# A Software Defined Wireless Networking for Efficient Communication in Smart Cities

Akram Hakiri *†, Aniruddha Gokhale ‡, and Prithviraj Patil§
* Univ de Carthage, ISSAT Mateur, Tunisia
† Univ de Toulouse, UPS, LAAS, F-31400 Toulouse, France.
‡ISIS, Dept of EECS, Vanderbilt University, Nashville, TN, USA.
§ The Mathworks, Stateflow Semantics Group, Natick, MA, USA.
Corresponding author: hakiri@laas.fr

***Abstract*—Smart cities represent a rich dynamic environment where multiple wireless mobile devices interconnect to each other to communicate and share their data. The growing demand and the diverse traffic patterns of these smart devices place an increasing strain on the wireless networks. Routing data among wireless routers becomes a challenging issue as traditional routing algorithms, which are based on the Ad-Hoc and LAN flavors, cannot fulfill the requirements of distributed communications in Smart Cities. The routing decisions are taken based on local knowledge of a wireless router about each of its neighbors to reflect a partial visibility of the network. Such a local visibility limits the ability of managing the huge number of wireless smart devices in Smart Cities. An attractive and more realistic alternative is the Software Defined Networking (SDN), which offers a centralized up-to-date view of the entire network by refactoring the wireless protocols into control and forwarding decisions. However, many challenges must first be overcome to realize this potential. Accordingly, this paper outlines important challenging issues for building efficient wireless communication in smart cities. In particular, we describe a novel network architecture that integrates SDN and the Wireless Mesh Networks (WMNs) to address these issues. Our approach proposes a novel way of performing network virtualization, routing and traffic engineering in SDN-based WMNs in smart cities thereby improving the performance and the flexibility of the network.**

***Index Terms*—Software Defined Network; Internet of Things; Wireless Mesh Networks; Smart Cities.**

## I. Introduction

Urban centers across the world continue to grow steadily as more than the half of the current world population are living in urban areas and the number is forecast to further increase by 2030 [1]. To help address various challenges due to the increased urbanization, innovative smart cities projects, such as VITAL [2] and Padova [3], have been adopted by local governments and private companies to provide new solutions, services and applications. To make such and future solutions, and ultimately to make cities livable and sustainable, smart cities need to be supported and integrated by an intelligent communication infrastructure that interconnects everything wirelessly. Wireless Mesh Networks (WMNs) have been considered as one of the most promising communication technologies to form the backbone of smart cities.

WMNs often consist of mesh clients, mesh routers and gateways. Mesh clients are mobile nodes such wireless camera, traffic signal controller, and other wireless devices while mesh routers forward data to and from gateways, which may connect to the Internet. The coverage area of those radio nodes working in a single network is called a mesh cloud. Such a mesh cloud allows monitoring traffic activity in cities and help to prevent traffic congestion. Traffic congestion is one of the most significant problems in cities, as it increases noise pollution and greenhouse emissions. Minimizing these negative effects of vehicular traffic jams with new perspectives on network traffic management will push the wireless mesh network performance and its capabilities to their extremes.

In this regard, although several wireless protocols such as AODV [4] (Ad hoc On Demand Distance Vector) and OLSR [5] (Optimized Link State Routing Protocol) have been investigated in the past decade, they were more influenced by the Ad-Hoc and LAN flavors. Indeed, the functionalities of the aforementioned routing approaches are limited and their extensions to support the newer and high volume network traffic patterns are very difficult. In particular, the routing decisions are taken in a distributed way based on the local knowledge of a mesh router about each of its neighbors to reflect a partial visibility of the network. This local visibility limits the ability of the WMN to perform traffic[1] engineering in distributed mesh clouds.

Additionally, the current routing protocols fail to provide sufficient, fast-failover to reroute failed nodes or broken links, and redistribute the orphaned clients among neighboring nodes. Furthermore, as most of the traffic is assumed flowing between the client nodes and the gateways, it is likely that the gateway becomes a network bottleneck in WMNs. Thus, selecting the best routes to the Internet in the mesh cloud for different traffic classes is an essential ingredient for QoS support. Besides, due to the link quality variation of the radio channels induced by the mobility and the topology changes, mesh cloud becomes more difficult to manage and configure. Specifically, managing and upgrading routers is a complex and error-prone task because nodes configuration should be performed manually and individually at each router.

To deploy new smart city services over WMNs, we need better manageability and flexibility in the network, which we surmise is possible using Software Defined Networking (SDN) [6]. SDN shows significant promise in meeting smart

[1]Unless otherwise stated, traffic refers to network traffic.

city needs by optimizing routing paths for information through the network [7]. Despite the promise of SDN, the challenges posed by smart city applications cannot be addressed by simply building SDN-based multipath routing algorithms. Rather, SDN should provide access to distributed wireless sensors that continuously monitor and control pollution, lighting, and vehicular traffic jams. Recent trends in programmable wireless networks reveal that SDN has been used to build relays between home gateways and the Internet [8], simplify the network management operations of the wireless access points [9] and enhance the traffic orchestration [10] in virtual access points.

Despite these advances, the aforementioned efforts used SDN only in a single wireless access point, which make their solutions unstable in a highly distributed wireless environment. First, SDN itself does not provide any abstracted programming interfaces for wireless communication. SDN was initially introduced for wired networks such as cloud computing and data centers to provide packet encapsulation and tunneling. 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 appropriate modules to support centralized interference management, node mobility, and topology discovery. Fourth, as smart cities interconnect distributed mesh clouds, it is not clear if a centralized SDN controller will be able to manage the entire network, or multiple distributed ones should coordinate their activities to perform router point cooperation.

To address those challenges, we present a novel approach that combines ideas from SDN with Wireless Mesh Network to define a powerful and easy to deploy smart city network. Our approach provides a novel way of performing routing, network monitoring, and traffic engineering by using modified OpenFlow protocol. It also supports both centralized and distributed SDN control planes based on a bootstrapping mechanism that decouples the orthogonal distributed systems concerns from the primary issues related to the controller.

The remainder of this paper is organized as follows: Section II compares related efforts to our proposed solution. Section III introduces the architecture of a futuristic urban scenario through a Smart Traffic Light System (STLS) and briefly discusses the role of SDN in enabling such a scenario. Section IV articulates some open issues related to the deployment of SDN-based wireless communication in such smart cities system. Section V 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 VI evaluates the framework along multiple dimensions including its performance, overhead and load balancing properties. Finally, Section VII provides concluding remarks describing potential future directions and open research problems in this realm.

## II. Related Work

A new research direction is needed to make it possible to deploy city-scale networks using SDN paradigm which includes supporting bandwidth reservation, load balancing, data security, etc. Wang et al. [7] proposed a SDN-based Internet of Vehicles (IoV) architecture that optimizes OpenFlow rules by introducing a compact flow rules. Sahoo et al. [11] introduced a SDN-based traffic engineering approach that solves the connectivity problems of vehicles in a smart city. Authors in Bozkaya et al. [12] have demonstrated the feasibility of combining SDN with wireless access in vehicular environments. They proposed a flow and power management model implemented into a SDN controller to enhance the connectivity of the Road-Side Units (RSU). Similarly, Xu et al. [13] proposed a cloud-based architecture to improve the capacity and the performance of vehicular network. Truong et al. [14] combined a SDN-based VANET and Fog Computing to offer delay-sensitive, location-awareness services, and optimize the resource utilization.

SDN is applied to improve network management between home gateway and the network edge in OpenRoad [8] by providing hierarchical logical layers that distinguishes flow layer for data forwarding. Likewise, a SDN-based lightweight virtual access point is proposed in the Odin Framework [9] to manage mobility and maintain client reachability by delegating the hand-off to a virtual AP. Furthermore, a CloudMAC [15] is introduced to offload the MAC layer processing to virtualized APs in the cloud. CloudMAC uses the OpenFlow protocol to forward MAC frames between virtualized APs in the cloud and the physical Wi-Fi stations. Authors in [16] proposed different design SDN approaches to accommodate dynamic conditions such as mobility and unreliable wireless connectivity.

Chung et al. [17] proposed a Hybrid Wireless Mesh Protocol (HWMP) defined in IEEE 802.11s to enable MAC layer based routing in wireless mesh SDN networks. A similar approach has been provided by Vagner et al. [18] that extended the OpenFlow semantics by adding new messages and rules to the OpenFlow protocol to include IEEE 802.11s MAC header. However, the link layer multi-hop routing has two major drawbacks: (i) limited number of nodes (i.e., maximum 32 nodes ) are allowed in a single network, which presents a scalability bottleneck in wireless networks; and (ii) the conflicting rules between 802.11s and OpenFlow introduce severe performance degradation. Besides, Huawei et al. [10] proposed a traffic orchestration architecture for WMNs based only on OpenFlow for both IP data forwarding and SDN control data signaling. Likewise, Donghaiet al. [19] presented the OpenCoding framework, which implements network coding functions at the data plane so that the SDN controller can access them for both hop counting and path selection.

Dely et al. [20] proposed an out-of-band approach by enabling a dual SSID in the same access point: one for data forwarding and the other for the signaling of control traffic. Likewise, authors in [21] proposed a hybrid OpenFlow communication using a single SSID. It benefits from the distributed IP routing algorithms like OLSR for the propagation of the control traffic and OpenFlow for data forwarding. A

centralized controller is used to forward data traffic, while the OLSR protocol is used to install OpenFlow rules along with IP forwarding strategy inside each SDN switch. In contrast to out-of-band signaling, the authors in [22] proposed in-band OpenFlow signaling where the centralized SDN controller configures all wireless nodes throughout the end-to-end path taken by the flow. However, such a configuration may become a network bottleneck since it may increase the network load.

All these approaches use a SDN controller to carry the signaling messages as well as the data packets across the wireless network. Unlike these efforts, our approach uses the SDN controller for the control traffic and IP-based data forwarding to transmit data in hop-by-hop fashion.

## III. Synergy Between SDN and Smart Cities

In this section, we describe a motivating urban scenario and show how SDN wireless communication can play an effective role in it. We first describe the network architecture of a Smart Traffic Light System (STLS) envisioned for smart city. Then, we discuss the value of SDN in enabling such a scenario.

### A. Motivating Example

The STLS depicted in Figure 12 collects data from diverse sensing devices, roadside equipment, and cameras to detect the presence of vehicles, cyclists, and pedestrians. Pedestrians may wear body-borne computers (wearable clothing) and their dogs may wear a dog collar (circle (1) in the figure). Motorcyclists can wear connected helmets and cyclists may ride 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 approaching vehicles from every direction of an intersection (circles (3)). It can also disseminate warnings 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.

The STLS can be considered as a wireless mesh network composed of distributed multi-hop wireless routers embedded in the light boxes that relay into partial or full mesh topologies. Those routers represent the SDN data plane that coordinate the communication activity (circles (4)) among routers, issues 'slow down' warning to vehicles at the risk of collision, create a sequence of green lights by adapting traffic lights to allow emergency vehicles to pass. Those vehicles are equipped with On-Broad Units (OBUs) to connect to various network interfaces, such as Global Positioning System (GPS), radio transceivers for Wireless Access Vehicle Environment (WAVE) [23], and Vehicular Ad-Hoc Network (VANET) [24] (circle (5)) to communicate with each other and connect to Road Side Units (RSUs) and mobile base stations. Each cluster of RSUs can be considered as a cluster of SDN-enabled wireless routers controlled by a SDN controller.

Besides, routers are connected to Internet gateways to provide diverse services such as route planning, traffic alert dissemination as well as mobile vehicular cloud services [25]. Finally, the access to the Internet (circle (6)) could be provided either by diverse access technologies such as xSDL, satellite

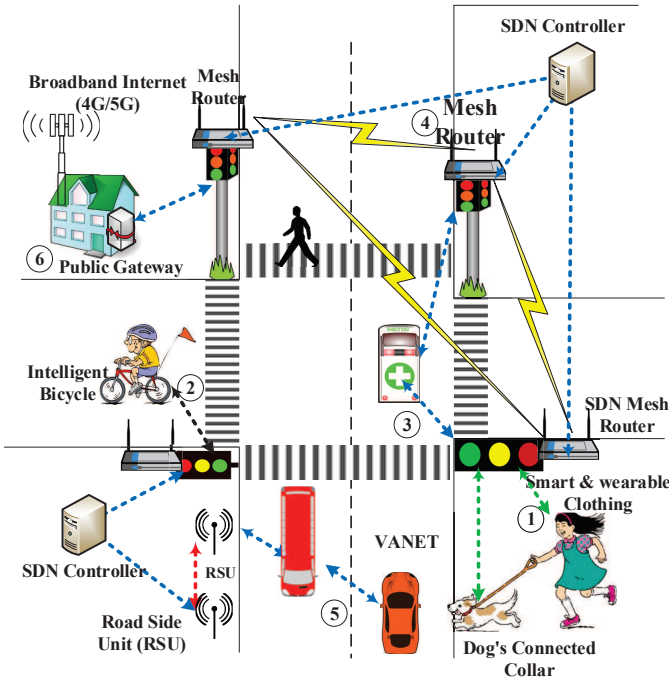


Fig. 1: Overview of the Intelligent Smart Traffic Light System

or by heterogeneous and multi-technology connectivity, such as 4G/5G cellular networks.

### B. Role of SDN in the STLS Scenario

To enable a global view and control of the entire STLS network, the SDN controller in Figure 12 implements routing algorithms to control every wireless router in the data path. It also connects to clusters of RSUs to enable forwarding local road information to connected vehicles and perform emergency services. By pushing the control plane in the controller, we benefit from the simplicity of the implementation as well as easier maintainenance and upgrades to the algorithms. The controller embeds all the network intelligence and maintains a global network-wide view of the routing devices and links connecting them.

At the SDN data plane, i.e., the wireless router, OpenFlow [26], which is the dominant SDN technology, is used to replace the hardware pipeline of the monolithic routers in traditional networks by software pipelines 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. The SDN controller can dynamically install OpenFlow rules in the data plane to decide specific behaviors of the wireless routers and RSUs. It also keeps an up-to-date network view of the vehicles and enables forwarding data like road condition, traffic jam, and road maps to vehicles.

Another complementary technology to SDN called called Network Function Virtualization (NFV) [27] provides a powerful way to run multiple virtual network partitions over shared substrates that run concurrently. The network traffic is thus separated between isolated substrates to avoid leak among partitions. Through network virtualization, it becomes

possible to refactor networking functionalities by virtualizing as many network functions as possible. NFV advocates the virtualization of network functions as software modules that can be assembled and/or chained to create new services.

Decoupling virtual network functions from the hardware allows the network to be tweaked as smart cities requirements change. Specifically, the network underling the STLS should be able to retrieve, store and forward high levels of contextual insight through real-time analytics conducted on extremely large datasets if systems are to be able to problem-solve in real-time; for example, automatically diverting traffic away from a street where a traffic incident has taken place.

## IV. Networking Challenges and Opportunities for Smart Cities

In this section we show how the diverse communication patterns of the STLS scenario introduces a plethora of challenging issues along multiple dimensions, such as wireless network virtualization, controller placement problem, traffic monitoring, and traffic engineering. We also allude to potential opportunities.

### *A. Wireless Network Virtualization*

Smart clients in the STLS scenario in Figure 12 end up repeatedly triggering the embedded controller for marshaling and unmarshaling data. Consequently, the network overhead will increase significantly which combined with the limited computing capabilities and resources of the physical wireless router, will significantly degrade network performance. Wireless router virtualization can increase network capacity and allow high volume of traffic in the STLS scenario by offloading the MAC layer processing to virtualized APs and simplify network management operations. Running multiple non-overlapping isolated wireless networks will provide airtime fairness for multiple different groups of wireless smart clients. Each virtual router will have its own radio configuration, capabilities of advertisements, and a set of distinguished services.

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.

### *B. Efficient Routing*

As the STLS network in Figure 12 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 [18], the link layer multi-hop routing suffers from two shortcomings. First, in MAC layer-based routing, a limited number of wireless nodes (up until 32 nodes as a maximum) are allowed in single network. Second, the conflicting rules between 802.11s and OpenFlow introduce severe performance degradation.

Unfortunately, there are 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.

### *C. Distributed vs Centralized Network Control and Management*

In the STLS scenario of Figure 12, 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.

### D. Wireless Monitoring

The wireless routers need to report their status as well as route modifications after failure or congestion and send them back to an SDN controller. The controller can take its routing decisions based on the collected statistics and faults to optimize the traffic engineering. However, diagnosing the network performance and bottlenecks is a challenging issue in wireless mesh networks. In particular, estimating wireless channel status is hard because these status change frequently due to fading, interference and multi-path selection. Additionally, monitoring wireless channels without having visibility into the traffic characteristics, the network topology, and the link characterization, i.e., latency, load, and stability, can disturb the consistency of wireless networks.

Accordingly, the design of any monitoring algorithm should include software plugins that allow topology and neighbor discovery. Such a monitoring tool should be able to retrieve every wireless device's configuration parameters, collect traffic statistics, link utilization as well as modifying flow table to update wireless routing strategy. It also needs to cover monitoring functionality such as flow display, topology discovery, and check, process and monitor the data generation during the network operations.

### E. Traffic Engineering

Wireless routers and gateways in the STLS scenario depicted in Figure 12 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 [28] so that heuristic algorithms should take into account both wireless channels and routing algorithm.

## V. SDN-Enabled Wireless Architecture

In this section we describe our SDN-enabled wireless architecture to address the aforementioned challenges and avail of the outlined opportunities.

### A. Proposed Architecture

Figure 2 depicts the architecture of the blended SDN-OLSR architecture. At the core of this design is the centralized controller, i.e., the control plane, which communicates with

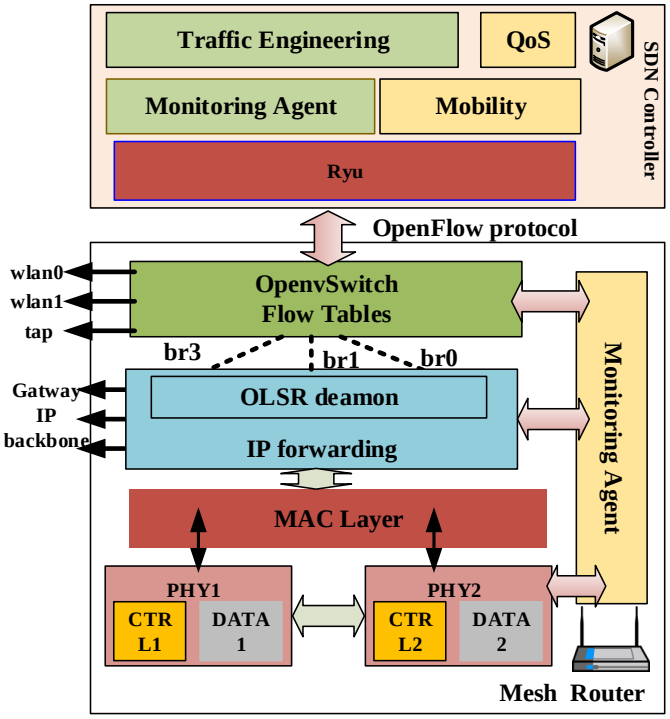


Fig. 2: Architecture of the Joint SDN-IP solution

the underlying mesh routers using the OpenFlow protocol. The controller includes several network modules:

- **Topology discovery module:** which uses the Link Layer Discovery Protocol (LLDP) to perform automatic discovery of joining and leaving mesh routers. The controller broadcasts OpenFlow *PACKET_OUT* messages to all connected routers, which in turn respond by sending ARP messages to notify their liveliness.
- **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. It also supervises the path reservation and modification at run-time. This module allows the controller to query a mesh router to gather individual statistics.
- **Traffic engineering module:** which supports load balancing to offload mesh cloud devices in case of traffic congestion. It also performs traffic redirection based on the optimized routing strategy used in the routing module.

On the data plane, each mesh router 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, i.e., *br0, br1, and br2.*, to exploit

the capacity of IP networks to route packets via the shortest path. Additionally, to enable multiple virtual routers inside the same physical node, the data plane implements two virtual radio interfaces, i.e. *PHY1* and *PHY2* shown in Figure 2. Using virtual radio interfaces allows efficient sharing of the downlink bandwidth between multiple clients and airtime fairness scheduling with the help of channel sharing.

The remainder of this section describes how our architecture resolves the challenges described in Section IV.

### *B. Wireless Network Virtualization: Splitting Routers into Two Virtual Ones*

In order to support wireless virtualization, we slice each physical router into two virtual routers; each has its own virtual hardware resources and virtual radio interface. Hence, each physical access point is split into two non-overlapping virtual APs, i.e., ESSID 1 and ESSID 2. 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, in order to separate the control traffic, i.e., signaling, from the data traffic, each SSID forwards the traffic independently from the other. The benefit of splitting an AP into two virtual ones is twofold. First, we provide an efficient downlink bandwidth sharing between multiple smart clients due primarily to the efficient airtime fairness scheduling with the help of channel sharing. Second, we solve the challenges of uplink channel access for multi-clients simultaneous transmission, and we enable high data rates as well as low latency for those smart clients.

Additionally, allowing two virtualized access point inside the same wireless router 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 a mesh routers to keep alive all the clients attached to a wireless router. To that end, each virtual router advertises Beacon frames in the air so that smart clients can easily and seamlessly associate and connect to it. Such an approach allows the use of existing link layer protocols and while changing the MAC settings simultaneously.

Furthermore, we enhanced the physical layer to transmit the SNR (Signal to Noise Ratio) values, not only to the MAC layer to improve packet delivery, but also we carried these values to the SDN controller to allow the centralized management of the radio interference so that packets are routed along the best path based on the highest SNR values.

### *C. Efficient Routing: Composite Routing for Wireless Networks*

To support efficient routing in SDN-enabled wireless routers, we divide the routing functionalities into two layers as shown in Figure 2. The upper layer supports SDN routing by enabling 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.

Figure 2 depicts the architecture of a SDN-enabled wireless router for smart city network that implements both OpenFlow and OLSR. To allow the controllers to reach all the geographically distributed routers, we used an in-band control approach in a way to provide long distance wireless connectivity among the wireless mesh backhaul. There are two advantages of cohabitating IP-based routing and SDN routing. On the one hand, the controller implements its own routing algorithms for best path selection, and configures mesh routers by adding/removing/updating OpenFlow rules. It also can retrieve the current network states from the nearest mesh router. On the other hand, packets can be routed according to OLSR routing tables under the instruction of the controller through OpenFlow.

Hence, 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, builds periodically a new refreshed routing table, and selects the newest shortest path to all destinations. Thereafter, the controller retrieves the topology information from its nearby mesh routers.

### *D. Centralized versus Distributed: the Controller Placement Problem*

To support a dense wireless communication in smart cities, we propose a hybrid network controller that combines centralized and distributed SDN and attempts to gain the merits of both. To that end, we leverage our prior work on the *'InitSDN'* framework as illustrated in Figure 3. InitSDN [29] is a meta-controller layer based on the boot loading mechanism adopted by operating systems. First, a single centralized controller is deployed at the initialization phase to control and manage the entire network. Then, in case of controller failure or overhead, additional controllers are added at runtime as required to balance the network performance. Additionally, a set of coordination mechanisms are deployed between the distributed controllers to ensure the network consistency. In particular, these mechanisms include an election process that allows electing the closest 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, and fault-tolerant.

InitSDN divides the wireless network into two slices: a date 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.

### *E. Wireless Monitoring: Monitoring the Routers with OpenFlow*

To enable fine-grained control and monitoring of the traffic in SDN-enabled smart cities, the controller implements a monitoring agent as shown in Figure 2. It uses OpenFlow messages to supervise the path reservation, modification and installation.

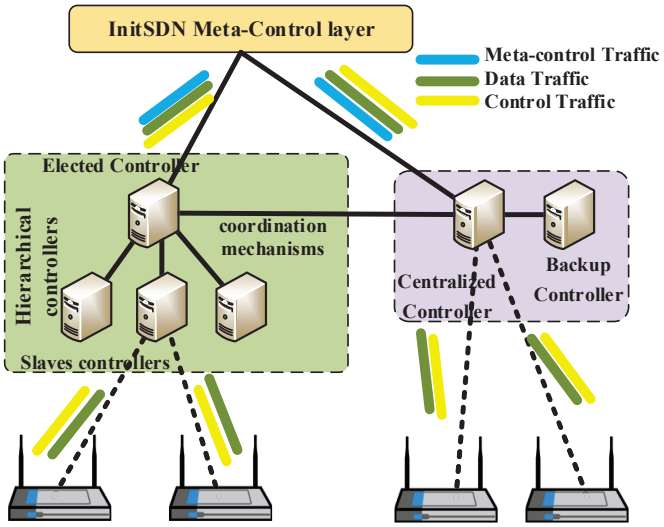


Fig. 3: InitSDN Hybrid Control Plane for the STLS

In particular, the monitoring agent uses two types of messages illustrated in Figure 4 to gather statistics from mesh routers: *STATISTICS* messages and *FlowRemoved* messages. The monitoring agent sends a *FlowStatsRequest* message to the wireless router to query individual flow statistics. The router replies with *FlowStatsReply* to respond to that request. Additionally, when a flow timeout expires in the switch or flow entries are deleted, the switch notifies the monitoring agent with *FlowRemoved* message. This message includes the duration, and packet and byte counts of a recently removed flow. This information is used to monitor the delay between the controller and its corresponding router.

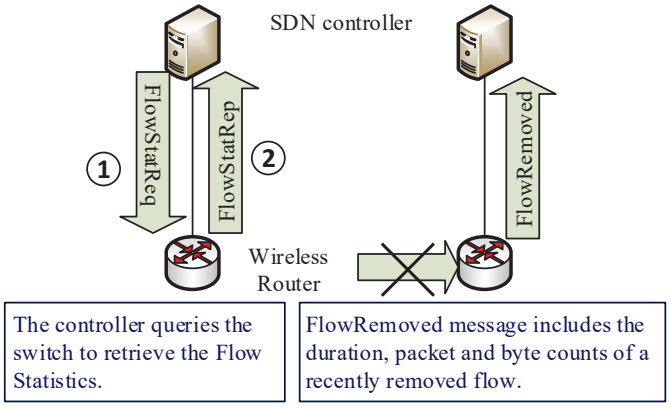


Fig. 4: Retrieving Traffic Statistics by the Monitoring Agent

### *F. Traffic Engineering: Adding Support for Load Balancing*

To address the network overflow issues, we introduce at the controller side a traffic-engineering algorithm that performs load balancing as depicted in Figure 2. Figure 5 depicts the principle of the load balancing approach: the SDN controller connecting the edge routers of the mesh clouds tries to establish a routing path between mesh cloud 1 and mesh cloud across the link ⓐ connecting router 1 and router 4. Links *a*, *b*, *c*, *d*, *e*, and *f* establish the communication paths across the mesh clouds in the STLS.

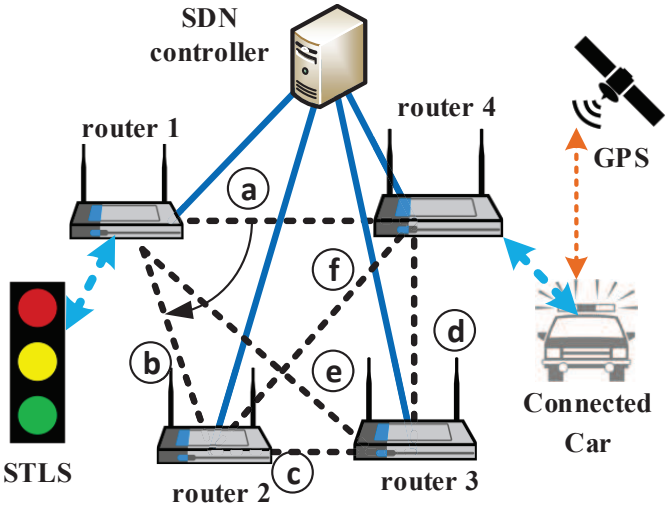


Fig. 5: Load Balancing in the STLS network

As soon as the link becomes a bottleneck, i.e., congestion, connection loss, interference, etc, the load-balancing algorithm is activated in the controller side. Thereafter, the controller can easily decide on switching the data to the next best available path as illustrated by the bow in Figure 5. Algorithm 1 shows the load balancing algorithm to select the optimal path. It calculates the new rules, i.e., the MAC and IP addresses, of the new path towards the new mesh routers, i.e., ⓑ, ⓕ and ⓒ, ⓓ as shown in Figure 5. Once the new path is established end-to-end by sending FlowMod messages, the controller floods all ports towards the selected virtual router, open the client's connection to enable packets reaching their destination, and simultaneously continue discovering and monitoring the network topology. The controller calculates the new optimal path based on the graph topology, which includes all available routers $\Re$ as well as the links $\aleph$ connecting them. Then, it installs new OpenFlow rules to program the flow entries inside the software pipeline in each router.

Table I 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 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.

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

**Algorithm 1:** Load Balancing Algorithm

**Data**: Th, $\Re, \aleph$
**Result**: Rerouting traffic to the optimal path

```
installDefaultFlowRules(ℜ);
while Listening to LLDP packets do
    if TrafficCongestion(Th) then
        calculateNewOFRules(ℜ, ℵ);
        FloodPackets(ℜ);
        calculateOptimalPath(source, destination,ℜ, ℵ);
        if isBestPATH then
            InstallnewOFRules(ℜ);
        else
            goto:
            calculateNewOFRules(ℜ, ℵ);
        end
    else
        monitoring();
    end
end
```

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

TABLE I: Flow entries the controller install in the routers

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 I.

## VI. Evaluating the Blended SDN-OLSR Architecture

This section describes our proposed architecture that blends SDN and OLSR. Our evaluations center around 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.

### A. Testbed Settings

To evaluate our architecture along the different dimensions, we have developed a prototype with NS-3 simulator [30] and Mininet [31] SDN emulator. Mininet is the reference SDN simulator which provides a simple and inexpensive network testbed for developing OpenFlow applications. However, Mininet does not support a realistic wireless protocol stack to enable mesh networks. Therefore, we extended it with the NS-3 simulator. The latter natively supports IP forwarding protocols such as OSLR and AODV. We leveraged the *TapBridge* functionality in NS-3 to integrate NS3 with Mininet so that our SDN-enabled mesh routers support both OpenFlow-based OpenVSwitch routing at the upper layer as well as OLSR protocol as a default IP routing protocol.

At the controller side, we enhanced the Ryu SDN controller to support our approach described in Section V. For example, the monitoring agent is used to supervise the network topology changes and discover any failure event in the mesh cloud using a web-based interface. Additionally, the traffic engineering module includes the load balancing algorithm to offload the mesh routers in case of buffer overflow. Furthermore, to support network virtualization, the Ryu controller can cooperate with *OpenStack* using the *Quantum Ryu plugin* [2] to support Mobile Cloud communication. The extension can be easily integrated into *OpenStack++* [32] platform for enabling mobile Cloudlets.

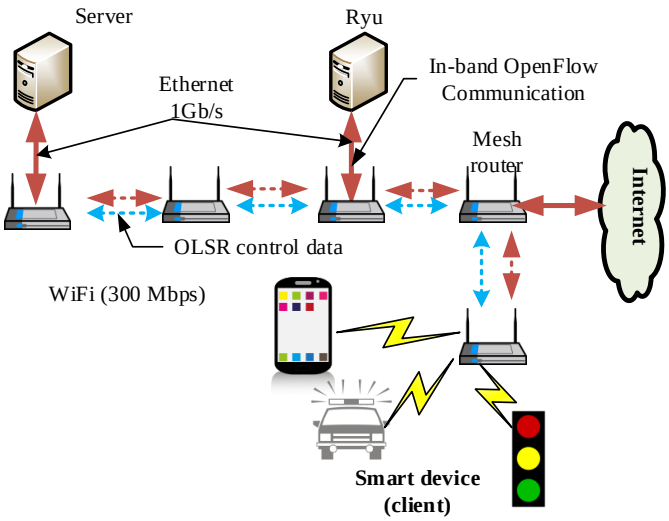


Fig. 6: Experimental setup for the wireless mesh network

To evaluate the proposed solution, we consider the experimental setup depicted in Figure 6. 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 communication with the gateway, which acts as the access point to the Internet. To be able to reach its destination, i.e., the Cloudlet server or the Internet, the Ryu controller should be able to install OpenFlow roles to its neighbor router.

### B. Evaluating Throughput Performance

*a) Rationale:* To evaluate the throughput performance and robustness of our proposed architecture, we consider a UDP traffic between end hosts and the packet size is set to 1,500 bytes. We also consider that each wireless node uses the 802.11b standard to exchange 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.

[2]https://github.com/osrg/ryu/wiki/OpenStack

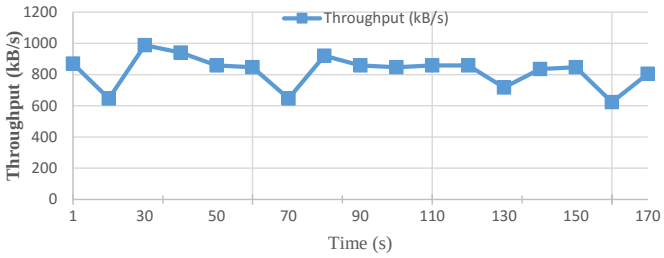


Fig. 7: UDP Throughput

We assume that the controller has already pushed down and installed the flow rules in the OpenFlow tables of the underlying 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.

*b) 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 850 KB/s while the maximum expected throughput is bounded by 988 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 850 KB/s, which we consider as an acceptable value for such an unreliable traffic. The evaluation data confirms our claims on ensuring the fairness of the global optimization of our approach.

### C. Evaluating the Average Relative Error

*a) Rationale:* In order to provide in-depth inspection of the average relative error in the throughput described in Section VI-B, 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.

*b) Analysis:* The packet loss measurements depicted in Figure 8 show that the average error is close to 10%. 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. Nevertheless, the packet error in our experiments does not drastically degrade the performance of the communication because 85% or more of the traffic is sent to the corresponding destination. The observed degradation is due mainly to the additional processing and overhead required by OpenFlow to forward packets between different wireless hops. A close inspection of these results shows that our solution is successfully able to to support SDN-based communication in the smart cities scenarios.

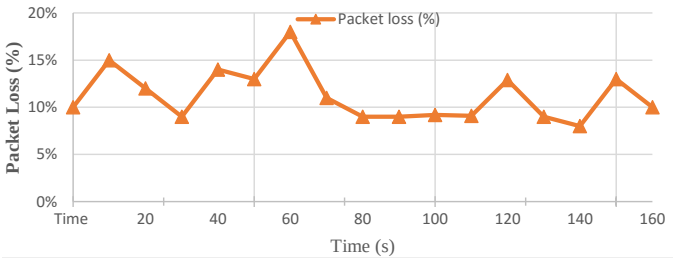


Fig. 8: Packet Loss

### D. Evaluating the End-to-end Delay

*a) Rationale:* We consider the end-to-end delay 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 kept the average latency. The measurement of one-way delay 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 its close for router.

*b) Analysis:* Figure 9 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 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 130 seconds. In all those cases, the controller-router latency remains bounded to 12 ms during the setup phases and close to 3 ms otherwise. Therefore, the controller-router latency does not present a network bottleneck.

The end-to-end latency between remote hosts is shown in Figure 9. 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 40 ms so we consider this value acceptable in the presence of the controller regular operations.

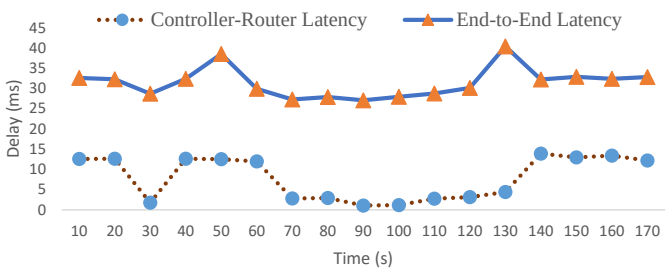


Fig. 9: delay

### E. Evaluating the Router overhead

*a) Rationale:* We consider the performance overhead in each mesh router when using 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 OpenFlow 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.

*b) Analysis:* Figure 10 depicts the total traffic rates generated by OLSR and OpenFlow. After the initiation phase, OpenFlow creates control traffic at time 30 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 9 KBytes/s and the total traffic is 6 times higher compared to a case when OLSR is used as a routing protocol. At time 46 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 flow, OpenFlow adds some extra control flow at the new rules installation phase, but this quantity is relatively low compared to the overall throughput. Therefore, both OpenFlow control flow and OLSR IP forwarding flows do not contribute to the network overhead.

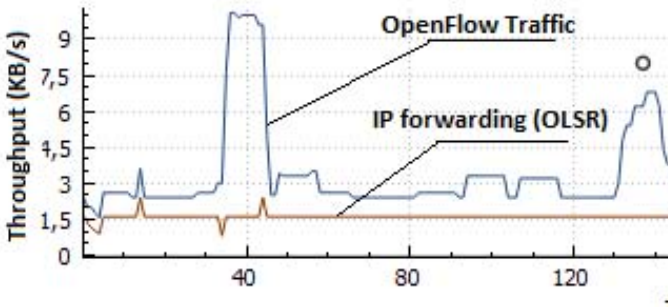


Fig. 10: Network overhead

### F. Evaluating the controller overhead

*a) Rationale:* 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 installed the 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.

*b) Analysis:* Figure 11 shows the control traffic overhead along with the data traffic through a router. The control OpenFlow traffic is close to 35% of the overall traffic exchanged in the wireless network, the data traffic close to 80%, and the Ethernet traffic, i.e., ARP traffic, is close to 20%. The initialization phase requires exchanging layer 2 ARP data to perform host reachability between remote hosts.

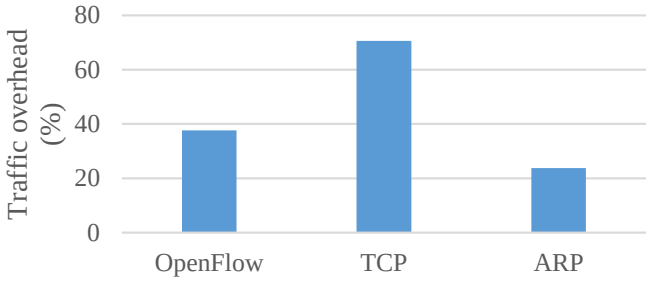


Fig. 11: Controller overhead

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.

### G. Evaluating the Load Balancing

*a) Rationale:* 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 path 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,

we inject after 40 seconds a competing flow into router 4 to simulate network congestion and introduce a performance degradation in this node.

*b) Analysis:* Figure 13 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 200 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.

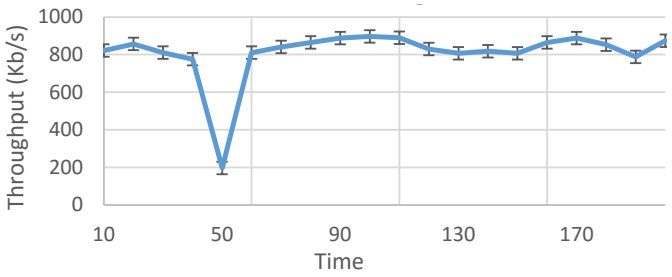


Fig. 12: TCP Throughput

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 I. The IP and MAC addresses of router 2 are added in the new rules. The bow in Figure 5 shows the new path selected by the controller by installing new OpenFlow rules in node 1.

A close inspection of Figure 13 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. Our results shows that our approach to provide traffic engineering in wireless networks succeeds in redirecting packets to the new selected path when multiple wireless hops are available in the network.

## VII. Conclusions

This paper described several important challenging issues that need to be tackled for efficient support of wireless communication in Smart Cities. To address these challenges and avail of the opportunities, we introduced a novel architecture based on a symbiotic relationship between wireless mesh networks (WMNs) and software defined networking (SDN) that enables agile and flexible communication.

The following is a summary of the insights we gained from this research and directions for future work.

***Bringing the cloud computation to the network edge:*** The proposed architecture provides new opportunities to build vehicular cloud services for data collection and analytics by forwarding all the traffic data to the network core through the Internet gateways. Nonetheless, sending data to remote cloud servers consumes higher bandwidth and introduces extra latency for real-time applications like video streaming inside the cars. We believe that with the emergence of self-driving vehicles, real-time data processing, and safety information and storage for clusters of RSUs should be provided at the road side using mobile Cloudlets and Fog devices to offer low-latency, context- and location-awareness to fulfill the future needs of smart cities.

***Cross-Layer design and optimization:*** The proposed approach allows splitting the wireless data plane into several separate channels to improve the QoS. 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) could unify the network resource management and the radio resource management. We can rethink on how a cross-layering design could interoperate both SDN and SDR for better spectrum utilization and channel interactions, which forms additional dimensions of our future work.

## Acknowledgment

This work was supported in part by the Fulbright 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] U. Nations, "World population prospects: The 2015 revision, methodology of the united nations population estimates and projections."

[2] A. Gyrard and M. Serrano, "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)*, March 2016, pp. 796–802.

[3] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, 2014.

[4] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," RFC 3561 (Experimental), Internet Engineering Task Force, Jul. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3561.txt

[5] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," RFC 3626 (Experimental), Internet Engineering Task Force, Oct. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3626.txt

[6] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *Communications Surveys Tutorials, IEEE*, vol. 16, no. 3, pp. 1617–1634, 2014.

[7] X. Wang, C. Wang, J. Zhang, M. Zhou, and C. Jiang, "Improved rule installation for real-time query service in software-defined internet of vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. PP, no. 99, pp. 1–11, 2016.

[8] K.-K. Yap, M. Kobayashi, R. Sherwood, T.-Y. Huang, M. Chan, N. Handigol, and N. McKeown, "Openroads: empowering research in mobile networks," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 125–126, 2010.

[9] J. Schulz-Zander, L. Suresh, N. Sarrar, A. Feldmann, T. Hühn, and R. Merz, "Programmatic orchestration of wifi networks," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. USENIX Association, 2014.

[10] H. Huang, P. Li, S. Guo, and W. Zhuang, "Software-defined wireless mesh networks: architecture and traffic orchestration," *Network, IEEE*, vol. 29, no. 4, pp. 24–30, July 2015.

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

[12] E. Bozkaya and B. Canberk, "Qoe-based flow management in software defined vehicular networks," in *2015 IEEE Globecom Workshops (GC Wkshps)*, 2015, pp. 1–6.

[13] K. Xu, R. Izard, F. Yang, K. C. Wang, and J. Martin, "Cloud-based handoff as a service for heterogeneous vehicular networks with openflow," in *2013 Second GENI Research and Educational Experiment Workshop*, March 2013, pp. 45–49.

[14] N. B. Truong, G. M. Lee, and Y. Ghamri-Doudane, "Software defined networking-based vehicular adhoc network with fog computing," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 1202–1207.

[15] P. Dely, J. Vestin, A. Kassler, N. Bayer, H. Einsiedler, and C. Peylo, "Cloudmac: An openflow based architecture for 802.11 mac layer processing in the cloud," in *Globecom Workshops (GC Wkshps), 2012 IEEE*, Dec 2012, pp. 186–191.

[16] I. Ku, Y. Lu, and M. Gerla, "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*, 2014, pp. 1–6.

[17] J. Chung, G. Gonzalez, I. Armuelles, T. Robles, R. Alcarria, and A. Morales, "Experiences and challenges in deploying openflow over a real wireless mesh network," in *Communications (LATINCOM), 2012 IEEE Latin-America Conference on*, Nov 2012, pp. 1–5.

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

[19] D. Zhu, X. Yang, P. Zhao, and W. Yu, "Towards effective intra-flow network coding in software defined wireless mesh networks," in *Computer Communication and Networks (ICCCN), 2015 24th International Conference on*, Aug 2015, pp. 1–8.

[20] P. Dely, A. Kassler, and N. Bayer, "Openflow for wireless mesh networks," in *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*, July 2011, pp. 1–6.

[21] A. Detti, C. Pisa, S. Salsano, and N. Blefari-Melazzi, "Wireless mesh software defined networks (wmsdn)," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on*, Oct 2013, pp. 89–95.

[22] I. Brito, S. Gramacho, I. Ferreira, M. Nazare, L. Sampaio, and G. Figueiredo, "Openwimesh: A framework for software defined wireless mesh networks," in *Computer Networks and Distributed Systems (SBRC), 2014 Brazilian Symposium on*, May 2014, pp. 199–206.

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

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

[25] M. F. Feteiha and H. S. Hassanein, "Enabling Cooperative Relaying VANET Clouds Over LTE-A Networks," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 4, pp. 1468–1479, April 2015.

[26] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[27] T. Wood, K. K. Ramakrishnan, J. Hwang, G. Liu, and W. Zhang, "Toward a software-based network: integrating software defined networking and network function virtualization," *IEEE Network*, vol. 29, no. 3, pp. 36–41, May 2015.

[28] B. Mumey, J. Tang, I. Judson, and D. Stevens, "On Routing and Channel Selection in Cognitive Radio Mesh Networks," *Vehicular Technology, IEEE Transactions on*, vol. 61, no. 9, pp. 4118–4128, Nov 2012.

[29] P. Patil, A. Gokhale, and A. Hakiri, "Bootstrapping Software Defined Network for flexible and dynamic control plane management," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, April 2015, pp. 1–5.

[30] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley, "Ns-3 project goals," in *Proceeding from the 2006 Workshop on Ns-2: The IP Network Simulator*, ser. WNS2 '06, 2006.

[31] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX, 2010.

[32] K. Ha and M. Satyanarayanan, "Openstack++ for cloudlet deployment," pp. 1–24, Aug.