Managing Wireless Fog Networks using Software-Defined Networking

Akram Hakiri, Bassem Sellami, Prithviraj Patil, Pascal Berthou, Aniruddha Gokhale

Managing Wireless Fog Networks using Software-Defined Networking

Akram Hakiri*, Bassem Sellami*, Prithviraj Patil†, Pascal Berthou‡ and Aniruddha Gokhale§
*University of Carthage, SYSCOM ENIT, ISSAT Mateur, Tunisia.
†The MathWorks Inc, Natick, MA, USA.
‡CNRS, LAAS, UPS, 7 Avenue du colonel Roche, F-31400 Toulouse, France.
§ISIS, Dept of EECS, Vanderbilt University, Nashville, TN, USA.
Corresponding author: akram.hakiri@gmail.com

***Abstract*—Fog computing has recently emerged as a new cyber foraging technique to offload resource-intensive tasks from mobile devices to mobile cloudlets in close proximity to end-users. Since the one-hop communication in the network edge is predominantly wireless, Wireless Mesh Networks (WMNs) are being considered to build wireless fog networks. However, WMNs use distributed hop-by-hop routing protocols to reflect a partial visibility of the network, which limits their ability to perform global network management and monitoring needed by fog networks. Software Defined Networking (SDN) provides a centralized control and management of the entire network, which makes it a good candidate to support fog communication. Unfortunately, the SDN OpenFlow protocol does not support any functionalities for wireless fog networks as it is primarily targeted to wired networks. To address these issues, this paper presents a SDN-enabled wireless fog architecture that combines both OpenFlow and distributed wireless protocols. The proposed solution provides lower latency and efficient load balancing to offload the network load by enabling programmable fog routers.**

***Index Terms*—Wireless Fog Networks, SDN, Hybrid Control Plane, Hybrid Routing, Load Balancing.**

I. Introduction

Fog Computing [1] is an emerging technology to bring cloud computing services, such as communication, computation, network control and storage, to the network edge. Fog services can be hosted at users' edge wireless devices to improve network reliability and latency, and overcome the issues stemming from geographically distributed locations in cloud computing. Additionally, fog computing relies on discoverable, generic, forward-deployed servers located in single-hop proximity of mobile devices [2]. These servers should be used to offload expensive computation at the network edge, perform data filtering to remove unnecessary data from streams intended for dismounted users, and serve as collection points for data heading for enterprise repositories. Obviously, the number of users in the fog network is typically bounded since we can estimate in advance what the size of the network is, how big a group of wireless users, similar to traditional wireless access points in public environment such as airports, cyber-coffee, etc.

An additional trend reveals that wireless fog devices are increasingly used locally, e.g., for intra-vehicle communication [3], intra-sensor communication [4] and smart energy management in intra-building [5], where data are generated and consumed locally. To make such and future solutions, fog computing architectures should enable real-time data sharing across a range of platform such as mobile severs and embedded sensor, wirelessly. In this setup, each fog node is envisioned to act as a wireless router to its neighbors to provide a resilient network with high capacity data transfer, fault tolerance as well as higher availability. Meanwhile, Software Defined Network (SDN) [6] has been envisioned as a new approach to enable network programmability to test out new protocols. SDN allows creating modular and declarative programming interfaces across the wireless stack by refactoring the wireless protocols into processing and decision planes. On one hand, an external SDN controller that holds all the routing intelligence is used to centralize the management and the control of the network. On the other hand, the hardware pipelines in the traditional routers are replaced by software pipelines holding abstracted flow tables. The controller can program those flow tables to enable a customized and fine-grained control of the traffic.

So far, the above two technologies, i.e. Fog computing and SDN, have been studies separately. The wireless SDN approaches predominantly focus on using only OpenFlow to interconnect virtual public mesh network [7] and offloading the router's computation to the cloud [8]. Furthermore, SDN has been used to maintain the session continuity from the application perspective, ensure mobility management [9], and maintain the network connectivity [10]. Similarly, fog computing technologies are used to extend the capabilities of mobile devices by enabling real-time analytics and performing computational functions at the edge of a network. For example, Datta et al. [11] introduced a Fog computing architecture for connected vehicles to optimize data dissemination over single-hop communication. Hong et al. [12] proposed a high level programming model for data staging in geographically distributed mobile devices. Elgendi et al. [13] proposed a SDN architecture to interconnect distributed fog cloudlets.

Despite the promise, all these efforts used a centralized SDN control plane to manage isolated islands of fog computing infrastructure, which may result in adverse consequences to the reliability and performance due to increased traffic and can become a single point of failure. Conversely, a distributed control plane is more responsive to handle network events

because its proximity to the fog island. However, managing multiple independent distributed controllers can become a hard task as i) it incur a different set of complexities to provide a global optimal view of the whole network; and ii) developers should take care of all the concerns that arise out of distributed nature of the system including controller synchronization, controller replication, controller logic partitioning and controller placement. Furthermore, the aforementioned efforts use OpenFlow protocol to statically establishes paths to every SDN switch so that it can run centralized routing algorithms, which is orthogonal to the fundamental distributed wireless routing functionality. Despite some wireless routing protocols (e.g. AODV [14], OLSR [15], etc.) can be used to perform distributed routing, they however incur significant issue: their routing decisions are taken based on local knowledge of each router's neighbors. Thus, this neighborhood reflects only a partial visibility of the network without providing a global view of the entire wireless backhaul. In other words, this local visibility contracts the SDN requirements for centralized management of the network, as it limits the ability to perform network traffic engineering tasks and best path selection across the available SDN-enabled Fog computing devices.

Consequently, we need an intelligent approach that can i) provide an hybrid control plane across the distributed Fog computing devices, ii) exploits the advantages of existing routing protocols to perform flexible and fine-grained flow control across the wireless fog infrastructure, and iii) selecting the best forwarding path that increases the radio channel transmission capacity. To realize such a capability, we present in this paper an integrated architecture for software-defined networking and virtualization for wireless fog computing. The main contribution of this paper are:

- We present a novel architecture to mange wireless fog infrastructure using a hybrid SDN control plane to perform a flexible deployment and management.
- We propose a hybrid SDN routing protocol that combines the OLSR data forwarding and OpenFlow to perform global and optimal path selection as well as monitoring the entire network.
- We propose a network traffic engineering approach to perform load balancing for offloading fog devices in the edge network. It also allows transmitting the Signal-to-Noise Ratio (SNR) to the controller to perform best path selection based on the highest SNR values.
- We have implemented our solution on a SDN emulation testbed and evaluated our approach for various QoS metrics like latency, router overhead, and bandwidth utilization.

The remainder of this paper is organized as follows: Section II details the design rationale and the implementation for our architectural decisions; Section III evaluates our solution to validate its claim about flexible data delivery and low-latency communication; Section IV compares our approach with the related work; and finally Section V presents concluding remarks alluding to lessons learned and future work.

II. Enabling SDN for Wireless Fog Networks

In this section we describe the design principles and details of our SDN-enabled solution for managing fog computing infrastructure.

A. Architecture

Figure 1 depicts the architecture of the SDN-enabled Wireless Fog Network. At the core of this design is the SDN controller, i.e., the control plane, which communicates with the underlying fog routers using the OpenFlow protocol. The controller includes several network modules:

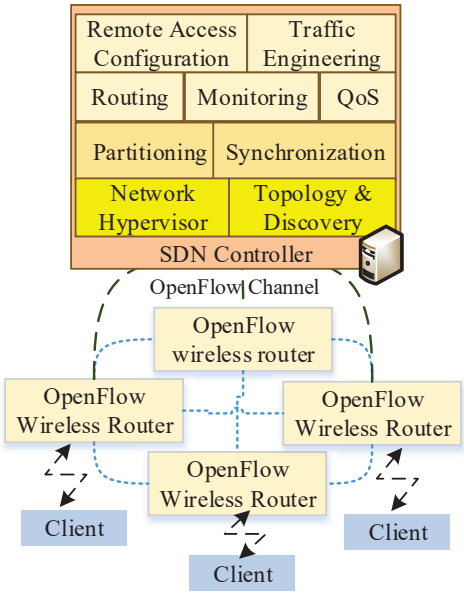

Fig. 1: Architecture of the SDN-enabled Wireless Fog Network

- **Routing Module:** which implements the shortest path algorithm to build optimal routing strategy to route packets across the fog 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 fog router to gather individual statistics.
- **Traffic engineering module:** which supports load balancing to offload fog devices in case of traffic congestion. It also performs traffic redirection based on the optimized routing strategy used in the routing module.
- **Partitioning Module:** is responsible for slicing the control plane logic. This module requires the controller to expose an API to perform flow slicing, i.e, separating data flow, control flow and meta-controller traffic.
- **Host Remote Access:** allows the access to remote hosts to install or initialize remote distributed controllers. It provides new API to the controller by combining SSH and SCP through the Python command line tool Fabric.

- **Synchronization Module:** it used to specify the synchronization mechanism to be used in the control plane, e.g., how to synchronize the backup controller, i.e. using Apache Zookeeper.
- **Topology & Discovery module:** which uses the Link Layer Discovery Protocol (LLDP) to perform automatic discovery of joining and leaving fog routers. The controller broadcasts OpenFlow *PACKET_OUT* messages to all connected routers, which in turn respond by sending ARP messages to notify their liveliness.
- **Network Hypervisor:** it provides the access to the underlying network hypervisors, which are used to slice the network into control and data slice based on OpenVirtex [16].

On the data plane, each fog 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, etc.*, 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 3. Using virtual radio interfaces allows efficient sharing of the downlink bandwidth between multiple fog clients and airtime fairness scheduling with the help of channel sharing.

B. Hybrid SDN Control Plane

To address the limitation of both the centralized and distributed SDN controllers, we introduced a hybrid control plane insofar as it helps to decouple the orthogonal distributed systems concerns from the primary issues related to the controller. The hybrid SDN approach, depicted in Figure 2, is designed to make SDN more flexible, reliable, fault-tolerant without adding complexity to the controllers. Specifically, we introduces a meta-controller layer based on the bootstrapping mechanism adopted by operating systems, so that a centralized controller is deployed at the initialization phase to control and manage the entire network. Then, in case of a controller failure or overhead, additional controllers are added at runtime as required to balance the network performance.

Indeed, our hybrid approach divides a single network into two logical slides: a control slice that contains one or more distributed SDN controllers; and ii) a data slice that contains other clients and routers. Based on the network requirements, i.e. traffic load and the availability of the other controllers, it allocates the right number of clients and routers to each controller, while offering a set of coordination mechanisms to ensure the network consistency. In particular, these mechanisms include an election process that allows electing the closest controller as a master. Furthermore, this hybrid approach can increase or decrease the size of the control plane, change

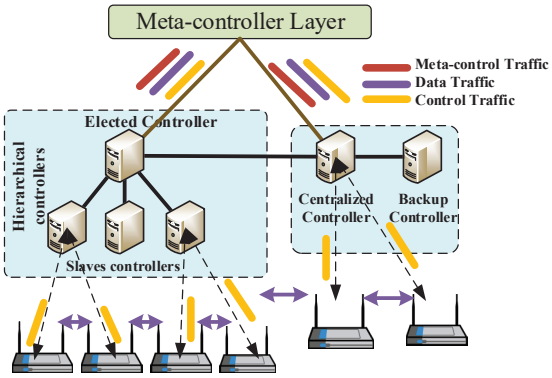

Fig. 2: Hybrid Control Plane

the coordination mechanism among the controllers, i.e. using Apache Zookeeper to adapt to network topology changes or to dynamic network loads or simply as part of an upgrade.

In the large scale wireless fog infrastructure, SDN will need to provide services for creating virtual SDNs (vSDN). In such environments, individual vSDN will be managed by different independent controllers and may require different control plane topology. For example, client *A* and client *B* lease the one vSDN each from the SDN service provider *C*. *C* hosts both the vSDN on the same physical network hardware. The client *A* wants hierarchical control-plane with POX controllers, while B wants centralized control-plane with RYU controller. We can accomplish such requirements easily as follows:

1) *Case when vSDN is created before SDN bootstraps i.e. statically.* A network hypervisor first creates two vSDNs on the data slice. It will then inform the meta-control layer about switches and hosts used to create these two vSDNs. It will also provide control-plane topology requirements of these two vSDNs. The, the meta-control layer can calculate the total number of controllers needed to satisfy requirements of both vSDN. This calculation depends upon the switches used by the both vSDNs. It can optimize the number of hosts required for control plane based on switch sharing between two, for example if vSDNs share a switch between them, e.g. port 1 of switch S1 goes to first vSDN and port 2 of the same switch goes to other vSDN etc.
2) *case when vSDN is created dynamically i.e. after the SDN is booted.* In this case, similar approach is used, however the meta-control layer needs to use the existing control plane or scale up if required. This decision is based on whether control-plane requirement of new vSDN is met by the existing control-plane. In this way, the meta-control layer provides more flexibility to the SDN control plane management.

C. The Routing Approach

The routing approach is shown in Figure 3 and is divided into two sub logical layers: traditional OpenFlow-enabled SDN data forwarding at the upper sublayer and the OLSR routing protocol at the bottom layer. The former is responsible for communicating OpenFlow policies with the controller in the

upper layer. The latter is responsible for handling IP routing among the wireless interfaces. We use an in-band approach to forward signaling packets across different fog routers. This design decision is motivated by the desire to provide long distance wireless connectivity among the wireless backhaul. Additionally, the controller can implement its own routing algorithms to select the best paths that packets should follow. The advantages of this composite architecture are twofold: (i) the IP forwarding using OLSR allows reporting of every changes in the wireless routers' topology graph, such as addition/removal of a fog router and/or wireless link; and (ii) OLSR routing increases the availability and dependability of the fog network since even if a SDN controller fails or becomes unavailable, the IP routing continues to manage the network. Moreover, packets can be routed according to OLSR routing tables under the instruction of the SDN controller by using OpenFlow. The controller configures the wireless

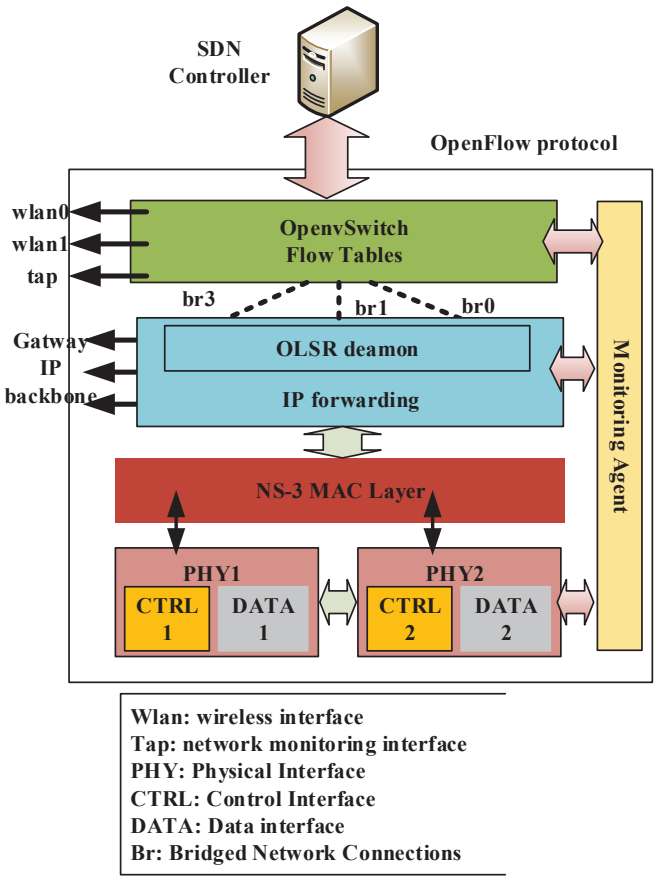

Fig. 3: Hybrid Routing Algorithm

fog routers by adding, removing, and/or updating OpenFlow rules and retrieves the current network states from the nearest wireless router. Since OLSR is a proactive routing protocol, each fog router acquires the topology information from other nodes using multi-points relays (MPRs). The MPRs are used to reduce the network overhead and provide the shortest path routes for all fog routers selected as a destination in the network. Each router keeps a list of its neighbors, which are selected by the MPR using the MPR-selector list. Each node can periodically refresh its routing table, i.e., after exchanging *HELLO* messages periodically with other neighbors, and selecting the new shortest path to all destinations. Subsequently, the SDN controller can retrieve topology information from its nearby router using the topology discovery service implemented in the controller.

D. Monitoring Fog Devices with OpenFlow

After the startup phase, a client sends a request for a service from the remote fog server, but there no end-to-end connection between them. As depicted in circle ② of Figure 4, the request is redirected by the fog router towards the controller using *PACKET_IN* message. Then, the controller responds to this request to establish the path for data transfer between the client and the server. The controller first examines the packet header and checks whether a new flow entry needs to be created and new actions should be applied on these packets. The controller sends a *Flow_Mod* message (③ in Figure 4) to the fog router that includes the new actions to perform on the packets belonging to this entry. Each flow entry contains a set of instructions to apply immediately to the packet or forward them to the next match table in the pipeline. Each pipeline consists of multiple OpenFlow tables which in turn contain multiple flow entries. A "goto" instruction in the pipeline indicates the next table where the lookup object is found and match actions are performed at each stage.

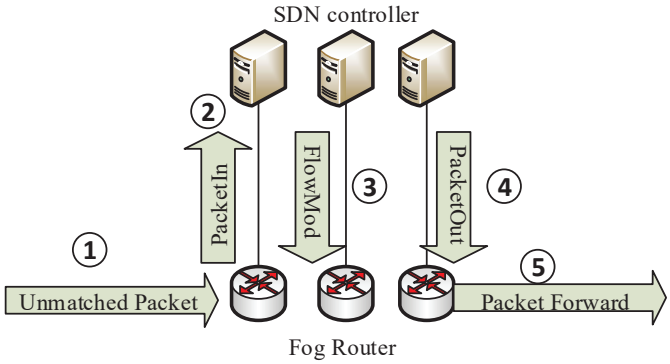

Fig. 4: Installation Procedure of a new flow

Meanwhile, the controller injects *PACKET_OUT* command messages (④ in Figure 4) into the data plane of the fog router. *PACKET_OUT* command messages are not processed the same way as packets that arrive on standard ports. These packets jump to the action sets application and instructions are checked to determine how a packet and its associated data will be processed. All subsequent packets in both directions, i.e., between the client and the server, are matched at the MAC Ethernet port. At the client side, the data received from the server are matched against the IP source-destination addresses and ports.

E. Traffic Engineering

To illustrate traffic engineering, consider a scenario from Figure 5 depicting four wireless fog routers connected in mesh

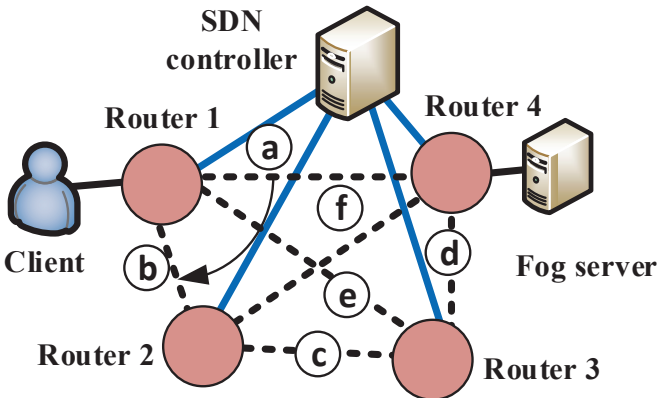

Fig. 5: Load Balancing Scenario

topology. Links *a*, *b*, *c*, *d*, *e*, and *f* establish the communication paths across all the wireless fog routers.

Algorithm 1 shows the load balancing algorithm to select the optimal path between the client and the fog server in Figure 5. Assume that the first optimal path is across the wireless link *a* between *router 1* and *router 4*. The controller has already installed the default parameters to establish the path *a*, and periodically discovers the link states. Once the bandwidth utilization reaches the threshold "Th" and the client experiences traffic congestion, the load balancing algorithm detects the network bottleneck and starts calculating new OpenFlow rules to reroute the traffic across new links, i.e., links *b, f* or *e, d*. Thereafter, the controller floods all the ports (i.e., using OpenFlow FlowMod packets) towards the selected fog routers in the path to the server. 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.

Algorithm 1: Load Balancing Algorithm

```
Data: Th, ℜ, ℵ
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
```

Furthermore, we modified the MAC layer of the wireless devices to allow them sending information about the interference towards the controller. Algorithm 2 depicts the SNR-based route optimization approach, which calculates the best path that has the highest SNR ratio. In particular, rather than storing the SNR values in the MAC layer, we have modified the "Maclow" and "MacRxMiddle" modules to allow them forwarding the SDN radios to the "StaWifiMac" module of each wireless fog router. Such an approach allows centralizing information about the interference in the SDN controller side, by modifying the "WifiNetDevice" and "TabBridge" modules in the SDN emulator.

Algorithm 2: SNR-based routing optimization algorithm

```
Data: SNR
Result: Best_SNR_Path(ℜ, ℵ);
if ∃ path (SNR) then
    path ⟵ Find_Best_SNR (rules)
    return path
else
    rules ⟵ calc_new_rules(SNR);
    FlowMod_router();
    best_path(rules);
end
```

Table I depicts the flow entries 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.

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

TABLE I: OpenFlow entries the controller installs inside the fog routers

Upon the failure of radio link *a*, the controller installs new OpenFlow rules to redirect the flow from router *1* to router *4* through router *2* with ID *dpID2*. Since the new available forwarding path should pass through router 2, the controller should program both routers 2 and 4 with the new flow entries as described in the "After" column of Table I. Since the controller have a centralized view of the network, he can easily decide the network bottleneck and switch data to the best available new path. To do so, the controller calculates the new rules, i.e., the MAC and IP addresses for the new fog routers in the new path and send OpenFlow FlowMod messages to select the new end-to-end path. Thereafter, it floods all the ports towards the selected fog routers, open the TCP connection to allow fog clients reach each other's, while it continue performing node discovery for monitoring the network topology.

III. Performance Evaluation

In this section we show results of experiments that validate our claims.

A. Experimental Setup

To evaluate the performance of our approach, we implemented a software integration layer that combines Mininet [17], the reference SDN emulator, along with the NS3 [18] simulator. The main reason behind developing the integration layer is to complete the lack of the wireless routing capabilities in Mininet, which are only supported by NS3. Conversely, NS3 does not support any feature for emulating SDN networks natively. This integration layer uses the NS3 *TapBridge* functionality to integrate real wireless fog infrastructure nodes in the SDN environment. Additionally, the integration layer offers new capabilities to integrate the SDN controllers along with the OpenFlow-enabled switch, i.e., the OpenVSwitch [19].

At the controller side, we enhanced the Ryu [20] SDN controller. As described earlier in Section II-A, we developed our networking modules with Ryu to perform network monitoring and debugging, discovering the topology changes in the network, recovering the network from failure as well as traffic engineering throughout the load balancing module.

B. Evaluating the Network Latency

We consider the Round Trip Time (RTT) as the time taken by a data packet to be sent from a client to the fog server and the time it takes to be received back by that client. We report the RTT measurement in Figure 6 after conducting the experiments multiple times. At the startup phase, the controller is listening to OpenFlow PACKET_IN messages from the router to learn about the MAC and IP addresses of the incoming packets. Then, it sends PACKET_OUT messages to open the path for those packets. The controller-router delay during this procedure is close to 10 milliseconds. Once the setup phase is finished and the traffic is being sent to the destination, i.e., the controller has already installed OpenFlow rules into the router, the controller latency decreases to about 3 milliseconds. The controller continues performing topology discovery using the LLDP discovery protocol. Hence, only the OpenFlow *keep-alive* messages are exchanged to check whether an idle connection occurs to indicate a loss of controller-router connectivity.

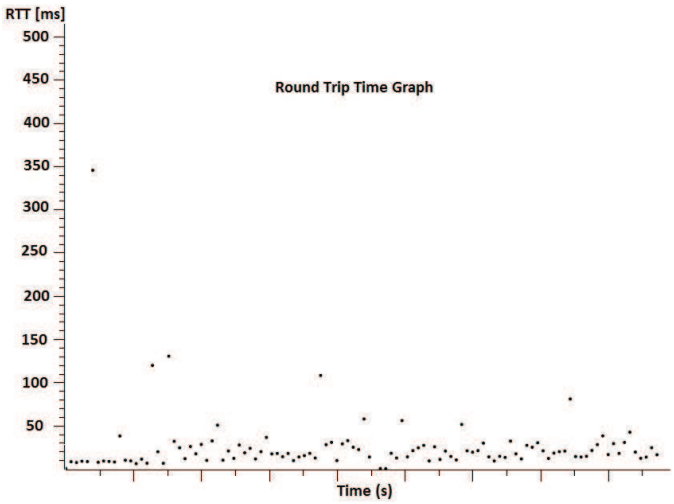

Fig. 6: Round Trip Time Between Client and a Fog Server

Once a new client joins the network, its forwarding rules are yet unknown to the router. The router blocks the traffic sent by that new client and queries the controller about the new rules to apply for the incoming packets. Thereafter, the end-to-end latency becomes close to 20ms. This latency does not depend on increasing the number of clients, but due mainly to the time required by the router to negotiate and process the new OpenFlow rules with the controller.

These results underscore another benefit of our hybrid routing approach: additional network processing delays are not incurred since OpenFlow messages sent by the controller to install new routing rules do not affect the performance of the communication.

C. Evaluating the Router Performance

Figure 7 depicts the traffic rates generated by OLSR and OpenFlow. At startup, the OpenFlow traffic is close to 2KB/s. This is because the traffic exchanged includes the discovery messages. At time 32 seconds, new OpenFlow messages are exchanged between the fog router and the controller. Those messages include new OpenFlow rules the controller has to install for programming the flow entries in the fog router. Hence, the OpenFlow traffic increases during this phase. Meanwhile, the OLSR traffic remains constant as it does not involve any message exchange with the controller. All the fog routers exchange their routing information including the routing tables and the neighbors tables. After programming the flow entries that finished at time 46s, the OpenFlow traffic decreases and the controller continues to listen to the LLDP packets sent by the fog routers. That is, our hybrid routing appraoch that uses OLSR for data forwarding performs lower bandwidth utilization compared to the traditional OpenFlow

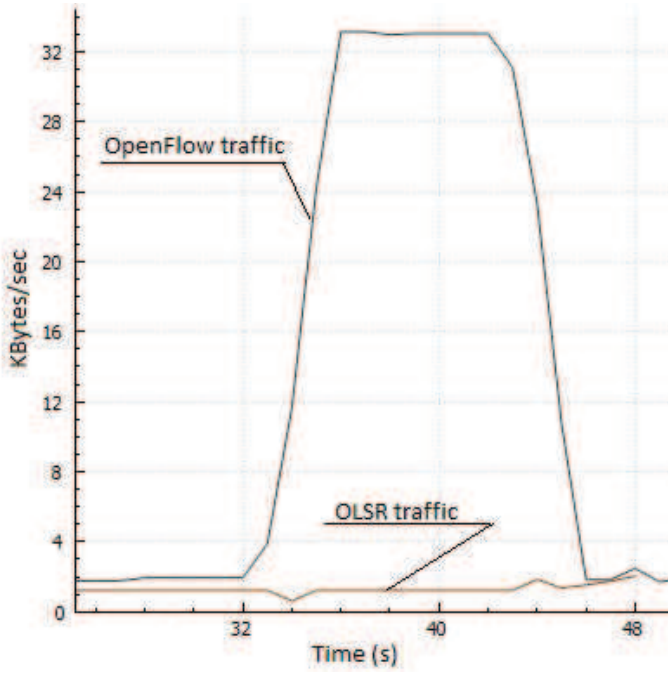

Fig. 7: Hybrid Routing Performance

only routing, which is used to exchange data between the SDN controller and the SDN-enabled fog router. Thus, our hybrid routing appraoch does not introduce a network overhead in the fog routers.

D. Evaluating the Load Balancing

To evaluate the performance of the load balancing approach, we inject a competing flow into router 4 to introduce network congestion and introduce a performance degradation in this node. Figure 8 shows the throughout observed in router 4. Due to buffer overflow, router 4 starts dropping packets and throughput decreases from 850 kB/s to 200 kB/s and a significant packet loss is observed as foreseen by our experiments.

At time 50 seconds, the load balancing algorithm at the controller, which we programmed to be activated when the bandwidth becomes 200 kB/s, is activated to redirect the traffic from radio link *a* to radio links *b* and *f* of Figure 5. The topology discovery module at the controller discovers 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 the destination. The new path is extracted from the routing table that is updated regularly by 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 installing 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 8 shows that the controller is able to make traffic adjustment using the load balancing algorithm. The traffic is balanced among the new wireless

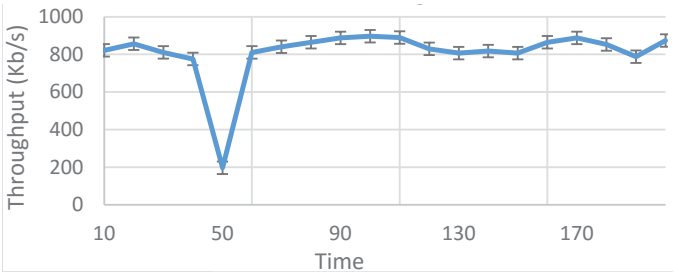

Fig. 8: TCP Throughput

links after establishing the new data path. The time delay required by the controller for deciding the new available path and forwarding data is close to 10ms. 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 show that our approach to provide traffic engineering in wireless fog networks succeeds in redirecting packets to the new selected path when multiple wireless hops are available in the fog network.

IV. Related Work

During the past few years, SDN has received unprecedented attention from the research community for developing network support for cloud and edge networks. The authors in [21] proposed using SDN to improve data transfers between mobile clients and the cloud, by temporarily staging data and transit them on intermediate Fog computing infrastructure. Likewise, Liang et al. [22] introduced OpenPipe framework that helps in virtualizing the radio access with fog computing. Sun et al. [23] introduced the edgeIoT framework which uses a centralized SDN controller to facilitate packet forwarding among fog nodes. He et al. [24] used SDN for the Internet of Vehicles (IoV) in which fog computing infrastructure is located the road size to perform vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), vehicle-to-base station communications. Lazar et al. [25] and Truong [26] concluded that both SDN and Fog computing wan work together to accommodate a large variety of vehicular networks and the Internet of Vehicles (IoV) as well. Besides, Authors in [27] modified the SDN switch code to integrate a fog computing prototype and a built-in controller that leverages the MQTT middleware to interconnect distributed fog nodes.

Similarly, Bruschi et al. introduced in [28] a SDN-enabled virtualization platform called OpenVolcano that exploits in-network programmability capabilities to operate inside fog environments in a close proximity to end users. OpenVolcano allows infrastructure virtualization and computation offloading, data staging and forwarding as well as preforming QoS/QoE provisioning and energy efficiency. Betzler et al. [29] proposes a novel path forwarding scheme based on SDN with wireless back-hauling and edge computing capabilities, which achieves low balancing and external interference mitigation. Huang et al. [30] studied the QoS provisioning of wireless sensor fog devices. A fog layer is content-aware as it can collects data from packet headers as well as the content of the sensing data

to allow a centralized SDN controller to perform define fine-grained QoS provisioning policies.

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.

V. Conclusion

In this paper we presented a SDN-enabled solution to manage wireless fog networks by blending the OpenFlow and IP forwarding protocols. Our approach provides a flexible and programmable wireless data plane for fog networks as well as intelligent traffic engineering to offload the network. The performance evaluation shows the efficiency of the proposed solution to perform lower-latency communication, a flexible load balancing to select the optimal shortest path, and a lower network overhead.

Our future work will focus on unifying the radio resource management and the network resource management by designing a cross-layer architecture that can interoperate between SDN and Software Defined Radio (SDR) for better spectrum utilization and channel interactions.

Acknowledgments

This work was partially funded by the Fulbright Visiting Scholars Program, the NSF CNS US Ignite 1531079 and the French National Research Agency (ANR), the French Defense Agency (DGA) under the project ANR DGA ADN (ANR-13-ASTR-0024) and the French Space Agency (CNES). 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 DGA, CNES or NSF.

References

[1] I. Stojmenovic and S. Wen, "The fog computing paradigm: Scenarios and security issues," in *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*, Sept 2014, pp. 1–8.

[2] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*, ser. Mobidata '15, 2015, pp. 37–42.

[3] K. Kai, W. Cong, and L. Tao, "Fog computing for vehicular ad-hoc networks: paradigms, scenarios, and issues," *The Journal of China Universities of Posts and Telecommunications*, vol. 23, no. 2, pp. 56 – 96, 2016.

[4] S. Ivanov, S. Balasubramaniam, D. Botvich, and O. B. Akan, "Gravity gradient routing for information delivery in fog wireless sensor networks," *Ad Hoc Networks*, vol. 46, pp. 61 – 74, 2016.

[5] M. A. A. Faruque and K. Vatanparvar, "Energy management-as-a-service over fog computing platform," *IEEE Internet of Things Journal*, vol. 3, no. 2, pp. 161–169, 2016.

[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] I. Ahmed, A. Mohammed, and H. Alnuweiri, "On the fairness of resource allocation in wireless mesh networks: a survey," *Wirel. Netw.*, vol. 19, no. 6, pp. 1451–1468, Aug. 2013.

[8] 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.

[9] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao, "Towards programmable enterprise wlans with odin," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12, 2012, pp. 115–120.

[10] 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.

[11] S. Datta, C. Bonnet, and J. Haerri, "Fog computing architecture to enable consumer centric internet of things services," in *Consumer Electronics (ISCE), 2015 IEEE International Symposium on*, June 2015, pp. 1–2.

[12] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, "Mobile fog: A programming model for large-scale applications on the internet of things," in *Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing*, ser. MCC '13, 2013, pp. 15–20.

[13] I. Elgendi, K. S. Munasinghe, and B. Mcgrath, "A heterogeneous software defined networking architecture for the tactical edge," in *2016 Military Communications and Information Systems Conference (MilCIS)*, 2016, pp. 1–7.

[14] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," Jul. 2003.

[15] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," Oct. 2003.

[16] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, W. Snow, and G. Parulkar, "Openvirtex: A network hypervisor," *Open Networking Summit*, 2014.

[17] 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.

[18] 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.

[19] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'15, 2015.

[20] R. project team, *RYU SDN FRAMEWORK*. [Online]. Available: http://osrg.github.io/ryu/resources.html#books

[21] 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.

[22] K. Liang, L. Zhao, X. Chu, and H. H. Chen, "An integrated architecture for software defined and virtualized radio access networks with fog computing," *IEEE Network*, vol. 31, no. 1, pp. 80–87, 2017.

[23] X. Sun and N. Ansari, "Edgeiot: Mobile edge computing for the internet of things," *IEEE Communications Magazine*, vol. 54, no. 12, pp. 22–29, 2016.

[24] X. He, Z. Ren, C. Shi, and J. Fang, "A novel load balancing strategy of software-defined cloud/fog networking in the internet of vehicles," *China Communications*, vol. 13, no. Supplement2, pp. 140–149, 2016.

[25] S. A. Lazar and C. E. Stefan, "Future vehicular networks: What control technologies?" in *2016 International Conference on Communications (COMM)*, 2016, pp. 337–340.

[26] 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)*, 2015, pp. 1202–1207.

[27] Y. Xu, V. Mahendran, and S. Radhakrishnan, "Towards sdn-based fog computing: Mqtt broker virtualization for effective and reliable delivery," in *2016 8th International Conference on Communication Systems and Networks (COMSNETS)*, 2016, pp. 1–6.

[28] R. Bruschi, P. Lago, G. Lamanna, C. Lombardo, and S. Mangialardi, "Openvolcano: An open-source software platform for fog computing," in *2016 28th International Teletraffic Congress (ITC 28)*, vol. 02, 2016, pp. 22–27.

[29] A. Betzler, F. Quer, D. Camps-Mur, I. Demirkol, and E. Garcia-Villegas, "On the benefits of wireless sdn in networks of constrained edge devices," in *2016 European Conference on Networks and Communications (EuCNC)*, 2016, pp. 37–41.

[30] L. Huang, G. Li, J. Wu, L. Li, J. Li, and R. Morello, "Software-defined qos provisioning for fog computing advanced wireless sensor networks," in *2016 IEEE SENSORS*, 2016, pp. 1–3.