Communication Overhead of an OpenFlow Wireless Mesh Network

Arun. K. P., Abhishek Chakraborty, and B. S. Manoj
Indian Institute of Space Science and Technology, Thiruvananthapuram, India 695547
kp.arun@outlook.com, {abhishek2003slg, bsmanoj}@ieee.org

***Abstract*—OpenFlow is a new paradigm for running experimental protocols on production networks, and is becoming a standard reference for implementation of the software defined networking. OpenFlow can be deployed on wireless mesh network, a multi-hop relaying wireless network, for efficient network traffic management. Furthermore, various challenges such as load-balancing and control traffic management of wireless mesh networks can be effectively handled with the OpenFlow central controller. In this paper, we develop an OpenFlow wireless mesh network testbed to investigate various performance metrics (e.g., throughput, latency, CPU usage, and control traffic) for full mesh and partial mesh network topologies. Furthermore, we measure empirical controller capacity, a new metric we defined to evaluate the OpenFlow wireless mesh network controller performance. We also address various implementation challenges of real-world OpenFlow wireless mesh networks.**

***Keywords*—OpenFlow, wireless mesh network, throughput, empirical controller capacity, control traffic overhead, CPU usage.**

I. Introduction

WIRELESS mesh networks (WMNs) is formed as distributed networks on multi-hop relaying architecture over partial or full mesh topologies. WMNs have three types of nodes: mesh router, mesh client, and gateway mesh router [1]. Mesh routers in a WMN are deployed as static nodes or nodes with minimal mobility in the network. Therefore, mesh routers are functioning as access points in WMNs and perform all routing and maintenance operations. Mesh clients, on the other hand, are typically end-user equipment and thus, they are mobile in WMNs. Mesh clients are communicating to each other with multi-hop relaying through mesh routers and mesh clients. Gateway mesh routers are connected to the Internet or other existing infrastructure networks by means of backbone wired interfaces. Due to the distributed nature of WMNs, it is difficult to implement solutions to problems such as priority based traffic, client mobility, and load-balancing. However, OpenFlow simplifies the network administration by moving the network intelligence to the central controller even when the network data path is maintained as a distributed multi-hop path.

McKeown et al. [2] first proposed OpenFlow as a method of running experimental protocols on production networks without interfering with production traffic. OpenFlow enables better control of networks with the help of a central controller by separating network control traffic and data traffic. OpenFlow switch consists of a Flow Table, a secure channel, and the OpenFlow protocol. OpenFlow Flow Table stores the processing information for arriving packets. When a packet with no associated rules arrives at the switch, the packet is encapsulated in an OpenFlow message and sent to the OpenFlow controller through the secure channel to make required routing actions. The Flow Table is then updated accordingly to route data packets in the network. OpenFlow protocol, on the other hand, is a means of communication between the central controller and the switch in an open and standard manner [2].

Due to the centralized nature, OpenFlow can be deployed to solve problems such as load-balancing and control overhead optimization in the context of WMNs. By identifying abnormal traffic patterns, OpenFlow can identify attacks on the network infrastructure. Moreover, OpenFlow implements wireless channel switching by exchanging information with the link-layer. Sharing information with the link-layer enables OpenFlow to identify paths with end-to-end assured connectivity in the network. However, presence of a central controller in the OpenFlow WMNs introduces a single point of failure.

The remainder of the paper is organized as follows. Section II discusses existing literature on OpenFlow WMNs. In Section III, we present implementation details of our OpenFlow WMN testbed. Section IV describes performance results of various parameters in the context of OpenFlow WMNs. We highlight overall observations from our experiments in Section V. Section VI concludes the paper.

II. OpenFlow WMNs

There exist very limited experimental studies in the area of OpenFlow WMNs. Here we discuss a few literature on wired and wireless OpenFlow testbeds and present implementation challenges of OpenFlow WMNs.

The Stanford OpenRoads project [3] is a wired infrastructure based OpenFlow testbed which uses FlowVisor [4], a network hypervisor. The study involved monitoring the traffic through the network and developing the tools for logging, plotting graph, and visualizing the network traffic. A nationwide OpenFlow based wired mesh topology network was deployed in the Japan Gigabit Network 2 plus (JGN2plus) Ethernet testbed [5]. IEEE802.1q Q-in-Q tunneling [6] technique was used to overlay mesh topology on the tree topology based JGN2plus network. An NEC prototype OpenFlow controller was used for visualizing the flow. Apart from that, to find the solutions for fast recovery from network failure, various experiments on wired OpenFlow virtual networks have been conducted [7].

OpenFlow can be deployed with either out-of-band control or in-band control approach. In out-of-band control, OpenFlow control traffic from central controller and data traffic are routed through different secured TCP channels. On the other hand, in-band control employs control and data traffic in the same connection. All of the above mentioned work [3], [5], [7] deploy out-of-band control in wired OpenFlow for communication between the controller and the routers.

There exist only limited work in the area of OpenFlow WMNs. The authors in [8] performed an experimental study on OpenFlow based WMNs to evaluate the forwarding performance, rule activation delay, and control traffic overhead of the OpenFlow switch. The study on client mobility management [8] suggested OpenFlow's ability to provide feasible solutions in the context of WMNs. The author in [9] identified a set of specific challenges and proposed a system architecture incorporating solutions to the problems for OpenFlow WMNs. Another study on OpenFlow based WMNs used Ethernet links or multiple wireless interfaces for the control traffic [10]. In this work [10], OpenFlow performance was compared against better approach to mobile ad-hoc networking-advanced (BATMAN-adv) and 802.11s standard.

Our study focuses on the challenges to develop in-band control path and its implications in the context of an OpenFlow WMN with respect to control overhead. We measure throughput, CPU usage, and control overhead in OpenFlow enabled mesh routers deployed in full mesh as well as partial mesh topologies.

III. Our Implementation of OpenFlow WMN

OpenFlow is deployed along with the optimal link state routing (OLSR) protocol [11] to establish connectivity among WMN routers and the central controller. Mesh routers use in-band control channel to establish connection with the OpenFlow central controller.

A. *OpenFlow WMN Routers*

We developed OpenFlow mesh routers with PC Engine's Alix3d3 board [12] which are small and low powered system-boards. Figure 1 shows a view of Alix3d3 board with omni-directional antennas attached. Table I shows the technical specifications of the WMN router where Voyage Linux 0.9.1 [13] is installed. The wireless interface (operating at 2.4GHz) in each mesh router is configured such that it can operate in ad-hoc mode. We choose Stanford OpenFlow version 1.0.0 [14] for our OpenFlow WMN routers where the wireless interfaces are configured for in-band control operation. The routers are mounted on a support stand such that they can be portable for setting up various topology configurations. Figure 2 shows the final deployed view of our OpenFlow WMN router.

In order to configure OpenFlow for in-band control, a datapath is setup by assigning a unique number for each OpenFlow mesh router. A *tun/tap* interface [15] is created for OpenFlow in-band control. Internet protocol (IP) address is assigned to the tun/tap interface of each mesh router in the network.

B. *The OpenFlow Controller*

We deploy Floodlight [16] which is a Java-based open source OpenFlow controller supporting OpenFlow-1.0.0 version. Floodlight controller is made of a number of modules to support additional functionality such as link discovery, flow-cache, and topology manager to manage the network. Moreover, it seamlessly works with both physical and virtual OpenFlow switches.

Floodlight supports two methods for inserting flows: proactive and reactive. Proactive flow insertion involves setting up flow rules in advance. In reactive flow insertion, new rules are added to the Flow Table as new flows are encountered by the switches. Representational state transfer (REST) API, an HTTP API to exchange information, is used for flow insertion.

Floodlight controller is installed in a laptop which runs Ubuntu Linux 12.04 LTS. The laptop has a 32-bit 1.2GHz Core™2 Duo processor with 2GB RAM. Floodlight uses OLSR protocol along with tun/tap interface to find path to each node.

TABLE I
WMN Router Hardware Specifications

Part	**Specification**
Processor	500MHz AMD Geode
RAM	256MB DDR DRAM
Storage	8GB CompactFlash socket
Expansion	2 miniPCI slots, LPC bus
I/O	DB9 serial, VGA, 2 USB ports
Board size	100 x 160 mm
Firmware	Award BIOS
Power	DC jack or passive POE, min. 7V to max. 20V

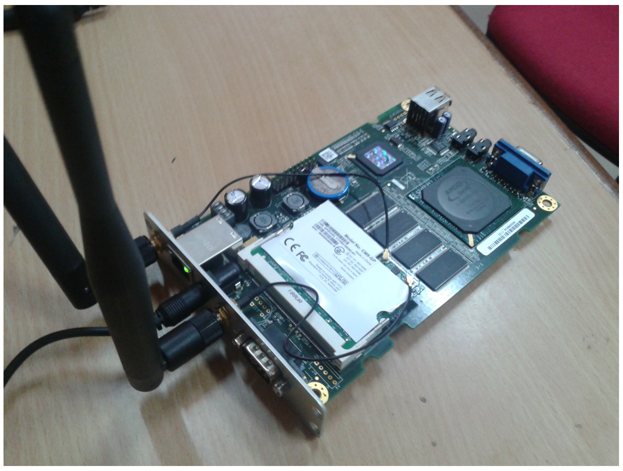
Fig. 1: A view of our OpenFlow WMN hardware platform (PC Engine Alix3d3 board).

IV. Experimental Results

Experiments are performed on OpenFlow WMN testbed to evaluate various performance parameters such as throughput, control traffic overhead, CPU workload, and transmission delay for full mesh and partial mesh topology networks. We measure the TCP performance of WMN routers in the context of control overhead with OpenFlow central controller. We also measure TCP performance with OLSR protocol installed on the WMN routers and then we compare network performance with respect to OpenFlow WMN.

In full mesh topology WMN, each mesh router is directly connected to remaining mesh routers in the network. On the other hand, in partial mesh topology, each mesh router may

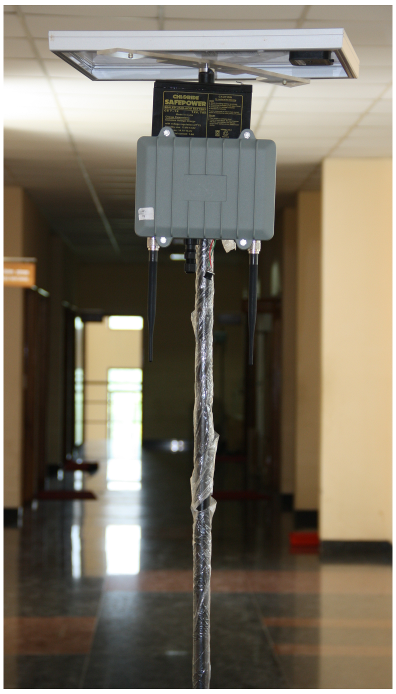
Fig. 2: OpenFlow WMN router setup used for experiments.

not have direct links to rest of the router nodes in the network. Measurements of router data traffic as well as control traffic are carried out by taking the average of several data transfer sessions in the network. Similarly, we measure the control traffic between the controller and routers over multiple seeds. Additionally, we compare OLSR and OpenFlow control traffic overhead in the context of WMNs.

In order to evaluate the impact of large OpenFlow control traffic on partial mesh topology, nodes are placed at various locations and then traffic is monitored at the central controller. An OpenFlow WMN router in the network is setup with no rules to handle the incoming traffic. Then control traffic is monitored by the central controller for all mesh routers to evaluate the impact of the control overhead in OpenFlow WMNs. Figure 3 shows the topology of the partial mesh OpenFlow WMN.

We measure TCP control traffic of all mesh routers in the context of OpenFlow WMNs. TCP control traffic refers to the TCP packets which are exchanged between the central controller and OpenFlow routers to maintain connectivity in the network. Further, we identify the maximum capacity of the central controller to handle incoming traffic. CPU usage of the OpenFlow routers and OpenFlow controller are measured for various scenarios. Moreover, we measure overhead generation by OLSR protocol in the context of Layer-3 deployment in WMNs.

A. *Node-pair TCP Throughput Performance*

TCP throughput performance of each node-pair in the OpenFlow WMN testbed is carried out for full mesh as well as partial mesh topology. Here, TCP throughput refers to the number of bits transferred per unit time from a source node to a destination node in the network. We use Iperf [17] tool to measure the link throughput of the OpenFlow WMNs.

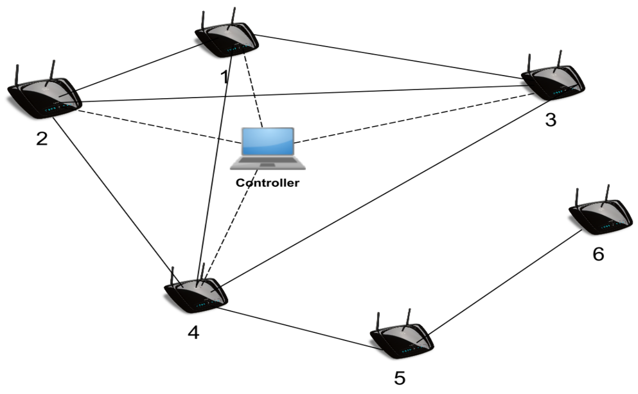

Fig. 3: Experimental setup for six OpenFlow WMN routers in a partial mesh topology.

TCP throughput measurement is carried out for 10 seconds interval which is averaged for each link over five sessions. The OpenFlow routers are configured to forward the incoming Iperf traffic to destination routers in order to take the measurements.

Table II shows the throughput measurement for all node pairs in full mesh topology WMN. The node-pair throughput varies greatly across various pairs in the network. We observe that there is a drop in node-pair throughput for most cases when OpenFlow is used. The mean drop in the throughput performance with OpenFlow is found to be 8.0% for the full mesh topology network.

TABLE II
TCP THROUGHPUT BETWEEN NODES FOR FULL MESH TOPOLOGY

Nodes	**Throughput (Mbps)**	**Standard Deviation**	**Throughput with Open-Flow (Mbps)**	**Standard Deviation**
1 $\leftrightarrow$ 2	16.24	0.93	10.84	0.08
1 $\leftrightarrow$ 3	7.14	0.76	6.24	0.16
1 $\leftrightarrow$ 4	19.19	1.02	20.78	0.18
1 $\leftrightarrow$ 5	9.75	1.57	11.53	0.18
1 $\leftrightarrow$ 6	15.17	0.17	13.92	0.25
2 $\leftrightarrow$ 3	17.27	0.22	13.76	0.46
2 $\leftrightarrow$ 4	20.87	0.44	20.08	0.92
2 $\leftrightarrow$ 5	17.49	1.25	16.50	0.36
2 $\leftrightarrow$ 6	11.11	0.22	9.29	0.34
3 $\leftrightarrow$ 4	15.14	0.87	15.53	0.33
3 $\leftrightarrow$ 5	14.47	0.29	13.10	0.50
3 $\leftrightarrow$ 6	19.62	1.32	19.38	0.82
4 $\leftrightarrow$ 5	13.79	0.31	11.74	0.05
4 $\leftrightarrow$ 6	14.96	0.21	13.56	0.10
5 $\leftrightarrow$ 6	17.29	0.05	14.88	2.20
Mean	**15.30**	**0.64**	**14.08**	**0.46**

Table III presents TCP throughput results of partial mesh topology OpenFlow WMNs. It is noticed that there is an overall reduction in the average node-pair throughput with and without OpenFlow switch implementation in the partial mesh topology. The degradation in the TCP throughput of partial mesh WMNs when using OpenFlow is 10.7%. The degradation is observed due to the additional processing required by OpenFlow to forward the packets due to relaying in the context of partial mesh topology.

TABLE III
TCP THROUGHPUT BETWEEN NODES FOR PARTIAL MESH TOPOLOGY

Nodes	Throughput (Mbps)	Standard Deviation	Throughput with Open-Flow (Mbps)	Standard Deviation
1 ↔ 2	20.31	1.44	15.09	2.44
1 ↔ 3	11.98	2.14	12.46	1.47
1 ↔ 4	11.19	0.54	8.88	0.63
1 ↔ 5	12.78	0.21	10.16	0.22
1 ↔ 6	6.30	0.80	7.03	2.04
2 ↔ 3	13.00	0.87	8.83	0.61
2 ↔ 4	10.35	2.53	12.04	0.66
2 ↔ 5	5.83	1.62	5.48	0.13
2 ↔ 6	4.67	0.48	6.01	2.64
3 ↔ 4	5.38	0.57	4.65	0.16
3 ↔ 5	10.73	1.16	8.13	1.11
3 ↔ 6	9.73	1.05	7.80	0.70
4 ↔ 5	7.42	2.96	7.63	0.57
4 ↔ 6	4.73	0.78	5.39	0.49
5 ↔ 6	6.31	2.11	6.06	0.51
Mean	**9.38**	**1.28**	**8.38**	**0.96**

B. Control Traffic Overhead

The control traffic of each node is measured by capturing the traffic using tcpdump [18]. The traffic at each node is analyzed and control traffic overhead is measured. Control traffic overhead measurement is averaged over five repeated sessions. Figure 4 shows the control traffic at each node of the OpenFlow WMNs along with a rule to handle all incoming traffic.

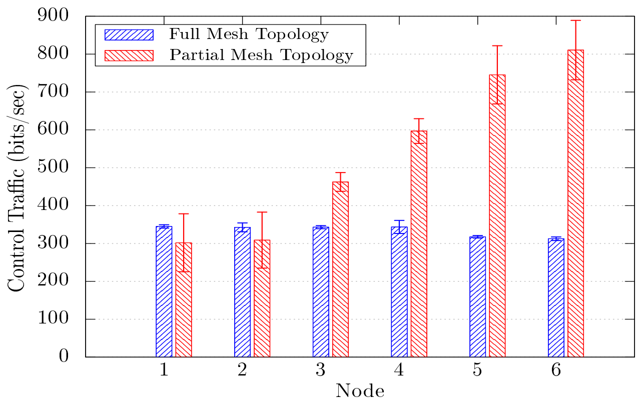

Fig. 4: Control traffic for each node in full and partial mesh topologies.

The data traffic overhead of partial mesh topology is lower compared to the data throughput of full mesh topology OpenFlow WMNs. However, the control traffic for the partial mesh topology is higher than the full mesh topology. Due to larger amount of relay traffic, control traffic for farther away nodes are high in partial mesh networks. For example, nodes that contribute higher relay traffic in Figures 4, 5, and 6 also lead to higher OpenFlow control overhead.

The control traffic in Figure 4 is minimal when compared to the data throughput (see Table III) for each node. Therefore, the control traffic does not contribute significant overhead. The average link throughput using OpenFlow full mesh topology WMN is nearly 14Mbps and the control traffic is only about 250Bps.

C. Empirical Controller Capacity (ECC)

We define empirical controller capacity (ECC) as the maximum number of OpenFlow routers that can be supported by an OpenFlow central controller for a given topology. Beyond the empirical controller capacity, the central controller becomes a bottleneck to handle the control traffic.

We estimate ECC based on the following procedure. ECC is estimated as the ratio of average data link throughput to the average controller overhead traffic. For example, average data throughput for our OpenFlow WMN is found to be nearly 14Mbps, and average control traffic is around 250Bps in our testbed when full mesh topology is concerned. Therefore, the maximum number of OpenFlow mesh routers that can be supported by the central controller, without impairing its normal operation, is nearly 7000 in our testbed. The control traffic in the context of partial mesh topology is higher compared to full mesh topology of same network size. Hence, ECC value in the context of partial mesh topology is lower than the full mesh topology.

D. CPU Usage for the OpenFlow Central Controller

CPU usage represents the percentage of total CPU capacity that is utilized. Figure 5 shows CPU usage for the central controller measured using the *top* utility [19]. CPU usage is measured for 30 seconds interval and then averaged over multiple seeds. We measure CPU usage values for three different states. In the first state, no OpenFlow routers are connected to the controller. Thus, the CPU is occupied only to run the Floodlight [16] in the controller. When the OpenFlow WMN routers are connected to the controller with no rules (i.e., the second state), all traffic of the network are routed through the central controller and thus, CPU usage is increased drastically. On the other hand, when all fields are wildcarded, mesh routers become capable of routing traffic based on the packet information. Therefore, in this state, OpenFlow controller burden is reduced. We took the measurement of percentage CPU usage for full mesh and partial mesh topologies (see Figure 5).

CPU usage for the central controller and the router is highest (nearly 100%) when no rules are set to handle the transmitted OpenFlow packets. However, the CPU usage drops to nearly 50%–60% when all fields of OpenFlow routers are wildcarded to handle the packets. On the other hand, when no routers are connected to the central controller, CPU usage is around 30%–40% (for partial mesh topology, CPU usage is around 10%–20%), suggesting that the CPU load is caused by the controller program running in the machine. The heavy CPU usage indicates how the controller can be affected when it is flooded with OpenFlow messages.

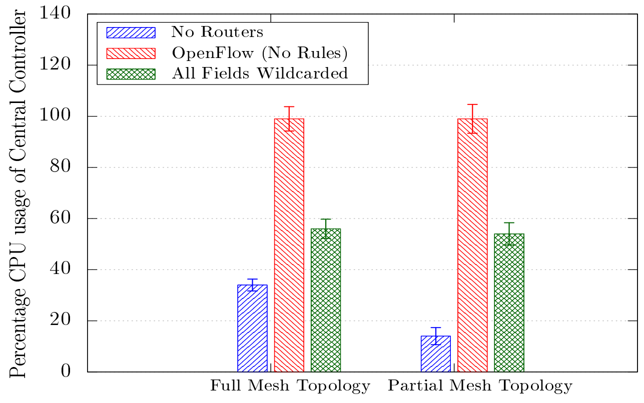

Fig. 5: CPU usage of OpenFlow controller with full mesh and partial mesh topologies.

E. *CPU Usage for OpenFlow Router*

Figure 6 depicts CPU usage of the routers for three states (e.g., disconnected state, OpenFlow with no rules, and all field are wildcarded) for full mesh as well as partial mesh topologies. CPU usage is monitored using the *top* utility [19] where measurements are taken for 30 seconds interval for each state and then averaged over multiple sessions. Even though there is an increase in CPU usage when flow rules are absent, the CPU usage remains low for the routers. OpenFlow routers are not affected with higher CPU usage on the absence of rules to handle incoming traffic.

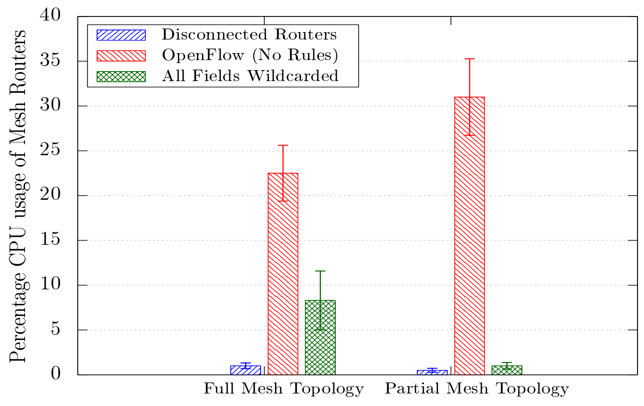

Fig. 6: CPU usage of mesh routers with full mesh and partial mesh topologies.

Data traffic in partial mesh topology is 40% less than that of full mesh topology. This reduction in incoming traffic is because of the weak network node inter-connectivity as compared to full mesh topology. Due to the reduced incoming traffic, CPU usage is significantly lower when partial mesh topology network is concerned.

F. *Control Traffic between OpenFlow Routers and Controller*

The control traffic between the OpenFlow routers and the controller is captured using *tcpdump* [18] at the controller. Figure 7 depicts the control traffic between the routers and the controller for full mesh topology. The control traffic is measured for various states of the controller. On connection establishment, the control traffic is enhanced when there is no rules to handle the incoming traffic. The peak control traffic is about 20-25Mbps. The control traffic is limited by the bandwidth available for the nodes.

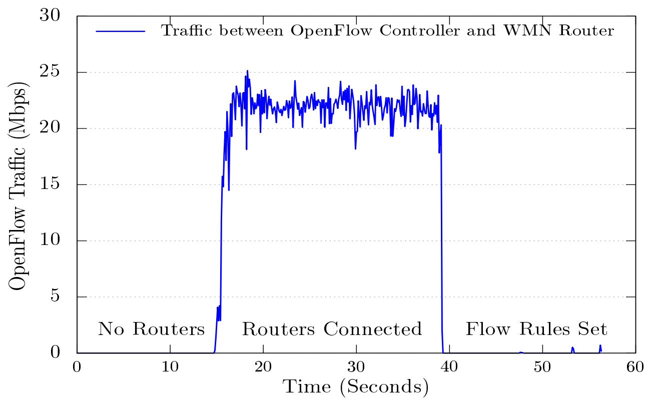

Fig. 7: Temporal behavior of traffic between OpenFlow controller and mesh routers in full mesh topology.

Figure 8 presents the control traffic between the routers and the controller for partial mesh topology. The traffic is approximately 15-20Mbps which is less than that of full mesh topology. The network throughput for partial mesh topology is less than full mesh topology, thus, reducing the available bandwidth for the control traffic.

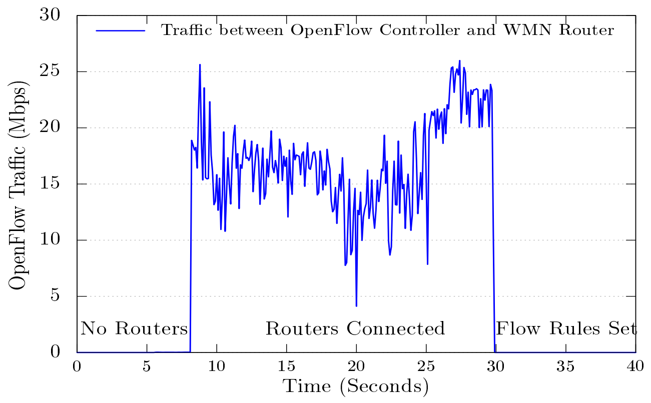

Fig. 8: Temporal behavior of traffic between OpenFlow controller and mesh routers in partial mesh topology.

G. *Overhead Issue: Layer-3 versus OpenFlow*

OLSR and OpenFlow overhead is measured by capturing the traffic at the central controller using *tcpdump* [18]. Packet capture files are examined to estimate the overhead of OpenFlow and OLSR traffic as follows. OLSR traffic is estimated at TCP port number 6633 and the overhead is estimated by counting the control packets (e.g., flooding, link-state update, "Hello", etc.) at that port. Similarly, OpenFlow traffic is measured at UDP port number 698 and then overhead is estimated by measuring various control packets at the port.

Figure 9 compares the traffic generated by OLSR and OpenFlow for full mesh and partial mesh topologies. The OpenFlow traffic is measured when there is rule to handle all incoming traffic. The traffic generated is minimal when compared to the link bandwidth. For partial mesh topology, the OpenFlow traffic is measured with a rule to handle all incoming packets. In case of the partial mesh architecture, there is an increase in the OpenFlow traffic (nearly 500Bps). The control traffic is minimal for both OLSR and OpenFlow with rules. The maximum overhead is around 900Bps by OpenFlow in partial mesh topology. The overhead barely affects available link bandwidth which is around 14Mbps for full mesh topology and 8Mbps for partial mesh topology.

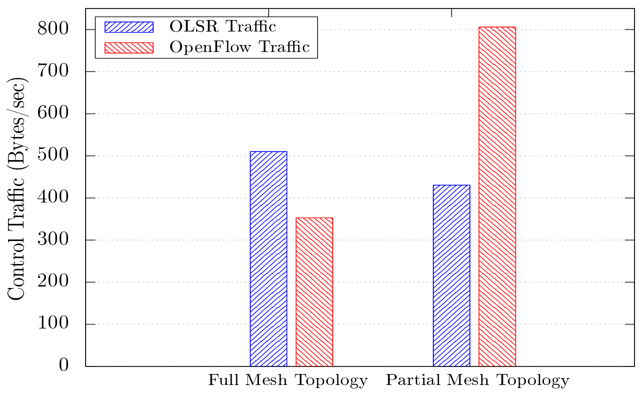

Fig. 9: OLSR and OpenFlow traffic between controller and routers in full and partial mesh topologies.

OpenFlow router with no rules to handle the traffic is placed at different locations in the mesh network. The router with no rules flood the network with OpenFlow control traffic. Moreover, it results in larger overhead due to multiple retransmissions required to forward the OpenFlow messages. The large amount of OpenFlow traffic generated negatively impacts the link bandwidth availability for the data traffic.

V. Observations and Discussion

We developed an OpenFlow based WMN testbed consisting of six WMN routers. Throughput, control overhead, and CPU usage measurements are carried out for full and partial mesh topologies using our testbed. In this Section, we analyze the experimental results obtained from Section IV.

The throughput performance of partial mesh topology based OpenFlow WMN is degraded approximately 10% with respect to full mesh topology WMN. Moreover, the control traffic overhead of partial mesh topology OpenFlow is higher than full mesh topology. The increased control traffic in partial mesh topology WMN restricts the ECC value for the network (ECC is approximately 7000 in our full mesh topology testbed).

When WMN routers are forwarding the traffic with no rules, percentage CPU usage of the OpenFlow central controller is maximum for full and partial mesh topology networks. The control traffic uses full link bandwidth for full mesh (about 20-25Mbps traffic) as well as partial mesh topology (about 15-20Mbps traffic) networks. For partial mesh topology, CPU usage for the router is significantly lower when flow rules are present to forward the packets.

The control traffic overhead of OLSR (nearly 400-500Bps) is minimal when compared to the data traffic in Layer-3 network. Furthermore, OLSR overhead for both partial and full mesh topology WMNs are comparable. On the other hand, OpenFlow control traffic overhead (approximately 800-900Bps) increases drastically in partial mesh topology WMN. Therefore, OLSR can be used for providing layer-3 infrastructure for OpenFlow WMN without affecting the link bandwidth availability for the OpenFlow control traffic.

VI. Conclusion

In this paper, we developed and deployed a testbed to study the performance of OpenFlow WMNs. The experiment was conducted for full mesh topology as well as partial mesh topology WMNs. We observed a marginal increase of the OpenFlow control traffic overhead (approximately 500Bps) in partial mesh topology WMN. This overhead restricts empirical controller capacity (ECC) of the partial OpenFlow WMN. We observed that ECC for the full mesh OpenFlow WMN is nearly 7000 in our testbed. However, throughput degradation of TCP over multi-hop scenario is still a bottleneck for the OpenFlow WMN implementation.

References

[1] I. Akyildiz and X. Wang, “Wireless mesh networks,” *John Wiley & Sons*, 2009.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling innovation in campus networks,” *SIKKIM Computer Communication Review*, vol. 38, no. 2, pp. 69-74, March 2008.

[3] K. K. Yap, M. Kobayashi, D. Underhill, S. Seetharaman, P. Kazemian, and N. McKeown, “The Stanford OpenRoads Deployment,” in *Proc. ACM 4th International Workshop on Experimental Evaluation and Characterization (WINTECH ’09)*, pp. 59-66, September 2009.

[4] FlowVisor, Website: https://openflow.stanford.edu/display/DOCS/Flowvisor.

[5] Y. Kanaumi, S. Saito, and E. Kawai, “Toward large-scale programmable networks: Lessons learned through the operation and management of a wide-area OpenFlow-based network,” in *Proc. IEEE International Conference on Network and Service Management (CNSM)*, pp. 330-333, October 2010.

[6] IEEE 802.1Q tunneling, Website: http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/12-2SX/configuration/guide /book/dot1qtnl.html.

[7] B. Heller, R. Sherwood, and N. McKeown, “The controller placement problem,” in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, pp. 7-12, August 2012.

[8] P. Dely, A. Kassler, and N. Bayer, “OpenFlow for wireless mesh networks,” in *Proc. IEEE 20th International Conference on Computer Communications and Networks (ICCCN)*, pp. 1-6, July 2011.

[9] P. Dely, “Towards an architecture for OpenFlow and wireless mesh networks,” in *Proc. Ofelia Summer School*, November 2011.

[10] J. Chung, G. Gonzalez, I. Armuelles, T. Robles, R. Alcarria, and A. Morales, “Characterizing the multimedia service capacity of wireless mesh networks for rural communities,” in *Proc. IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 628-635, October 2012.

[11] T. Clausen, P. Jacquet, C. Adjih, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L. Viennot, “Optimized link state routing protocol (OLSR),” *rfc. 3636, Project Hipercom, INRIA*, October 2003.

[12] PC Engines alix3d3 system board, Website: http://www.pcengines.ch /alix3d3.htm.

[13] VoyageLinux, Website: http://www.linux.voyage.hk.

[14] Openflow, Website: http://www.openflow.org.

[15] Universal Tun/Tap device driver, Website: https://www.kernel.org/doc /Documentation/networking/tuntap.txt.

[16] ProjectFloodlight, Website: http://www.projectfloodlight.org.

[17] Iperf, Website: http://www.ietf/iperf.

[18] Tcpdump, Website: http://www.ietf/tcpdump.

[19] Top utility, Website: http://www.unixtop.org.