A Software Defined Networking Architecture for the Internet-of-Things

Zhijing Qin*, Grit Denker§, Carlo Giannelli‡, Paolo Bellavista‡, Nalini Venkatasubramanian*
* University of California, Irvine, USA ‡ University of Bologna, Italy § SRI International, USA

***Abstract*—The growing interest in the Internet of Things (IoT) has resulted in a number of wide-area deployments of IoT subnetworks, where multiple heterogeneous wireless communication solutions coexist: from multiple access technologies such as cellular, WiFi, ZigBee, and Bluetooth, to multi-hop ad-hoc and MANET routing protocols, they all must be effectively integrated to create a seamless communication platform. Managing these open, geographically distributed, and heterogeneous networking infrastructures, especially in dynamic environments, is a key technical challenge. In order to take full advantage of the many opportunities they provide, techniques to concurrently provision the different classes of IoT traffic across a common set of sensors and networking resources must be designed. In this paper, we will design a software-defined approach for the IoT environment to dynamically achieve differentiated quality levels to different IoT tasks in very heterogeneous wireless networking scenarios. For this, we extend the Multinetwork INformation Architecture (MINA), a reflective (self-observing and adapting via an embodied Observe-Analyze-Adapt loop) middleware with a layered IoT SDN controller. The developed IoT SDN controller originally i) incorporates and supports commands to differentiate flow scheduling over task-level, multi-hop, and heterogeneous ad-hoc paths and ii) exploits Network Calculus and Genetic Algorithms to optimize the usage of currently available IoT network opportunities. We have applied the extended MINA SDN prototype in the challenging IoT scenario of wide-scale integration of electric vehicles, electric charging sites, smart grid infrastructures, and a wide set of pilot users, as targeted by the Artemis Internet of Energy and Arrowhead projects. Preliminary simulation performance results indicate that our approach and the extended MINA system can support efficient exploitation of the IoT multinetwork capabilities.**

I. Introduction

The recent surge in popularity of the Internet of Things (IoT) across multiple domains has stemmed from the spread of networking-enabled consumer devices that are deployed on a geographically wide-scale. For instance, within the context of two large EU Artemis Joint Undertaking projects, called Internet of Energy for Electric Mobility [1] and Arrowhead [2], we are addressing wide-scale deployment scenarios of IoT and the industrial interest for the near future where a large fraction of vehicular traffic will consist of electric vehicles. These scenarios give rise to many technical and organizational challenges - from the monitoring of current road traffic to the optimization of travel paths based on recharging availability; from the locating of target vehicles to the dissemination of alert message in an audio or text format; from the identification of spatio-temporal recharging patterns for enabling mass scale user behavior prediction to the optimization of smart grid management in order to adequately sustain the expected patterns of recharging requests from different geographical areas.

978-1-4799-0913-1/14/$31.00

Real world IoT deployments are fundamentally heterogeneous; they are often derived from the integration of already independently deployed IoT sub-networks, characterized by very heterogeneous devices and connectivity capabilities. The co-existence of different types of network technologies is due to legacy motivations and to different specializations in different sub-domains. Potential networks in the smart transportation case above may include single-hop wireless communications based primarily on Near Field Communications and ZigBee between neighboring cars and between cars and recharging sites (at least in the vision of big industrial players, such as Siemens and ENEL, involved in the above projects), while single/multi-hop WiFiDirect and IEEE 802.11p communications will enable the dissemination of useful recharging data among moving vehicles, as well as of user-generated entertainment content flow (e.g., tele audio or video streaming flows) between cars. In addition, 4G-based access to the standard Internet infrastructure will enable real-time collection of monitoring data at datacenters either directly from cars (often through smartphone-based gateways) or via intermediary collectors at road side units. The above 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 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 second issue is an interoperability challenge that arises when heterogeneous devices exploit different data formats for modeling information and diverse protocols for machine-to-machine (M2M) data exchange, often dictated by legacy needs and the specifics of the domain in which they are applied. The varying throughput, latency, and jitter requirements of applications' different requirements and properties enhance the complexity of state capture and resource provisioning. For instance, monitoring data between neighboring cars and a recharging site or between a recharging site and the smart grid infrastructure is often transmitted nowadays by adopting the relatively efficient machine-to-machine protocol called MQ Telemetry Transport (MQTT). However, our experience with interoperability challenges has revealed the need to adopt open and flexible protocols (even if format-inefficient and expensive), such as eXtensible Messaging and Presence Protocol (XMPP) for message exchange between cars and between a car user and a user-oriented service implemented over the support infrastructure, e.g., the recharging sites booking service.

The above articulated scenarios push for a novel software stack enabling effective resource provisioning in the **IoT Multinetworks** environment, to accomplish heterogeneous IoT tasks with various requirements. As a step towards this ambitious goal, we have developed MINA, a reflective self-observing and adapting middleware exploiting the Observe-Analyze-Adapt loop, to realize and manage dynamic and heterogeneous multi-networks in pervasive environments [3]. 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, [4] 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 [5] 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, [6] employs SDN techniques to support policies to manage Wireless Sensor Networks. In summary, 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 paper, 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 II). In Section III and IV, 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 V, and we have verified it in Section VI. Conclusive remarks are given in Section VII.

II. 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 control/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 [7][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 [7] and simulated annealing [8], are not directly applicable in IoT Multinetworks. Finally, the nature of interactions in current realizations of SDN (e.g., OpenFlow [9]) 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 [10]. Although there are proposals [11], [12] 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 [13] 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 [14] 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 [15] 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 paper, we propose a novel IoT Multinetworks controller architecture to overcome these limitations. As shown in Fig. 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 multi-network resources in a flexible manner. As shown in Fig. 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, 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.

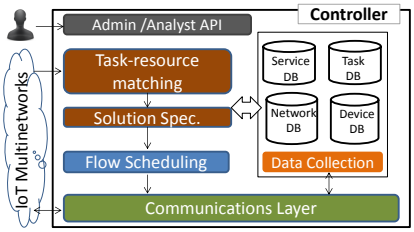


Fig. 1. IoT controller Architecture

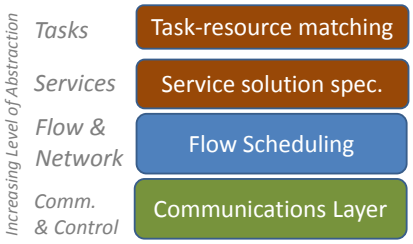


Fig. 2. Layering in the IoT controller

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 III).

- 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 he video surveillance service, given the video frame resolution, codec, and receiver's buffer (Section IV).
- 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 component (the flow scheduling algorithm is described in details in Section V).
- 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.

III. 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 [16], [17]. 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 [18], [19], [20].

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.

IV. 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.

V. 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 [21], 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.

A. Network Calculus-Based Model

Generally, there are two methodologies in analysing QoS in communication networks, one is Queueing Theory [22] and the other is Network Calculus [23]. 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 [24]. 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 [24]. 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. 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).

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. 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

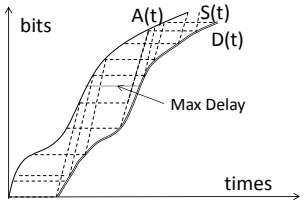


Fig. 3. System with Service $S(t) = R[t-T]^+$, arrive curve $A(t)$ and departure curve $D(t)$

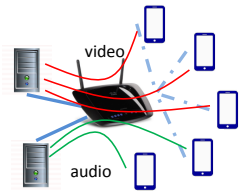


Fig. 4. Initial Validation Scenario

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:

$$D(t) = A(t) \otimes S(t) \tag{1}$$

which means:

$$D(t) \geq \inf_{s \leq t}(A(s) + S(t-s)) \tag{2}$$

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:

$$S_i = \frac{\theta^i}{\Sigma_{j \neq i}\theta^j} R[t-T]^+ \tag{3}$$

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. 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.

$$S(t) = S_1 \otimes S_2 \otimes ... \otimes S_n \tag{4}$$

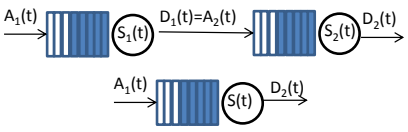


Fig. 5. Association of service curve

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 [25]. 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. 4. The video streaming and Skype voice services are based on real traces collected by the Arizona University [26] and the Polytechnic University of Turin [27]. 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: the average error rate of the delay, jitter, and throughput are 0.05, 0.08, and 0.03 respectively.

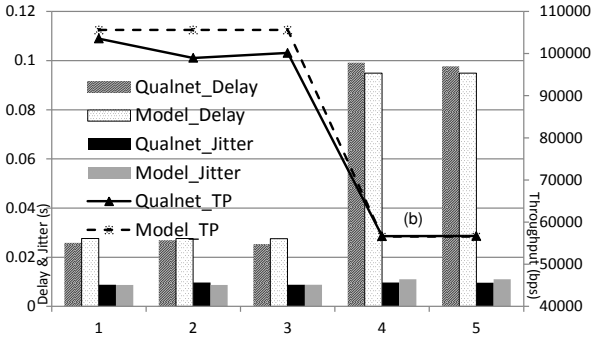


Fig. 6. Initial Validation Results

B. Genetic Algorithm-based Multi Constraints Flow Scheduling

Multiple constraints often make the routing problem intractable [28]. For example, finding a feasible path with two independent path constraints is NP-complete [21]. Traditional flow schedulers in DCNs employs heuristic algorithms such as bin packing [7] 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., [28], [29], [30]. 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.

1) 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:

$$X_i(p) \vdash Q_i, \text{for each flow} f_i \iff \quad (5)$$

$$x_d \leq w_d \text{ and } x_j \leq w_j \text{ and } x_t \geq w_t, \text{for each flow} f_i \quad (6)$$

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.

2) 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):

$$[\alpha \frac{x_d - w_d}{w_d} + \beta \frac{x_j - w_j}{w_j} + \gamma \frac{w_t - x_t}{w_t}]^+ \quad (7)$$

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 V-A 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.

VI. Customized Simulation Platform and Evaluation

We have implemented a prototype of the proposed controller on top of the Qualnet simulation platform [31]. 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. 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 V-B 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.

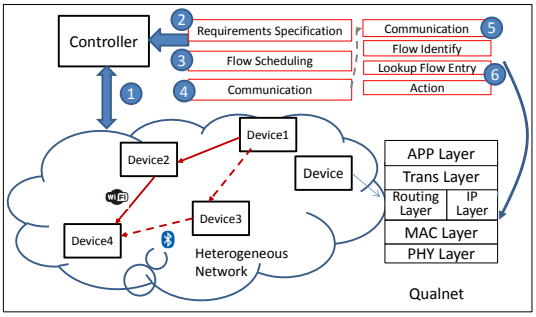
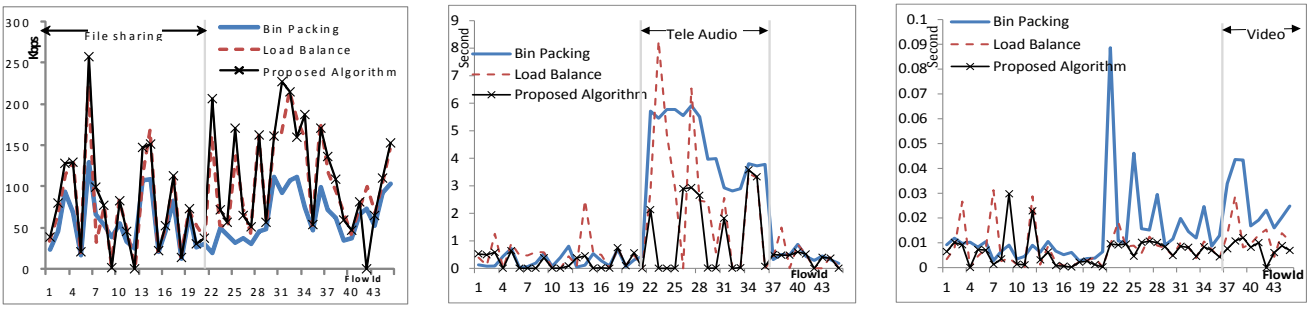


Fig. 7. Operational Flow Diagram

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. 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 [32]. 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



(a) End-to-End Throughput (b) End-to-End Delay (c) End-to-End Jitter

Fig. 8. Performance Comparisons Results

[0.01, 0.1] seconds. Tele audio and video streaming flows are from real traffic traces [26], [27].

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 V-B2 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 (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. 8(a) 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 [33]. Fig. 8(b) 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. 8(c) 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.

VII. Conclusions

In this paper, 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 [2] 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.

REFERENCES

[1] Internet of energy for electric mobility. [Online]. Available: http://www.artemis-ioe.eu/

[2] Arrowhead. [Online]. Available: http://www.arrowhead.eu/

[3] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian, "Mina: A reflective middleware for managing dynamic multinetwork environments," in *Proceedings of IEEE/IFIP Network Operations and Management Symposium 2014*, ser. NOMS 2014. Krakow, Poland: IEEE, 2014.

[4] A. Sydney, "The evaluation of software defined networking for communication and control of cyber physical systems," Ph.D. dissertation, Department of Electrical and Computer Engineering College of Engineering, KANSAS STATE UNIVERSITY, Manhattan, Kansas, 2013.

[5] 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*, ser. HomeNets '11. New York, NY, USA: ACM, 2011, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/2018567.2018569

[6] T. Luo, H.-P. Tan, and T. Quek, "Sensor openflow: Enabling software-defined wireless sensor networks," *Communications Letters, IEEE*, vol. 16, no. 11, pp. 1896–1899, November 2012.

[7] 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*, ser. SIGCOMM '11, 2011.

[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*, ser. NSDI'10, 2010.

[9] 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.*

[10] 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. [Online]. Available: http://hal.inria.fr/hal-00825087

[11] 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*, ser. HotSDN '12, 2012.

[12] 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*, ser. WREN 2009, 2009.

[13] 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*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 109–114. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342464

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

[15] 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.*, vol. 40.

[16] 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*, ser. ISWC '09, 2009.

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

[18] 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*, vol. 5, no. 3, pp. 230–245, 2011.

[19] ——, "Ontologies and tools for analyzing and synthesizing lvc confederations," in *Simulation Conference (WSC), Proceedings of the 2009 Winter*, Dec 2009, pp. 1387–1398.

[20] 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.

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

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

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

[24] 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*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, p. 37.

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

[26] Video streaming trace files. [Online]. Available: http://trace.eas.asu.edu/TRACE/ltvt.html

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

[28] R. Leela, N. Thanulekshmi, and S. Selvakumar, "Multi-constraint qos unicast routing using genetic algorithm (muruga)," *Appl. Soft Comput.*, vol. 11, no. 2, Mar. 2011.

[29] F. Xiang, L. Junzhou, W. Jieyi, and G. Guanqun, "Qos routing based on genetic algorithm," *Computer Communications*, vol. 22, no. 1516, pp. 1392 – 1399, 1999.

[30] 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*, vol. 1, 2004, pp. 23–28 Vol.1.

[31] "Scalable networks technologies http://www.scalable-networks.com."

[32] Uc berkeley campus network maps. [Online]. Available: http://net.berkeley.edu/netinfo/newmaps/

[33] 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*, vol. 9, no. 9, pp. 1212–1225, Sept 2010.