On the Benefits of Wireless SDN in Networks of Constrained Edge Devices

August Betzler*, Ferran Quer*, Daniel Camps-Mur*, Ilker Demirkol†, Eduard Garcia-Villegas†
*i2CAT Foundation, Barcelona, Spain
† Universitat Politecnica de Catalunya, Barcelona, Spain
{august.betzler, ferran.quer, daniel.camps}@i2cat.net, {ilker.demirkol,eduardg}@entel.upc.edu

***Abstract*—In this paper we study the benefits of applying Software Defined Networking (SDN) to control forwarding in a network of constrained wireless edge devices. The proposed architecture is applicable to dense Small Cell deployments featuring wireless backhauling and edge computing capabilities, or to wirelessly connected sensor nodes following the fog computing paradigm. The paper introduces a novel path forwarding policy based on SDN, and presents an experimental evaluation demonstrating the benefits of the proposed policy to mitigate external interference, achieve flow balancing, and cope with CPU constrained devices.**

I. Introduction

Applying Software Defined Networking (SDN) architectures to wireless mesh networks is a developing area of research that attempts to mitigate some of the shortcomings encountered in traditional wireless mesh network solutions [1]. Despite the performance penalty incurred by multi-hop forwarding, wireless mesh networks continue to offer an interesting value proposition due to their reduced costs.

Recently, two novel applications are reviving the interest on wireless mesh networks, while posing new challenges. First, the deployment of ultra dense networks of outdoor Small Cells to cope with the growth in mobile data requires affordable backhaul technologies. Second, the fog computing paradigm [2] proposes to transition from cloud based IoT architectures, to IoT architectures where intelligence is pushed to the edge, which requires more flexible and reliable networking solutions. Wireless mesh networking offers a viable solution to both paradigms.

In this paper, we study the benefits of applying SDN to wireless mesh networks of constrained edge devices, that is, devices with very limited processing power and memory capacities. In particular, we build upon the SDN architecture proposed in [3], and propose two novel SDN forwarding policies for multi-radio mesh nodes. The proposed policies consider both network state and available computing resources in the network elements, and have been evaluated in an testbed composed of multi-radio Raspberry Pi B+ devices (RPi).

Related to our work, [5] proposed a hybrid architecture comprising a distributed OLSR daemon to configure the in-band control network and a centralized SDN controller to configure the data plane, while demonstrating an Internet gateway balancing policy. The authors in [4] proposed an architecture for wireless backhauling, where the SDN controller operates on an abstracted view of the mesh network and is only used

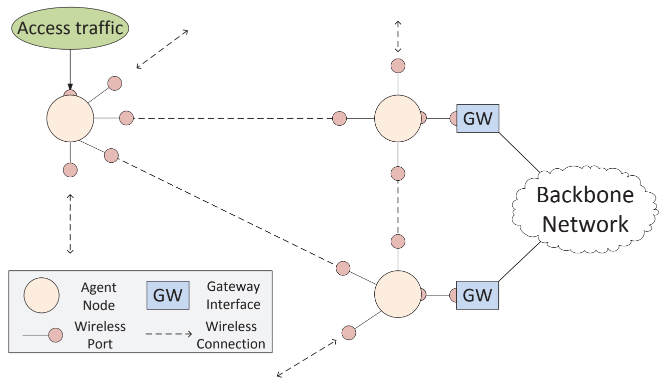


Fig. 1: General view on our system model: the wireless mesh network provides backhaul connectivity for external access traffic via gateway interfaces.

to configure end-to-end flows. To the best of the authors' knowledge this is the first paper to propose forwarding policies that consider both network and available computing resources enabled by specific OpenFlow wireless extensions.

This paper is structured as follows. Section II introduces the system model and describes our SDN forwarding policy. Section III provides an experimental evaluation that illustrates the benefits of the proposed forwarding policy in scenarios with external and internal interference, and in a network featuring CPU constrained devices. Finally Section IV concludes the paper and discusses future work.

II. System Design

A. System Model

Fig. 1 depicts the system model considered in this paper, consisting of a set of nodes forming a wireless mesh network using unlicensed spectrum. Following the architecture presented in [3], nodes contain a wireless port to represent each potential one-hop neighbor in the wireless mesh.

Two types of nodes are considered in the system: So called *agent nodes* serve as entry points for access traffic, while also acting as relay nodes in the network. Once traffic enters the wireless network via one of the agent nodes, it is forwarded to one of the gateway nodes where the traffic is bridged over to the wired network. It is considered that a flow can use any of the gateways to reach the wired network.

Each end-to-end flow in the mesh network, i.e. between *agent* and gateway, is defined using a unique identifier. Thus,

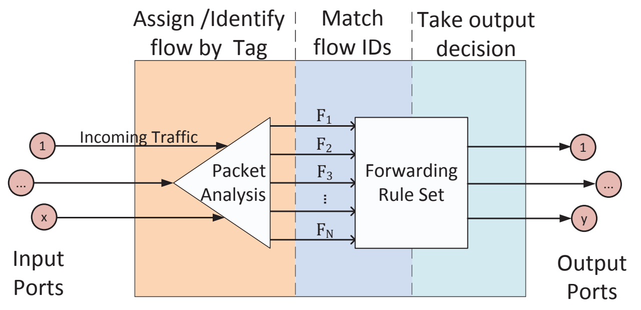


Fig. 2: Abstraction of an agent node and its forwarding mechanism that is capable of distinguishing between up to N different data flows.

access traffic entering the mesh network through an *agent* node is assigned to an end-to-end flow. Our architecture allows for N different unidirectional flows in each *agent* node, thus resulting in up to $N \times A$ end-to-end flows in the mesh network, where A is the number of *agent* nodes. Hence, N allows to trade-off flow granularity with forwarding table size.

When data packets are received by an agent node, the node matches the flow identifier and looks up the forwarding rules to decide the corresponding output interface. An abstract view of an agent node and the implementation of the packet forwarding process is given in Fig. 2. The SDN controller is responsible for determining the paths that the flows will follow throughout the wireless network and for installing the forwarding rules on the network nodes accordingly. For this purpose the controller runs an algorithm that consists of three main steps:

1) Bootstrapping: Upon initiating the network, the SDN controller installs an initial set of forwarding rules on the agent nodes for each flow. An arbitrary routing policy, such as minimum hop-count or WCETT [6], can be used for this purpose.
2) Network state monitoring: The SDN controller collects statistic reports from the agent nodes, which include information about their wireless ports, internal state (e.g. CPU load), and data flows traversing them. This information is used to get a holistic view of the system, and to trigger corrective actions if necessary.
3) Flow reallocation policy: If the SDN controller detects a precarious network state (e.g. link congestion), it reacts by applying a data flow reallocation policy. In this paper two such policies are introduced and evaluated.

B. Information available at the SDN controller

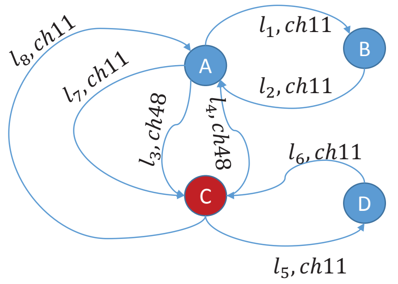
Each of the nodes in the network maintains information about its internal state, its wireless network interfaces and ports, and the observed network environment. The set of statistics gathered by each agent node is shown in Table I.

The SDN controller periodically polls each node for statistics. Nodes respond to such a request with statistic response packets that contain all the statistics gathered over the last 16 seconds[1]. The SDN controller calculates a moving average of the gathered values, stores them in a database, and processes the data in order to obtain a view of the network state that is used by the decision taking algorithm to manage the network data plane. For the reallocation algorithm to trigger and to start data flow redirections, it is necessary that a critical network state is observed.

[1]Up to 8 measurements taken every 2 seconds are buffered until receiving the statistics request

TABLE I: Overview of the Statistics Gathered by Each Node

Statistic	Explanation
Interface Statistics:	
Channel Usage	Occupancy of a wireless channel ($0 \leq u \leq 1$)
Channel Number	Operational channel of a wireless interface
Port Statistics:	
Signal Strength	RSSI value measured by a wireless interface
Tx/Rx Bitrate	Physical (PHY) Data rate used over a certain link
Device Statistics:	
CPU Load	A node's CPU load in %
Flow Statistics:	
Tx/Rx Packets	Number of packets transmitted via a flow
Tx/Rx Bytes	Bytes transmitted via a flow



$$I = \begin{bmatrix} 1&1&0&0&1&0&1&1 \\ 1&1&0&0&0&0&1&1 \\ 0&0&1&1&0&0&0&0 \\ 0&0&1&1&0&0&0&0 \\ 1&0&0&0&1&1&1&1 \\ 0&0&0&0&1&1&0&1 \\ 1&1&0&0&1&0&1&1 \\ 1&0&0&0&1&1&1&1 \end{bmatrix}$$

Fig. 3: Illustration of the SDN controller's view on the topology (left) and the corresponding interference matrix generated by the SDN controller(right).

C. Flow redirection policies

In order to be able to take meaningful path forwarding decisions, the SDN controller is continuously analyzing whether a flow reallocation would improve the overall network state.

The used network model builds upon the following assumptions. First, data flows are always established between an agent node and a gateway, traversing the network over one or several hops. Second, each wireless hop between two nodes a and b is composed of two independent, unidirectional links; one link that originates in node a and terminates in node b, and vice-versa. Since statistics of the wireless links are collected by both endpoints of a wireless connection, the controller has an independent view from each side of each link, which is important for the detection of asymmetric links and to be able to create asymmetric routes between two nodes. Further, it is assumed that links of neighboring nodes using wireless interfaces that are operating on the same channel interfere each other. Fig. 3 illustrates an example network model in the SDN controller representing a network of 4 nodes, and the corresponding interference matrix $\boldsymbol{I}$, identifying interfering links. The controller computes the interference matrix by determining which links are on the same channel in a node's neighborhood, and assuming that data transmission can interfere nodes up to 2 hops away.

1) Load Balancing Policy: The goal of the load balancing policy is to reduce the utilization level of individual links in order to minimize congestion events. For this purpose the controller monitors the load of individual links, and upon measuring a utilization level above a configurable threshold, u_{thr}, executes the following algorithm.

Let us start denoting the set of links in the network model kept at the SDN controller as $\mathcal{L} = \{l_1, ..., l_M\}$, and the network state vector as $\boldsymbol{ns} = \{u_1, ..., u_M\}^{\intercal}$, which contains the utilization levels of each link in the network, and is maintained through the periodic statistic reports issued by the nodes (see Table I).

Upon detecting a link l_i with a utilization level above u_{thr}, the controller constructs the set $\mathcal{F} = \{F_1, F_2, ..., F_T\}$ containing all flows that traverse the link l_i. Thus, the goal of this policy is to reallocate some of the flows in $\mathcal{F}$ over alternative paths in order to bring the utilization of all links in the network, below u_{thr}.

To decide which flows to reallocate, the controller implements a greedy heuristic whereby it starts considering the flow $F_j \in \mathcal{F}$ with the highest data rate. Let us assume that F_j traverses path $\boldsymbol{P_x}$, where $P_{x,y} = 1$, with $1 \leq y \leq M$, if $l_y \in \boldsymbol{P_x}$ and 0 otherwise. A *base* network state can be computed as $\boldsymbol{ns}^0 = \boldsymbol{ns} - \boldsymbol{I} \times \boldsymbol{u_{F_{j,x}}}$, where $\boldsymbol{u_{F_{j,x}}}$ is the vector containing the utilization level introduced by flow F_j on each link of $\boldsymbol{P_x}$, and $\boldsymbol{ns}^0$ represents the estimated network state without flow F_j. Notice that the controller can easily compute $\boldsymbol{u_{F_{j,x}}}$ because it continuously measures the load introduced by each flow and the data rate used in each link (see Table I).

As a next step, the controller computes the set $\mathcal{P} = \{\dot{\boldsymbol{P}}_{\boldsymbol{1}}, \dot{\boldsymbol{P}}_{\boldsymbol{2}}, ..., \dot{\boldsymbol{P}}_{\boldsymbol{K}}\}$ containing K candidate alternative paths between the source and destination nodes of F_j. $\mathcal{P}$ is computed using a K-shortest-path routing algorithm with the WCETT metric [6], thus $\mathcal{P}$ already contains paths with low expected utilization levels. In order to compare the candidate paths in $\mathcal{P}$, and select a new path $\boldsymbol{P_{new}}$ to reallocate F_j, the controller uses as metric the resulting network state when flow F_j is allocated through each candidate path $\dot{\boldsymbol{P}}_{\boldsymbol{l}} \in \mathcal{P}$, computed as $\boldsymbol{ns_{P_l}} = \boldsymbol{ns}^0 + \boldsymbol{I} \times \boldsymbol{u_{F_{j,l}}}$, which accounts for the interference that this flow will cause throughout the network. Consequently, the controller selects as alternative path, $\boldsymbol{P_{new}}$, the one minimizing the worst case link congestion in the resulting network state vector, i.e.:

$$\boldsymbol{P_{new}} = \min_{1 \leq l \leq K} \max \boldsymbol{ns}_{\dot{\boldsymbol{P}}_{\boldsymbol{l}}}$$

Finally, if the selected path $\boldsymbol{P_{new}}$ improves the current network state by a configurable threshold θ and it reduces the utilization of all links below u_{thr}, the corresponding set of forwarding rules is installed in the nodes to redirect the traffic. Else, the algorithm repeats the process for the next flow in $\mathcal{F}$ in decreasing order of introduced load. In case that no flow is chosen to be redirected, the controller does not take any action. Notice, that the complexity required by the described greedy searching process is affordable in an SDN architecture, where the controller has access to the required computational and storage resources.

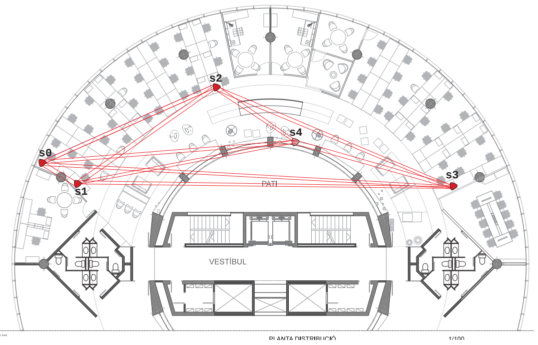


Fig. 4: Map of the office environment in which the experiments are carried out.

2) Avoiding Overloaded CPUs Policy: In fog computing scenarios, nodes may become temporarily CPU hogged due to the execution of local computations, which affects their data forwarding capabilities. Thus, this policy excludes nodes that suffer from a high CPU load from acting as forwarders in the data plane. This policy executes the following greedy algorithm:

1) Monitor CPU utilization in each node through periodic reports (see Table I).
2) Determine the heaviest flow that is passing through a node whose CPU utilization exceeds a configurable threshold cpu_{thr}[2]. Compute the set $\mathcal{C} = \{n_1, ..., n_L\}$ containing the list of nodes being CPU hogged and sorted in decreasing order of CPU utilization.
3) Compute a set $\mathcal{P} = \{\dot{\boldsymbol{P}}_{\boldsymbol{1}}, \dot{\boldsymbol{P}}_{\boldsymbol{2}}, ..., \dot{\boldsymbol{P}}_{\boldsymbol{K}}\}$ of K alternative paths that do not contain the node $n_i \in \mathcal{C}$ with highest CPU load. If no alternative paths can be found, skip to the next step.
4) Apply the same $\boldsymbol{P_{new}}$ selection method used in the load balancing policy to select a new path that avoids n_i, redirect the traffic and terminate the algorithm.
5) Repeat the process removing the next node, n_{i+1}, from $\mathcal{C}$, until the flow can be redirected or $\mathcal{C}$ is empty and the algorithm terminates.

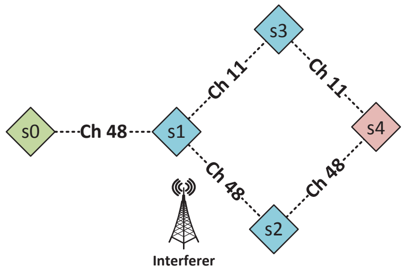
III. Evaluation

A. Testbed Description

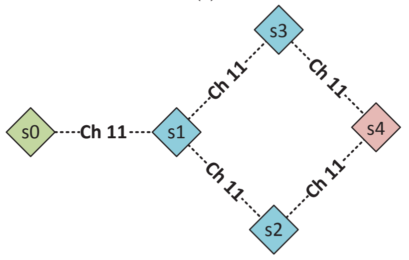
The testbed used for the experimental evaluations consists of 5 Raspberry Pi B+ [10] nodes that are distributed on a single office floor, building a mesh network as shown in Fig. 4. Each Raspberry is equipped with two Wi-Fi dongles with *ath9k*-compatible chipsets. For each Raspberry, the two wireless interfaces are chosen to operate at the 2.4 GHz band (channel 11) and at the 5 GHz band (channel 48), respectively.

While channel 48 has been found to be interference free, on channel 11 we observe a low, constant background traffic. Simultaneous communication over these two channels is possible. The double links between the nodes in Fig. 4 indicate

[2]A hysteresis margin is applied to filter out sporadic utilization peaks.

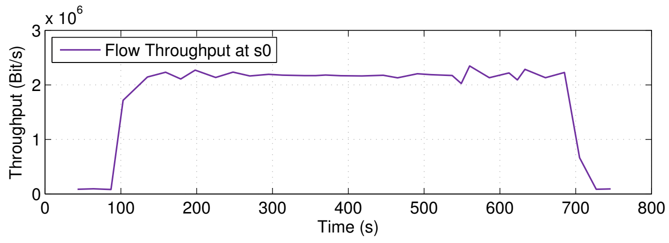


(a)

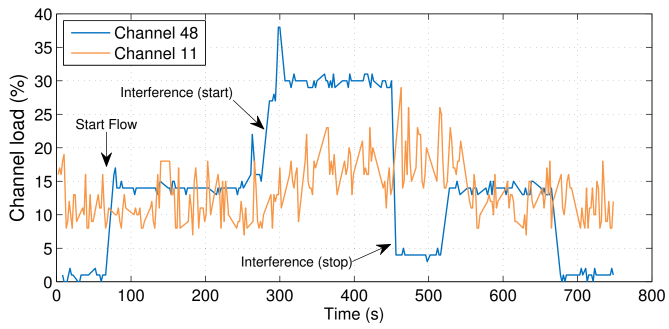


(b)

Fig. 5: (a) Bannergraph topology for the interference aware routing and flow balancing scenarios, using a mix of links on channels 11 and 48 and (b) Bannergraph topology for the CPU aware routing scenario, using only channel 11.



(a)



(b)

Fig. 6: (a) Throughput measured over the link from *s0* to *s1* and (b) channel load observed by s1 during the interference aware routing experiment.

connectivity between each of the nodes' wireless interfaces (on channel 11 and channel 48, respectively). In the chosen network setup, each node is capable of seeing all other nodes of the network. Thus, to generate topologies that are a subset of the basic experimental topology, we implement MAC filters that only allow logical connections between specific wireless interfaces. Applying these filters, we generate a $(4, 1)$-tadpole (bannergraph) topology, where we analyze three experimental scenarios to show and explain the reaction of the presented SDN forwarding policies to certain network states. In all of these scenarios, data flows are generated from agent node *s0* towards gateway node *s4*. The access traffic entering the network at *s0* is generated by a host device (laptop) that is connected over Ethernet with the Raspberry. Further, we set the utilization threshold u_{thr} to 15%[3], and the CPU threshold cpu_{thr} to 80%.

B. Experimental Evaluation

Next, we describe the three experimental scenarios and show the results of the evaluations.

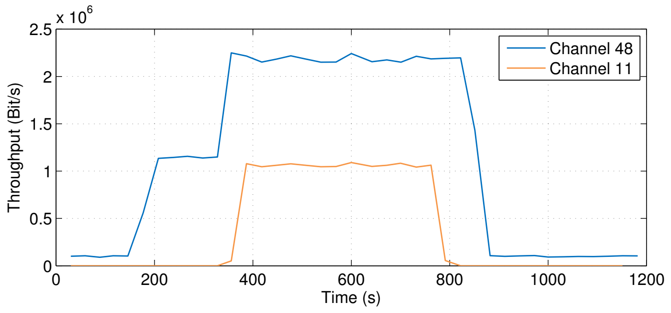
1) Interference Aware routing: In this scenario we illustrate how the SDN controller is able to redirect a data flow after detecting external interference. The setup used for this experiment is shown in Fig. 5a, which depicts the wireless channels of the links set up between the nodes, as well as an external interferer that operates on channel 48. During the bootstrapping phase, a set of forwarding rules is installed in the nodes directing traffic from *s0* to *s4* via *s1* and *s2* over channel 48, which does not suffer from noise. Then, a flow from *s0* to *s4* with a data rate of 2 Mbit/s is generated. Figure 6a depicts the throughput statistics gathered for the link between *s0* and *s1* at the controller. Figure 6b displays the channel loads on channels 11 and 48 reported by node *s1* to the controller, including the background traffic (channel 11, $\sim 10\%$) and signalling traffic (channel 48, $\sim 1\%$).

Once the 2 Mbit/s data flow enters the network, the load of channel 48 increases to approximately 15% (t=80 s). Around 300 s into the experiment, the interferer starts transmitting on channel 48, clearly visible by a sudden peak of channel load. At this point the SDN algorithm estimates a better network state and redirects the flow over channel 11, using *s3* as relay. After applying the reallocation, the load on channel 48 settles to a steady 30%. On the other hand, the channel load of channel 11 increases since now the flow is reaching *s4* via *s3*. At t=450 s, the interferer stops transmitting and the load on channel 48 drops to around 5%. Since the load on channel 11 is still above $u_{thr} = 15\%$, the controller decides to balance the network load again: shortly after the 500 s mark, the algorithm switches the data flow back to channel 48. This status is maintained until the flow stops at t=680 s and the network returns to its initial state.

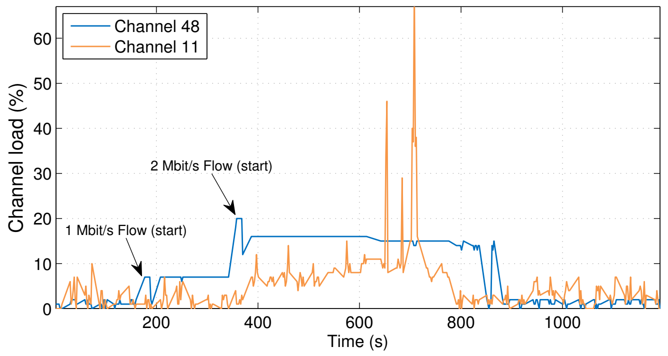
2) Flow balancing: The setup depicted in Fig. 5a is used again, however, without the interferer. At the beginning of the experiment, a 1 Mbit/s flow is generated, which is initially sent over channel 48 towards *s2* (Fig. 7a). The resulting channel loads observed by *s1*'s wireless interfaces are depicted in Fig. 7b.

A second 2 Mbit/s flow is introduced at around t=330 s and forwarded as well over *s2* on channel 48, leading to a peak in the reported channel load. Consequently, the SDN controller decides to balance the flows by redirecting the small 1 Mbit/s flow via *s3* over channel 11. This effectively balances

[3]A low threshold is chosen for the validation to provoke reallocation policies even at low traffic levels.



(a)



(b)

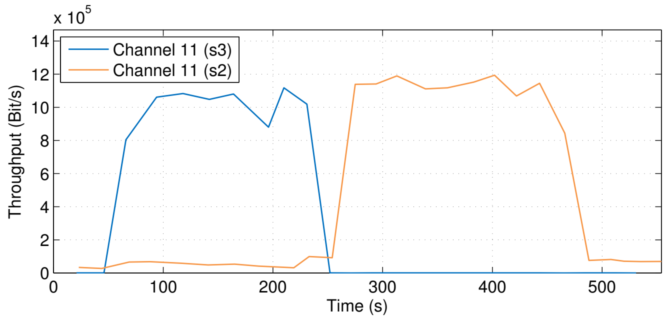
Fig. 7: Throughput measured over the links connecting *s1* with *s2* and *s3* (a) and channel load observed during the load balancing experiment.

the network load over the two available wireless channels (Fig. 7b). This configuration is maintained until the end of the experiment.

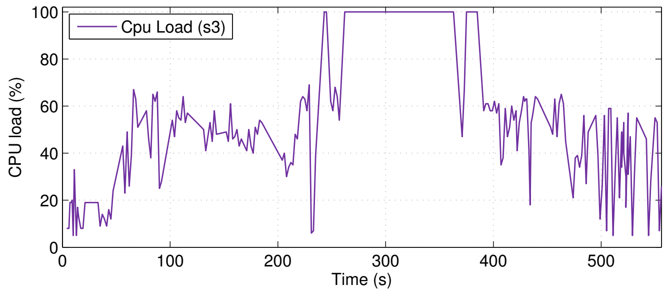
3) CPU aware routing: In the last experiment, the CPU aware routing policy is evaluated, using the setup depicted in Fig. 5b, where all links are on channel 11. A 1 Mbit/s flow is created at t=40 s, using *s3* as relay node between *s1* and *s4*, as can be derived from Fig. 8a. At approximately t=240 s, a CPU-intensive task is started in *s3*, boosting the CPU load to 100% (Fig. 8b). The SDN controller reacts to the increased CPU load by redirecting the flow over *s2*, thus alleviating *s3* from data forwarding tasks and granting it more processing power to perform the computing task. After *s3*'s processing finishes, the chosen path is maintained until the end of the experiment, because no further route changes are triggered due to CPU loads above the threshold.

IV. Conclusions

In this paper we present a wireless SDN-based architecture for networks of constrained edge devices. The proposed architecture features SDN control of wireless mesh networks, along with two path forwarding policies that are tailored to the requirements of networks of constrained devices: a network flow balancing policy that avoids heavy link congestion, and a CPU load aware policy that alleviates the workload of nodes by dynamically excluding from routing tasks for the data plane. Experiments carried out in a real testbed of constrained devices demonstrate that the architecture and the network policies are correctly implemented and executed.



(a)



(b)

Fig. 8: Throughput measured at node *s1* (a) and CPU load measured for node *s3* (b) during the CPU aware routing experiment.

V. Acknowledgements

The research leading to these results has received funding from the European Union under grant agreements 618098, and 671551 (H2020 5G-XHaul), and from the Spanish Ministry of Economy and Competitiveness (MINECO), under research grant TEC2013-47960-C4-4-P.

References

[1] Akyildiz, Ian F., Xudong Wang, and Weilin Wang. "Wireless mesh networks: a survey." Computer networks 47.4 (2005): 445-487.

[2] Bonomi, Flavio, et al. "Fog computing and its role in the internet of things." Proceedings of the first edition of the MCC workshop on Mobile cloud computing. ACM, 2012.

[3] Hurtado-Borras, A.; Pala-Sole, J.; Camps-Mur, D.; Sallent-Ribes, S., "SDN wireless backhauling for Small Cells," in Communications (ICC), 2015 IEEE International Conference on , vol., no., pp.3897-3902, 8-12 June 2015

[4] Seppnen, Kari, Jorma Kilpi, and Tapio Suihko. "Integrating WMN based mobile backhaul with SDN control." Mobile Networks and Applications 20.1 (2015): 32-39.

[5] Detti, Andrea, et al. "Wireless mesh software defined networks (wmSDN)." Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on. IEEE, 2013.

[6] Draves, Richard, Jitendra Padhye, and Brian Zill. "Routing in multi-radio, multi-hop wireless mesh networks." Proceedings of the 10th annual international conference on Mobile computing and networking. ACM, 2004.

[7] K shortest path implementation available at: http://jgrapht.org/javadoc/org/jgrapht/alg/KShortestPaths.html

[8] The OpenFlow Switch Specification. Available at: *https://www.opennetworking.org*

[9] OpenDayLight, available at: http://www.opendaylight.org/

[10] Raspbery B+, available at: https://www.raspberrypi.org/products/model-b-plus/