Multi Controllers Architecture with Adaptive Monitoring in SDWMN In-Band

Bruno Ramos e Silva[1], **Madson R. Araujo**[1], **Ibirisol F. Ferreira**[1], **Gustavo B. Figueiredo**[1]

[1]Institute of Mathematics and Statistics – Federal University of Bahia (UFBA)
Av. Adhemar de Barros, s/n, Ondina – 40170-115 – Salvador – BA – Brasil

{brunoramos,madsonra,ibirisol,gustavo}@dcc.ufba.br

Abstract. *Software Defined Wireless Mesh Networks (SDWMN) enable more intelligent and programmable wireless networks. SDWMN with in-band control plane can optimize resource utilization when compared to out-of-band network architectures, because in-band can exempt the need of additional physical or virtual interfaces and legacy protocols to coordinate the control plane traffic. This traffic can also be engineered by SDN and make use of its possible benefits. However, there are many challenges to be dealt with SDWMN in-band, such as network auto-configuration during start-up, scalability and control plane overhead over the data plane. To overcome these limitations, we propose in this paper a multi controllers architecture with adaptive monitoring in SDWMN in-band which objective is to reduce control plane overhead and delay while improving network scalability. The work is implemented as a proof-of-concept in the OpenWiMesh framework. Results show it effectively reduce switch-controller latency and control plane overhead, as well as enabling higher scalability. We also show our monitoring solution enables better wireless network view and uses at least near half the bandwidth compared to a SNMP solution.*

1. Introduction

A Wireless Mesh Network (WMN) is a multi-hop wireless network with decentralized architecture, where each node can either communicate directly to other nodes or through intermediary nodes. These networks are highly tolerant to failures, being able to self-organize and self-configure themselves. In addition, WMNs have qualities such as ease of installation and maintenance, scalability, reliability and low cost. Therefore, they are widely used in community networks, disaster and emergency scenarios, home networks or extending corporate networks [Chung et al. 2013, Brito et al. 2014].

The Software Defined Networks (SDN) paradigm allows networks to become more intelligent and programmable. It separates control and data planes in network devices, proposing a centralized architecture to control packet forwarding. Such architecture greatly simplifies the network management since the control of the network is centralized. In Software Defined Wireless Mesh Networks (SDWMN), in-band control plane signaling is preferable over an out-of-band one since the former incurs on lower deploying costs by using the same network resources for control and data traffic [Brito et al. 2014].

In contrast, out-of-band control plane signaling requires a secondary network (virtual or physical) to transmit the control plane traffic, thus increasing network costs.

However, utilizing SDN in WMNs brings many challenges. Some of them are related to scalability and performance of the control plane, especially as the network grows in size and traffic. This way, multiple controllers become a natural strategy to approach both challenges of SDN control plane. Different approaches of multiple controllers are used to tackle scalability, performance and availability problems [Koponen et al. 2010, Tootoonchian and Ganjali 2010, Hassas Yeganeh and Ganjali 2012, Bari et al. 2013, Berde et al. 2014, Mattos et al. 2015], but they do not focus on wireless networks. Once WMN are sensible to quality and performance variations, these networks demand a resilient control plane and show strict requirements when compared to wired networks, specially in logically centralized networks. For example, these works do not take into account wireless in-band control traffic impact over the data plane traffic, as well as characteristics inherent to wireless environment, such as interference and others. Thus, it is desirable to have SDN controllers developed specifically for WMN when managing these networks.

Additionally, network monitoring is important to keep global network view consistent and to choose when and how to distribute the control plane. Monitoring metrics can be used for many network tasks, such as statistics plotting, events and alerts generation, traffic pattern analysis and as input for control plane applications and traffic engineering algorithms. Thus, most SDN controllers should require control and data planes monitoring system. Considering SDN controllers for wireless networks, the metrics gathered should be related to those networks. Standard SNMP and OpenFlow are not well suited for specific wireless data collection according to [Duarte et al. 2007, Nascimento et al. 2014], so custom agents are often required. From the best of our knowledge there are works, such as [Dely et al. 2011, Nascimento et al. 2014, Kim et al. 2016], closely related to, although they do not fully wrap together distributed controllers, wireless and monitoring.

Monitoring requires information sharing that generates traffic and processing costs which can be too high if not well balanced between precision and resource costs. Coupled with the dynamic quality of WMN links, the monitoring should ideally increase the monitored wireless-specific metrics while reducing the impact of monitoring costs over the data plane in an SDWMN in-band. In addition, the monitoring costs should follow the quality changes of wireless links.

To overcome these limitations, we propose in this paper a multi controllers architecture with adaptive monitoring in SDWMN in-band. Our approach allows a hierarchical distribution of the control plane with: i) a top level global controller (GC), which provides automated coordination configuration and management and maintains consistent global network view; ii) several local controllers (LC), where each one controls a set of switches, reducing control plane delay by reducing the distance from switches to controllers, and these LCs are deployed on-demand by GCs depending on network dynamics. Thus, this multi controllers architecture provides better scalability, performance and availability; iii) a passive monitoring approach where switches periodically send monitoring metrics to their LC without polling requests, reducing traffic costs; iv) adaptive monitoring where LCs can define in what interval and which metrics will be sent by switches according to

network conditions; v) switches have an agent which reads and captures specific wireless metrics not available in common SNMP or OpenFlow implementations.

This paper is divided as follows. Section 2 shows the background study. In section 3 we present the design and implementation of our architecture. The section 4 presents the evaluation and results. The paper concludes on section 5.

2. Background

Physically distributed control plane with logically centralized control plane requires state distribution between controllers. However, it can impact the performance of logically centralized control applications, this way, strongly consistent distributed controllers always operate with a consistent global view and thus, ensure that controllers operate properly in a coordinated manner. This process imposes overhead and delay, which can be especially harmful in SDWMN in-band, therefore limiting the responsiveness and may lead to sub-optimal decisions. Eventually consistent distributed controllers integrate the information as they become available, thus, the controllers react faster and can handle higher update rates, but tend to have a temporarily inconsistent global vision and possibly leading to incorrect behaviors. This results in two trade-offs: the first is between the overhead to ensure the consistency of the state distribution and performance of control applications; the second relationship is between the complexity of the logic of control applications and robustness to handle inconsistencies [Levin et al. 2012].

A single OpenFlow controller brings scalability problems to networks. To tackle these problems, several research efforts have been made. Works such as [Koponen et al. 2010, Tootoonchian and Ganjali 2010, Berde et al. 2014] propose a physical control plane distribution while maintaining the logical centralization using a distributed file system. In spite of distributing the control plane, these approaches impose a global network view in all controllers, consequently increasing the amount of shared information in the network, which is not desired in SDWMN in-band. Kandoo [Hassas Yeganeh and Ganjali 2012] proposes a distributed control plane that contains two-level hierarchical controllers, local applications are controlled by local controllers and the latter redirect decisions that need the global network view to a root controller. All communications between controllers are asynchronous and event-based as a publish subscriber architecture. In this method, a local controller works as a proxy for the requests coming from switches which would need to go directly to the root controller, resembling and providing the benefits of a proxy. This approach helps reducing the amount of messages exchanged over the control plane, which is desired in WMN. Other works deal with the amount and placement of controllers [Heller et al. 2012, Bari et al. 2013, Mattos et al. 2015].

These multi controllers approaches are generic and do not solve specific SDWMN problems. For example in an in-band scenario, the control plane communication uses the same channel as the data plane and it could negatively impact network traffic, therefore, being able to keep the overhead of control plane as low as possible is important. Also, to improve multi controllers support in SDWMN, the network should be able to deal with: link quality variations and interruptions, which may physically isolate controller instances from network devices; wireless forwarding devices can be many hops away from the controller, which would increase the latency and network convergence time; inconsistency

in network global view leads to performance degradation and errors in applications that need consistent state.

In terms of monitoring, these works have monitoring modules, which do not go deep into the subject. They collect few metrics such as delay or OpenFlow statistics and most of these metrics are actively measured (with periodic heartbeat or request messages inserted into network), which mostly are not specific for wireless networks management, such as RSSI or neighbor bytes and packets sent/received. Ideally, the monitoring should gather as many metrics as possible, thus enabling even more possible uses for them, while reducing the impact of monitoring costs over the data plane in an SDWMN in-band. Furthermore, link quality variations are often present in WMN due to many reasons (one of them, variable intra and inter-nodes interference), so a static monitoring overhead may degrade even further poor WMN links. Consequently, an adaptive monitoring overhead is desired [Hava et al. 2012]. [Kim et al. 2016, Shah et al. 2016] recently addressed monitoring in ONOS framework [Berde et al. 2014], which has multi controllers feature. The first records OpenFlow message exchanged on the ONOS logging system, providing real time monitoring result. The latter is an adaptive monitoring solution to dynamically monitor the overall load of individual controllers working in a logically centralized SDN environment. However, both of them do not focus on monitoring wireless-specific metrics, neither on network performance metrics.

Common protocols used for network monitoring in SDWMN are SNMP and OpenFlow. However, they are not well suited for specific data collection from wireless physical and link layers [Duarte et al. 2007, Nascimento et al. 2014] and it directly impacts the flexibility benefits that SDN could have in WMN. [Nascimento et al. 2014] implements extensions to OpenFlow to contemplate specific wireless functionalities, including new messages into OpenFlow to query nodes for wireless interfaces physical layer metrics, but it was only tested in an out-of-band scenario. [Dely et al. 2011] and [Rethfeldt et al. 2015] insert custom SNMP agents in WMN client and uses non OpenFlow router nodes, respectively, to collect specific wireless metrics from the WMN. Although, the first is too end-user intrusive and the latter has a relative high monitoring traffic cost of 80Kb for each mesh router per monitoring cycle.

Works focused on reducing network resource utilization in WMN or on adaptive monitoring overhead according to network quality are not based on SDN. A hierarchical WMN monitoring is presented in [Nanda and Kotz 2008] where a node is responsible for monitoring itself and nodes k-hopes distant. The monitoring data gathering is passive, decentralized and uses a combination of passive and active monitoring techniques depending if the amount of monitoring overhead in a link is high or low, respectively.

Related works show that there are few approaches aggregating granular metrics from physical and link layers of SDWMN while reducing network resource utilization in these networks. The latter is important because it can negatively impact network performance in a SDWMN in-band [Hava et al. 2012].

3. General Architecture

Our work proposes a multi controllers architecture with adaptive monitoring in SDWMN in-band. Figure 1 shows an example of this architecture with a global controller (GC), a monitoring database (MonitDB) and *N* control domains. Each control domain consists of

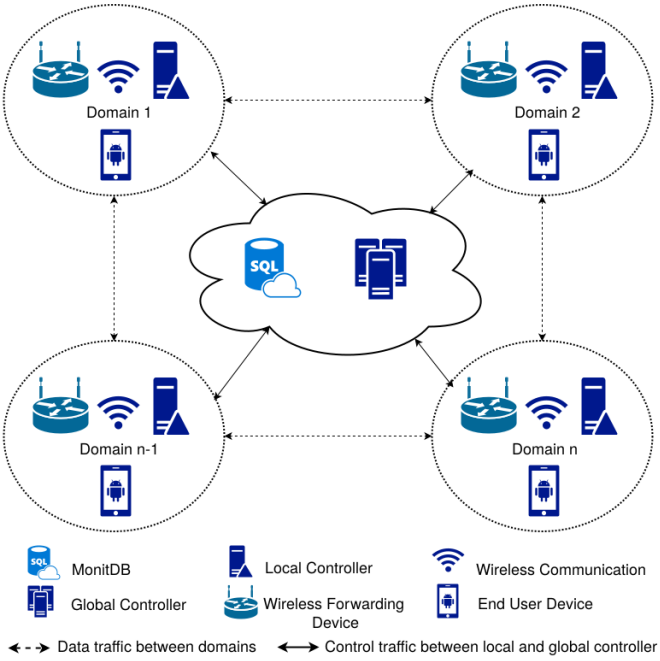


Figure 1. System Architecture

a Local Controller (LC) and wireless forwarding devices (WFD) with OpenFlow support, such as wireless mesh router (WMR) and access points (AP). Lastly, the end user devices connect to one of the forwarding devices inside a control domain, as if the end user was connected to a common access point. This architecture was implemented as a proof-of-concept on OpenWiMesh [Brito et al. 2014], a framework for developing SDN in WMN in-band, extending its features.

Hierarchical Architecture: To enable faster answers to WFD requests and to reduce control plane overhead in the rest of the network, our architecture is based on a hierarchy of GC and LCs. This approach offloads control plane applications load over the multiple controllers, which is more suitable for a WMN unlike the horizontal approach that impose the network-wide state synchronization cost.

Global Controller: One top level global controller provides automated coordination, configuration and management of local controllers and maintains consistent global network view. The GC provides top-level applications to LCs through an Global Control API (GCAPI). GCs can reactively respond to LC's GCAPI queries, e.g inter-domain routes discovery, or proactively distribute commands to LCs, e.g coordinate the deploying and installation of OpenFlow rules in LCs and delegation of data plane devices to LCs. The GC subscribes to each LC to receive periodical topology information updates from them, so the global network view is updated in an event-based communication, as later seen in Figure 2. We assume that the GC is deployed in a cloud cluster with high availability, for example, in one or more data centers.

Local Controllers: LCs are dynamically provisioned and deployed on-demand by the GC when and where needed according to the network dynamics possibly based on data plane overhead, network fault tolerance or simply by network policies. Each LC controls a set of WFD through OpenFlow (thus creating a control domain) and manages

the entire intra-domain information. LCs do not communicate with each other and they only know about nodes inside their domain. This way, the GC must be queried by LCs, for example, about packet forwarding to nodes outside LC's domain. In our implementation, the LC is represented by the controller element that is inside the WMR as shown in [Brito et al. 2014] and our work extends the WMR, making it remotely accessible and having its controller element started/stopped by the GC. Although, a LC could also be deployed in a similar way by the Network Function Virtualization (NFV) concept in a server close to domain devices. Moreover, every WMR is a potential controller, so the number of LC can follow the network growth.

Global Control API: LCs can access the applications provided by GC using a remote access API. The GCAPI is implemented using a Python library called Pyro[1]. Each LC has a GCAPI client installed that calls the methods implemented in GC.

Inter Domain Routing Table: The Inter Domain Routing Table (IDRT) is present in LCs and provides instructions required to forward packets to outside domain destinations. In a few words, entries in IDRT are made of different metrics and each entry correlates two domains, for example there must be a field for the local domain edge node and another for neighbor domain edge node, if additional metrics are required they may be requested to the MonitDB. This table can optionally lower the amount of control traffic over the network by preventing LCs from constantly requesting outside domain information to the GC in a short spam of time, by working as a cache. The IDRT implemented in this work is adapted from [Phan et al. 2013].

Inter Domain Routing: If a new flow arrives at a WFD, and if this flow does not match any flow entry, the WFD forwards a PacketIn message to LC that controls him. If the destination of this flow is not within the local domain, the LC looks up in the IDRT for instructions on how to forward this flow to another domain. Once the destination is not in IDRT, the LC sends this request to GC by using the GCAPI. Then, the GC performs path computations and send the routing instructions to all LCs which control the domains where the flow request will be forwarded to. So each LC parses GC's instructions and install the forwarding rules of this flow into their local domain WFDs.

Fault Tolerance: We consider 4 types of failures: path, LC and GC failures and GC isolation. In the first one, OpenWiMesh framework can find new routes once network links of a path fail. Secondly, if a LC fails, the nodes of its domain must reconnect to another LC. So, local domain edge nodes of the failed LC will connect to the LC of the closest neighbor domain and the local nodes near the edge nodes will successively follow the edge nodes by connecting on same LC. The GC is responsible to define whether the failed domain nodes will be integrated into the new domain or if a new controller will be instantiated to manage these nodes. Thirdly, as described before, we assume that the GC is deployed in a high availability (HA) environment. Finally, in case of a domain isolation, when a LC loses communication with GC and there is no neighbor domain to forward GC's traffic, the LC of this isolated domain keeps accepting new flows for local traffic only. Meanwhile, the LC keeps trying to communicate with the GC through every local domain edge node and, once it succeeds, the communication with GC is reestablished.

Monitoring Database: As shown in Figure 2, the Monitoring Database

[1]https://pythonhosted.org/Pyro4/

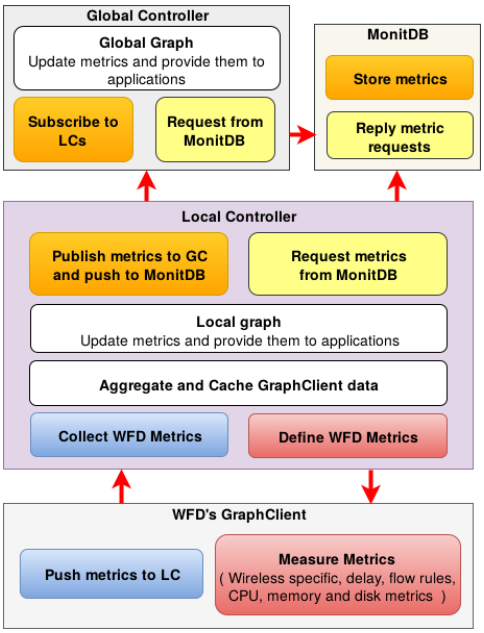


Figure 2. Proposed monitoring system

(MonitDB) can be consulted by GCs and LCs and have data inserted by LCs with monitoring data metrics from their local network view. Apart from sending simple local domain network topology data to GC, LCs periodically write network metrics from its domain to the MonitDB. The periodic push mode from LC refers to the LC working as a monitoring proxy, temporarily aggregating monitoring data before transmitting it to the MonitDB, this way no data is lost in case of communication problems between them. LCs can send their raw or calculated network metrics, such as bytes transferred or average local domain intra-node delay. The Monitoring Database can be accessed by LCs or GCs when querying for historical data metrics or inter domain routing metrics, respectively.

Network Graph: Data structure responsible for storing local network view data in the LCs (local network graph) and global network view in GC (Global network graph), as shown in Figure 2. LCs update their local view with the information periodically sent by the GraphClient module running in every local domain WMR, as described in [Brito et al. 2014]. LCs also periodically send their local view to the GC, but any change in network topology of any domain is immediately reported to the GC. GC has the entire network topology view, but specific domain metrics are consulted on the MonitDB, which are periodically populated by LCs.

Proactive Push-Based Monitoring: In a WMN, a single fault can affect several neighboring nodes in a network, and even disconnecting them, leading to isolated partitions. These faults could make the gathering of accurate network information challenging or even impossible after the occurrence of the fault. Hence we advocate the use of a periodic and proactive approach to monitor the network in order to be able to detect fault conditions before it actually happens. Aiming for monitoring traffic overhead reduction, a push-based mechanism is used, where nodes proactively send monitoring information to the LC without receiving queries messages from it.

GraphClient: GraphClient is an agent module installed in every WFD in the

network to passively collect specific metrics from wireless interfaces. The wireless monitoring metrics are collected via the Linux firmware nl80211 and sent to the LC using a periodic proactive push-based approach. GraphClient's role is not to distribute monitoring information to other WFDs, but to keep the LC graph up-to-date with network quality variations (as seen in Figure 2) and to enable a historical view on network quality. Our paper extends GraphClient functionalities [Brito et al. 2014] by inserting automatic/dynamic variation of data pushing intervals and number of metrics monitored (based on LC's control), as well as new monitoring metrics as shown in Table 1[2]. The new metrics enable better network traffic and load behavior view as well as providing ground for new different management applications and routing algorithms. With these metrics one can, for example, detect overloaded WFD nodes or links (as done by our adaptive monitoring) or create paths over low latency links or low load nodes.

Type of data	Description
Node metrics	CPU usage* (%)
	TX Power (dB)
	Free Memory (MB)*
	Free disk space (MB)*
	Openflow flow rules statistics*
Association metrics	RSSI
	Bytes sent*
	Bytes received*
	Packets sent*
	Packets received*
	Packets retransmitted
	Max bandwidth (Mbps)
	Inactive time (s)
	Bidirectional delay (s)*

Table 1. Metrics captured by GraphClient.

Adaptive Monitoring: The monitoring overhead inside a local domain follows the network quality of the own local domain. Each LC monitors the network quality inside its domain and, based on the quality of each WFD and network wireless links, it controls in what interval and which variables each GraphClient will send its monitoring metrics back to the LCs. If one of the network wireless links drops in quality and there is no other better route for the monitoring traffic, then the LC tell all WFDs reachable through that link to increase the interval of monitoring or, at the same time, lower the number of metrics monitored. Once the link quality gets back to normal, LC can tell the node to monitor again all variables back at the lowest interval. One thing to keep in mind is to not over increase the measurement interval of metrics used by routing or network quality detection algorithms in LCs, thus avoiding long network graph inconsistencies. Its encouraged to keep these routing metrics always monitored in fixed intervals, while others can be changed or not even monitored.

[2]In Table 1, the fields with '*' are the new metrics of our GraphClient extension.

In our implementation, LCs can set 3 different monitoring states, *normal(N)*, *higher interval(HI)* and *minimum(M)*. At *N* state, all monitoring variables are measured and collected at the lowest pre-defined interval. If the bandwidth consumption of a link goes past a threshold (25%) of the residual bandwidth, LCs order WFDs to change to the *HI* state. In *HI* state, all monitoring variables have their collection interval set *Y* times higher, except the ones that are being used for the routing algorithms, otherwise the network graphs gets less updated and the network may become less stable. If the bandwidth consumption of a link is over 50%, only the variables used for routing are kept being monitoring in the normal interval while all others are not measured or collected anymore. The thresholds selected are not based on researches, to the best of our knowledge, due to the lack of it. Nevertheless, the chosen thresholds aims to lower the monitoring traffic overhead, consequently lowering the possibility of losses and physical modulation changes in a saturated wireless link before it happens. All states can have intervals and amount of metrics configurable.

4. Evaluation and Results

We evaluated our proposal in two stages. In the first stage we execute three experiments, the first two of them to demonstrate the scalability improvement, based on [Hassas Yeganeh and Ganjali 2012] evaluations, and a third to show the lower switch-controller latency brought by our architecture. In all three experiments we compare the results to a single OpenWiMesh controller (Normal OWM). In the second stage, we show that our monitoring solution captures different metrics including wireless ones as expected and then, we compare the amount of monitoring bandwidth a node would expend using either SNMP or our monitoring solution to show how much our solution saves network bandwidth compared to common solutions. These evaluations demonstrate the feasibility of multi controllers and monitoring of specific wireless metrics of a SDWMN in-band, in order to distribute the control plane in an WMN and provide scalability and enhanced control plane management. The proposed architecture has a lot of features, for example, fault tolerance and adaptive monitoring. But, for the sake of space, we are focusing on testing scalability, latency and monitoring metrics.

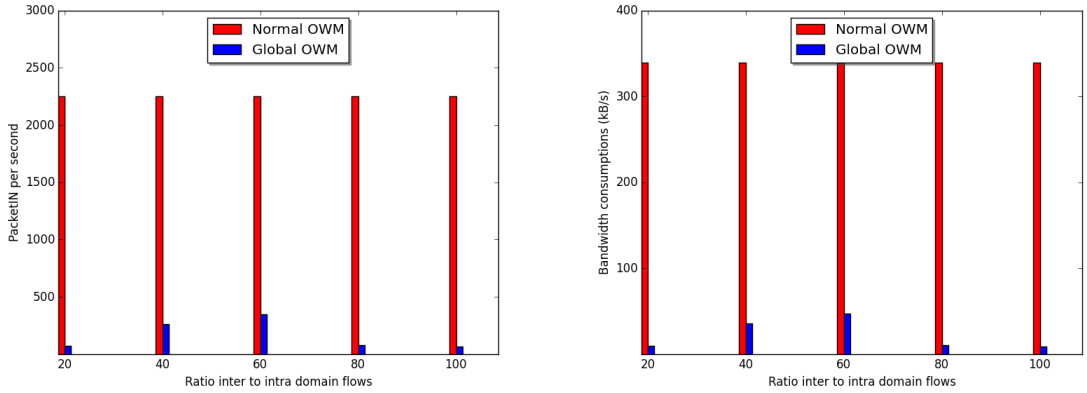
The adopted topology for the experiments is equivalent to the depicted in Figure 1, with a fan out factor varying from 2 to 4 (i.e. 2 to 4 local controllers and 10 WFD per local domain). Additionally, nodes dispersion were randomly done by CORE Emulator and we considered the following assumptions: Every node is at most 5 hops away from its controller and every node has 10 ms delay from each other configured from Linux traffic control *tc* command.

We evaluate our proposal in an emulated environment using the same real time emulator described in [Brito et al. 2014] [3]. This emulator is capable of emulating devices operating in data and network layers, including the physical layer operating properties of them. At its emulation scenario in GNU/LINUX, it utilizes kernel namespaces to create devices and Linux network bridges to create link connections between nodes. Each node has its own processes and environment variables, thus creating a totally isolated environment between network nodes. To enable the collection of neighbors bytes and packets (as shown in Table 1), we had to setup the emulator to, instead of creating namespaces with

[3]CORE Emulator v4.8

its default network wireless interfaces, create network interfaces using the kernel module called mac80211_hwsim. We plan to implement link losses and medium access delay in our future tests. To run the experiments, we used a Personal Computer with an Intel(R) Core(TM) i7-3630QM 2.40GHz CPU and 8 GB of RAM, running Debian 9. We executed every test 10 times and we only show here the averages taken with 95% confidence interval.

In the first two experiments of the first stage, we measure the number of requests (PacketINs) processed by each controller and their bandwidth consumption. Our main goal is to decrease the load on our global OpenWiMesh controller (Global OWM) compared to the Normal OWM. We evaluate how the control plane scales against the ratio of inter and intra domain flows (variating inter domain traffic from 20% to 100%). Each WFD starts fifty TCP flows to any other WFD on the network. Figure 3 shows that the Global OWM load is much lower than the Normal OWM, even when 100% of traffic are inter domain. This happens because the IDRT presents on LCs work as a cache reducing the amount of requests to the GC, as soon as the inter domain traffic grows and more "cache hits" happen. In the second experiment, the ratio of inter and intra domain flow is set to 20% using different fanout (from 2 to 4). The goal of this experiment is to measure our proposal's control plane scalability as the number of nodes increases. As seen in Table 2, the larger the network is, the larger the load difference is between Normal OWM and our proposal. It happens because a local controller works as a proxy for the requests coming from WFDs protecting the GC from high request load. Meaning that our proposal scales better than a single OpenFlow controller, such as the Normal OWM.



(a) Average number of PacketIns received by the controllers

(b) Average number of bytes received by the controllers

Figure 3. Control Plane load for inter and intra domain routing flows scenario. The load is based on the number of inter domain flows.

In the third experiment of stage one, we measure the average latency between WFD and their respective controllers via ICMP ping's RTT. Each WFD is in ping loops for 10 minutes, sending 3 ping packets in 1 second interval each to WFDs, in a crescent IP order. There is a 10 ms delay configured on each hop and ARP resolutions delay are accounted. The goal here is to show the positive impact in switch-controller latency as more LCs are deployed in the network. Table 3 presents the average switch-controller latency varying the fan out factor, as seen in the second experiment. The result shows lower latencies for our architecture in every fan out factor. It happens because each LC

Number of nodes	Normal OWM (kB / s)	Global OWM (kB / s)	Normal OWM (pktINs / s)	Global OWM (pktINs / s)
Fan out 2	156	8.8	1020	60
Fan out 3	238	9.6	1625	65
Fan out 4	339	11.3	2250	73

Table 2. Control Plane load for inter and intra domain routing flows scenario. The load is based on the number of nodes.

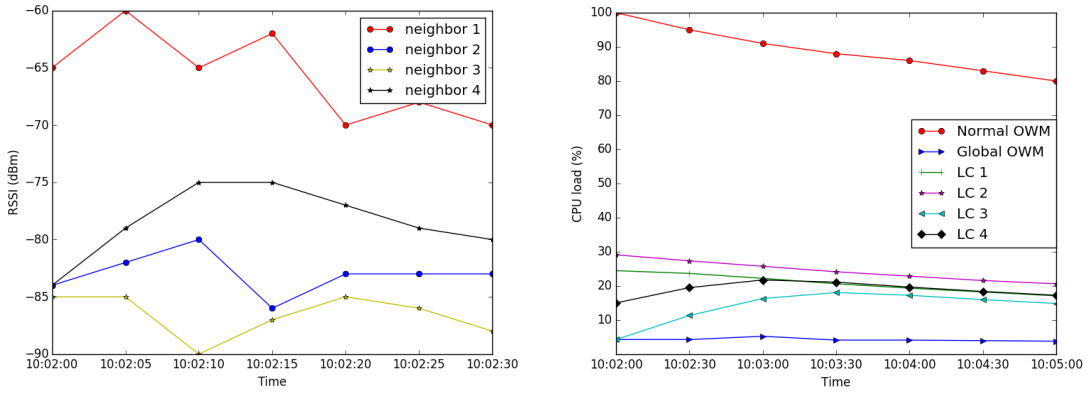
controls a set of switches, reducing control plane delay by reducing the distance from switches to controllers compared to a single controller topology.

Number of nodes	Normal OWM	Multiple Controllers
Fan out 2	23 ms	15 ms
Fan out 3	29 ms	16 ms
Fan out 4	34 ms	18 ms

Table 3. Differences in average controller-switch latency with a single and multiple controllers.

The experiments of the second stage are done in a fan out 4 topology (40 nodes) with 20% inter domain flows ratio. In the first experiment of the second stage, we show the time series monitoring metrics captured by our solution, including the wireless-specific metrics and the CPU use of normal controllers and our solution's. The goal here is to show that our solution scales linearly and enables better network view with wireless-specific metrics. For the sake of space, we show here some of the metrics captured. Figure 4 shows two time series with metrics captured from GraphClient. In Figure 4(a) we show a time series with the RSSI values related to each neighbor of one WFD in the network, proving us with all link connection qualities in the network and helping, for example, the decision of routing algorithms. This kind of time series can be done with all metrics monitored, providing historical and better network view as a whole. Figure 4(b) shows the CPU load behavior of a single Normal OWM and our proposal's controllers. The average CPU load in the global and local controllers of our proposal is lower than the load on the single OWM controller, demonstrating our load distribution and scale potential. CPU load metrics can help us deciding when and where to deploy these multiple controllers.

In the second experiment, we compare the average monitoring bandwidth costs of a WFD trying to deliver all its possible monitoring metrics running either our proposed monitoring system or a common implementation of SNMP. We also change the number of wireless neighbors of a WFD from 1 to 5, basically repeating the metrics for each neighbor and rising the packet size. The goal is to show how much our proposal's monitoring system can save bandwidth compared to a common monitoring protocol. The WFD's metrics monitored are the ones shown in Table 1, except the "Openflow flow rule statistics" field, because it varies too much depending on the number of OpenFlow rules installed in the WFD. Our calculations in this experiment are based in the protocols overheads shown in Figure 5 and we consider each metric as a SNMP OID where each OID has a fixed fair size of 10bytes. As seen in Table 4, our solution saves, at minimum, near 50% in



(a) RSSI time series of neighbors of a WFD

(b) CPU load time series on a Normal OWM and our solution's controllers

Figure 4. Time series graphs with different monitoring metrics captured by GraphClient.

bandwidth compared to a SNMP reply, because there's a considerable OID overhead in SNMP which is minimum in our solution. Besides, in periodical monitoring using SNMP protocol, there is also the request messages overhead in the network, while there are none periodic requests in our solution.

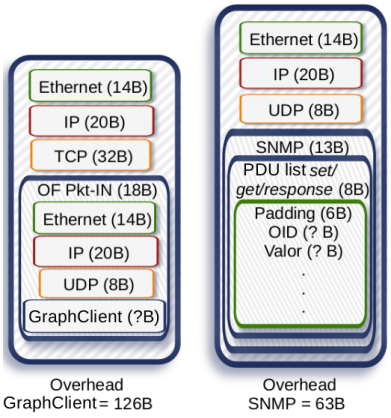


Figure 5. Protocols headers overhead on monitoring packets of our monitoring system's and SNMP's packets

5. Conclusion and Future Work

SDWMN in-band can join the benefits from SDN and WMN to make WMN more intelligent and programmable without high deployment costs. However, there are some issues about control plane scalability, performance, and availability. To address these issues several works in the literature proposed multiple controllers architectures, but these proposals were not implemented to SDWMN in-band. The implementation of a multi controllers architecture in SDWMN in-band requires to lead with challenges associated with the wireless mesh environment. Besides, a monitoring system able to gather specific wireless metrics without overloading the communication channel is indispensable.

We proposed a multi controllers architecture with adaptive monitoring in SDWMN in-band which objective is to reduce control plane overhead and delay while im-

Bandwidth cost (bytes)	**1 Neighbor**	**2 Neighbors**	**3 Neighbors**	**4 Neighbors**	**5 Neighbors**
SNMP	387	615	854	1084	1274
Our solution	211	286	355	415	497

Table 4. Comparison of different monitoring packet size averages

proving network scalability. The results show that our architecture linearly scales the control plane while reduces switch-controller latency, as well as offering better network view and management for wireless networks while providing low monitoring overhead compared to common monitoring protocols. To the best of our knowledge, this is the first proposal of distributed controllers for SDWMN in-band with wireless-specific metrics and adaptive monitoring.

As future work, we plan to evaluate the elasticity of our architecture with several dynamic controller placement algorithms and evaluate the benefits of adaptive monitoring in lowering packet drops in the network. We also plan to implement link losses and medium access delay in our tests.

References

Bari, M., Roy, A., Chowdhury, S., Zhang, Q., Zhani, M., Ahmed, R., and Boutaba, R. (2013). Dynamic controller provisioning in software defined networks. In *Network and Service Management (CNSM), 2013 9th International Conference on*, pages 18–25.

Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W., and Parulkar, G. (2014). Onos: Towards an open, distributed sdn os. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 1–6, New York, NY, USA. ACM.

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

Chung, J., Gonzalez, G., Armuelles, I., Robles, T., Alcarria, R., and Morales, A. (2013). Experiences and challenges in deploying openflow over real wireless mesh networks. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 11:955–961.

Dely, P., Kassler, A., and Bayer, N. (2011). Openflow for wireless mesh networks. In *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*, pages 1–6. IEEE.

Duarte, J. L., Passos, D., Valle, R. L., Oliveira, E., Muchaluat-Saade, D., and Albuquerque, C. V. (2007). Management issues on wireless mesh networks. In *2007 Latin American Network Operations and Management Symposium*, pages 8–19.

Hassas Yeganeh, S. and Ganjali, Y. (2012). Kandoo: A framework for efficient and scalable offloading of control applications. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 19–24, New York, NY, USA. ACM.

Hava, A., Ghamri-Doudane, Y., and Murphy, J. (2012). On monitoring overhead impact in wireless mesh networks. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2012 8th International*, pages 487–492. IEEE.

Heller, B., Sherwood, R., and McKeown, N. (2012). The controller placement problem. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 7–12, New York, NY, USA. ACM.

Kim, W., Li, J., Hong, J. W. K., and Suh, Y. J. (2016). Ofmon: Openflow monitoring system in onos controllers. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 397–402.

Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., and Shenker, S. (2010). Onix: A distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA. USENIX Association.

Levin, D., Wundsam, A., Heller, B., Handigol, N., and Feldmann, A. (2012). Logically centralized?: State distribution trade-offs in software defined networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 1–6, New York, NY, USA. ACM.

Mattos, D. M. F., Lopez, M. A., Ferraz, L. H. G., and Duarte, C. M. B. (2015). Controlador resiliente com distribuição eficiente para redes definidas por software. In *Computer Networks and Distributed Systems (SBRC), 2015 Brazilian Symposium on*.

Nanda, S. and Kotz, D. (2008). Mesh-mon: A multi-radio mesh monitoring and management system. *Comput. Commun.*, 31(8):1588–1601.

Nascimento, V., Moraes, M., Gomes, R., Pinheiro, B., Abelém, A. J. G., Borges, V. C. M., Cardoso, K. V., and Cerqueira, E. (2014). Filling the gap between software defined networking and wireless mesh networks. In Raz, D., Nogueira, M., Madeira, E. R. M., Jennings, B., Granville, L. Z., and Gaspary, L. P., editors, *CNSM*, pages 451–454. IEEE Computer Society.

Phan, X. T., Thoai, N., and Kuonen, P. (2013). A collaborative model for routing in multi-domains openflow networks. In *Computing, Management and Telecommunications (ComManTel), 2013 International Conference on*, pages 278–283.

Rethfeldt, M., Danielis, P., Moritz, G., Konieczek, B., and Timmermann, D. (2015). Design and development of a management solution for wireless mesh networks based on ieee 802.11s. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 902–905.

Shah, S. A. R., Bae, S., Jaikar, A., and Noh, S.-Y. (2016). An adaptive load monitoring solution for logically centralized sdn controller. In *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 1–6.

Tootoonchian, A. and Ganjali, Y. (2010). Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, INM/WREN'10, pages 3–3, Berkeley, CA, USA. USENIX Association.