Work-in-Progress: Towards Real-time Smart City Communications using Software Defined Wireless Mesh Networking

Akram Hakiri* and Aniruddha Gokhale†
*University of Carthage, SYSCOM ENIT, ISSAT Mateur, Tunisia.
†ISIS, Dept of EECS, Vanderbilt University, Nashville, TN, USA.
Email: akram.hakiri@ieee.org, a.gokhale@vanderbilt.edu

***Abstract*—Effective management and provisioning of communication resources is as important in meeting the real-time requirements of smart city cyber physical systems (CPS) as managing computation resources is. The communication infrastructure in Smart cities often involves wireless mesh networks (WMNs). However, enforcing distributed and consistent control in WMNs is challenging since individual routers of a WMN maintain only local knowledge about each of its neighbors, which reflects only a partial visibility of the overall network and hence results in suboptimal resource management decisions. When WMNs must utilize emerging technologies, such as time-sensitive networking (TSN) for the most critical communication needs, e.g., controlling traffic and pedestrian lights, these challenges are further complicated. An attractive solution is to adopt 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. This paper presents ongoing work to overcome the key challenges and support the end-to-end real-time requirements of smart city CPS applications.**

***Index Terms*—Software Defined Networking; Wireless Mesh Networks; Smart Cities Cyber Physical Systems.**

I. Introduction

Several smart city initiatives have been undertaken worldwide to address the myriad of problems that arise due to rapid urbanization and the ensuing challenges. Wireless Mesh Networks (WMNs) often serve as the backbone communication technology for the communication networks of smart city cyber physical systems (CPS) [1]. 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. Such a mesh cloud allows monitoring of vehicular traffic activity in cities to help alleviate congestion.

Many smart city applications, such as intelligent traffic light control and traffic coordination, have stringent end-to-end real-time requirements. Such a wide array of smart city CPS applications, each with their own performance requirements, makes it hard for WMNs to enforce effective and distributed allocation and management of communication resources. For example, consider the consequences if WMNs used known wireless routing protocols such as AODV (Ad hoc On Demand Distance Vector) and OLSR (Optimized Link State Routing Protocol) in making effective routing decisions and satisfying the end-to-end timeliness constraints. The design of these protocols are influenced primarily by the *ad hoc* nature and local area network (LAN) constraints. Thus, any routing decision is taken in a distributed manner based only on local knowledge of a mesh router about its neighbors, which reflects only a partial visibility of the network. Consequently, WMNs would make only suboptimal decisions in supporting the real-time needs of smart city applications and their high volume network traffic patterns. 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. Finally, WMNs may need to utilize emerging technologies, such as Time-Sensitive Networking (TSN) [2], to satisfy the stringent real-time communication needs.

Therefore, to deploy new smart city services over WMNs, we need better manageability, control and flexibility in the network, which is feasible using Software Defined Networking (SDN) [3]. SDN shows significant promise in meeting smart city needs by optimizing routing paths for information through the network [4]. SDN decouples the control plane from the data plane for distributed, networked applications so that all network management, such as routing, 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. The controller can manage a variety of network elements, such as routers, switches, and different types of middle boxes.

Recent promising approaches for programmable wireless networks reveal the use of SDN to build relays between home gateways and the Internet [5], simplify the network management operations of the wireless access points [6] and enhance the traffic orchestration [7] 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.

To address these challenges, we present ongoing work on a novel approach that incorporates SDN into WMNs to define and manage a powerful and easy-to-deploy CPS network that meets the real-time requirements of CPS applications.

II. SDN-enabled Wireless Mesh Networks for CPS

We are designing our SDN-enabled wireless mesh network solution for Smart City CPS as shown in Figure 1. 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 [8]. We propose an efficient hybrid routing scheme using SDN-enabled wireless routers, where we divide the routing functionality into two layers. 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.

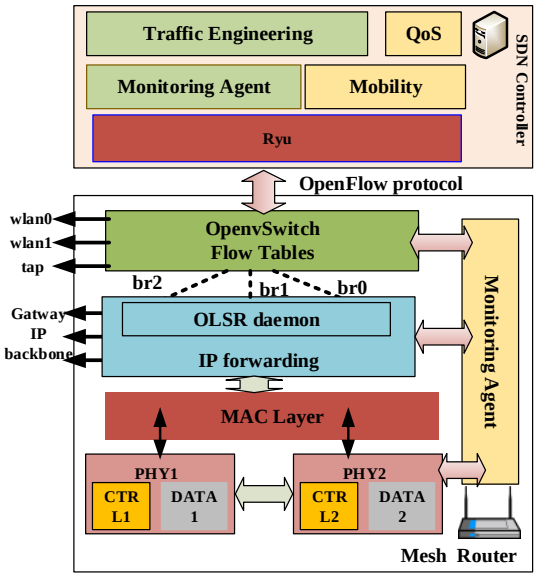

Fig. 1: Architecture of the Joint SDN-WMN solution

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.

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 can also 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.

III. Ongoing Work and Research Evaluation

We are using emulation as a means to initially validate our architecture. For the emulation testbed, we combined the NS-3 simulator [9] and Mininet SDN emulator [10]. Mininet is the reference SDN emulator which provides a simple and inexpensive network testbed for developing OpenFlow applications. Since Mininet does not support a realistic wireless protocol stack to enable mesh networks, we integrated Mininet with NS-3. The latter natively supports IP forwarding protocols such as OSLR and AODV. We leveraged the *TapBridge* functionality in NS-3 to integrate it 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. To support network virtualization, the Ryu controller can cooperate with *OpenStack* using the *Quantum Ryu plugin* (https://github.com/osrg/ryu/wiki/OpenStack) to support Mobile Cloud communication. The source code for this research is available at https://github.com/doc-vu.

To evaluate the proposed solution, we used an initial experimental setup as depicted in Figure 2. 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 must install OpenFlow rules to its neighbor routers.

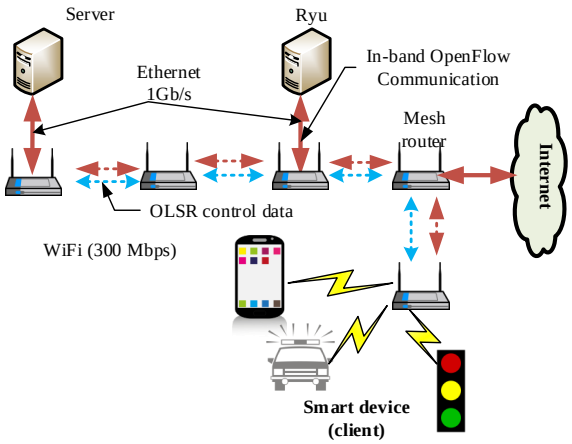

Fig. 2: Experimental setup for the wireless mesh network

Measuring the Overhead of the Solution: Before real-world deployment, it is important to first gauge the overhead imposed by the additional infrastructure elements our solution introduces. To that end, we first evaluated 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 were conducted five times and the average values were 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 the routers was captured using Wireshark and the analyses were performed with the Tcpdump packet analyzer.

Figure 3 shows the control traffic overhead and 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 close to 10%. The initialization phase requires exchanging Ethernet traffic to perform host reachability between remote hosts.

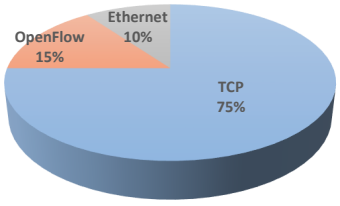

Fig. 3: Controller overhead

Indeed, the first host sends ARP requests across the networks, which generate broadcast of *PACKET_OUT* messages to all nodes in that 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 still minimal when compared to the TCP data traffic. Therefore, the control traffic does not contribute significant overhead, which indicates that our solution does not impose any undue overhead on the system.

We also measured the overhead on existing routers. To that end, we considered the performance overhead in each mesh router when using our hybrid routing approach versus the traditional 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 1. 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.

Figure 4 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 does not contribute to router overhead.

We also measured the overhead on the SDN control traffic and found out that the 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 around 10%.

Evaluating Throughput Performance: Scalability and throughput of applications is a key requirement in CPS systems given the scale of CPS and the presence of a large number of competing applications. To evaluate the throughput performance and robustness of our proposed architecture, we consider UDP traffic between end hosts (e.g., video traffic for smart traffic light) 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

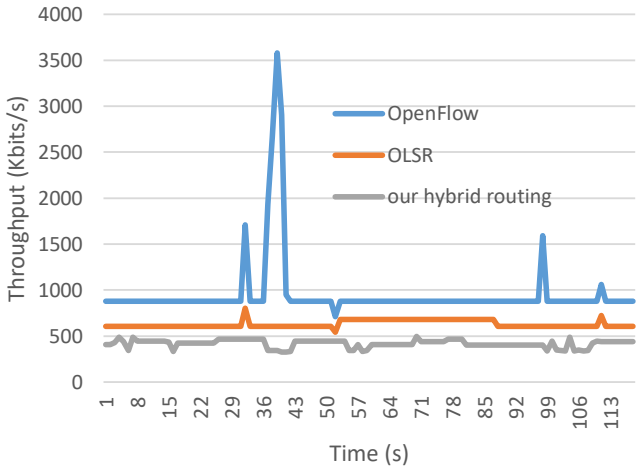

Fig. 4: Evalution of the network overhead

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.

Figure 5 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 cause the throughput to decrease: 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.

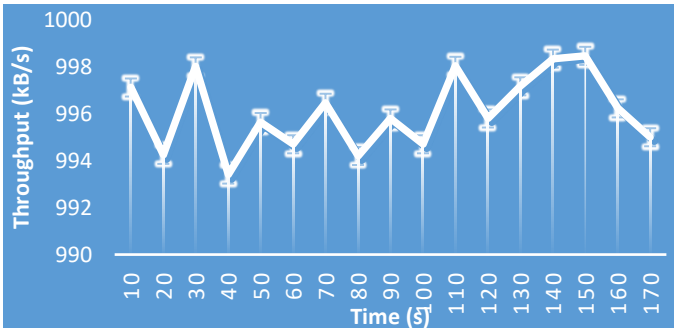

Fig. 5: UDP Throughput

Ongoing Work: Although our empirical validations to date are emulation-based, we are prototyping our solution on a collection of Raspberry Pi-2 running the OpenWRT Linux OS. We are also incorporating TSN [2] evaluation boards and integrating them into our SDN-enabled WMN design for traffic that needs stringent real-time properties. We are conducting a variety of additional experiments including measuring the end-to-end latencies and their predictability for CPS applications. Additionally, we are incorporating TSN in the networks and measuring the performance and predictability improvements using TSN. Third, we aim to provide a more holistic resource management solution where multiple different resource types, e.g., CPU, networks, etc are managed through a single framework. Finally, we are seeking opportunities to deploy our solution in Smart City scenarios for which we are seeking ongoing efforts such as US Ignite Smart Cities.

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] S. Vural, D. Wei, and K. Moessner, "Survey of experimental evaluation studies for wireless mesh network deployments in urban areas towards ubiquitous internet," *IEEE Communications Surveys Tutorials*, vol. 15, no. 1, pp. 223–239, 2013.

[2] A. Nasrallah, A. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. E. Bakoury, "Ultra-low latency (ull) networks: The ieee tsn and ietf detnet standards and related 5g ull research," *IEEE Communications Surveys Tutorials*, pp. 1–1, 2018.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.