the `sourceNodeUID` inherited from `ManagementMessage`. Each managed device is modeled at the Server side as an instance of the Node Class (described in subsection 7.1) and is persisted (and updated) into the Network state Information Database through the `static` method `updateNode` of the *Persistence manager* (see below).

7.2.5 Persistence Manager

Persistence manager cooperates with the *State manager* in order to persist, into the Network state Information Database, the state information received at runtime by the managed devices. As mentioned in the previous paragraph, in the Java Object-oriented environment, each managed device is modeled as an instance of the Class Node with its associations, as described in Figure 7.1. In order to interact with the Relational environment typical of a database, we use the services offered by Java Persistence API (JPA) [75] that allow to automatically map a Class and its associations into specific tables of a database, depending on the cardinality of the associations themselves.

7.3 Client-side Components:

The description of Client software stack will follow the general architecture that we showed inChapter 3. The client-side software stack is responsible for collecting local network state information on the managed devices and forwarding it to the server counterpart (see Fig. 7.2(b)). From an implementation perspective, the client- and server-side stacks present many simi-

larities; we focus here on only key client-side components.

7.3.1 Connectivity Manager

Connectivity manager represents the lowest layer component in the client side architecture and is responsible to interact with the TCP/IP stack, creating an abstraction layer to the upper components. Each device is supposed to have more than one network interface and this component hides the heterogeneity providing a common API for communicating with neighbor nodes, regardless the specific kind of physical link used. In order to make a node able to communicate over heterogeneous links, it is first necessary to establish a physical connection to an available wireless connector (e.g., 802.11 AP/Ad-hoc, Bluetooth). To accomplish this step, Connectivity manager is able to dynamically discover new network opportunities (either Infrastructure or Ad-hoc) and tries to connect to them according to a given policy (e.g., choosing the connector with best signal quality). Using a Linux-based machine, for example, it is possible to automatically discovery wireless networks and configure compatible network interfaces accordingly. The *Linux Wireless Extension* [51] provides some tools (e.g., `iwlist scan`, and `iwconfig`) that have been used to automatically configure the network interface cards available on the node, if not manually configured by the user. To accomplish the same task for Bluetooth interfaces, we have used the tools provided by the Linux BlueZ Stack (e.g., `hciconfig`) [50]. In the specific case of Bluetooth devices, we leverage on the Bluetooth Network Encapsulation Protocol (BNEP) [86] that allows to encapsulate packets from various networking protocols and transport them directly over the Bluetooth Logical Link Control and Adaptation Layer Protocol (L2CAP). In this way it is possible to create a Personal Area Network (PAN) that allows to send and receive IP packets over Bluetooth, hiding the details of physical and link layers. Most of this component is based on RAMP [12] Core API and exposes the primitives to send and receive messages from the managed devices in the overlay network.

7.3.2 State Manager

State manager is one of the key components at the Client side because it is responsible for collecting local state information (filling its own instance of the NIB) and forwarding them to the Server. The most challenging part for this component is to calibrate the state forwarding policies (e.g., update frequency and data accuracy) in order to achieve a tradeoff between state accuracy collected at the Server and bandwidth consumption (i.e., fine-grained or coarse-grained state collection). It is clear that would be too much expensive in terms of bandwidth utilization and energy consumption (especially for battery-powered devices) if every managed device sent a complete image of its NIB every time that even a single property changed. Thus, the first optimization that we have introduced is the opportunity, for each Client, to send a set of `PartialInfo` messages that contains only the differential information about the state properties that have actually changed. Regarding the policies that we have used in order to forward the state updates to the Server, we argue that there are three different potential ways that we can adopt:

- *Push*
- *Pull*
- *Push with error margin (tolerance)*

With the *Push* policy, every state update is immediately sent to the Server, using the appropriate method of the Adapter component, as soon as a change in the local NIB is detected. On the other hand, with the *Pull* policy, a state update is forwarded only when explicitly requested by the Server (e.g., with a `Query` message). These two policies are completely antithetical: the former allows to have an always up-to-date fine-grained global picture of the Multinetwork environment at the Server side, at the expense of a great bandwidth utilization and energy consumption. The latter, instead, is the optimal solution if the objective is only

to minimize the resource utilization. However, in our case, it does not allow to achieve some of the most important objectives in terms of monitoring and timely fault detection because the Server may not be aware of what is happening in the managed environment. Thus, we have proposed the third policy that is an hybrid between the first two and aims to achieve a reasonable tradeoff between resource consumption and state accuracy. Each node periodically refreshes its own instance of the NIB and automatically forwards a `NodeStateUpdate` message to the Server only if the new collected values exceed the error margin (expressed in percentage). In general, the update is not forwarded to the Server if the new collected value falls into a tolerance range, as expressed by 7.1, that can be tuned in order to achieve better performance:

$$Value_{old} - \%_{error} \leq Value_{new} \leq Value_{old} + \%_{error} \tag{7.1}$$

where $Value_{new}$ is the new collected value of a NIB property, $Value_{old}$ is the corresponding previous value, and $\%_{error}$ is the percentage error tolerated. In this way, depending on $\%_{error}$, it is possible to control how many updates are actually forwarded to the server. If $\%_{error}$ is 0, the policy becomes equivalent to the *Push* one because no error is tolerated and every update is forwarded to the server. On the other hand, if $\%_{error} \rightarrow \infty$, the policy becomes equivalent to the *Pull* one because every new updated value falls into the tolerance range. Thus, tuning $\%_{error}$, it possible to achieve the desired accuracy of the network state information persisted into the Network state Information Database. The forwarding policy can be modified at runtime locally by the Client itself depending on current resource utilization (e.g., bandwidth, power level) or after receiving a configuration command by the Server in order to configure the managed devices to achieve a global goal. By delving into finer details, the *State manager* is composed by some Thread of the `LocalStateCollector` Class; each Thread is responsible for updating specific objects of the local NIB. There are several parameters that affect the state collection process and can be combined in order to real-

ize one of the aforementioned forwarding policies. `collectionPeriod`, for example, defines the state collection frequency (how often the `collectLocalState()` method is executed), whereas `errorPercentage` defines the $\%_{error}$ described in the previous paragraphs. Finally, the boolean flag `autoSendStateUpdateToServer` allows to choose between the Push and Pull forwarding policy. If the flag is true, the `LocalStateCollector` realizes the Push policy; in this case, tuning `collectionPeriod` and `errorPercentage`, it is possible to reduce the forwarding overhead, realizing the hybrid policy described before.

One important feature that characterizes only the Client software stack deployed on Tier 2 nodes is the *message aggregation*. In order to reduce the transmission overhead between these nodes and the Server, some update messages can be aggregated and transmitted as a unique `NodeStateUpdate` message. In this way, the transmission overhead can be reduced in terms of packets sent and average packet size. Therefore, it is possible to reduce the resource consumption both in terms of bandwidth and energy consumption. The main drawback of message aggregation is that it usually introduces a delay between the message sending and receiving time at the Server. This is because the `LocalStateCollector` stores the messages received from Tier 3 nodes in a queue and sends them only after a timeout expires or the queue itself reaches its capacity limit. In order to control the message aggregation, two parameters have been defined for the local state collection: `maxLocalAggregationTime` and `maxLocalAggregationSize`. The former defines how long `LocalStateCollector` waits for further messages from Tier 3 nodes (to aggregate them) before than actually forwarding them to the Server, whereas the latter defines the aggregation queue capacity.

7.4 System Validation

We have implemented a Java-based prototype of the MINA server and client - the initial implementation uses, on the server side, an Intel Core Duo 2.93 GHz, 4G RAM, Windows

7 with a MySQL 5.1 database as NIB and and Intel Core Duo 1.83GHz, 1G RAM, Ubuntu 10.04 for the client. The MINA client is also being currently ported to Android and MacOS platforms. In the following subsections, we will show 1) how does MINA improve the application performance by promptly switching networks? 2) how does the forwarding policies affect the data collection efficiency?

7.4.1 Rapid Network Switching

To evaluate the feasibility and value of rapid real-time rerouting in MINA on multiple physical networks, we deployed MINA in lab multi-network testbed and conducted experiments. Consider a setting with two multi-network paths between source S and destination R. One path uses Ethernet and IEEE 802.11g links; the other has a BlueTooth link in place of IEEE 802.11g. With the IEEE 802.11g link deactivated, we send packets (20 packets per second, 100B each) from S to R. Abrupt IEEE 802.11g link disruption makes the path unavailable, affecting service provisioning in the basic case without MINA. Instead, with MINA and the support of the **Observe** step, S quickly switches towards the alternative path (**Adapt** step). Packet delivery shows a regular trend briefly after path disruption even if the lower performance of Bluetooth 2.1 connectivity increases inter-packet arrival interval variability. Without MINA, S has to reactively find alternative paths by sending a broadcast discovery message and waiting for replies, resulting in greater packet delivery delay (Fig 7.3). (note that packets from #14 to #27 are delayed and jointly sent in a burst fashion after alternative path discovery). Without MINA, node X spends 150ms to perceive link failure, i.e. the timeout of the TCP connect phase. This only takes 35ms with MINA. Moreover, without MINA, there is an additional delay of 300ms due to alternative path reactive discovery. As an additional scenario, we experimented with the activation of a Web server on node R accessed from a browser on node S. S takes advantage of the availability of multiple paths by concurrently dispatching multiple requests towards different paths. In particular, we tested

our solution when accessing an HTML page with 15 pictures (page size 5MB); performance management reduced page rendering time by 28% from 2.9 to 2.1s.

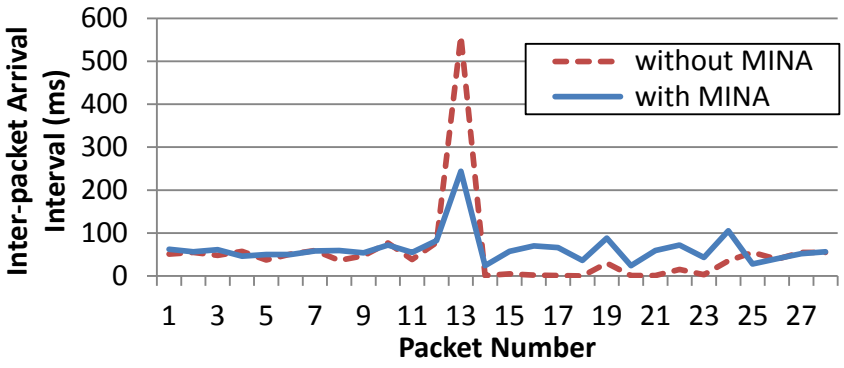


Figure 7.3: Path switch

7.4.2 Forwarding Policies

Another important feature of the system that we have evaluated is related to the *state forwarding policies*. In particular we are interested to show the difference between the pure *Push* and *Push with error margin* forwarding policies. Fig 7.4 depicts the three-node test bed that we have installed for our tests. Node A (MacBook Pro with Intel Core i7 2.7 GHz CPU and 8GB of RAM on Mac OS X 10.7.3) runs the Server software stack, whereas Node B and C (Lenovo ThinkPad X61 with Intel Core 2 Duo T7300 2.0 GHz CPU and 1 GB RAM on Ubuntu Linux 10.04 LTS) run the Client software stack Each node is equipped with IEEE 802.11b/g WiFi and Bluetooth network interfaces. The overlay network has been configured in a static way (using Java System Properties to define the relationship between nodes) [53].

Figure 7.6 underlines the effectiveness of the *Push with error margin* forwarding policy in function of the % error range (setting the % error range to 0 actually corresponds to the pure Push forwarding policy).As it is possible to notice, the "filtering" introduced by the error

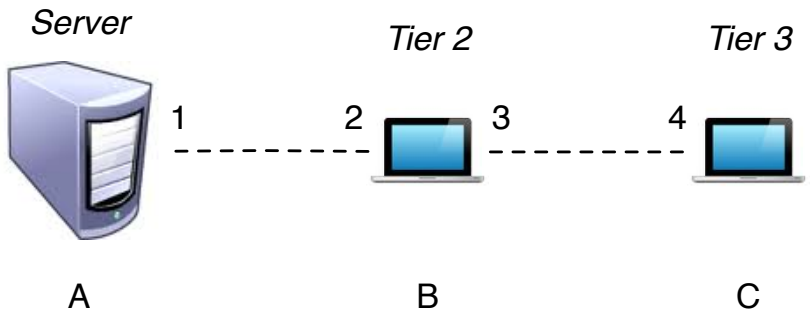


Figure 7.4: Network topology

range is almost negligible if message aggregation is enabled, whereas it can strongly reduce the number of received message by the Server if not. In the first case, Node B aggregates messages and forward them as a unique `NodeStateUpdate` message. Thus, regardless the % error range, the number of received messages by the Server is quite constant (what is actually changing is the aggregation ratio, that is inversely proportional to the % error range). However, there is still a non-negligible difference given by the aggregation mechanism for lower values of the % error range. The impact of this policy is more evident if not considering message aggregation. In this case, the higher the % error range, the lower is the actual number of received messages. We argue that message aggregation gives a higher level of determinism to our forwarding mechanism, making the state collection process more periodical. This behavior is also confirmed by the graph plotted in Figure 7.8 which shows the inter-message receiving time, that is calculated as the time period that occurs at Server side between two consecutive receptions of a `NodeStateUpdate` message. Considering a 10 seconds aggregation timeout, it is possible to notice that each single message is received at least every 10 seconds (aggregation enabled), with the inter-message receiving time increasing proportionally to the % error range. This behavior would make easier to implement a scheduling algorithm that allows to collect information from several Tier 2 nodes in different moments, reducing the chances of collisions at the physical layer and, thus, improving the overall performance of the network. On the contrary, if aggregation is not enabled, it is more

difficult to schedule the collection process from Tier 2 nodes because the global behavior of the system is more unpredictable. On the one hand, the *Push with error margin* forwarding policy allows to reduce the state collection overhead because it limits the number of messages that are actually forwarded to the Server. On the other hand, as we stated before, it affects the accuracy of state information persisted into the Server database. However, Figure 7.9

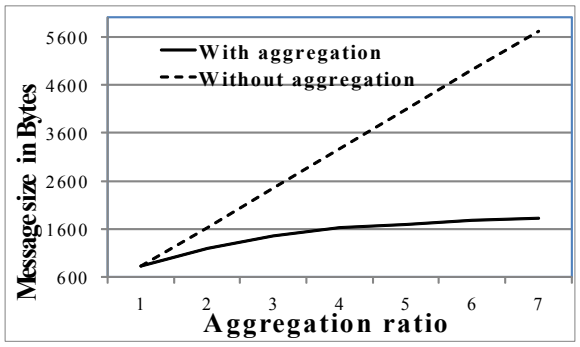


Figure 7.5: Message Size

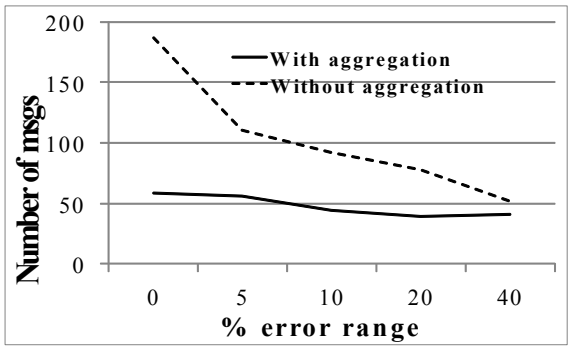


Figure 7.6: Message Number

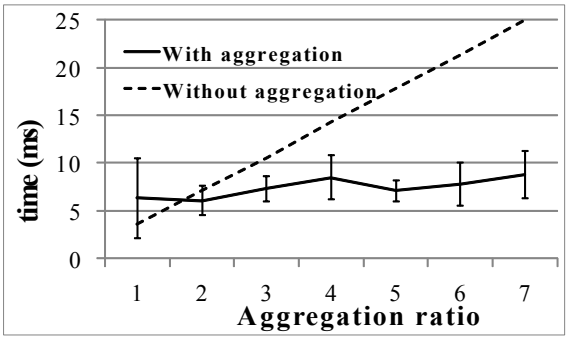


Figure 7.7: Processing Time

shows that, even with a 40% error range, the accuracy of information stored at Server side is still above the 90%. We have calculated the accuracy percentage as the ratio between the values stored into the database and the corresponding values collected from the managed devices. Finally, we are going to show how the *Push with error margin* forwarding policy significantly reduce the bandwidth utilization and how leveraging on heterogenous networks allows to balance the traffic load over the networks themselves. Figure 7.10 highlights the aforementioned behavior comparing the bandwidth utilization calculated over the WiFi link between interface 1 and 2 in Figure 7.4. As stated before, we have run two different rounds

of tests that differ by the wireless technology used to connect Node B and C. In both cases, as it is possible to see in Figure 7.10, the *Push with error margin* forwarding policy allows to reduce the bandwidth utilization increasing the % error range. Furthermore, for lower values of the % error range (i.e., when network traffic is high), using heterogeneous networks allows to significantly decrease the bandwidth utilization of the WiFi network because part of the traffic is transmitted over the Bluetooth link. For higher values of the % error range (i.e., when network traffic is low), as expected, using heterogeneous links is not so useful anymore in terms of bandwidth utilization but has still a positive impact on other metrics (e.g., interference and collisions, energy consumption, etc.).

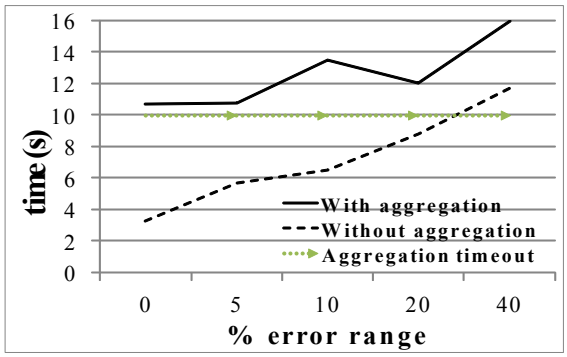


Figure 7.8: Inter Message Time

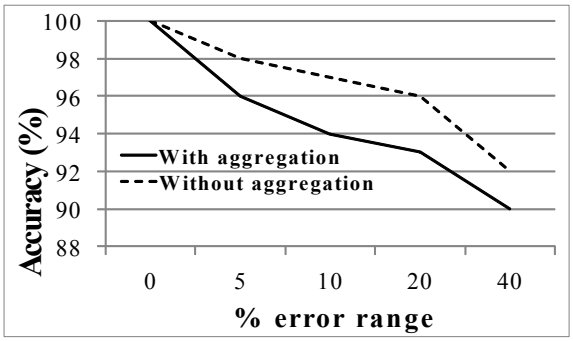


Figure 7.9: Accuracy

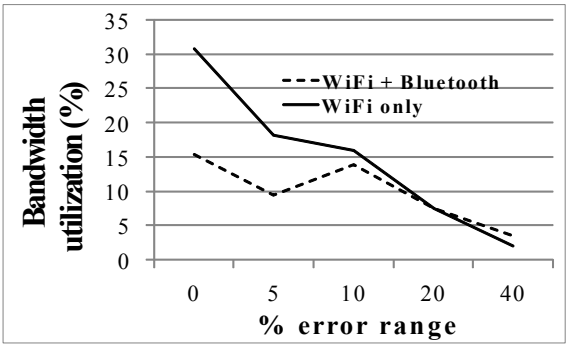


Figure 7.10: Bandwidth Utilization

Chapter 8

Conclusion and Future work

In this chapter, we first conclude our research contributions on multinetwork management in CPS/IoT environment. Then we identify and present several open areas that have not been touched or completely solved in this dissertation.

8.1 Conclusion

In this thesis, we have presented the MINA multi-network management framework whose primary goal is to organize heterogeneous devices over different networks, analyze network state information and hence provide control and adaptations for heterogeneous application flows with different QoS requirements. Its central, novel feature is the use of a reflective OAA approach. In this loop, we did 1) Efficient Topology Management on Heterogenous Devices and Network Monitoring in Convergecast; 2)Formal Method Based Multinetwork Analysis. 3) Software Defined Networking Based Flow Scheduling

For Efficient Topology Management, we presents the design, extensions/enhancements, and simulation-based evaluation of the MINA overlay network solution. The key aspect of

the MINA overlay approach is the development of a novel, dynamically constructed, and mobility-aware tree-based overlay structure that can effectively balance end-to-end data collection delay and overhead. The encouraging results achieved are stimulating our further research activities along the path duration modelling direction. We proposed a novel path duration time model for data collection in convergecast network. We claim that the probability of the multi-hop path duration time is not merely a product of each link, instead, the n-hop path duration time always based on its previous $n-1$ hop path. Besides, we show that how network density affects the path duration time. The results demonstrate that our model can accurately reflect the path duration time in simulation. We also give the analysis on the correlations between end-to-end delay and the path duration time, which will help to understand the relationship between delay, path duration time and nodes density in convergecast network. This work can be served as a guidance of link availability based routing protocol design and link quality aware system.

For Formal Method Based Multinetwork Analysis, we proposed and evaluated a novel what-if analysis tool based on the formal language Maude. We modeled heterogeneous networks in Maude using a general flow and queue model and provided rules that compute how end-to-end delay in the network. Using these models, we can evaluate various kinds of network changes (i.e. failures, backup and reconfigurations) and determine improved network configurations. For example, we can determine what is the best position for additional resources or which nodes are most critical when it comes to failures. Experiments from three case studies have shown that our tool outperforms traditional approaches. The what-if analysis tool could be extended to adopt more QoS example parameters and heterogeneous applications in the future. Another area of future work is to investigate flow models for other types of traffic and formalize those in Maude. This would allow us to handle network traffic more comprehensively.

For Software Defined Networking Based Flow Scheduling, we have presented an original SDN

controller design in IoT Multinetworks whose central, novel feature is the layered architecture that enable flexible, effective, and efficient management on task, flow, network, and resources. We gave a novel vision on tasks and resources in IoT environments, and illustrated how we bridge the gap between abstract high level tasks and specific low level network/device resources. A variant of Network Calculus model is developed to accurately estimate the end-to-end flow performance in IoT Multinetworks, which is further serving as fundamentals of a novel multi-constraints flow scheduling algorithm under heterogeneous traffic pattern and network links. Simulation based validations have shown that our proposed flow scheduling algorithm has better performance when compared with existing ones.

Finally, We point out that MINA is effectively a reflective middleware system for multi-network management; the OAA approach inherently embodies computational reflection principles [59]. The reflective feature in MINA is realized through the interaction between the MINA middleware (the meta-level) and the underlying multi-network environment (the base-level). MINA observes network state (implementing reification), analyses it to determine what is adapted and implements adaptations based on application context and observed network state (implementing reflection). The multi-network state is stored in a DB, as a meta-level representation of the underlying system state. The reflective approach is also a natural fit for the growing Software Defined Networking paradigm that aims to manage heterogeneous networks in an abstract, high-level, and logically centralized way. To the best of our knowledge, MINA is the first system to implement a reflective middleware approach and utilize on-the-fly, lightweight formal methods in the context of multi-network management.

8.2 Future Work

We identify a few open research directions listed below. The directions either extend our research in multinetwork management or study new problems involved with network analysis

and actuation.

- There are plenty of work on sensor data collection mechanisms in the literature. However, network state information collection has its own characteristics: a) network state information is the meta data of its carriers–the network, hence analysing the data content can lead to improve its non functional requirements. Utilizing the knowledge of the network state information to adapt the collection overlay is an interesting problem. b). we proposed a path duration model in covergecast networks, which can be a guide of designing advanced collected data routing protocols. Path predictable routing protocol in convergecast is another interesting open problem,

- In this thesis we employed a formal method based approach to analyze the network state information. Potential machine learning based analysis tools can be used here as well to perform more sophisticated proactive or reactive analysis. For example, by observing the 802.11 signal strength, it can predict the bandwidth of nearby bluetooth network since 802.11 radio can be interfered with the bluetooth radio.

- Employing SDN paradigm in IoT network management is new. In this thesis we only show how to reschedule flows using SDN paradigm. In fact there are more tasks we can do. For example, SDN based fault detect and recovery, malicious detection via global information analysis, etc

Bibliography

[1] Arrowhead, http://www.arrowhead.eu/ http://www.arrowhead.eu/.

[2] Model-based environment for validation of system reliability, availability, security, and performance https://www.mobius.illinois.edu/.

[3] Skype tele audio trace files http://tstat.polito.it/traces-skype.shtml.

[4] Uc berkeley campus network maps http://net.berkeley.edu/netinfo/newmaps/.

[5] Video streaming trace files http://trace.eas.asu.edu/trace/ltvt.html.

[6] A.Agarwal and P.R.Kumar. Capacity bounds for ad-hoc and hybrid wireless networks. *ACM SIGCOMM Comput. Commun. Rev., Special Issue on Science of Networking Design*, 34(3):71–81, 2004.

[7] A.B.McDonald and T.F.Znabi. A path availability model for model for wireless ad hoc networks. In *Proc. IEEE WCNC*, 1999.

[8] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, 2010.

[9] G. Ananthanarayanan and I. Stoica. Blue-fi: Enhancing wi-fi performance using bluetooth signals. In *MobySis*, 2009.

[10] M. Bansal, J. Mehlman, S. Katti, and P. Levis. Openradio: A programmable wireless dataplane. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 109–114, New York, NY, USA, 2012. ACM.

[11] H. Bany Salameh and M. Krunz. Channel access protocols for multihop opportunistic networks: challenges and recent developments. *Network, IEEE*, 23(4):14 –19, july-august 2009.

[12] P. Bellavista, A. Corradi, and C. Giannelli. The real ad-hoc multi-hop peer-to-peer (ramp) middleware: an easy-to-use support for spontaneous networking. In *ISCC10*, 2010.

[13] P. Bellavista, A. Corradi, and C. Giannelli. Middleware for differentiated quality in spontaneous networks. In *IEEE Pervasive Computing*, volume 11, pages 64 – 75, 2012.

[14] Z. B.Liu and D.Towsley. On the capacity of hybrid wireless networks. In *INFOCOM*, 2003.

[15] P. Bonacich. Some unique properties of eigenvector centrality. *Social Networks*, 29(4):555 – 564, 2007.

[16] S. P. Borgatti. Centrality and network flow. *Social Networks*, 27(1):55 – 71, 2005.

[17] U. Brandes. On variants of shortest-path betweenness centrality and their generic computation. *SOCIAL NETWORKS*, 30(2), 2008.

[18] e. a. Caesar, M. Virtual ring routing: Network routing inspired by dhts. In *ACM Sigcomm*, 2006.

[19] e. a. Castro, M.C. Performance evaluation of structured p2p over wireless multi-hop networks. In *Sensor Technologies and Applications*, 2008.

[20] A. Chadda. Quality of service testing methodology, 2004.

[21] P. C. Chakeres I. Dynamic manet on-demand (dymo) routing, 2008.

[22] C. Chen and C. A. Pomalaza-Raez. Design and evaluation of a wireless body sensor system for smart home health monitoring. In *GLOBECOM'09*, pages 1–6, 2009.

[23] Cisco prime infrastructure. http://www.cisco.com/en/US/products/ps12239/index.html/.

[24] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[25] M. Clavel, F. Durán, and et. al, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[26] U. community. Ubuntu networkmanager. *https://help.ubuntu.com/community/NetworkManager*, 2011.

[27] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, 2011.

[28] C. Davison, D. Massaguer, and et.al. Practical experiences in enabling and ensuring quality sensing in emergency response applications. In *PERNEM10*, 2010.

[29] G. de Silva, P. Matousek, O. Rysavy, and M. Sveda. Formal analysis approach on networks with dynamic behaviours. In *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2010 International Congress on*, pages 545 –551, oct. 2010.

[30] M. Di Francesco, K. Shah, M. Kumar, and G. Anastasi. An adaptive strategy for energy-efficient data collection in sparse wireless sensor networks. In *EWSN'10*, pages 322–337, 2010.

[31] Y. D.M.Shila and T. Anjali. Throuput and delay analysis of hybrid wireless networks with multi-hop uplinks. In *INFOCOM*, 2011.

[32] V. Dyo and C. Mascolo. Efficient node discovery in mobile wireless sensor networks. In *DCOSS 2008. LNCS*, pages 478–485. Springer, 2008.

[33] D. Elenius, R. Ford, G. Denker, D. Martin, and M. Johnson. Purpose-aware reasoning about interoperability of heterogeneous training systems. In *The Semantic Web*, volume 4825 of *Lecture Notes in Computer Science*, pages 750–763. Springer Berlin Heidelberg, 2007.

[34] D. Elenius, D. Martin, R. Ford, and G. Denker. Reasoning about resources and hierarchical tasks using owl and swrl. In *Proceedings of the 8th International Semantic Web Conference*, ISWC '09, 2009.

[35] E. Estrada, D. J. Higham, and N. Hatano. Communicability betweenness in complex networks. *Physica A: Statistical Mechanics and its Applications*, 388(5), 2009.

[36] R. Ford, D. Hanz, D. Elenius, and M. Johnson. Purpose-aware interoperability: The onistt ontologies and analyzer. In *Simulation Interoperability Workshop, 07F-SIW-088*. Simulation Interoperability Standards Organization, September 2007.

[37] R. Ford, D. Martin, D. Elenius, and M. Johnson. Ontologies and tools for analyzing and synthesizing lvc confederations. In *Simulation Conference (WSC), Proceedings of the 2009 Winter*, pages 1387–1398, Dec 2009.

[38] R. Ford, D. Martin, D. Elenius, and M. Johnson. Ontologies and tools for analysing and composing simulation confederations for the training and testing domains. *J. Simulation*, 5(3):230–245, 2011.

[39] P. Frossard, J. de Martin, and M. Reha Civanlar. Media streaming with network diversity. *Proceedings of the IEEE*, 96(1):39 –53, jan. 2008.

[40] Q. Gan, Bjarne, , and E. Helvik. Dependability modelling and analysis of networks as taking routing and traffic into account. In *Next Generation Internet Design and Engineering, 2006. NGI '06. 2006 2nd Conference on*, 2006.

[41] S. Gandham, Y. Zhang, and Q. Huang. Distributed time-optimal scheduling for convergecast in wireless sensor networks. *Comput. Netw.*, 52:610–629, February 2008.

[42] R. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: current state and future challenges. *Communications Magazine, IEEE*, 49(11):32–39, 2011.

[43] L. Gargano. Time optimal gathering in sensor networks. In *Proceedings of the 14th international conference on Structural information and communication complexity*, SIROCCO'07, pages 7–10, 2007.

[44] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris. *Fundamentals of queueing theory*. Wiley. com, 2013.

[45] R. S. Gruber, I. and W. Kellerer. Performance evaluation of the mobile peer-to-peer service. In *Cluster Computing and the Grid*, 2004.

[46] P. Hage and F. Harary. Eccentricity and centrality in networks. *Social Networks*, 17(1):57 – 63, 1995.

[47] Q. Han, Y. Bai, L. Gong, and W. Wu. Link availability prediction-based reliable routing for mobile ad hoc networks. *Communications, IET*, 5(16):2291 –2300, 4 2011.

[48] B. P. Han Q., Hakkarinen D. and S. J. Quality-aware sensor data collection. *Int. J. Sensor Networks*, 7(3):127–140, 2010.

[49] Hinrich, N. Gude, M. Casado, J. Mitchell, and S. Shenker. Practical declarative network management. In *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking, pp. 110. ACM, New York*, WREN 2009, 2009.

[50] M. Holtmann and al. Bluez: official linux bluetooth protocol stack. *http://www.bluez.org/*, 2012.

[51] HP. Wireless tools for linux. *http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html*, 1996.

[52] I-sensorium. http://i-sensorium.ics.uci.edu/.

[53] L. Iannario. Design and implementation of effective monitoring solutions for heterogeneous wireless networks. Master's thesis, University of Bologna, Italy, 2012.

[54] F. Ingelrest, N. Mitton, and D. Simplot-Ryl. A turnover based adaptive hello protocol for mobile ad hoc and sensor networks. In *MASCOTS '07*, pages 9–14, 2007.

[55] Y. Jiang. Network calculus and queueing theory: two sides of one coin: invited paper. In *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, page 37. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[56] V. Kalnikaite, A. Sellen, S. Whittaker, and D. Kirk. Now let me see where i was: understanding how lifelogs mediate memory. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 2045–2054, New York, NY, USA, 2010. ACM.

[57] W. Kellerer and R. Schollmeier. Proactive search routing for mobile peer-to-peer networks: Zone-based p2p. In *ASWN*, 2005.

[58] C. L. Klemm, A. and O. Waldhorst. A special-purpose peer-to-peer file sharing system for mobile ad hoc networks. In *Vehicular Technology Conference*, 2003.

[59] F. Kon, F. Costa, G. Blair, and R. H. Campbell. The case for reflective middleware. *Commun. ACM*, 45(6):33–38, June 2002.

[60] A. Koyama, L. Barolli, K. Matsumoto, and B. Apduhan. A ga-based multi-purpose optimization algorithm for qos routing. In *Advanced Information Networking and Applications, 2004. AINA 2004. 18th International Conference on*, volume 1, pages 23–28 Vol.1, 2004.

[61] J.-Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*, volume 2050. Springer, 2001.

[62] R. Leela, N. Thanulekshmi, and S. Selvakumar. Multi-constraint qos unicast routing using genetic algorithm (muruga). *Appl. Soft Comput.*, 11(2), Mar. 2011.

[63] L. Li, Z. Mao, and J. Rexford. Toward software-defifned cellular networks. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, 2012.

[64] X. Li, N. Mitton, and D. Simplot-Ryl. Mobility prediction based neighborhood discovery in mobile ad hoc networks. In *Proceedings of the 10th international IFIP TC 6 conference on Networking - Volume Part I*, NETWORKING'11, pages 241–253, 2011.

[65] H. Luo, R. Ramjee, P. Sinha, L. Li, and S. Lu. Ucan: a unified cellular and ad-hoc network architecture. In *MOBICOM 2003*, 2003.

[66] T. Luo, H.-P. Tan, and T. Quek. Sensor openflow: Enabling software-defined wireless sensor networks. *Communications Letters, IEEE*, 16(11):1896–1899, November 2012.

[67] P. Matouek, J. R, O. Ryavy, and M. Svéda. A formal model for network-wide security analysis. In *Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, ECBS '08, pages 171–181, Washington, DC, USA, 2008. IEEE Computer Society.

[68] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*

[69] A. Mclver and A. Fehnker. Formal techniques for the analysis of wireless networks. In *Leveraging Applications of Formal Methods, Verification and Validation, 2006. ISoLA 2006. Second International Symposium on*, pages 263–270. IEEE, 2006.

[70] M. Mendonça, B. N. Astuto, X. N. Nguyen, K. Obraczka, and T. Turletti. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks, 2013. In Submission In Submission.

[71] M. Menth, M. Duelli, R. Martin, and J. Milbrandt. Resilience analysis of packet-switched communication networks. *Networking, IEEE/ACM Transactions on*, 17(6):1950 –1963, dec. 2009.

[72] E. Miluzzo and et.al. Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pages 337–350. ACM, 2008.

[73] N.Bansal and Z.liu. Capacity,delay and mobility in wireless ad hoc networs. In *INFOCOM*, 2003.

[74] Hp openview. http://www8.hp.com/us/en/software/enterprise-software.html.

[75] Oracle. The java persistence api - a simpler programming model for entity persistence. *http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html*, 2009.

[76] J. Pang, B. Greenstein, M. Kaminsky, D. McCoy, and S. Seshan. Wifi-reports: Improving wireless network selection with collaboration. In *MobySis*, 2009.

[77] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (aodv) routing, 2003.

[78] P.Gupta and P.R.Kumar. The capacity of wireless networks. *IEEE Trans. Inf.Theory*, 46(2):388–404, 2000.

[79] Qualnet. www.scalable-networks.com/.

[80] R. Raghavendra, E. Belding, K. Papagiannaki, and K. Almeroth. Unwanted link layer traffic in large ieee 802.11 wireless networks. *Mobile Computing, IEEE Transactions on*, 9(9):1212–1225, Sept 2010.

[81] responsphere project. http://www.responsphere.org/.

[82] J. R. S. Nikoletseas. *Theoretical Aspects of Distributed Computing in Sensor Networks*. Springer, 2011.

[83] Safire. http://cert.ics.uci.edu/SAFIRE/index.html.

[84] P. Samar and S. B. Wicker. Link dynamics and protocol design in a multihop mobile environment. *IEEE Transactions on Mobile Computing*, 5:1156–1172, September 2006.

[85] R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.-Y. Huang, P. Kazemian, M. Kobayashi, J. Naous, S. Seetharaman, D. Underhill, T. Yabe, K.-K. Yap, Y. Yiakoumis, H. Zeng, G. Appenzeller, R. Johari, N. McKeown, and G. Parulkar. Carving research slices out of your production networks with openflow. *SIGCOMM Comput. Commun. Rev.*, 40.

[86] B. SIG. Bluetooth network encapsulation protocol. *http://grouper.ieee.org/groups/802/15/Bluetooth/BNEP.pdf*, 2001.

[87] T. N. Smyth and et.al. Where there's a will there's a way: mobile media sharing in urban india. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 753–762, New York, NY, USA, 2010. ACM.

[88] spider cloud wirless. http://www.spidercloud.com/.

[89] W. Stallings. Queuing analysis, 2000.

[90] A. Sydney. *THE EVALUATION OF SOFTWARE DEFINED NETWORKING FOR COMMUNICATION AND CONTROL OF CYBER PHYSICAL SYSTEMS*. PhD thesis, Department of Electrical and Computer Engineering College of Engineering, KANSAS STATE UNIVERSITY, Manhattan, Kansas, 2013.

[91] A. Ten Teije, Marcos, et al. Improving medical protocols by formal methods. *Artificial intelligence in medicine*, 36(3):193–209, 2006.

[92] C. K. Toh, A.-N. Le, and Y.-Z. Cho. Load balanced routing protocols for ad hoc mobile wireless networks. *Communications Magazine, IEEE*, 47(8):78 –84, august 2009.

[93] S. Upadhyayula, V. Annamalai, and S. Gupta. A low-latency and energy-efficient algorithm for convergecast in wireless sensor networks. In *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, volume 6, pages 3525 – 3530 vol.6, dec. 2003.

[94] R. Vaisenberg, S. Ji, B. Hore, S. Mehrotra, and N. Venkatasubramanian. Exploiting semantics for sensor recalibration in event detection systems. In *MMCN'08*, 2008.

[95] R. Vaisenberg, S. Mehrotra, and D. Ramanan. Exploiting semantics for scheduling data collection from sensors on real-time to maximize event detection. In *MMCN'09*, 2009.

[96] A. Voellmy, H. Kim, and N. Feamster. Procera: a language for high-level reactive network control. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, 2012.

[97] Z. Wang. and J. Crowcroft. Quality of service routing for supporting multimedia applications. In *JSAC 14 (7)*, 1996.

[98] H. Wu, Y. Liu, Q. Zhang, and Z.-L. Zhang. Softmac: Layer 2.5 collaborative mac for multimedia support in multihop wireless networks. *Mobile Computing, IEEE Transactions on*, 6(1):12 –25, 2007.

[99] X. Wu, H. R. Sadjadpour, and J. J. Garcia-Luna-Aceves. From link dynamics to path lifetime and packet-length optimization in manets. *Wirel. Netw.*, 15:637–650, July 2009.

[100] F. Xiang, L. Junzhou, W. Jieyi, and G. Guanqun. Qos routing based on genetic algorithm. *Computer Communications*, 22(1516):1392 – 1399, 1999.

[101] G. G. Xie, J. Zhan, D. A. Maltz, H. Zhang, A. Greenberg, G. Hjalmtysson, and J. Rexford. On static reachability analysis of ip networks. In *in Proc. IEEE INFOCOM*, 2005.

[102] Y. Yiakoumis, K.-K. Yap, S. Katti, G. Parulkar, and N. McKeown. Slicing home networks. In *Proceedings of the 2nd ACM SIGCOMM workshop on Home networks*, HomeNets '11, pages 1–6, New York, NY, USA, 2011. ACM.

[103] D. Yu, H. Li, and I. Gruber. Path availability in ad hoc network. In *Telecommunications, 2003. ICT 2003. 10th International Conference on*, volume 1, pages 383 – 387 vol.1, feb.-1 march 2003.

[104] H. Zhang, A. Arora, Y. ri Choi, and M. G. Gouda. Reliable bursty convergecast in wireless sensor networks. *Computer Communications*, 30(13):2560 – 2576, 2007.