Software-defined Wireless Mesh Networking for Reliable and Real-time Smart City Cyber Physical Applications

Akram Hakiri
Univ de Carthage, ISSAT
Mateur, Bizerte, Tunisia
akram.hakiri@gmail.com

Aniruddha Gokhale
Dept of EECS, Vanderbilt University
Nashville, Tennessee, USA
a.gokhale@vanderbilt.edu

Pascal Berthou
CNRS, LAAS, UPS
Toulouse, France
berthou@laas.fr

ABSTRACT

The growing demand for and the diverse mobility patterns of smart devices place an increasing strain on the wireless mesh networks (WMNs) of smart city cyber physical systems (CPS). Realizing reliable and real-time smart city CPS applications is challenging because routing the data among wireless routers using existing routing algorithms that are based on Ad-Hoc and local area network flavors cannot make effective routing decisions due mainly to only local knowledge maintained by an individual router about each of its neighbors, which reflects only a partial visibility of the network. An attractive and more realistic alternative is to adopt Software Defined Networking (SDN), which offers a logically centralized, up-to-date view of the entire network by refactoring the wireless protocols into control and forwarding decisions. This paper presents solutions to overcome key challenges that must first be overcome to realize the potential of SDN in WMNs for smart city applications. Specifically, we describe a novel network architecture that integrates SDN and WMNs to perform network virtualization, routing and network traffic engineering thereby improving the predictability, reliability and the flexibility of the communication network. The benefits of this approach are demonstrated and evaluated for an emulated smart cities use case.

CCS CONCEPTS

• **Networks** → **Network resources allocation**; **Cloud computing**; **Data center networks**; • **Computer systems organization** → *Cloud computing*; *Fault-tolerant network topologies*;

KEYWORDS

Software Defined Networking; Internet of Things; Wireless Mesh Networks; Smart Cities Cyber Physical Systems.

1 INTRODUCTION

Context: Urban centers across the world continue to grow steadily as more than the half of the current world population is living in urban areas and the number is forecast to further increase by 2030 [18]. To help address various challenges due to the increased urbanization, innovative smart cities projects, such as VITAL [9] and Padova [35], have been commissioned by local governments and private companies to provide new solutions, services and applications. To further refine such solutions, and ultimately to make cities livable and sustainable, smart cities need real-time and reliable communication capabilities to support the quality of service (QoS) needs of such smart city Cyber Physical Systems (CPS), e.g., intelligent traffic light control and traffic coordination.

Wireless Mesh Networks (WMNs) often serve as the backbone communication technology for smart city communications [32]. WMNs often consist of mesh clients, mesh routers and gateways. Mesh clients are mobile nodes such as wireless cameras, traffic signal controllers, and other wireless devices. Mesh routers forward data to and from gateways, which in turn may connect to the Internet. The coverage area of the radio nodes operating in a single network is called a mesh cloud, which allows monitoring of vehicular traffic activity in cities to help alleviate congestion.

Challenges: Supporting the myriad of smart city CPS applications with their individual QoS properties is, however, stretching the WMN's capabilities to its limit for a variety of reasons. First, although several wireless protocols such as AODV [22] (Ad hoc On Demand Distance Vector) and OLSR [6] (Optimized Link State Routing Protocol) have been investigated in the past decade for WMNs, their designs were influenced primarily by the *ad hoc* nature and local area network (LAN) constraints. Their routing decisions are based only on local knowledge of a mesh router about its neighbors, which reflects only a partial visibility of the network and hence may result in suboptimal routing that can be detrimental to the real-time needs of smart city applications. Consequently, extending these protocols to support the timeliness needs of such applications and their high volume network traffic patterns is very difficult, which limits the ability of the WMNs to dynamically adapt to and prioritize varying network traffic streams.

Furthermore, existing routing protocols fail to provide real-time failover to reroute failed nodes or broken links, and redistribute the orphaned clients among neighboring nodes. Since most of the network traffic tends to flow between the client nodes and the gateways, the gateway will become a bottleneck in WMNs. Thus, selecting the best routes to the Internet in the mesh cloud for different traffic classes is needed for QoS support. Besides, due to link quality variations in the radio channels induced by mobility and topology changes, a mesh cloud becomes more difficult to manage and configure, e.g., managing and upgrading routers is a complex and error-prone task because the configuration of nodes must be performed manually and individually at each router.

To deploy new smart city services over WMNs, we need better manageability, control and flexibility in the network, which

RTNS'19, , November 6–8, 2019, Toulouse, France.

ACM ISBN 978-1-4503-5149-2/17/12…$15.00
https://doi.org/10.1145/3147213.3147220

is feasible using Software Defined Networking (SDN) [19]. SDN decouples the control plane from the data plane for distributed, networked applications so that all network management can be enforced from a single, logically centralized and programmable controller that resides in the control plane, while application-level messaging is carried out in the data plane. Hence, SDN shows significant promise in meeting smart city needs by optimizing routing paths for information through the network [31]. OpenFlow [15] is a dominant SDN technology for communication between control and data plane elements.

Recent promising approaches for programmable wireless networks reveal the use of SDN to build relays between home gateways and the Internet [34], simplify the network management operations of the wireless access points [25], to ensure high priority network traffic is assured network resources even during reconfiguration periods [8] and enhance the traffic orchestration [11] in virtual access points. Despite these advances, these efforts used SDN only in a single wireless access point, which makes their solutions unstable in a highly distributed wireless environment. Moreover, since SDN was initially introduced for wired networks such as cloud computing and data centers to provide packet encapsulation and tunneling, it does not yet provide any abstract programming interfaces for wireless communication. Second, the SDN requirements of centralized control and simple router design contradict with the distributed routing algorithms and sophisticated switch design of the wireless network architecture. Third, the characteristics of wireless channels, e.g., fading, interference, and broadcast require that the SDN controller offer modules to support centralized interference management, node mobility, and topology discovery. Fourth, as CPS scale up to interconnect distributed mesh clouds, it may not be feasible for a centralized SDN controller to manage the entire network, however, distributed controllers will require sophisticated coordination mechanisms that preserve application QoS.

Contributions: To address these challenges, we present a novel approach that incorporates SDN into WMNs to define and manage a powerful and easy-to-deploy CPS network that is both reliable and supports the QoS needs of CPS applications. Our approach provides a novel way to perform routing, network monitoring, and traffic engineering by defining a modified OpenFlow protocol. It also supports both centralized and distributed SDN control planes based on a bootstrapping mechanism we developed in prior work [20] that decouples the distributed systems concerns from the primary issues related to the controller.

The remainder of this paper is organized as follows: Section 2 introduces the architecture of a futuristic urban scenario through a Smart Traffic Light System (STLS) and articulates some open issues related to the deployment of SDN-based wireless communication in such smart cities system. Section 3 describes the architecture of our SDN-enabled solution for efficient support of wireless networking in smart cities and discusses the role of our approach in solving the aforementioned challenging issues. Section 4 evaluates the framework along multiple dimensions including its performance, overhead and load balancing properties. Section 5 compares related efforts to our solution. Finally, Section 6 provides concluding remarks describing potential future directions and open research problems in this realm.

2 PROBLEM DESCRIPTION AND KEY CHALLENGES

In this section we use a smart city use case to illustrate a plethora of challenges along multiple dimensions, such as wireless network virtualization, controller placement problem, traffic monitoring, and traffic engineering and allude to solution requirements.

2.1 Smart City Motivating Example

Figure 1 shows the network architecture of a Smart Traffic Light System (STLS) in a Smart City CPS, which we use as our motivating example. The STLS collects data from diverse sensing devices, roadside equipment, and cameras to detect the presence of vehicles, cyclists, and pedestrians. Circle (1) in the figure shows pedestrians wearing body-borne computers (wearable computing) and their dogs wearing a dog collar . Motorcyclists are shown wearing connected helmets and cyclists are shown riding smart bicycles equipped with smart pedal for connecting to the STLS as well as providing real-time location through a smartphone (Circle (2)). The STLS measures the distance and the speed of the approaching vehicles from every direction of an intersection (Circles (3)). It can also disseminate warnings via publish/subscribe messaging to vehicles to inform them about the possible crossing risks as well as the possibility to change the routes in case of vehicular traffic jams.

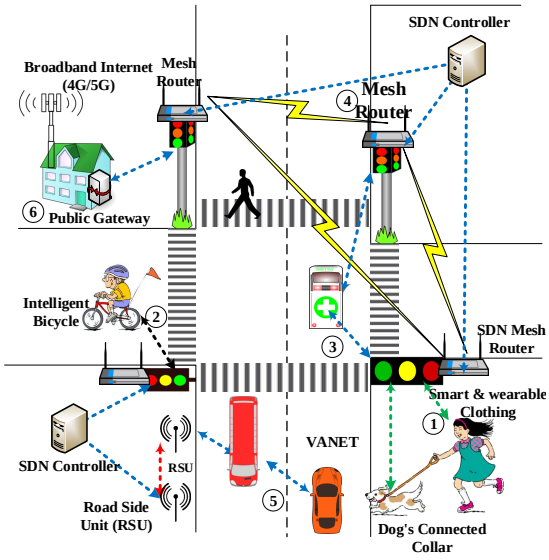

Figure 1: Intelligent Smart Traffic Light System Usecase

The STLS is envisioned as a wireless mesh network comprising distributed multi-hop wireless routers embedded in the traffic light boxes that relay into partial or full mesh topologies. These routers represent the SDN data plane that participate in the application-level messaging (Circles (4)) by routing packets between routers. This setup enables a variety of performant applications, e.g., for issuing 'slow down' warning to vehicles which are at a risk of collision, or for dynamically creating a sequence of green lights by adapting traffic lights to allow emergency vehicles to pass. The vehicles are assumed to be equipped with On-Board Units (OBUs) to connect to various network interfaces, such as Global Positioning

System (GPS), radio transceivers for Wireless Access Vehicle Environment (WAVE) [1], and Vehicular Ad-Hoc Network (VANET) [2] (Circle (5)) to communicate with each other, and connect to Road Side Units (RSUs) and mobile base stations. Each cluster of RSUs is treated as a cluster of SDN-enabled wireless routers controlled by a SDN controller.

The routers are connected to Internet gateways to provide services such as route planning, traffic alert dissemination and mobile vehicular cloud services. Finally, access to the Internet (Circle (6)) can be provided by access technologies such as xSDL, satellite or by heterogeneous and multi-technology connectivity, such as 4G/5G cellular networks.

2.2 Challenges Realizing Real-time and Reliable Smart City Applications

Using the STLS motivating use case, we now highlight the key challenges and solution requirements to resolve them.

Challenge 1: Wireless Network Virtualization. Smart clients in the STLS scenario end up repeatedly triggering the embedded controller for marshaling and unmarshaling the data thereby creating additional overhead on an already resource-constrained wireless router. Wireless router virtualization has the potential to increase network capacity and allow high volume of traffic in the STLS scenario by offloading the MAC layer processing to virtualized access points (APs) and simplifying network management operations. Running multiple non-overlapping isolated wireless networks can provide airtime fairness for multiple different groups of wireless smart clients [28]. Wireless virtualization includes virtualizing both the infrastructure, i.e., processors, memory, network interfaces, and wireless radio, and the spectrum. Spectrum virtualization has the potential to provide better utilization of wireless resources, channel isolation, control signaling, QoS allocation, and mobility management [12]. Hence, each virtual router should have its own radio configuration, capabilities for notifications, and set of distinguished services. This is a difficult task because using a large number of independent wireless channels induces channel fading due to multipath propagation and shadow fading that affects wave propagation.

Wireless virtualization should be applied to both the infrastructure and spectrum sharing. Virtualizing the infrastructure means that processors, memory, network interfaces, and wireless radio have to be virtualized. Since the spectrum is a scarce resource, spectrum virtualization should bring the potential to provide better utilization of wireless resources, channel isolation, control signaling, QoS allocation, and mobility management. Hence, each virtual router should have its own radio configuration, capabilities for notifications, and set of distinguished services. This is, however, a very difficult task because using a large number of independent wireless channels induces channel fading due to multi-path propagation and shadow fading that affects wave propagation.

Challenge 2: Lack of Efficient and Scalable Routing. As the STLS network in Figure 1 brings together diverse applications that use the wireless technologies, e.g. RSUs, wearable computing clothes, connected helmets, and connected vehicles, the design of routing protocols in such smart city networks should be sensitive about how the network can handle data as well as the speed and the processing capabilities of the wireless routers. Another challenging issue stems from enabling SDN routing in the presence of existing wireless routing protocols. Although some approaches use the IEEE 802.11s MAC layer for routing the traffic in SDN-enabled WMNs [17], the link layer multi-hop routing suffers from two shortcomings. First, in MAC layer-based routing, a limited number of wireless nodes (maximum 32 nodes) are allowed in a single network. Second, the conflicting rules between 802.11s and OpenFlow introduce severe performance degradation.

Many other interesting SDN opportunities that are not yet addressed to deal with rapid client association and re-association, and predicting the network traffic to keep all the flows between clients and the wireless routers in the network. Despite the presence of several routing protocols for IoT systems, such as LoWPAN and RPL, these routing protocols must be made dynamically adaptive to any change in the network devices over the time. Therefore, more research efforts are required to address such routing issues. Further, an important issue that needs to be addressed is the cohabitation between existing wireless routing protocols and SDN data forwarding to ensure interoperability, scalability and reliability of IoT technologies in smart cities.

Challenge 3: Centralized versus Distributed SDN Control. In the STLS scenario of Figure 1, geographically distributed mesh routers should coordinate their activities to provide a global network view and simplify their management and configuration. Nonetheless, this task is complex and hard to achieve because coordination mechanisms are necessary at each router. Although SDN can bring the benefits of the network centralization through the centralized controller, this is however contrary to the distributed nature of wireless mesh networks. First, the simplicity of the centralized controller can come at a cost of network scalability, which could deteriorate the network performance. Second, the centralized controller presents a single point of failure, which could affect availability of the network. Conversely, distributed controllers aim at eliminating the single point of failure and scale up the network. Despite the advantages of distributed SDN control to improve the scalability and the robustness of networks, several key challenges should be addressed to obtain a consistent and a global optimal view of the entire network.

Accordingly, it is difficult to decide whether a single controller will be able to manage distributed islands of wireless devices or multiple controllers should coordinate their activities to perform cooperation between wireless mesh routers and enable zone specific controllers. To derive the advantages of both approaches, a new hybrid control plan can be developed that benefits from the simplicity of the centralized management and the scalability and resilience of the distributed model coordination.

Challenge 4: Lack of Effective Traffic and Resource Management. Wireless routers and gateways in the STLS scenario depicted in Figure 1 should forward the incoming traffic either between each other in case of mesh routers or to the Internet when the traffic reaches the gateways. Nevertheless, both the gateways and the routers can become a potential network bottleneck due to their high traffic overload. In particular, the concentration of traffic on the gateways, which act as central points of attachment to the Internet, may increase the network load on certain paths, which leads

to saturation of the links as well as generating buffers overflows. Moreover, traffic overload in the routers affects the performance of the overall mesh backbone if routing protocols are unable to provide network offload. Although, increasing the number of wireless routers can help to distribute load among them, mitigating the problem by increasing the number of routers does not necessarily increase the capacity of the network. Additionally, traffic forwarding in the STLS scenario requires selecting the best paths from smart cars towards their nearest routers. However, the best path selection in such a scenario seems to be NP-hard problem [16] so that heuristic algorithms should take into account both wireless channels and routing algorithm.

3 SDN-ENABLED WIRELESS MESH NETWORKS FOR CPS

We present our SDN-enabled wireless mesh network solution for CPS that meets the reliability and performance requirements outlined in Section 2. Figure 2 depicts our blended SDN-WMN architecture. At the core of this design is a logically centralized controller, i.e., the control plane, which communicates with the underlying mesh routers using the OpenFlow protocol.

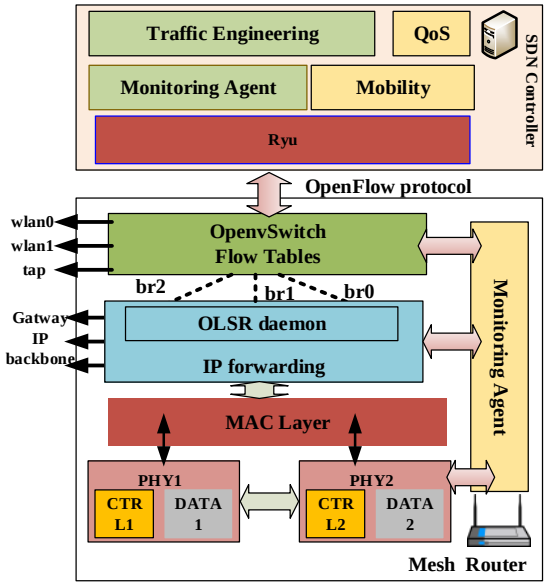

Figure 2: Architecture of the Joint SDN-WMN solution

The SDN controller comprises the following modules:

- **Topology discovery module:** which uses the Link Layer Discovery Protocol (LLDP) to perform automatic discovery of joining and leaving wireless mesh routers.
- **Routing module:** which implements the shortest path algorithm to build the optimal routing strategy to route packets across the mesh routers. It builds a network graph of connected routers, removes a node from the graph when a router leaves the network, and activates/deactivates links to force packets to follow an optimal path.
- **Monitoring module:** which enables fine-grained control and monitoring of the OpenFlow traffic by querying a mesh router to gather individual statistics. It also supervises the path reservation and modification at run-time.
- **Traffic engineering module:** which supports load balancing to offload mesh routing devices in case of traffic congestion. It also performs traffic redirection based on the optimized routing strategy used in the routing module.

In the data plane, each mesh router (shown in the bottom box) forwards OpenFlow messages using the OpenVSwitch soft router. OpenVSwitch implements a software pipeline based on flow tables. These flow tables are composed of simple rules to process packets, forward them to another table, and finally send them to an output queue or port. Furthermore, the data plane includes an IP-based forwarding daemon running the OLSR routing protocol. OpenVSwitch bridges OpenFlow and OLSR using virtual network interfaces, shown as *br0, br1, and br2*, to exploit the capacity of IP networks to route packets via the shortest path.

3.1 Addressing Challenges to Realize Real-time and Reliable Smart City Applications

We now show how our architecture resolves the challenges outlined in Section 2.

Resolving Challenge 1: Wireless Network Virtualization by Splitting Routers into Two Virtual Ones. To support wireless virtualization, we slice each physical router into two virtual routers; each having its own virtual hardware resources and virtual radio interface, i.e. *PHY1* and *PHY2* shown in Figure 2. Each physical access point (AP) in turn can be split into two non-overlapping virtual APs, i.e., ESSID 1 and ESSID 2 thereby enabling four virtual APs on a node. Each virtual ESSID has its virtual wireless channel so that mobile clients can switch between them seamlessly and can communicate using the virtualized channels. Moreover, to separate the control traffic, i.e., signaling, from the data traffic, each SSID forwards the traffic independent of the other. The benefit of splitting an AP into two virtual ones is twofold. First, it provides an efficient downlink bandwidth sharing between multiple smart clients due primarily to the efficient airtime fairness scheduling with the help of channel sharing. Second, it resolves the challenges of uplink channel access when multiple clients are simultaneously transmitting while also enabling high data rates and low latency for those smart clients.

Allowing two virtualized access points inside the same wireless router also allows each virtual AP to deliver its traffic indication map, i.e., broadcast Beacon messages, and enables the synchronization of its clients with the wireless network. These beacon frames are management frames used in mesh routers to check liveness of all the clients attached to a wireless router. Such an approach allows the use of existing link layer protocols while allowing MAC settings to be changed simultaneously.

Resolving Challenge 2: Efficient Routing by Blending OpenFlow and OLSR Routing. To support efficient and scalable routing in SDN-enabled wireless routers, we divide the routing functionality into two layers as shown in Figure 2. The upper layer supports SDN routing using the OpenFlow protocol for data forwarding. The bottom layer uses IP-based forwarding with the OLSR routing protocol. The former is responsible for communicating OpenFlow policies with the SDN controller. The latter is responsible for handling IP

routing among OLSR interfaces inside the mesh routers. To allow the controllers to reach all the geographically distributed routers, we use an in-band control approach in a way that provides long distance wireless connectivity among the wireless mesh backhaul. There are two advantages of cohabitating IP-based routing and SDN routing. First, the controller can implement its own routing algorithms for best path selection, and configure the mesh routers by adding/removing/updating OpenFlow rules. It can also retrieve the current network states from the nearest mesh router. Second, packets can be routed according to OLSR routing tables under the instruction of the controller through OpenFlow.

OLSR reports every change in the topology graph, such as adding/removing new mesh router and/or wireless link. Each wireless router keeps a list of its neighbors – the so called multi-points relays (MPR) selector list, periodically builds a new refreshed routing table, and selects the new shortest path to all destinations. Thereafter, the controller retrieves the topology information from its nearby mesh routers.

Resolving Challenge 3: Adapting between Centralized and Distributed SDN Control. We propose a hybrid network controller that combines centralized and distributed SDN controllers to derive the benefits of both approaches. To that end, we leverage our prior work on the *'InitSDN'* framework [20] as illustrated in Figure 3. InitSDN is a meta-controller layer based on the concept of boot loading used in operating systems. First, we start with a single centralized controller that is deployed during the initialization phase to control and manage the entire network. Then, in case of controller failure or overload, additional controllers are introduced at runtime as required to balance the network load and scaled elastically.

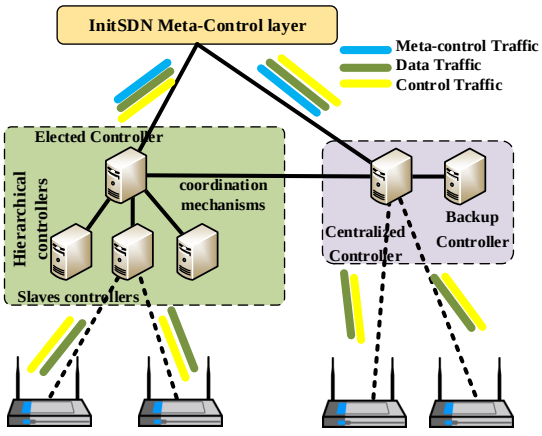

Figure 3: InitSDN Hybrid Control Plane for the STLS

Coordination mechanisms are deployed to ensure consistency among the distributed controllers. In particular, these mechanisms include an election process that allows electing a SDN controller as a master. Such a hybrid control strategy allows allocating and assigning the right traffic to the right number of controllers, while making the network more flexible, reliable, predictable, and fault-tolerant. Details on the load balancing solution are discussed next.

InitSDN divides the wireless network into two slices: a data slice to control the traffic exchanged between users applications and a control slice for managing the controllers. It allows selecting the optimum initial topology of the control slice, i.e., the number of controllers, based on the current network conditions, i.e., network overhead, failure, etc.

Resolving Challenge 4: System Monitoring and Load Balancing. Detecting faults and balancing load requires effective monitoring of system resources. To that end, the controller implements a monitoring agent as shown in Figure 2. It uses OpenFlow messages to supervise the path reservation, modification and installation. Based on the collected statistics the controller determines if the system is overloaded. To address network overload issues, we introduce a traffic-engineering algorithm at the controller to perform load balancing. Figure 4 depicts the principle behind our load balancing approach using an example: the SDN controller that connects the edge routers of the mesh tries to establish a routing path between mesh router 1 and mesh router 4 across the link ⓐ connecting router 1 and router 4. Links *a* through *f* establish the communication paths across the mesh routers in the STLS. As soon as a link becomes a bottleneck, e.g., because of congestion, connection loss, interference, etc, the load-balancing algorithm is activated on the controller side. Thereafter, the controller can easily decide the next best available path to switch the data as illustrated by the curved arrow in Figure 4.

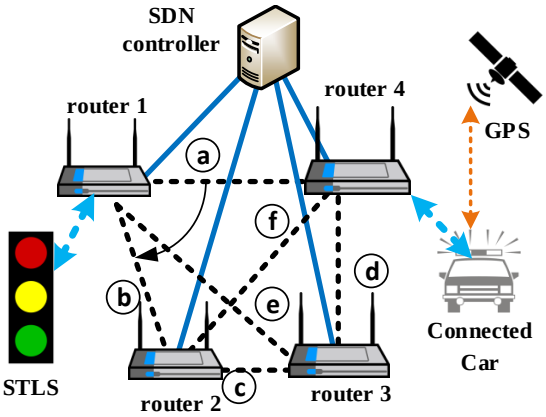

Figure 4: Load Balancing in the STLS network

Algorithm 1 shows the load balancing algorithm to select the optimal path. It calculates the new rules, i.e., the MAC and IP addresses, for the new path towards the new mesh routers, i.e., ⓑ, ⓕ and ⓒ, ⓓ shown in Figure 4. Once the new path is established end-to-end by sending FlowMod messages, the controller floods all ports towards the selected virtual routers, opens the client's connection to enable packets to reach their destination, and simultaneously continues discovering and monitoring the network topology. The controller calculates the new optimal path using the graph topology, which includes all available routers as well as the links connecting them (Algorithm 2). Then, it installs new OpenFlow rules to program the flow entries inside the software pipeline in each router.

Algorithm 1: Load Balancing Algorithm

```
1  rules ⟵ DefaultRules();
2  trafficSchudeling();
3  while Listening to LLDP packets do
4      isBestPATH = best_path(rules);
5      if ≠ isBestPATH then
6          rules ⟵ calculateNewRules();
7          FlowMod_router(); path ⟵ bestPath(rules);
8      else
9          installOFRules(path);
10     end
11     hostsReachable();
12     monitoringPath();
13 end
```

Algorithm 2: Function bestPath(rules)

```
   Data: rules, PATH
   Result: Function to find optimal path
1  bestPath(rules);
2  if (∃ PATH in (rules) then
3      PATH ⟵ find(rules) ;
4      return PATH
5  else
6      rules ⟵ calculateNewRules(); FlowMod_router();
7      return rules
8  end
9  best_path(rules);
```

Algorithm 1 uses the function in Algorithm 2 to find the optimal end-to-end path to the destination. This recursive function makes it possible to look for the best path for each iteration. On the controller side we implemented all the mechanisms and functions required for routing data toward the selected path. In particular, it implements the routing function to forward data towards the SDN routers. First, the data path is extracted from in the incoming packets, then data and protocols are extracted to initialize the SDN controller. Thereafter, OpenFlow rules are added to all routers in the destination path.

Table 1 depicts the flow entries that the controller can program before traffic congestion and after triggering the load balancer algorithm. At startup time, the controller has already installed the data path between router 1 with ID *dpID1* and router 4 with ID *dpID4*. When router 1 receives incoming packets in its virtual port, i.e., *ingress-Port: virtual port 1*, the headers of those packets are inspected to check whether they match the OpenFlow rules in the flow entries. The action sets are provided through the physical port of router 1, i.e., *output: To port router 4* and the destination of packets from router 1 is the next nearest hop, i.e., the router 4. Thus, packets from router 1 should encapsulate in their headers the IP and MAC destination addresses of router 4. Hence, the flow entries are injected by the controller to allow forwarding the data to router 4 using both its IP, i.e., *SetDestIP: IP router 4,* and its MAC, i.e., *SetDestMAC: MAC router 4,* destination addresses.

OF	Before	After
OpenFlow rules	router1: dpID1 router4: dpID4 ingress-Port: virtual port 1	router1: dpID1 router2: dpID2 router4: dpID4 ingressPort: virtual port 1 ingressPort: virtual port 2
OpenFlow entries	SetDestIP: IP router 4 SetDestMAC: MAC router 4 output: To port router 4	setDestIP: IP router 2 SetDestMAC: MAC router2 output: To port router 2 setDestIP: IP router 4 SetDestMAC: MAC router4 output: Port router 4

Table 1: Flow entries the controller install in the routers

Upon the failure of radio link *a*, the controller installs new OpenFlow rules to redirect the flow from router *1* to router *4* through router *2*. Since the new available forwarding path should pass through router 2, the controller should program routers 2 and 4 with the new flow entries as described in the "After" column of Table 1.

4 EVALUATING THE BLENDED SDN-OLSR ARCHITECTURE

This section evaluates our solution that blends SDN and OLSR by measuring the performance, evaluating the claims on the architecture's properties, and the overhead, if any, that is imposed by the overall design and its architectural elements.

4.1 Testbed Settings and Experimental Setup

We have prototyped our solution using OpenWRT in Raspberry Pi 2s. At the controller side, we enhanced the Ryu SDN controller to support our approach. Furthermore, to support network virtualization, the Ryu controller can cooperate with *OpenStack* using the *Quantum Ryu plugin* to support Mobile Cloud communication. The extension can easily be integrated into *OpenStack++* [10] for enabling mobile cloudlets (which are edge-based micro data centers).

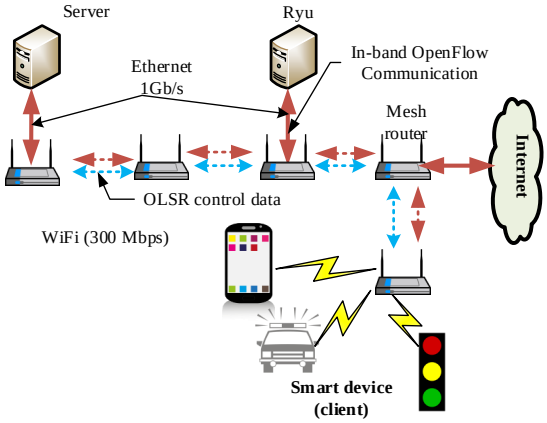

Figure 5: Experimental setup for the wireless mesh network

To evaluate the proposed solution, we consider an emulation-based experimental setup depicted in Figure 5. We consider the mesh client as an autonomous car which communicates using its radio interfaces with the cloudlet server across multi-hop routers.

This smart car can also communicate with the gateway, which acts as the access point to the Internet. To reach its destination, i.e., the cloudlet server or the Internet, the Ryu controller should be able to install OpenFlow rules to its neighbor router.

4.2 Evaluating the Predictability of End-to-end Latency

Rationale and Methodology. Real-timeliness can be gauged based on how predictable are the response times for smart city applications. To that end, we consider the latency as the time duration from a packet to be sent from the source mesh client until it is received by the destination server which executes the services. We conducted this experiment multiple times and recorded the average latency. The measurement of one-way latency is not straightforward because packets experience different network delays including processing delay, queuing delay, transmission and propagation delays. Thus, we have calculated the Round Trip Time (RTT), which then estimates the one-way latency by assuming half of the RTT. Additionally, we calculated the delay required for a packet to be sent by the controller until it is received by its nearest router.

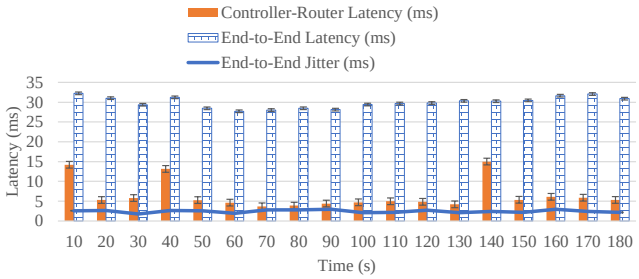

Figure 6: Evaluating the controller-router Latency and the end-to-end latency

Analysis. Figure 6 depicts the latency between the SDN controller and its corresponding router as well as the end-to-end latency between the client and the gateway. At the startup phase, the controller-router delay is close to 10 milliseconds and decreases close to 3 milliseconds after the controller has installed new OpenFlow rules into the router. At this time only the OpenFlow *keepalive* messages are exchanged to check whether an idle control connection occurs to indicate a loss of controller-switch connectivity. At time 40 seconds, a new mobile client joins the network, but its forwarding rules are still unknown for both the controller and the switch. Thus, they start exchanging messages to setup new forwarding rules for packets belonging to that client. The same behavior occurs at time 140 seconds. In all those cases, the controller-router latency remains bounded to 15ms during the setup phases and close to 5 ms otherwise. Therefore, the controller-router latency does not present a network bottleneck.

The end-to-end latency and the bounded jitter between remote hosts is also shown in Figure 6. In the regular case where no setup traffic is injected into the network, the delay is close to 30 ms. It becomes close to 38 ms each time new OpenFlow rules are being negotiated between the controller and the switches. In both cases, the latency remains bounded to 35ms. Similarly, the experiment illustrated a lower jitter, i.e. around 2.5ms, which is necessary to support predictable latencies in real-time applications.

4.3 Evaluating Throughput Performance

Rationale and Methodology. Scalability and throughput of applications is another key requirement. To evaluate the throughput performance and robustness of our proposed architecture, we consider UDP traffic between end hosts (e.g., video traffic in STLS scenario) and the packet size is set to 1,500 bytes. We also consider that each wireless node exchanges data at a transmission rate of 1 Mbps. In such a full mesh topology, we consider all routers connected to each other and the measurements of the data traffic is taken by the average of different packets' forwarding sections. To evaluate the impact of using OLSR forwarding and OpenFlow, the routers are placed in different locations and traffic monitoring is performed at the controller side.

We assume that the controller has already pushed down and installed the flow rules in the OpenFlow tables of the underlying mesh routers. Hence, the incoming packets in a given ingress port of a router are directly forwarded to its physical output port to enable the packets to reach the next wireless hop.

Analysis. Figure 7 shows the throughput measured with the Iperf measurement tool on the client side. We repeated the experiments multiple times to ensure the consistency of the results. In each run, there are three different traffic types: (i) the OpenFlow control traffic, (ii) the OLSR forwarding traffic; and (iii) the UDP/IP data traffic exchanged between end users. The average throughput is close to 950 KB/s while the maximum expected throughput is bounded by 998 KB/s at time 30 seconds. There are many reasons that may lead to the decrease of the throughput: data plane to control plane encapsulation, thread priorities, CPU interrupts, amount of OLSR traffic and OpenFlow control data exchanged across the network. The average throughput drops closer to 950 KB/s, which we consider as a good value for such an unreliable traffic. The evaluation data confirms our claims on ensuring the fairness of the global optimization of our approach.

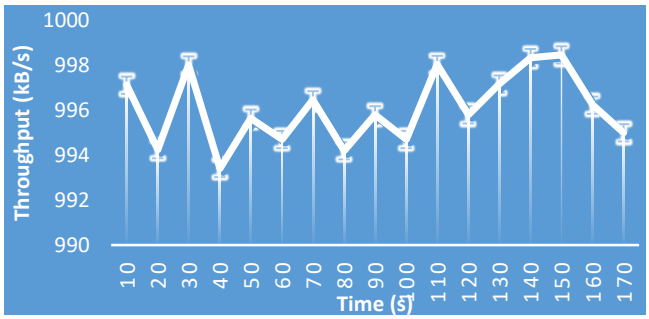

Figure 7: UDP Throughput

4.4 Evaluating the System Reliability

Rationale and Method. Reliability is another key requirement for CPS. To that end, to provide an in-depth inspection of the average relative error in the throughput described in Section 4.3, we estimated the per-flow packet loss by polling the flow statistics in

the edge routers assuming a relationship between the link packet loss and the throughput. The packet loss can be obtained by calculating the difference between the average throughput in the edge router on the client side and the edge router on the server side. Throughput results using UDP traffic are shown in Figure 7.

Analysis. The packet loss measurements depicted in Figure 8 show that the average error is close to 1%. The average packet loss is calculated by subtracting the difference of packet counters in edge routers between the client and the server. These measurements give a sufficient estimate about service degradation. The current version of the OpenFlow specification does not include any QoS service differentiation to enable per-class packet classification, scheduling and forwarding. Thus, traffic prioritization is not applied to protect packets against any computing flows. A close inspection of these results shows that our solution is successfully able to support SDN-based communication in the smart cities scenario.

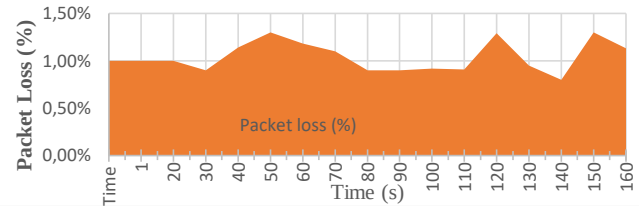

Figure 8: Packet Loss

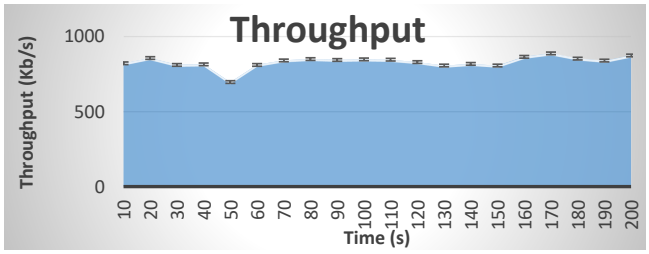

Figure 9: TCP Throughput

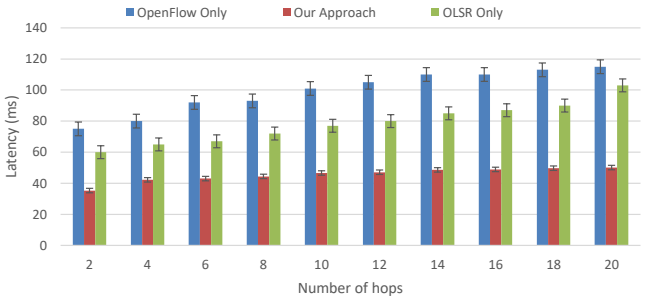

Figure 10: Latency when increasing the number of wireless hops

4.5 Evaluating the Load Balancing Strategy

Rationale and Methodology. Overload and failure management is critical for smart city applications. OpenFlow allows setting up of flow paths by inserting flow entries at the controller. Each connected node to the controller is considered as a mesh router so that any incoming flow that matches the OpenFlow flow rules is redirected by the controller based on the OpenFlow actions. Redirecting flows between routers is essential to enable traffic engineering in mesh networks. It allows offloading certain paths to allow fairness among different flows. Recall that all routers are OpenFlow-enabled and each has an OLSR instance to allow IP-based data forwarding and routing table updates. To evaluate the performance of the load balancing approach, after 40 seconds we inject a competing flow into router 4 to simulate network congestion and introduce a performance degradation in this node.

Analysis. Figure 9 shows the throughout observed in router 4. Due to buffer overflow, router 4 starts dropping packets so that the throughput decreased from 800 kB/s to 697 kB/s and a significant packet losses is observed. At time 50 seconds, the load balancing algorithm at the controller is activated to redirect the traffic from radio link *a* to radio links *b* and *f*. The topology discovery module at the controller discovered the disconnection of the wireless radio between routers 1 and 2, checks the new available path based on the graph its has and selects router 2 as new shortest path to destination.

The new path is extracted from the routing table updated regularly by the OLSR protocol. Then, the controller needs to remove the old OpenFlow rules in router 1, i.e., those used for sending the traffic across link *a*, pushing down and installs new forwarding rules as described in column 3 of Table 1. The IP and MAC addresses of router 2 are added in the new rules. The bow in Figure 4 shows the new path selected by the controller by installing new OpenFlow rules in node 1.

A close inspection of Figure 9 shows that the controller is able to make traffic adjustment using the load balancing algorithm. The traffic is balanced among the new wireless links after establishing the new data path. Furthermore, we find the delay required by the controller for deciding the new available path and forwarding data is close to 6 milliseconds. Therefore, the controller-router communication does not degrade the performance of the network during the traffic engineering process. The redirection delay is composed of the delay required to drop the old rules from routers and pushing down the new rules in the flow tables of each router. The results show that our approach succeeds in redirecting packets to the new selected path when multiple wireless hops are available in the network.

4.6 Evaluating the Impact on Latency with Increasing Hops and Link Failures

Rationale and Methodology. To evaluate the latency in a distributed wireless network, we increased the number of wireless routers for the scenario in Figure 1. We also compared the latency of our solution with both the OLSR-only and the OpenFlow-only latencies. We repeated this experiment multiple times and recorded the average. Additionally, we measured the controller-switch latency for reconnecting them during failures. In particular this latency is evaluated against the number of broken links.

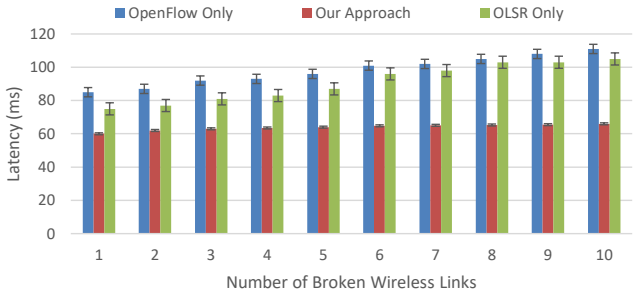

Figure 11: Latency after Links failure and reconnection

Analysis. The controller attempts to connect to all switches using discovery messages to find the shortest path to all the underlying SDN routers. Figure 10 illustrates the controller connection latency against the number of hops towards the remote SDN routers. Our approach shows better latency compared to both the OpenFlow-only and the OLSR-only approaches. The latency incurred by our approach is approximately 50% less compared to the other approaches, which is around 40ms, while it is about 60ms and 75ms for OSLR-only and OpenFlow-only approaches, respectively. Additionally, we observed the same behavior when we measured the latency after the controller re-connection against the number of broken wireless links.

Figure 11 illustrates that the latency observed in our approach is close to 60ms, while both the OSLR-only and OpenFlow-only approaches incur 75ms and 85 ms, respectively. Therefore, our approach outperforms the two other approaches as the number of hops increase between switches. The reason for this result is that the SDN routers have better connection to the controller than in the in the OSLR-only and OpenFlow-only approaches

4.7 Evaluating the Router overhead

Rationale and Methodology. We consider the performance overhead in each mesh router when using our hybrid routing approach and we compare it with OLSR IP routing along with the OpenFlow forwarding. Each gateway is connected to the Internet and announces the default route, i.e. 0.0.0.0/0, through OLSR, which inserts this default route in the routing table of each router. Moreover, each router can carry OpenFlow messages using OpenVSwitch, which is bridged to the IP forwarding using the *br* network interfaces as shown in Figure 2. This scenario makes it possible to perform flow-based forwarding operations using our hybrid routing approach while still routing those flows between different mesh routers using OLSR to better exploit the capacity of IP networks to route packets to the shortest path between the source and destination.

Analysis. Figure 12 depicts the total traffic rates generated by our approach and OpenFlow. After the initiation phase, OpenFlow creates control traffic at time 38 seconds when new rules are installed by the SDN controller into its corresponding router. As expected, the OpenFlow traffic increases as the installation of new rules is performed, while the OLSR traffic remains the same. The additional control traffic introduced by OpenFlow is about 3580 Kbits/s (447.5 KB/s) and the total traffic is 6 times higher compared to a case when OLSR is used as a routing protocol. At time 42 seconds, the OpenFlow control traffic decreases as all the new OpenFlow rules are installed in the router and the controller has no new flow entries to inject into it. Compared to OLSR and OpenFlow, our approach does not add any extra control flow at the new rules installation phase. Thus, our approach do not contribute to the router overhead.

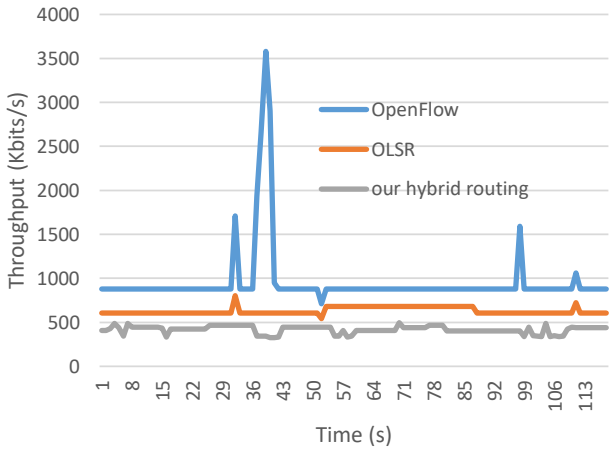

Figure 12: Evalution of the network overhead

4.8 Evaluating the Controller Overhead

Rationale and Methodology. It is also important to gauge the overhead of the additional infrastructure elements. Hence, to evaluate the controller overhead, we measured the amount of control data exchanged between the controller and the underlying routers. We also compared this traffic to data traffic exchanges when the controller installs new flow entries in the routers' flow tables. These experiments are conducted five times and the average values are taken for the evaluation. The captured controller traffic includes three different matching actions: OpenFlow packets, Ethernet packets (i.e., ARP) and data packets (i.e., TCP packets). The controller traffic through routers is captured using Wireshark and analysis are performed with Tcpdump packet analyzer.

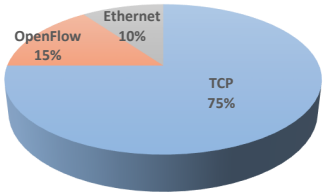

Figure 13: Controller overhead

Analysis. Figure 13 shows the control traffic overhead along with the data traffic through a router. The control OpenFlow traffic is close to 15% of the overall traffic exchanged in the wireless network, the data traffic close to 75%, and the Ethernet traffic is close to 10%. The initialization phase requires exchanging Ethernet traffic to perform host reachability between remote hosts.

Indeed, the first hosts send ARP requests across the networks, which generate broadcast of *PACKET_OUT* messages to all nodes in the network. The routers will examine these requests to know the source port mapping. Then, ARP responses come back with all Ethernet addresses known to the controller, i.e., the source MAC address will be associated with the port. The controller can now flood *Flow_Mod* messages on all ports of the underlying router. Due to broadcasting OpenFlow messages the control overhead is almost two times the Ethernet traffic, which is minimal when compared to the TCP data traffic. Therefore, the control traffic does not contribute significant overhead.

5 RELATED WORK

Realizing and sustaining smart city-scale networks that are reliable and support predictable response times requires new approaches which can support timely bandwidth reservation, load balancing, data security, etc. Many efforts that use SDN in this context exist. Wang et al. [32] proposed a SDN-based Internet of Vehicles (IoV) architecture that optimizes OpenFlow rules by introducing compact flow rules. Sahoo et al. [23] introduced a SDN-based traffic engineering approach that solves the connectivity problems of vehicles in a smart city. Likewise, Bozkaya et al. [4] have demonstrated the feasibility of combining SDN with wireless access in vehicular environments. They proposed a flow and power management model implemented in a SDN controller to enhance the connectivity of the Road-Side Units (RSU). Similarly, Xu et al. [33] proposed a cloud-based architecture to improve the capacity and performance of vehicular networks. Truong et al. [29] combined SDN-based VANETs with fog Computing to offer delay-sensitive, location-aware services, while optimizing resource utilization. Greff et al. [7] combined online flow admission control and pathfinding algorithms to address real-time flow allocation problem in SDN-enabled mesh networks. Similarly, the authors [8] accomplished path redundancy in mesh network to handle fault tolerance in SDN-aware Real-Time mesh networks.

Venkatramana et al. [30] proposed a centralized SDN controller deployed on a cloud to perform geographical routing protocol in Vehicular ad hoc network (VANET). The controller maintains a global topology routing table to perform an optimal routing path within the considered vicinity of mobile cars. Nonetheless, deploying the controller in remote cloud is more sensitive to the latency since it needs powerful cloud resources such as communication, computation, network control and storage. Similarly, Wang et al. [32] introduced a real-time query services for SDN-based Internet of Vehicles (IoV). A cloud-hosted SDN controller uses a multicast communication pattern to send and retrieve information from mobile vehicles. Likewise, Bi et al. [3] proposed a SDN architecture to support smart city services. A network control layer includes a SDN controller is deployed in cloud data center for controlling big data transfer centrally.

A fault-tolerant SDN routing mechanism was introduced in [21] to construct elastic-aware routing tree and perform routers selections. In [24], the authors introduced a structured scheme to handle user's demands over SDN-aware WMNs based on multichannel multiradio WMNs. Likewise, the authors in [14] proposed an approach to offload 3GPP Radio Access Network (RAN) traffic through SDN-enabled WMNs to facilitate fast devices configuration and services deployment. Authors in [13] proposed different design SDN approaches to accommodate dynamic conditions such as mobility and unreliable wireless connectivity. Venkatramana et al [30] introduced a SDN-aware WMNs backbone to support intelligent transportation system that is envisaged to play a significant role in the futuristic smart cities for safety and traffic management.

In comparing our work to all the prior efforts, the SDN controller in our work is hosted at the network edge in close proximity to wireless devices to improve network reliability and latency, and overcome the issues stemming from geographically distributed locations in cloud computing. Additionally, in the aforementioned approaches, the controller is used to carry both signaling messages and data packets. In contrast to these efforts, our approach uses the SDN controller only for the control traffic and whereas we use IP-based data forwarding to transmit data in hop-by-hop fashion. We show that this approach provides superior performance.

6 CONCLUSIONS

Cyber physical system wireless communication networks, such as those in smart cities, must be scalable, reliable and predictable to support real-time applications. To address these needs, we introduced a novel architecture based on a symbiotic relationship between wireless mesh networks (WMNs) and software defined networking (SDN). Our experimental results validate our claims. Although our empirical validations are emulation-based, we are prototyping the capabilities on Raspberry Pi-2 running the OpenWRT Linux OS. Although our research is validated in the context of smart city CPS, the work is broadly applicable to other domains such as Industry 4.0 or smart grids. The source code for this research is available at https://github.com/hakiri/sdn-ns-3.

Several new directions for additional research exist. First, we need more experiments to evaluate the distributed controllers. Second, recent trends in fog/edge computing have focused primarily on resource management of fog compute resources (including our recent work [5, 26, 27]) but new research is needed to also include wireless network resource management for CPS applications. Second, although SDN allows the programmability of the data plane, current wireless devises employ diverse modulation protocols to comply with a specific radio interface, which limits their flexibility and versatility to respond to the increasing demands on bandwidth and frequency spectrum resources. We believe that the coexistence of SDN and the Software Defined Radio (SDR) can unify the network resource management and the radio resource management, which may require a cross layer design. Incorporating time sensitive networking is an additional dimension of future work. Finally, and more importantly, our solutions need to be tested in real smart city deployments.

ACKNOWLEDGMENT

This work was supported in part by the Fullbright Visiting Scholars Program and NSF CNS US Ignite 1531079. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF or the Fulbright program.

REFERENCES

[1] 2016. IEEE Approved Draft Standard for Wireless Access in Vehicular Environments (WAVE) - Networking Services. *IEEE P1609.3v3/D6, November 2015* (Jan. 2016), 1–162.

[2] K. M. Alam, M. Saini, and A. E. Saddik. 2015. Toward Social Internet of Vehicles: Concept, Architecture, and Applications. *IEEE Access* 3 (2015), 343–357.

[3] Y. Bi, C. Lin, H. Zhou, P. Yang, X. Shen, and H. Zhao. 2017. Time-Constrained Big Data Transfer for SDN-Enabled Smart City. *IEEE Communications Magazine* 55, 12 (2017), 44–50.

[4] E. Bozkaya and B. Canberk. 2015. QoE-Based Flow Management in Software Defined Vehicular Networks. In *2015 IEEE Globecom Workshops (GC Wkshps)*. 1–6.

[5] Faruk Caglar, Shashank Shekhar, Aniruddha Gokhale, and Xenofon Koutsoukos. 2016. An Intelligent, Performance Interference-aware Resource Management Scheme for IoT Cloud Backends. In *1st IEEE International Conference on Internet-of-Things: Design and Implementation*. IEEE, Berlin, Germany, 95–105.

[6] T. Clausen and P. Jacquet. 2003. Optimized Link State Routing Protocol (OLSR). RFC 3626 (Experimental). http://www.ietf.org/rfc/rfc3626.txt

[7] F. Greff, Y. Song, L. Ciarletta, and A. Samama. 2017. A dynamic flow allocation method for the design of a software-defined real-time mesh network. In *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*. 1–11. https://doi.org/10.1109/WFCS.2017.7991949

[8] Florian Grefff, Ye-Qiong Song, Laurent Ciarletta, and Arnaud Samama. 2017. Combining Source and Destination-tag Routing to Handle Fault Tolerance in Software-defined Real-time Mesh Networks. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems (RTNS '17)*. ACM, New York, NY, USA, 257–266. https://doi.org/10.1145/3139258.3139264

[9] A. Gyrard and M. Serrano. 2016. Connected Smart Cities: Interoperability with SEG 3.0 for the Internet of Things. In *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. 796–802.

[10] Kiryong Ha and Mahadev Satyanarayanan. [n. d.]. OpenStack++ for Cloudlet Deployment. , 24 pages.

[11] Huawei Huang, Peng Li, Song Guo, and Weihua Zhuang. 2015. Software-defined wireless mesh networks: architecture and traffic orchestration. *Network, IEEE* 29, 4 (July 2015), 24–30.

[12] S. N. Khan, A. Kliks, Tao Chen, M. Mustonen, R. Riggio, and L. Goratti. 2017. Virtualization of spectrum resources for 5G networks. In *2017 European Conference on Networks and Communications (EuCNC)*. 1–5.

[13] Ian Ku, You Lu, and Mario Gerla. 2014. Software-Defined Mobile Cloud: Architecture, services and use cases. In *International Wireless Communications and Mobile Computing Conference, IWCMC 2014, Nicosia, Cyprus, August 4-8, 2014*. 1–6.

[14] M. Labraoui, M. M. Boc, and A. Fladenmuller. 2017. Opportunistic SDN-controlled wireless mesh network for mobile traffic offloading. In *2017 International Conference on Selected Topics in Mobile and Wireless Networking (MoWNeT)*. 1–7.

[15] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 38, 2 (2008), 69–74.

[16] B. Mumey, Jian Tang, I.R. Judson, and D. Stevens. 2012. On Routing and Channel Selection in Cognitive Radio Mesh Networks. *Vehicular Technology, IEEE Transactions on* 61, 9 (Nov. 2012), 4118–4128.

[17] V. Nascimento, M. Moraes, R. Gomes, B. Pinheiro, A. Abelém, V. C. M. Borges, K. V. Cardoso, and E. Cerqueira. 2014. Filling the gap between Software Defined Networking and Wireless Mesh Networks. In *10th International Conference on Network and Service Management (CNSM) and Workshop*. 451–454.

[18] United Nations. [n. d.]. World Population Prospects: The 2015 Revision, Methodology of the United Nations Population Estimates and Projections.

[19] B.A.A. Nunes, M. Mendonca, Xuan-Nam Nguyen, K. Obraczka, and T. Turletti. 2014. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. *Communications Surveys Tutorials, IEEE* 16, 3 (2014), 1617–1634.

[20] Prithviraj Patil, Aniruddha Gokhale, and Akram Hakiri. 2015. Bootstrapping Software Defined Network for flexible and dynamic control plane management. In *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. 1–5.

[21] Yuhuai Peng, Xiaoxue Gong, Lei Guo, and Dezhi Kong. 2016. A survivability routing mechanism in SDN enabled wireless mesh networks: Design and evaluation. *China Communications* 13, 7 (July 2016), 32–38. https://doi.org/10.1109/CC.2016.7559073

[22] C. Perkins, E. Belding-Royer, and S. Das. 2003. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental). http://www.ietf.org/rfc/rfc3561.txt

[23] P. K. Sahoo and Y. Yunhasnawa. 2016. Ferrying vehicular data in cloud through software defined networking. In *2016 IEEE 12th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 1–8.

[24] D. Sajjadi, R. Ruby, M. Tanha, and J. Pan. 2018. Fine-Grained Traffic Engineering on SDN-Aware Wi-Fi Mesh Networks. *IEEE Transactions on Vehicular Technology* 67, 8 (Aug. 2018), 7593–7607. https://doi.org/10.1109/TVT.2018.2832010

[25] J Schulz-Zander, L Suresh, N Sarrar, A Feldmann, T Hühn, and R Merz. 2014. Programmatic Orchestration of WiFi Networks. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. USENIX Association.

[26] Shashank Shekhar, Hamzah Abdel Aziz, Aniruddha Gokhale, and Xenofon Koutsoukos. 2018. Online Performance Model Learning for Dynamic Resource Management in Cloud Computing Infrastructure. In *To Appear in IEEE International Conference on Cloud Computing (CLOUD)*. San Francisco, CA, USA, 8.

[27] Shashank Shekhar, Ajay Chhokra, Anirban Bhattacharjee, Guillaume Aupy, and Aniruddha Gokhale. 2017. INDICES: Exploiting Edge Resources for Performance-Aware Cloud-Hosted Services. In *IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*. Madrid, Spain, 75–80. https://doi.org/10.1109/ICFEC.2017.16

[28] K. Tan, H. Shen, J. Zhang, and Y. Zhang. 2012. Enable flexible spectrum access with spectrum virtualization. In *2012 IEEE International Symposium on Dynamic Spectrum Access Networks*. 47–58.

[29] N. B. Truong, G. M. Lee, and Y. Ghamri-Doudane. 2015. Software defined networking-based vehicular Adhoc Network with Fog Computing. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 1202–1207.

[30] D. K. N. Venkatramana, S. B. Srikantaiah, and J. Moodabidri. 2017. SCGRP: SDN-enabled connectivity-aware geographical routing protocol of VANETs for urban environment. *IET Networks* 6, 5 (2017), 102–111.

[31] X. Wang, C. Wang, J. Zhang, M. Zhou, and C. Jiang. 2016. Improved Rule Installation for Real-Time Query Service in Software-Defined Internet of Vehicles. *IEEE Transactions on Intelligent Transportation Systems* PP, 99 (2016), 1–11.

[32] X. Wang, C. Wang, J. Zhang, M. Zhou, and C. Jiang. 2017. Improved Rule Installation for Real-Time Query Service in Software-Defined Internet of Vehicles. *IEEE Transactions on Intelligent Transportation Systems* 18, 2 (2017), 225–235.

[33] K. Xu, R. Izard, F. Yang, K. C. Wang, and J. Martin. 2013. Cloud-Based Handoff as a Service for Heterogeneous Vehicular Networks with OpenFlow. In *2013 Second GENI Research and Educational Experiment Workshop*. 45–49.

[34] Kok-Kiong Yap, Masayoshi Kobayashi, Rob Sherwood, Te-Yuan Huang, Michael Chan, Nikhil Handigol, and Nick McKeown. 2010. OpenRoads: empowering research in mobile networks. *SIGCOMM Comput. Commun. Rev.* 40, 1 (2010), 125–126.

[35] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi. 2014. Internet of Things for Smart Cities. *IEEE Internet of Things Journal* 1, 1 (2014), 22–32.