DiMob : scalable and seamless mobility in SDN managed wireless networks

DiMob: Scalable and Seamless Mobility in SDN Managed Wireless Networks

Ian Vermeulen, Patrick Bosch, Tom De Schepper, and Steven Latré
University of Antwerp - imec, IDLab, Department of Mathematics and Computer Science, Belgium
fistname.lastname@uantwerpen.be

***Abstract*—Wi-Fi network roaming is the act of moving a wireless device from one Wi-Fi access point (AP) to another Wi-Fi AP. In urban environments, where APs are densely deployed, users would greatly benefit from roaming between these APs. Standards for Wi-Fi-network roaming have been developed (e.g. IEEE 802.11r), but are rarely implemented. The absence of a widely used standard leads to device-dependent roaming mechanisms, which brings numerous disadvantages. 5G-EmPOWER is an example of a framework that brings the Software-Defined Networking (SDN) paradigm to wireless networks. The framework solves the problem of network roaming by allowing users to connect to their own unique virtual AP and managing their connection to the WAN behind the scenes. This allows the 5G-EmPOWER controller to seamlessly handover users from one physical AP to another. Currently, a single physical controller manages the 5G-EmPOWER control plane. The use of a single system over a distributed system has known disadvantages (e.g. greater cost, single point of failure). In this paper, we present DiMob, which distributes the SDN control plane among multiple controllers. We show that DiMob maintains a seamless handover, while offering the advantages of a distributed system. We demonstrate, for example, that adding an additional node can save approximately 30 % in CPU usage for each controller.**

I. Introduction

Urban places have become filled with Wi-Fi APs [1]. Typically, these APs only have a range of a few meters. As a result, users in motion are handed over from one Wi-Fi network to another. When Wi-Fi was first introduced, a mobile device could connect with an AP using only four messages. However, as new features were added, the amount of messages required for a device to connect with an AP greatly increased. This increase in messages implied that connections could be terminated for seconds before the users were reconnected [2]. The standard IEEE 802.11r was introduced to seamlessly handover devices from one AP to the other by reducing the amount of messages that are required to connect with an AP. However, the standard is barely implemented by vendors [3]. This causes devices to fall back to the legacy Wi-Fi handover method, which waits for the current connection to drop and then reconnects the device to the other network.

In a networking context, we can differentiate between two planes: the control plane and the forwarding plane. The control plane perceives the network situation and commands the forwarding plane to perform certain actions (e.g. send packet to a certain port, drop packet, ...). In traditional networks, network devices contain both a control plane and a data plane. This architecture has slowed down innovation of the computer networking landscape for years [4]. For wired networks, the implementation of SDN paradigm elevates the problems of this inflexible architecture. SDN logically centralize the control plane in a single controller that commands the switches. OpenFlow is the established protocol to communicate between controllers and switches.

5G-EmPOWER brings the SDN paradigm to the wireless networking environment and improves Wi-Fi network roaming by introducing a seamless handover for clients. 5G-EmPOWER is an SDN based framework for managing wireless networks (i.e. Wi-Fi, LTE). Currently, the implementation of this framework uses a single physical controller to manage its operations. Such an architecture, where a system relies on a single computing node, has the disadvantage of having a single point of failure. Furthermore, this type of architecture is not scalable. As an increasing number of users and APs join the network, the resources of the single controller will eventually be exhausted and a maximum supported load will thus be reached. Moreover, this increasing load also increases the delay of the communication with the central controller, resulting in a less responsive system. These problems can be solved by distributing the responsibilities of the single controller. Distributed computing has certain advantages compared to traditional computing, such as: financial benefits, added fault-tolerance and scalability [5].

We present DiMob, an API for distributing the 5G-EmPOWER control plane. DiMob transforms 5G-EmPOWER into a scalable solution: controllers can be added to the network to support more users and APs at a lower CPU and memory footprint per controller. At the same time, DiMob maintains a seamless handover. We demonstrate the performance of DiMob by implementing a distributed mobility use case. To the best of our knowledge, DiMob is the first distributed SDN application that allows seamless Wi-Fi roaming. Our experiments show that as more APs and clients become active in the network, the addition of new controllers benefit the load of each controller. Furthermore, the handover of a mobile device between controllers has no significant impact on the connection of this device. This shows that the handover remains seamless in DiMob.

The remainder of this paper is organized as follows: Section II elaborates on related work. In Section III we outline the design and implementation of DiMob. Section IV discusses the results of our experiments. Finally, Section V concludes.

II. Related work

In this section, we will elaborate on several projects related to the distributed SDN mobility manager.

A. Wi-Fi handover

Murty et al. propose DenseAP to improve the capacity of a wireless enterprise network [6]. In DenseAP, Wi-Fi APs are installed densely in an office environment. This allows clients to be load-balanced among the various APs, resulting in a higher throughput. During load-balancing or when a client changes location, it is possible that clients are handed off from one AP to another. This handoff can take approximately 1.5 seconds [6]. In contrast, the handover in 5G-EmPOWER and DiMob do not introduce a significant delay.

Dyson is a software architecture for building customized WLANs [7]. The architecture was introduced by Murty et al.. Dyson provides APIs that allow wireless clients and APs to communicate with a central controller. This central controller is then able to build a complete overview of the network and create policies to manipulate the behavior of the network. Murty et al. demonstrate how Dyson manages to reduce the handoff delay between APs to approximately 6 milliseconds [7]. However, Dyson requires heavy modifications of the wireless client whereas DiMob requires no such modification.

B. SDN

Odin is a SDN based framework for wireless networks [8]. Whereas 5G-EmPOWER supports both Wi-Fi and LTE, Odin focuses only on Wi-Fi. The framework also introduces a mobility manager that supports a seamless handover between Wi-Fi APs. However, the project is no longer actively maintained.

Onix is a distributed SDN control platform [9]. The key element of Onix is the Network Information Base (NIB) [9]. This NIB is a graph that describes the network status. The applications that run on Onix can modify the NIB and Onix assures that the graph is properly distributed and replicated among the various Onix nodes. The platform brings scalability by balancing the work load among different nodes and making the nodes transparent by exposing them as one node [9]. While both Onix and DiMob distribute the SDN control platform, DiMob focuses on a wireless network environment, whereas Onix does not have this focus.

III. DiMob

In this section, we will discuss the design of DiMob. We first elaborate on 5G-EmPOWER, the framework of which we distribute the control plane. We then describe concepts such as virtual AP ownership, handovers, dummy APs and the details on how the different controllers share their information.

A. 5G-EmPOWER

5G-EmPOWER is an SDN framework for managing wireless networks (e.g. LTE and Wi-Fi). The framework introduces high-level programming abstractions to manipulate various control aspects of a wireless network [10]. The framework introduces a mobility manager application that actively hands over devices from one AP to the other, based on which AP offers the best connection. We build DiMob on top of 5G-EmPOWER. This allows us to update the distribution logic and the controller logic independently.

B. Ownership

In our design, every user is owned by one and only one controller. The device can only make a connection to the WAN through one of the APs of that controller. This will prevent the client from receiving duplicate replies through different controllers. When a wireless client joins the network, the controllers should consent on the controller that takes the initial ownership of the corresponding virtual AP. The most common consensus algorithm is Paxos [11]. Paxos is known to be an efficient and highly fault-tolerant consensus algorithm [12]. However, the algorithm is also known to be highly complex [11]. To this end, Ongaro and Ousterhout introduced Raft [13]. Raft is based on Paxos, but has a fundamentally different structure. The algorithm divides consensus into two major components: leader election and log replication. As we only need to select an owner (i.e. leader) for a certain virtual AP, we can simply implement the leadership election part, without needing to implement the log replication. The structure of Raft makes the algorithm less complex than Paxos while it maintains the same performance and fault-tolerance [13]. In Raft, all nodes are initially followers. The followers start an election timer that triggers after a randomized time $t \in [250, 400]$ ms. When the election timer triggers, the corresponding follower becomes a candidate. The candidate starts a new term, votes for itself, requests other nodes to vote for him to become a leader and finally, starts a new election timer for a randomized time $t \in [250, 400]$ ms. A follower that receives a vote request from a candidate and has not voted yet this term, replies to the candidate with a vote. If a candidate receives votes from the majority of the nodes before its election timer expires, it becomes the leader. This node then broadcasts to all other nodes that it has become the leader.

C. Dummy AP

If a controller does not own a certain user, then it should not send/receive network traffic to/from the client (e.g. send a web page in response to an HTTP request.). However, this controller still needs to monitor and store signal information of this user and therefore cannot simply ignore him. For this reason, we introduce the concept of a dummy AP that disconnects the connected users from the WAN. When a user joins the network, every controller that detects this client (i.e. gets a signal for the corresponding device on one of its APs), schedules the user on the dummy AP. The controllers then vote for ownership of the user using the Raft method which was described in Section III-B. The controller that becomes the leader of the client, schedules the client on a physical AP. All other controllers keep the wireless client scheduled on their dummy APs. We note that this dummy AP

is also needed because 5G-EmPOWER requires programmers to always schedule users on a valid AP.

D. Information sharing

The controllers communicate with each other using the ZeroMQ[1] library. This library is a high-performance messaging library created to build distributed applications [14]. They can either broadcast a message to all other controllers or they can unicast a message to a specific controller. To enable controllers to broadcast messages, we implemented a publish-subscribe environment where every controller is a publisher and subscriber to every other controller. This means that controllers can publish a message and every other controller will receive this published message. However, using this environment, the network administrator would have to configure every controller with the addresses of all other controllers. This implies a certain amount of effort of the administrator. We therefore used the XPUB-XSUB pattern from ZeroMQ. This pattern allows both subscribers and publishers to connect to a central node, the hub. The hub will then forward the published messages from the connected publishers to the connected subscribers. Now, the administrator is only required to configure each controller with the location of this central hub. The effort for the administrator has thus been decreased significantly. We implement the hub by defining a master controller. A master controller is a DiMob controller that also serves as a hub. To avoid a single point of failure, we can introduce several master controllers and configure each controller with the IP addresses of these master controllers. As soon as one master controller fails, the controllers will use the next master controller. Subscribers listen to all master controllers and will therefore not be affected by the crash of a master controller.

Controllers can also send other controllers messages directly. The IP address of the sender is included in the messages so that the receiver can reply to the message.

Controllers share their signal information every 500ms, by default. This signal information is a list of tuples with a tuple for each client. The tuple consists of the MAC address of the user, the maximum of every signal that the APs get for that user and a flag that indicates whether the publishing controllers owns the user. With these broadcasts, controllers update their internal signal table. This signal table stores the Received Signal Strength Indicator (RSSI) value of the connection between users and controllers.

E. Handover

A controller can handover one of its user between its APs. We refer to this handover as the *local handover*. It is also possible for the controller to send a request to another controller to take ownership of one of its users. We call this type of handover a *foreign handover*.

The flow to perform a foreign handover for a user is illustrated in Figure 1. The owning controller C_1 first disowns the user, then sends a handover message to the destination

[1]http://zeromq.org/

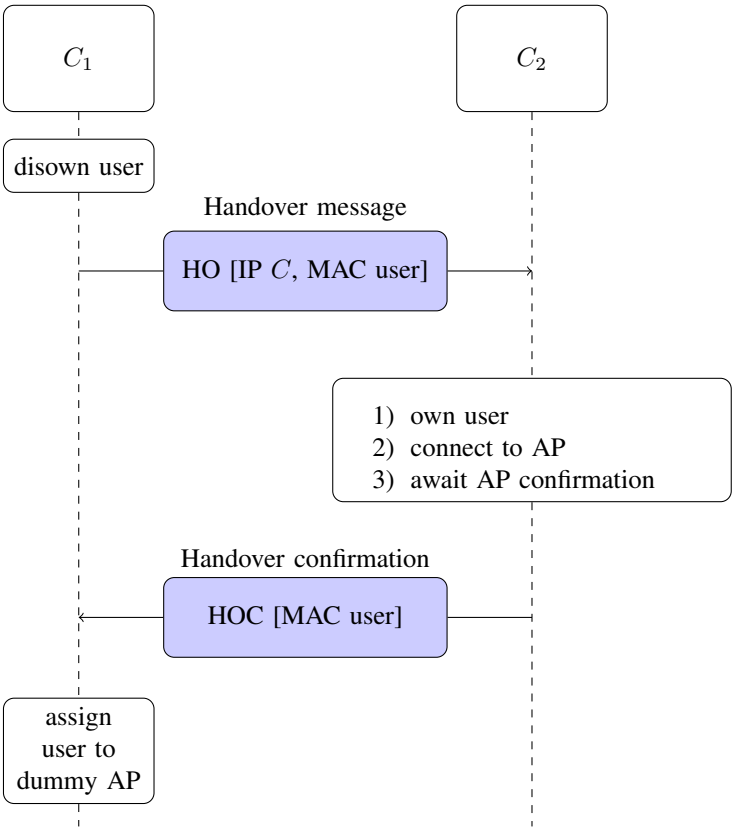


Fig. 1. The message flow for a foreign handover of a user.

controller. The receiving controller C_2 claims the user as his and performs a local handover to schedule the client on one of the APs of C_2. When the reply of an AP confirms that the user is indeed connected to this AP, the controller confirms the foreign handover to C_1 by sending a handover confirmation message. Finally, the controller C_1 schedules the user on the dummy AP. Note that this flow is not fault-tolerant. If a controller crashes while the flow has not been completed, unexpected behavior may occur. We consider the creation of a fault-tolerant flow for a foreign handover part of future work.

IV. Evaluation

In this section we will discuss the performance of the mobility use case of DiMob. In our application, the controller periodically hands over each of its users to the AP that offers them the best possible signal. If no AP offers a decent signal for a certain user, the controller will perform a foreign handover for this user. These handovers are also performed when an AP notifies its controller that the signal of one of its connected clients falls beneath a configured threshold. We refer to the non-distributed version of the 5G-EmPOWER mobility manager application as *Vanilla*. We will compare the throughput and packet loss of the Vanilla and DiMob network. Furthermore, we will perform load tests where a high number of users that connect to DiMob are simulated. During these tests, we will measure the CPU, memory and network usage.

We then analyze how the amount of users, amount of APs and amount of controllers influence these statistics.

In our test environment three computers with Intel Core 2 Quad CPUs (3,00GHz x 4) serve as the controllers. The APs are two Alix boards that support wireless connections and have a wired backbone connection.

A. Throughput and loss

In this section we will compare the throughput and packet loss between a DiMob network with two controllers and a Vanilla network with one controller. We verify that we succeeded in extending the seamless handover between APs in Vanilla to a seamless handover between controllers (and implicitly between APs of different controllers) in DiMob.

The setup for testing for Vanilla and DiMob is illustrated in Figure 2. The controllers (or controller, in case of Vanilla) and both APs are connected to a switch, which in turn is connected to a router. The router connects the LAN to the WAN. The setup is illustrated in Figure 2.

We configure the test server to be an iPerf server and the test subject to be the iPerf client. The iPerf client sends test packets to the server, who will report the received packets every .1 seconds. These test packets are sent by either using TCP or UDP. We did run our test for 30 seconds. During this time span, the test subject is handed over twice: once after 10 seconds and another time after 20 seconds. The tests were executed five times for each configuration.

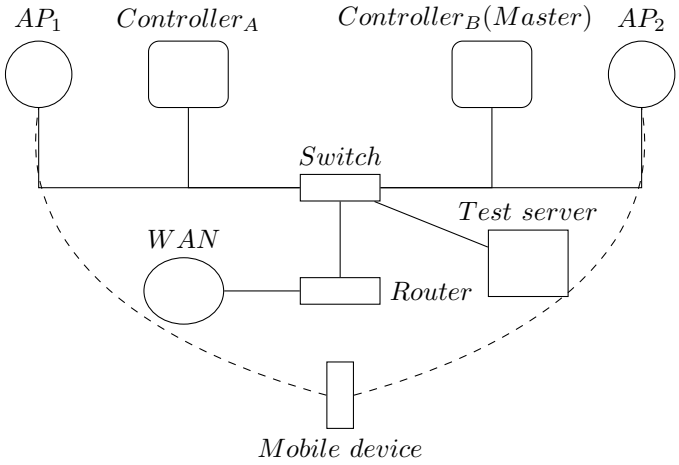
Figure 3a and Figure 3b show the throughput over time of the test performed where the client sends traffic to the server in a single TCP or UDP flow. We see that the bandwidth over time is similar for Vanilla and DiMob. The results show no connection losses (i.e. a bandwidth of 0 Mbit/s) or, more general, no noticeable drops of bandwidth during any particular moment in time. The deviation between test runs was negligible. We note that the UDP flow in DiMob has a slightly lower bandwidth than in Vanilla, however this can be explained with our test environment, which was an office space. Multiple other devices were present and caused interference which we were not able to negate. We did not do an interference analysis though, as this was not in the scope of the paper. The difference in bandwidth should therefore be nonessential for our conclusion.

Figure 3c shows the datagram loss experienced during the UDP iPerf test. Again, we see that both graphs are similar and that the loss of DiMob is slightly higher than that of Vanilla. This higher loss is due to the presence of interference and is nonessential for our conclusion.

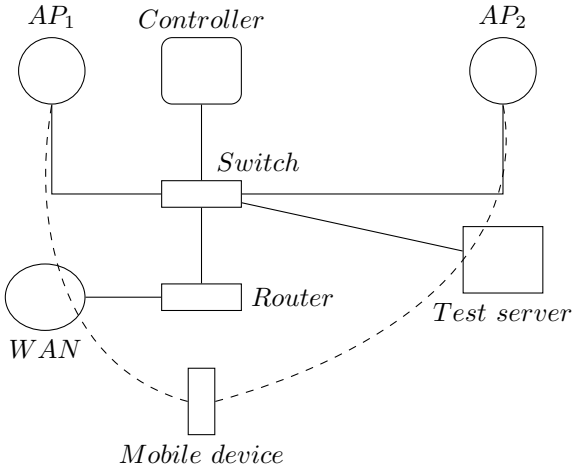
We conclude that a foreign handover (DiMob) does perform the same as a local handover (Vanilla) and that the handover mechanism is seamless in both cases.

B. Load

In this section, we will discuss the effect on system resources when dividing the responsibilities of a single DiMob controller to multiple controllers. More specifically, we shall evaluate how CPU, memory and network usage are affected by



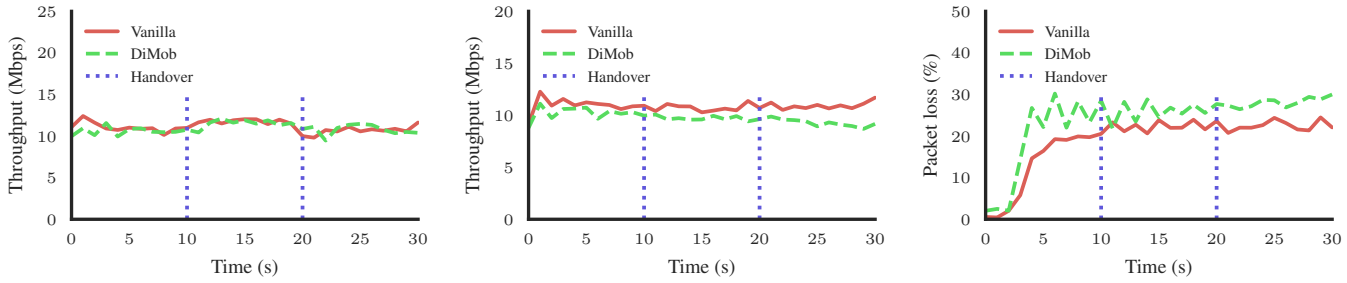
(a) DiMob test environment. The two nodes serve as the controllers.



(b) Vanilla test environment. One node serves as the single controller.

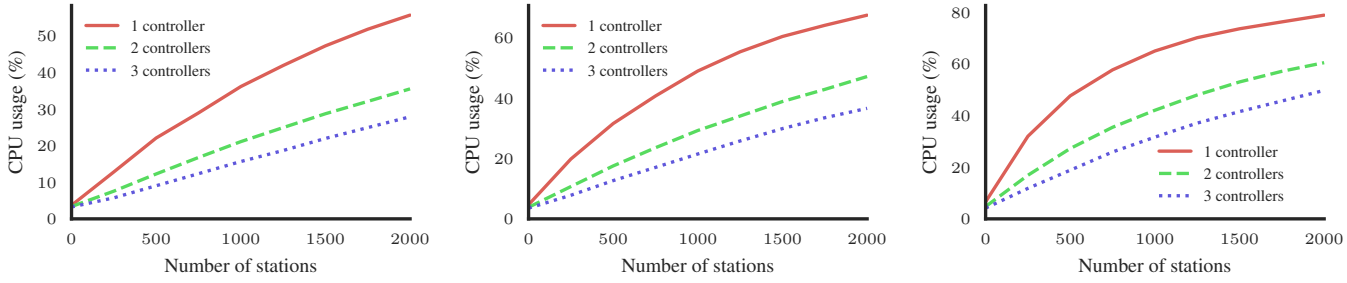
Fig. 2. DiMob and Vanilla test environments. Two Alix boards act as the APs. These devices are connected to a switch, which is connected to a router. The router is connected to the WAN. One wireless device attempts to connect to the test network.

distributing APs among more controllers while an increasing number of clients join the network. To properly test the controllers under high load, we implemented the emulation of users and APs in DiMob. Each emulated user connects to an emulated AP. This AP emulates a real signal for its connected users. The controllers will then treat the emulated users as any other user and make the appropriate decisions. Note that the computation time and memory usage of the simulation itself is included in the results. However, the computation needed for the emulation is limited and our results are still valid. We execute the test for the total AP amounts of 48, 96 and 192 and five times for each configuration. These APs are equally divided among the DiMob controllers. The first test runs with an amount of 0 users. This allows us to capture the minimum load of DiMob. We then increase the amount of users by 250 for each test until a maximum of 2000 users. The results of the CPU usage, memory usage and the network usage during the



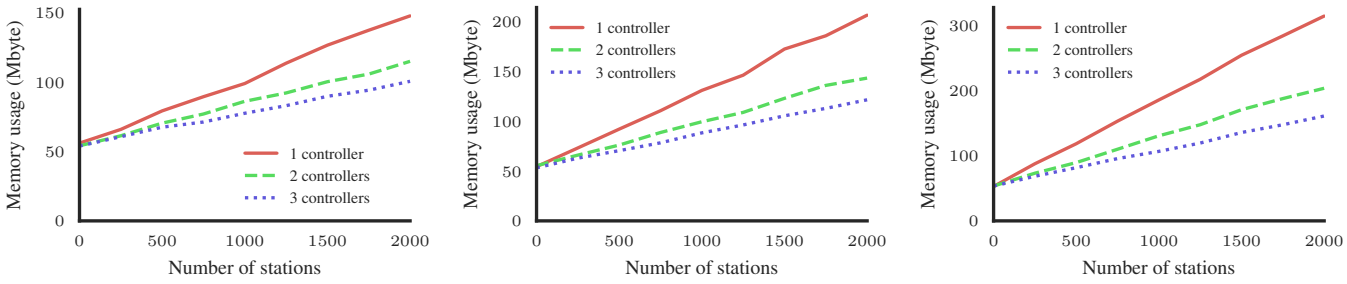
(a) TCP throughput. (b) UDP throughput. (c) UDP loss.

Fig. 3. Throughput and loss over time for UDP and TCP. Handovers (local for Vanilla and foreign for DiMob) at timestamps 10 s and 20 s are completely smooth.



(a) 48 APs. (b) 96 APs. (c) 192 APs.

Fig. 4. CPU usage for an increasing amount of stations shows the benefit of multiple controllers in reducing load.



(a) 48 APs. (b) 96 APs. (c) 192 APs.

Fig. 5. Memory usage for an increasing amount of stations shows the benefit of multiple controllers in reducing memory.

experiment are illustrated in Figure 4, Figure 5 and Figure 6 respectively.

We see that the CPU usage increases as the number of users and APs increase. We observe that the average CPU and memory usage of 1 controller is greater than that of 2 controllers and that the average CPU and memory usage of 2 controllers is always greater than that of 3 controllers. Again, the deviation between runs was too small to be mentioned. We can thus conclude that we succeeded in reducing the load of each controller by adding additional controllers to the environment. Furthermore, we note that the difference in load between all three scenarios slowly increases as the number of APs and users increase, e.g.: the percentage of CPU usage saved in the scenario of 2 controllers versus 1 controller is approximately 7 % for 250 users and 48 APs and increases to approximately 30 % for 2000 users and 192 APs. We conclude that, as more APs and users are added to the system, the benefit of our solution increases. Finally, we note that the average memory usage of DiMob, for all scenarios, is very low.

We see that the amount of network traffic scales approximately linear with the amount of controllers. It is up to the network administrator to consider whether the additional traffic that a controller introduces is tolerable on the network. Furthermore, we see that an increase of users also increases

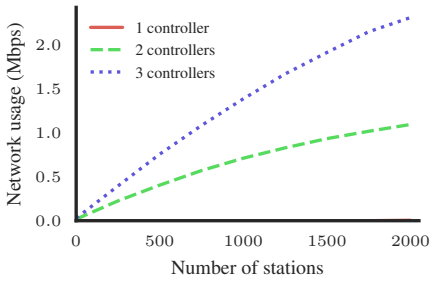


Fig. 6. Network usage for controller communication for an increasing amount of stations is very small.

the amount of network traffic. This can be explained the fact that if more users are added, the list of signal tuples that the controllers broadcast increases in size, which in turn increases the network traffic rate. We note that the network usage, for all scenarios, is very low, below 2.5 Mbit/s, and that only a small amount of network traffic is added when an additional controller is introduced to the network.

V. Conclusion

In this paper, we demonstrated DiMob, a solution to easily distribute and scale wireless SDN. In particular, we focused on reducing the CPU and memory usage of each controller by distributing the APs and clients among all controllers. DiMob introduces an API to distribute the control plane among multiple controllers and each DiMob controller manages a subset of the APs and clients. In the mobility use case of DiMob, the controllers make sure that their clients are continuously connected to the AP that provides them with the best possible connection to the network.

DiMob maintains a seamless handover mechanism while introducing scalability. The CPU and memory usage can be reduced by up to 30 %, while introducing only a small amount of network overhead. For systems that should support a large amount of users, network administrators can now easily introduce several nodes to scale their wireless network instead of one immensely powerful one, resulting in a financial benefit. Furthermore, the presence of multiple nodes prevent a node from becoming a single point of failure, making the network more robust.

DiMob can be further extended with features such as smart load-balancing between the various controllers and a fault-recovery mechanism for controllers.

References

[1] M. Afanasyev, T. Chen, G. M. Voelker, and A. C. Snoeren, "Usage patterns in an urban wifi network," *IEEE/ACM Trans. Netw.*, vol. 18, no. 5, pp. 1359–1372, Oct. 2010. [Online]. Available: http://dx.doi.org/10.1109/TNET.2010.2040087

[2] C. R. Wright and C. Polanec, "Metrics for characterizing bss transition time performance." [Online]. Available: \url{https://mentor.ieee.org/802.11/file/04/11-04-0989-01-000r-metric-characterizing-transition-time-performar ppt}

[3] D. D. Coleman, D. A. Westcott, B. E. Harkins, and S. M. Jackman, *CWSP Certified Wireless Security Professional Official Study Guide: Exam PW0-204*. John Wiley & Sons, 2011.

[4] D. Kreutz, F. M. V. Ramos, P. E. Verssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

[5] A. D. Kshemkalyani and M. Singhal, *Distributed computing: principles, algorithms, and systems*. Cambridge University Press, 2011.

[6] R. Murty, J. Padhye, R. Chandra, A. Wolman, and B. Zill, "Designing high performance enterprise wi-fi networks." in *NSDI*, vol. 8, 2008, pp. 73–88.

[7] R. Murty, J. Padhye, A. Wolman, and M. Welsh, "Dyson: An architecture for extensible wireless lans." in *USENIX Annual Technical Conference*, 2010.

[8] 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*. ACM, 2012, pp. 115–120.

[9] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks." in *OSDI*, vol. 10, 2010, pp. 1–6.

[10] R. Riggio, M. K. Marina, J. Schulz-Zander, S. Kuklinski, and T. Rasheed, "Programming abstractions for software-defined wireless networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 2, pp. 146–162, June 2015.

[11] D. Ongaro, "Consensus: Bridging theory and practice," Ph.D. dissertation, Stanford University, 2014.

[12] R. D. Prisco, B. Lampson, and N. Lynch, "Revisiting the paxos algorithm," *Theoretical Computer Science*, vol. 243, no. 1, pp. 35 – 91, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304397500000426

[13] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, 2014, pp. 305–319. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

[14] M. Sstrik, "Zeromq: The design of messaging middleware," http://www.drdobbs.com/architecture-and-design/zeromq-the-design-of-messaging-middlewar/240165684?pgno=1, 2014.