SDN Wireless Backhauling for Small Cells

Anna Hurtado-Borràs*, Jordi Palà-Solé*, Daniel Camps-Mur*, and Sebastià Sallent-Ribes*
*i2CAT Foundation
{anna.hurtado.borras, jordipalasole}@gmail.com, {daniel.camps, sebastia.sallent}@i2cat.net

***Abstract*—Small Cells are recognized as one of the key enablers to address the forecasted exponential traffic increase in future mobile networks. However, several major technical hurdles need to be overcome to fulfill the promise of Small Cells. In this paper we introduce a novel architecture that enables the application of Software Defined Networking (SDN) techniques on the wireless Small Cell backhaul. We demonstrate our architecture in a testbed prototype, where we analyze the costs incurred in the additional processing and the centralization of state imposed by SDN, while motivating the benefits of our architecture in terms of improved network management and control.**

I. Introduction

A $1000x$ increase in mobile traffic demand is considered one of the major challenges to be addressed by the mobile industry in the next ten years [1]. In order to tackle this problem, Small Cells, which increase capacity by means of reducing cell size and densifying access networks, are seen as one of the most promising solutions. However, a massive deployment of Small Cells poses significant technical hurdles to current network architectures. In particular, backhauling outdoor Small Cells, which may be mounted on lamp posts or street furniture, is a challenging problem that needs to be addressed with efficient wireless technologies. Figure 1 illustrates an example dense Small Cell deployment where wireless transport units are used between Small Cells to backhaul traffic in a multi-hop fashion until a wired transport unit connecting to the core network can be reached.

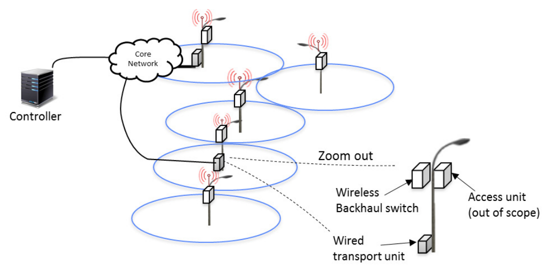


Fig. 1. Outdoor Small Cell deployment with wireless backahuling

Another important aspect of Small Cell deployments is the increased dynamics of the transported traffic flows. Specifically, reducing cell sizes results in an increase of traffic burstiness, as less terminals are simultaneously active in any given cell. Consequently, a tighter control of the resources in the wireless backhaul is needed, where instead of statically provisioning links for peak rates as in current mobile networks, links should be monitored and adapted to network dynamics. In this context, we claim that Software Defined Networking (SDN), where network state is conveyed to a central controller that obtains a holistic view of the network, is a key technology to enable the adaptability to network dynamics required in the Small Cell wireless backhaul.

Application of SDN techniques into wireless networks is currently a hot research topic, with contributions available in different domains. In [2] the authors propose an SDN based architecture for future mobile networks, where a controller is used to orchestrate heterogeneous wireless access technologies including aspects such as QoS and mobility. In this work though, only a high level architecture is proposed without details on the employed protocols or algorithms. In [3] the *Odin* controller is introduced, which implements certain upper MAC components, and can be used to orchestrate access points in a WLAN access network. A controller like Odin could also be used in Small Cell networks to control the access units illustrated in Figure 1. This work is further extended in [4]. Unlike [3] or [4] though, in this paper we investigate the application of SDN to control the backhaul network instead of the access network. To the best of the authors' knowledge the work in [5] is the closest to the one presented in this paper, since it introduces an architecture for Openflow based forwarding in mesh networks. However, the paper at hand advances the work in [5] in several aspects, such as minimizing the management overhead to configure the wireless nodes, and most significantly by extending Openflow to transport wireless parameters.

In particular, in this paper we introduce a novel SDN based architecture for the wireless Small Cell backhaul, which allows to easily reuse available SDN controllers to control wireless switches. In addition, we describe a prototype implementation and evaluate the potential benefits of our architecture using a virtual and a physical testbeds. This paper is organized as follows. Section II introduces our SDN architecture. Section III describes the implemented prototype and reports performance results. Finally, section IV summarizes the main findings and concludes the paper.

II. System Design

A. Overall architecture

Our goal is to design a system that allows an SDN controller to control the data plane forwarding in a wireless backhaul network. In order to implement this control effectively, the controller must be aware of the radio conditions. Figure 2 depicts the architecture of our system, where we can observe an SDN controller C that controls the operation of several wireless switches in the network through a south-bound interface Ext_{SB}. The middle switch in Figure 2 details the internal architecture of a wireless switch while identifying the following main components:

- One or more wireless devices D controlling access to the wireless medium in each wireless switch, where each device can be configured to operate independently. Without loss of generality we consider hereafter standard 802.11 devices as the wireless devices in our architecture, and note that while the implementation of some mechanisms in our system has been optimized for 802.11 radios, our proposed architecture is generic enough to encompass other radio technologies.
- One SDN agent A controlling the forwarding plane in the wireless switch and communicating to the remote SDN controller (C).
- A functional entity MUX that multiplexes multiple wireless links over a single wireless device D. For example in Figure 2 the wireless device operating in channel 1 is in communication range of two other devices, hence the MUX multiplexes two interfaces.
- A pair of interfaces Int_{DP_i} and Int_{CP} running between the SDN agent A and wireless device D, where: i) Int_{DP_i} transports the data plane packets to be transmitted over the wireless network, and ii) Int_{CP} is a control plane interface that allows the SDN agent to poll the wireless device D about the instantaneous radio conditions in the network.

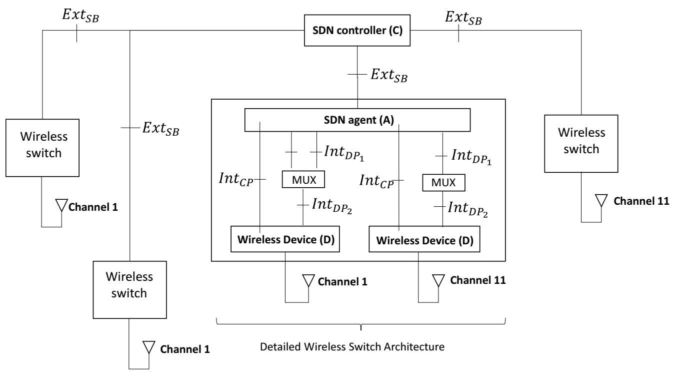


Fig. 2. Considered architecture.

A major requirement of our architecture is that it needs to be easily implementable using available SDN technologies and software components, which have been mostly developed for wired environments. This assumption is important because it simplifies the deployment of our architecture, it allows to reuse the numerous SDN developments performed for wired networks, and is inline with the work carried out in the ONF [6]. For this purpose, the southbound interface Ext_{SB} in our architecture is based on the OpenFlow protocol [7].

B. Data plane functions and interfaces

Given that Ext_{SB} is implemented using OpenFlow, the SDN agent A is essentially an OpenFlow switch. OpenFlow switches control forwarding through a set of Ethernet ports that compose the switch datapath, where each port provides a physical point to point connection to another switch in the network. Notice though that if instead of comprising a traditional Ethernet adapter an OpenFlow ports consists of a wireless interface, as in Figure 2, the situation changes because being wireless a broadcast medium, multiple switches in the network could be physically reached from a single OpenFlow *wireless* port.

However, considering physical point to multi-point connectivity in an OpenFlow network is potentially disturbing to existing SDN software components, as many applications (see for instance LLDP discovery [8]) heavily rely on the fact that physical links provide point to point connections. Therefore, in our architecture we decide to hide the point to multi-point capabilities of wireless devices from the SDN agent through the introduction of the MUX.

The MUX is a functional entity in charge of multiplexing data packets coming from the various Int_{DP1} interfaces over a single Int_{DP2} interface, and de-multiplexing them in the reverse direction. Due to its widespread availability, we have decided to use 802.1Q VLANs as a multiplexing mechanism. Notice that embedding a $4B$ VLAN tag in the data packets transmitted over the air incurs an additional overhead that we consider acceptable given the provided functionality. In addition, a VLAN tag allows to multiplex up to 4096 different Int_{DP1} interfaces, which is more than enough in our scenarios of interest. Thus, upon receiving a packet from an Int_{DP1_i} interface the MUX pushes $VLAN_i$ to the packet and forwards the packet through Int_{DP2}. On the other hand, upon receiving a packet from the Int_{DP2} interface, the MUX pops the packet's VLAN tag and forwards it to the appropriate Int_{DP1_i} interface. Later on we will discuss how VLAN tags are agreed upon between switches.

Finally, the interface Int_{DP2} carries VLAN tagged packets to the wireless device D, which, as will be explained later, decodes the neighboring switch the packet needs to be transmitted to from the embedded VLAN.

C. VLAN allocation

Given that VLAN tags are used in the MUX to multiplex data packets coming from the SDN agent, and de-multiplex packets from the wireless device, a mechanism is required to derive the appropriate VLAN tags to be used to represent each Int_{DP1} interface. In particular the designed mechanism should fulfill the following requirements:

- Upon receiving a packet from the MUX with $VLAN_i$, the wireless device D should be able to decode the destination neighboring switch for the packet, i.e. forwarding information is encoded in the VLAN.
- Upon receiving a packet from the wireless device D with $VLAN_i$, the MUX should be able to decode the Int_{DP1} interface the packet should be forwarded to. Notice that the MUX receives an Ethernet frame, i.e. the wireless header is strip by the wireless device D, and thus the

MUX is unaware of the address of the wireless switch that transmitted this packet.

Similar problems exist for instance in MPLS where a Label Distribution Protocol (LDP) is used to distribute labels among nodes [9]. Thus, a similar concept is used in our architecture, but instead of defining a new label distribution protocol we build upon the *Peer Link Management* (PLM) protocol available in any wireless device supported by the Linux kernel. PLM is a protocol defined as part of the 802.11s standard [10] (mesh extensions for 802.11), which allows wireless devices to establish a logical link with other neighboring devices configured to operate in the same wireless network. PLM works by monitoring the Beacon frame transmissions from neighboring devices, and implementing a two way handshake where devices exchange wireless capabilities and a locally unique identifier for the peer link, known as Local Link Identifier (LLID). Thus, after the PLM handshake completes a wireless device obtains a unique local identifier for the peer link (LLID), and the LLID value used by the neighboring device to refer to the same link, which is locally stored as the Peer Link ID (PLID). A simplified[1] example of the operation of the PLM protocol is depicted in the left part of Figure 3, where we can see how each wireless device D identifies potential neighboring links with a pair of $LLID/PLID$ values.

Our system builds on the exchanged $LLID/PLID$ values to derive the VLAN tags used to multiplex several Int_{DP_1} interfaces over a single wireless device. For this purpose though, the MUX needs to be aware of the $LLID/PLID$ values collected by the wireless device, which is possible using management tools for wireless interfaces available in the Linux kernel [11].

Algorithm 1 contains the description of the specific procedures executed in the MUX and in the wireless device D. In addition, the operation of the algorithm is illustrated through an example in the right part of Figure 3. As we can see in the figure, the MUX uses the LLID[2] value assigned by the wireless device to multiplex and de-multiplex the transmitted and received packets. In addition, upon having a new packet to transmit the wireless device D inspects the packet's VLAN tag, uses the contained $LLID$ value to discover the neighboring device the packet needs to be transmitted to, and finally overwrites the carried VLAN value to the corresponding $PLID$ for that link. Notice that the wireless device D needs to swap the VLAN value to $PLID$ because $PLID$ is the identifier for that link assigned by the device we are transmitting the packet to, and therefore it is ensured to be unique in the receiving device. Finally, notice that every packet received by the wireless interface is forwarded up to the MUX and the SDN agent where the forwarding intelligence is located.

[1]In reality for each link each device must initiate a two-way handshake, but only one handshake is shown in Figure 3 for simplicity.

[2]The wireless device D is modified to only assign LLIDs between 0 and 4096, which is the maximum VLAN ID value.

Algorithm 1: Mux/Demux and Inband setup algorithms

```
Variables:
$A \leftarrow$ SDN agent
$D \leftarrow$ wireless device
$MUX \leftarrow$ MUX function in Figure 2
$e_{MD} \leftarrow$ Ethernet device connecting the $MUX$ to $D$
$l \leftarrow$ Single wireless link characterized by $LLID/PLID/neighbour\_address$
$L \leftarrow$ List of all known wireless links by device $D$
$parent\_port \leftarrow$ Parent OF port in $A$ for inband set up
$child\_ports \leftarrow$ List of child OF ports in $A$ for inband setup
$IP_{ctrlr} \leftarrow$ IP address of the SDN controller $C$
$IP_{sw} \leftarrow$ IP of this wireless switch $A$
Executed in the $MUX$
Add $e_{MD}$ to $MUX$'s datapath
$L \leftarrow$ Poll $D$ for wireless links
for $l \in L$ do
    $(e_1, e_2) \leftarrow$ Create new veth pair to represent link $l$
    Add $e_1$ to $A$'s datapath
    Add $e_2$ to $MUX$'s datapath
    $rule_{mux} \leftarrow in = e_2 \Rightarrow out = e_{MD}$, push VLAN=$l.LLID$
    $rule_{demux} \leftarrow in = e_{MD}, VLAN = l.LLID \Rightarrow out = e_2$, pop VLAN
    Install $rule_{mux}$ and $rule_{demux}$ to $MUX$
Executed in the wireless device $D$
//Upon having to transmit a packet $P$
for $l \in L$ do
    if $P.VLAN = l.LLID$ then
        $P.VLAN = l.PLID$
        $P.dst\_addr = l.neighbour\_address$
Executed in the SDN agent $A$ for inband setup
// Rules dealing with ARP to/from controller
$arp_1 \leftarrow prot = arp, arp\_tpa = IP_{ctrlr} \Rightarrow out = parent\_port$
$arp_2 \leftarrow prio = 2, prot = arp, arp\_tpa = IP_{sw} \Rightarrow out = LOCAL$
$arp_3 \leftarrow prio = 1, prot = arp, arp\_spa = IP_{ctrlr} \Rightarrow out = child\_ports$
// Rules dealing to IP to/from controller
$ip_1 \leftarrow prot = ip, ip\_dst = IP_{ctrlr} \Rightarrow out = parent\_port$
$ip_2 \leftarrow prio = 2, prot = ip, ip\_dst = IP_{sw} \Rightarrow out = LOCAL$
$ip_3 \leftarrow prio = 1, prot = ip, ip\_src = IP_{ctrlr} \Rightarrow out = child\_ports$
Install $arp_1$, $arp_2$, $arp_3$, $ip_1$, $ip_2$ $ip_3$ in $A$
```

D. Control plane interface and OpenFlow extensions

In order to effectively control the wireless medium, the SDN controller C in Figure 2 needs to be aware of the radio conditions. Thus, two interfaces in Figure 2 tackle the problem of bringing radio information to the controller. Firstly, the interface Int_{CP} allows the SDN agent A in the wireless switch to retrieve measured radio conditions from the wireless device D. Secondly, the interface Ext_{SB} is used to transport the collected radio parameters from the SDN agent to the SDN controller.

Thus, the SDN agent may use at any time interface Int_{CP} to poll the wireless device for the latest radio conditions, where the wireless device reports radio conditions for each link that has been discovered through PLM handshakes. The implementation of this interface is described in section III-A.

Finally, the Ext_{SB} interface is implemented using OpenFlow 1.0 extended to be able to transport radio parameters. Specifically, in our implementation the ofp_port_stats structure defined in OpenFlow, which contains per-port statistics and is issued by the SDN agent A upon receiving a port statistics request from the controller is modified to include

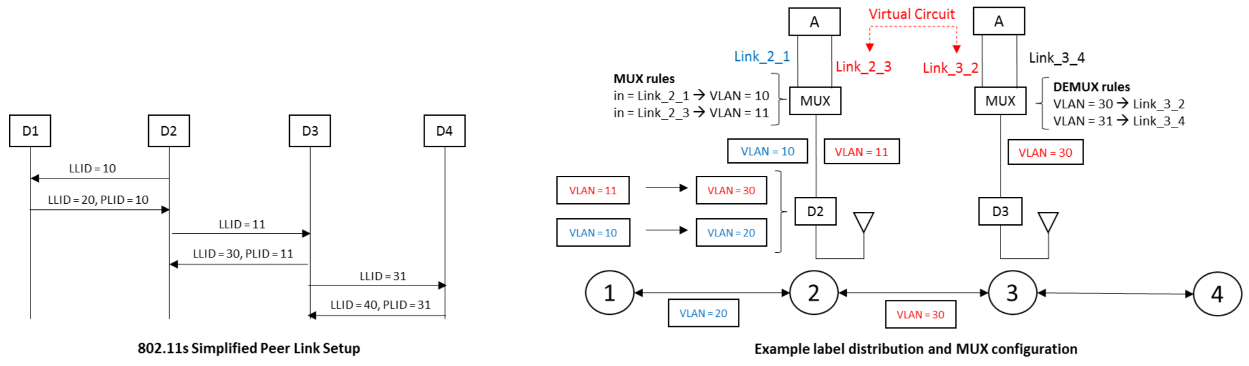


Fig. 3. Example VLAN distribution mechanism

the per-port radio parameters depicted in Table I.

TABLE I
OPENFLOW PER-PORT RADIO PARAMETERS

Parameter	Explanation
rx_bytes	Received bytes for this port in the wireless device
rx_packets	Received packets for this port in the wireless device
tx_bytes	Transmitted bytes for this port in the wireless device
tx_packets	Transmitted packets for this port in the wireless device
tx_retries	Retried packets for this port in the wireless device
tx_failed	Failed packets for this port in the wireless device
signal	Last signal level in dBm received in this link
avg_signal	Average signal level in dBm received in this link
tx_bitrate	Last transmitted PHY bitrate in this link

E. In-band connectivity to the controller

In Data-Centers where OpenFlow is currently being deployed it is common to use a separate network to transport the signaling connection between the OpenFlow switches and the OpenFlow controller. Notice though, that this is not possible in our Small Cell scenario where the signaling connection needs to be transported through the same wireless interfaces that compose the OpenFlow datapath. Such a way of connecting to an OpenFlow controller is known as an *in-band* connection.

In-band connectivity though poses a bootstrapping problem, because a path needs to be available for an OpenFlow switch to establish an initial contact to the controller, but in a normal deployment the switch tables are initially empty and can only be populated after establishing the OpenFlow channel with the controller. To resolve this dilemma, in our implementation the SDN agent A pre-populates the OpenFlow tables according to the rules depicted in the lower part of Algorithm 1. The employed algorithm relies on having an OpenFlow port in the agent A designated as the *parent* port to reach the controller, and a set of other ports designed as *child* ports that will be used to relay controller signaling to other wireless switches in the network. The algorithm requires to perform the selection of parent and child interfaces offline, and to provision the wireless switches in sequential fashion starting from a switch that has a wired backhaul connection to the controller (see Figure 1). Notice, though that this process is required only once at installation time, and that the actual path followed by the signaling connection can be optimized from the controller once the initial connection has been established. We leave the study of optimized bootstrapping mechanisms as future work.

TABLE II
PROTOTYPE COMPONENTS

Component	**Open Source**	**Custom extensions**
Wireless Device	mac80211 [11]	Extended to process incoming packets according to Algorithm 1
MUX	Openvswitch [12]	Configured according to Algorithm 1
SDN Agent	Openvswitch [12]	Extended to query wireless driver and report wireless parameters to SDN controller
SDN Agent ↔ Wireless Device interface	nl80211 [11]	No extensions required
SDN controller	OpenDayLight [13]	OSGI bundle defined to control forwarding

F. SDN controller

Following our architecture, any available SDN controller based on Openflow can be used to control the wireless switches in the Small Cell backhaul network.

III. PERFORMANCE EVALUATION

Next we describe a prototype implementation of our architecture, and an evaluation carried out to evaluate the costs and benefits introduced by SDN in the wireless backhaul.

A. Prototype and testbeds

Our wireless SDN implementation is based on open source components available in Linux platforms. Table II details the software components used to build a prototype following the architecture defined in Figure 2, where the basic software components and the required customizations are indicated.

In addition, our prototype implementation has been evaluated in two different testbeds:

- A *virtual* testbed that allows us to test arbitrary network topologies on a single machine. This testbed is implemented using virtualization capabilities of the Linux

kernel such as *mac80211_hwsim* [11] regarding wireless drivers and *namespaces* to instantiate several switches on a single machine. A Toshiba Vaio at 2.3 GHz and 4 GB RAM is used for this purpose.
- A *physical* testbed illustrated in Figure 4, implemented using laptops running Linux and commercial 802.11g wireless interfaces.

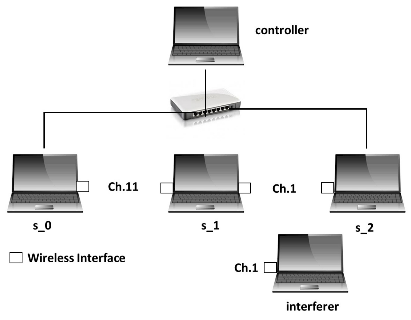


Fig. 4. Physical testbed

B. Results

1) Virtual Testbed: The architecture illustrated in Figure 2 enables SDN control of wireless switches, but introduces new processing elements in the data plane that may impair the achieved wireless throughput. In order to evaluate the introduced penalty, Figures 5(a) and 5(b) depict respectively the maximum throughput and average delay achieved in a linear chain topology, while comparing our SDN solution to 802.11s [10], which is a distributed mesh forwarding solution available in the Linux kernel.

Notice that our virtualization testbed emulates the wireless medium using the *mac80211_hwsim* kernel module, which forwards packets between wireless interfaces of different nodes. Consequently, the maximum achievable throughput in this case is not limited by the capacity of the wireless interface but rather by the processing capacity of the *mac80211_hwsim* module, and the rest of data plane processing elements. Figure 5(a) illustrates how the 802.11s solution achieves a higher forwarding throughput than our SDN solution, where throughput is measured between the first and last node of the linear chain using the $iperf$ tool to perform a UDP transfer. This result is as expected because in our architecture additional modules like the MUX and the SDN agent are involved in packet processing. However, the achieved throughput in the SDN case, which is above 250 Mbps for a single hop, is sufficiently high not to become a bottleneck when realistic wireless interfaces are considered.

Figure 5(b) compares the latencies between the first and last nodes in our linear chain experiment measured with the $ping$ utility. As expected, 802.11s delivers a reduced delay, because less processing elements are involved in the data plane. However, in the SDN case the introduced end to end delay is below 2 ms for a linear chain of six nodes, which is a long chain for our scenarios of interest.

Another penalty introduced by any SDN solution is the signaling cost of transferring network state from the network elements to the central controller. Figure 5(c) measures the *over-the-air* overhead introduced by the SDN controller, which periodically retrieves statistics from the wireless switches in our linear chain topology. A polling interval in the controller of 10 seconds has been considered in this experiment, which is the default value in our OpenDayLight controller implementation [13]. We leave as future work an optimal tuning of this parameter. As depicted in the figure, signaling increases linearly with the number of switches. The results are compared to the signaling introduced by 802.11s, which is configured as a reactive protocol triggering path set-up upon reception of new packets. As expected an SDN solution introduces higher signaling overhead than a distributed protocol like 802.11s, but still limited to a 0.25% over the air overhead for a linear chain of six nodes.

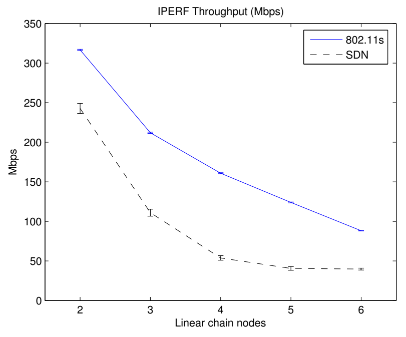
2) Physical Testbed: To illustrate the potential benefits of our wireless SDN solution, we consider the testbed illustrated in Figure 4 and two example applications. These applications are included hereafter to illustrate the potential benefits of the developed architecture, while an exhaustive study of the applications themselves is left as future work.

The first application consists of using the SDN controller as a *logging* element, which tracks and records relevant metrics of each wireless link. For this purpose Figure 6(a) illustrates the *PHY rate* and *Signal Level* metrics reported by the left and right links of the s_1 device in Figure 4, operating respectively in channels 1 and 11 in the 2.4 GHz band, where metrics are reported at a 10 second interval. Gathering performance metrics of each wireless link in the central controller can enable multiple applications such as network diagnostics, forensic or traffic pattern analysis, which can be used to optimize the performance of the wireless Small Cell backhaul.

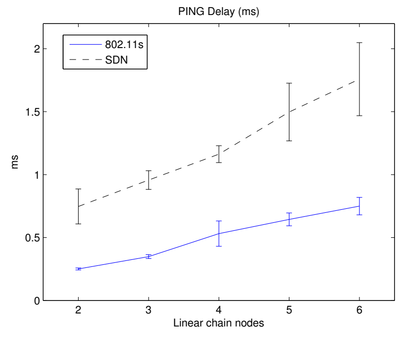
While the first application dealt with network management and offline optimization, the second application illustrates the use of the SDN controller for online network optimization. In Figure 6(b) the SDN controller is used to re-route backhaul flows in presence of interference. In particular, the upper part of Figure 6(b) depicts the instantaneous throughput achieved by 802.11s when transmitting a UDP transfer between device s_1 and the controller node in Figure 4, where the UDP transfer is overlapped by a backlogged interferer in channel 1 operating between 300 seconds and 450 seconds. As depicted in the figure, throughput is heavily impaired in presence of the interferer. Instead, the lower part of Figure 6(b) illustrates the potential of SDN to mitigate interference in this case, where upon detecting a reduction in throughput in the $s_1 \leftrightarrow s_2$ link the controller immediate switches the flow through the $s_1 \leftrightarrow s_0$ link, which operates in a different channel, thus leaving the flow unaffected by the interferer.

IV. Conclusions

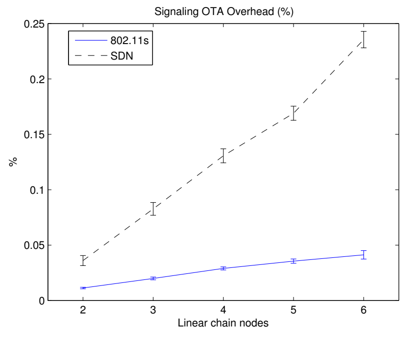
In this paper we have introduced a novel architecture that enables SDN control of wireless switches, and allows to report wireless specific parameters between a wireless device and an



(a) Linear chain Throughput (Mbps)

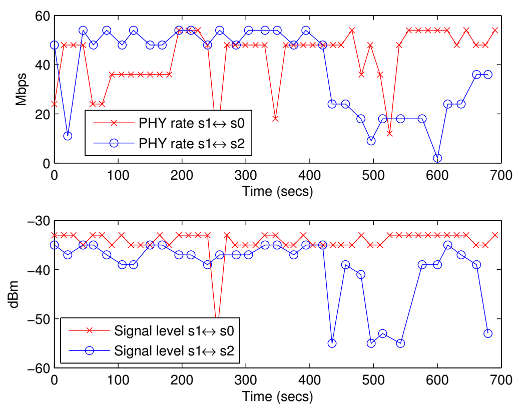


(b) Linear chain Delay (ms)

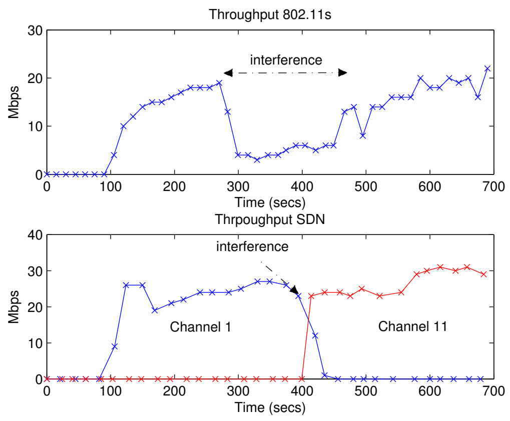


(c) Linear chain signaling OTA overhead

Fig. 5. SDN vs 802.11s evaluation on virtual testbed.



(a) Per-link wireless metrics collected at the SDN controller



(b) Throughput under interference for 802.11s (above) and SDN (below)

Fig. 6. SDN benefits evaluated on physical testbed.

SDN controller. Our architecture has been evaluated by means of quantifying the introduced packet processing and signaling overhead, whilst illustrating the potential of the presented architecture to improve management and control of the Small Cell wireless backhaul. Evaluating the proposed architecture in a larger wireless testbed, as well as further investigating the presented applications, are considered the main directions for future work.

V. Acknowledgements

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement n 618098, and from the Spanish Ministry of Economy and Competitiveness (MINECO), under research grant TEC2013-47960-C4-4-P.

References

[1] Qualcomm, *Rising to Meet the 1000x Mobile Data Challenge*, available at:https://www.qualcomm.com/media/documents/files/rising-to-meet-the-1000x-mobile-data-challenge.pdf

[2] Bernardos, C.J.; De La Oliva, A; Serrano, P.; Banchs, A; Contreras, L.M.; Hao Jin; Zuñiga, J.C., *An architecture for software defined wireless networking,* Wireless Communications, IEEE , vol.21, no.3, pp.52,61, June 2014

[3] S. Lalith, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao, *Towards Programmable Enterprise WLANs With Odin,* in Proc. Workshop on Hot Topics in Software Defined Networking (HotSDN '12), 2012.

[4] J. Schulz-Zander, N. Sarrar, S. Schmid, (2014, August). *Towards a scalable and near-sighted control plane architecture for WiFi SDNs.* In Proceedings of the third workshop on Hot topics in software defined networking (pp. 217-218). ACM.

[5] Peter Dely, Andreas Kassler, Nico Bayer. *OpenFlow for Wireless Mesh Networks,* IEEE International Workshop on Wireless Mesh and Ad Hoc Networks (WiMAN 2011), Hawaii, USA, Aug. 2011.

[6] ONF Wireless and Mobile Working Group, available at: *https://www.opennetworking.org/working-groups/wireless-mobile*

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

[8] Sherwood, R., Gibb, G., Yap, K. K., Appenzeller, G., Casado, M., McKeown, N., and Parulkar, G. (2009). *Flowvisor: A network virtualization layer*. OpenFlow Switch Consortium, Tech. Rep.

[9] Luc De Ghein, *MPLS Fundamentals*, Cisco Press, 2006.

[10] Hiertz, G. R., Denteneer, D., Max, S., Taori, R., Cardona, J., Berlemann, L., and Walke, B. (2010). *IEEE 802.11 s: the WLAN mesh standard.* Wireless Communications, IEEE, 17(1), 104-111.

[11] Vipin, M., and Srikanth, S. (2010, January). *Analysis of open source drivers for IEEE 802.11 WLANs.* In Wireless Communication and Sensor Computing, 2010. ICWCSC 2010. International Conference on (pp. 1-5). IEEE.

[12] The Openvswitch project. Available at: *http://openvswitch.org/*

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