WiseHAUL: An SDN-empowered Wireless Small Cell Backhaul testbed

José Núñez-Martínez, Jorge Baranda, Iñaki Pascual, Josep Mangues-Bafalluy
Centre Tecnològic de Telecomunicacions de Catalunya (CTTC)
Av. Carl Friedrich Gauss, 7, 08860 Castelldefels (Barcelona), Spain
e-mail: [mails: name.surname]@cttc.cat

***Abstract*—The deployment of dense networks of Small Cells (SC) is one of the key components of 5G mobile networks and will pose several challenges to the backhaul network. Software-Defined Networking (SDN) is a key technique to cope with the increased management complexity of such heterogeneous deployments, helping in the task of achieving a system-wide network management, which includes the 5G mobile wireless backhaul. Therefore, it is fundamental to devise the tools for the evaluation of 5G backhauling SDN architectures. This paper presents WiseHAUL: an SDN-empowered Wireless Small Cell Backhaul testbed. We present the design considerations of WiseHAUL, deployed in the form of an hypercube mesh that combines heterogeneous wireless technologies with the final goal of conducting SDN research and experimentation. Preliminary experimental results serve to identify the main research challenges faced by canonical SDN management techniques in an all-wireless backhaul deployment, hence acting as key drivers for further re-design and testbed development towards hybrid SDN models.**

I. Introduction

Dense deployments of small cells (SC) are seen as one of the most promising solutions to cope with the forecasted increase in mobile traffic demands. Such densification in the access network segment affects the backhaul segment. These deployments will be driven by several challenges, such as fiber availability or installation costs. Hence, it is recognized that wireless backhaul deployments, particularly those close to the edge, will be cost-effective, fast, and flexible. In such a context, the programming capabilities brought by Software Defined Networking (SDN) may become a key component to improve the management of wireless backhaul resources in addition to ease their dynamic optimization.

Currently, the application of SDN techniques in wireless networks is receiving a lot of attention not only from the academic sector, but also from industry. For instance, the Open Networking Foundation (ONF) has a dedicated group to cover the wireless and mobile use case [1]. In this short paper, the authors present the current deployment status and initial experiments of a testbed aiming to study the application of SDN techniques for the control of a wireless mesh backhaul. This paper is organized as follows. Section II explains the design conditions driving the deployment of this testbed. Section III presents the characteristics of the deployed infrastructure from the hardware and the software perspectives. Section IV describes the initial experiments conducted in the testbed, which illustrate the problems faced when applying the canonical SDN concept to an all-wireless deployment. These problems are the main drivers for further testbed development. The paper concludes with Section V.

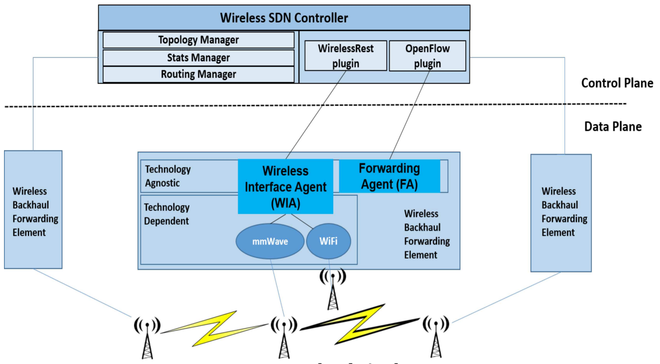


Fig. 1. Wisehaul Architecture.

II. Testbed Design

WiseHAUL follows the canonical principles of SDN models, where control plane and data plane are decoupled. Specifically, the control plane is performed by the Wireless SDN controller, while the data plane forwarding is carried out by the Wireless Backhaul Forwarding Elements (WBFEs), as depicted in Figure 1. The following subsections present a description of these components.

A. Wireless Backhaul Forwarding Element

As showed by Figure 1, the WBFEs forming the data plane include technology agnostic and technology dependent agents. The former is the Forwarding Agent (FA), whereas the latter is the Wireless Interface Agent (WIA). Each WBFE will feature one (or more) logical switch(es) that will be configured by these two agents in accordance with the rules received from the wireless SDN controller.

The FA, based on OpenFlow (OF) protocol [2], is in charge of adding, modifying, and deleting technology agnostic flow entries to performs action on packets in transit during initialization and run time. They are technology agnostic since the matching headers taken into account for datapath configuration have an equivalent frame format (e.g., IP, UDP, TCP, headers), which is agnostic of the underlying transport wireless technology (e.g., mmWave, WiFi, and microwave). It is important to note that the OF protocol was not designed to address the configuration of heterogeneous wireless interfaces. For this reason, we propose to embed an additional agent: the Wireless Interface Agent (WIA).

The WIA is a technology dependent agent in charge of configuring the underlying heterogeneous wireless technologies, which form the ports of the logical switches in the WBFE. The WIA integrates interfaces of heterogeneous technologies (e.g., 802.11ac and mmWave), which will be configured in a technology-agnostic or technology-specific way depending on the parameter to configure. The features of these wireless ports are exposed to the controller by means of the appropriate information models. In these information models, there are wireless configuration parameters that are common to several technologies, that is, technology-agnostic (e.g., modulation and coding scheme, channel bandwidth, transmission power), while other parameters are technology dependent (e.g., the beamwidth or the algorithm used to split the available bandwidth among the associated neighbors in the case of mmWave). These wireless link configuration parameters can be subject of modifications not only during configuration time but also during run time by the controller or by the own WBFE.

B. Wireless SDN Controller

Three basic modules compose the wireless SDN controller:

- A *topology manager module*, which is a database with the global network view of the wireless backhaul.
- The *stats manager module*, which contains a database with statistics of the network. The wireless SDN controller programs the nodes so they periodically report statistics to the SDN controller.
- The *routing manager module*, which based on the network vision and the reported statistics, configures the forwarding tables of the nodes with the corresponding *OFPT_FLOW_MOD* messages.

Furthermore, the wireless SDN controller interacts with the WBFEs through two different plugins:

- An *OF plugin* is in charge of interacting with the FA in the WBFE for modifying the forwarding rules. An OF interface is selected to allow the configuration of forwarding rules of different hardware and software switches.
- A *REST plugin* to interact with the wireless interface agent (WirelessRest plugin in Figure 1) for configuring the datapath. A REST API was elected in this case due to its simplicity, since it runs over HTTP and allows dealing with data models defined using Yang modeling language [3]. These data models can define the internal structure of a wireless backhaul interface or the number of virtual switches instantiated in that forwarding element.

Regarding the control plane network, Wisehaul proposes three different options, which may be understood as a stepwise approach from more ideal to more constrained deployments:

- *Wired out-of-band control network*. This would allow focusing on the data plane performance and could be used as benchmark for the more realistic options described below.
- *Wireless out-of-band control network*. Some deployments may allow an out-of-band wireless control network.
- *Wireless in-band control plane*. The control plane will share wireless resources with the data plane, and given the dynamicity and lower reliability of wireless links, this may pose some problems to the performance of the whole network. In this sense, this constitutes the more demanding case.

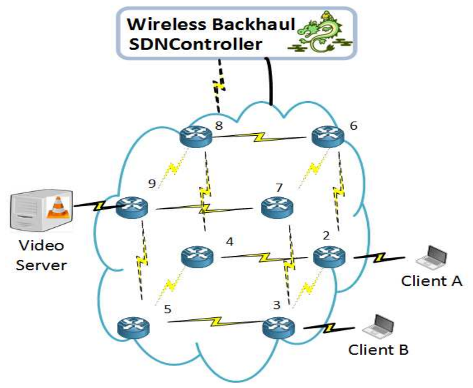


Fig. 2. Initial testbed deployment in an hypercube disposition counting with 8 forwarding nodes controlled by a single SDN controller.

III. Testbed Implementation

A. Hardware platform

The testbed consists of 9 programmable backhaul nodes based on PC engines Intel Core i7 platform (6-Core 3.3Ghz x86 CPU and 32GB of RAM), where each node runs Ubuntu 15.04 as operating system, as depicted in Figure 2.

Currently, each node is equipped with three Compex WLE900VX IEEE 802.11ac/a/b/g/n cards, one integrated Intel IEEE 802.11ac/a/b/g/n card and two Gigabit Ethernet ports. In particular, Compex cards will be used for data plane forwarding in the wireless backhaul network while the control plane can be wireless or wired, depending on the configuration of the remaining wireless interface and one of the Gigabit Ethernet ports. The remaining wired interface is used for the remote management of the different backhaul nodes. In a near future, each node will also include additional IEEE 802.11ad wireless cards (i.e., mmWave), increasing the heterogeneity of technologies included in the backhauling nodes.

B. Software platform

Regarding software components, our testbed relies on available SDN technologies and open source software components to simplify its deployment and allowing the reuse and extension of such elements for the wireless case. In particular, the SDN controller runs the latest available version of the Ryu SDN framework [4]. It is worth stressing that, in the architecture, new services and algorithms are likely to be deployed in the form of network applications on top of the Ryu controller by exploiting its native REST interface. At the data plane level, each node uses xDPd version 0.7.5 [5] as software switch running OF v1.3 specification [2].

IV. Preliminary Evaluation Results

A. End-to-end path establishment

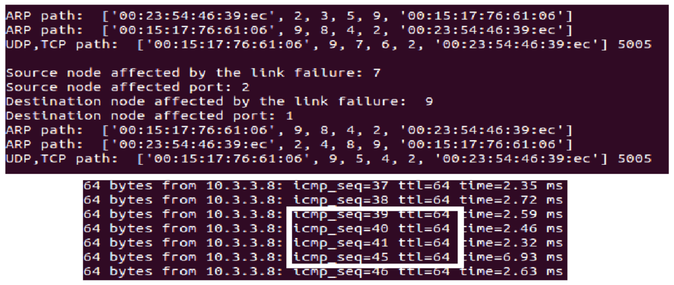
We evaluate our fully reactive OF setup by measuring the connection time between the *client A* establishing an ICMP connection to the *video server* in Figure 2 when using a wired and a wireless control plane. In the wired case, each forwarding node is connected to the SDN controller by means of a dedicated Gigabit Ethernet link, while in the wireless case, WFBEs use a WiFi link sharing the same access medium to connect with the SDN controller.

We observe that the wired control plane is able to set up the connection before the one using the wireless control plane. In particular, for the wired case, the client node loses 6 ICMP packets before obtaining an answer from the server node, while in the wireless case, 14 ICMP packets are lost. The contrast between results corresponds to the reliability and different characteristics of the considered control plane networks, hence, affecting to the establishment of forwarding rules. For instance, the measured time for the initialization of ARP rules takes around 38.4ms in the wireless case, while 27.98ms in the wired case. In next experiments, the control plane is performed by means of the wired connection because we want to focus on data plane aspects.

B. Path recomputation

This experience evaluates the recovery time after wireless link failures in the data plane, which are more likely to happen than in a wired data plane. We emulate a link failure by bringing down manually a wireless interface. This event is reported to the Wireless SDN controller which triggers the update of the network vision at the topology vision module. Then, the controller deletes the flow tables of all the WBFE and recalculates paths upon reception of new OF *OFPT_PACKET_IN* messages.

Figure 3 represents both the path recomputation process (above) and the time required to reestablish paths after a wireless link failure (below). The video server sends two simultaneous flows (ICMP and UDP video streaming) to the client. The upper part of the figure illustrates the routes in place before link failure occurs (including the MAC addresses of source and destination). The ICMP trace in the lower part of Figure 3 shows that around three seconds (three ICMP packets lost) are required to reestablish the path.



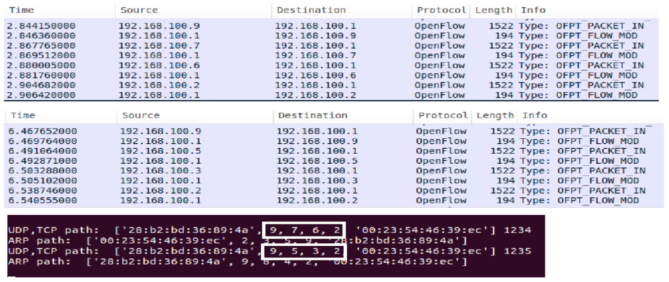
```
ARP path:  ['00:23:54:46:39:ec', 2, 3, 5, 9, '00:15:17:76:61:06']
ARP path:  ['00:15:17:76:61:06', 9, 8, 4, 2, '00:23:54:46:39:ec']
UDP,TCP path:  ['00:15:17:76:61:06', 9, 7, 6, 2, '00:23:54:46:39:ec'] 5005

Source node affected by the link failure: 7
Source node affected port: 2
Destination node affected by the link failure:  9
Destination node affected port: 1
ARP path:  ['00:15:17:76:61:06', 9, 8, 4, 2, '00:23:54:46:39:ec']
ARP path:  ['00:23:54:46:39:ec', 2, 4, 8, 9, '00:15:17:76:61:06']
UDP,TCP path:  ['00:15:17:76:61:06', 9, 5, 4, 2, '00:23:54:46:39:ec'] 5005
```

```
64 bytes from 10.3.3.8: icmp_seq=37 ttl=64 time=2.35 ms
64 bytes from 10.3.3.8: icmp_seq=38 ttl=64 time=2.72 ms
64 bytes from 10.3.3.8: icmp_seq=39 ttl=64 time=2.59 ms
64 bytes from 10.3.3.8: icmp_seq=40 ttl=64 time=2.46 ms
64 bytes from 10.3.3.8: icmp_seq=41 ttl=64 time=2.32 ms
64 bytes from 10.3.3.8: icmp_seq=45 ttl=64 time=6.93 ms
64 bytes from 10.3.3.8: icmp_seq=46 ttl=64 time=2.63 ms
```

Fig. 3. Path recomputation in the case of detecting a link event failure.

C. Load balancing

We implemented a load balancing application on top of the SDN controller to select the more proper paths on a hop-by-hop basis. Based on the utilization observed in the available ports of a switch this application selects the port that presents the lower utilization and minimizes the number of hops to reach the destination. In particular, we transmit two video streams from the *server node* to *client A* in Figure 2. In order to do so, the SDN controller configures the switches to report statistics via OF *OFMP_PORT_STATS* messages to the SDN controller every second, adding further control overhead. Based on the utilization rate of the ports, the routing manager embedded in the SDN controller is able to compute the less loaded available shortest path, that is, the one presenting less utilization on its ports on a hop-by-hop basis. The upper part of Figure 4 shows the Wireshark trace of the exchanged OF messages between the SDN controller and each traversed switch forming the path showing the pair (OF *OFPT_PACKET_IN*, OF *OFPT_FLOW_MOD*), hence demonstrating the hop-by-hop nature of the load balancing policy. The lower part illustrates the two disjoint paths formed by OF switches, that is, the nodes labeled as the set (9,7,6,2) and (9,5,3,2), as showed by Figure 4.



```
Time          Source           Destination      Protocol Length Info
2.844150000   192.168.100.9    192.168.100.1    OpenFlow   1522 Type: OFPT_PACKET_IN
2.846360000   192.168.100.1    192.168.100.9    OpenFlow    194 Type: OFPT_FLOW_MOD
2.867765000   192.168.100.7    192.168.100.1    OpenFlow   1522 Type: OFPT_PACKET_IN
2.869512000   192.168.100.1    192.168.100.7    OpenFlow    194 Type: OFPT_FLOW_MOD
2.880005000   192.168.100.6    192.168.100.1    OpenFlow   1522 Type: OFPT_PACKET_IN
2.881760000   192.168.100.1    192.168.100.6    OpenFlow    194 Type: OFPT_FLOW_MOD
2.904682000   192.168.100.2    192.168.100.1    OpenFlow   1522 Type: OFPT_PACKET_IN
2.906420000   192.168.100.1    192.168.100.2    OpenFlow    194 Type: OFPT_FLOW_MOD
```

```
Time          Source           Destination      Protocol Length Info
6.467652000   192.168.100.9    192.168.100.1    OpenFlow   1522 Type: OFPT_PACKET_IN
6.469764000   192.168.100.1    192.168.100.9    OpenFlow    194 Type: OFPT_FLOW_MOD
6.491064000   192.168.100.5    192.168.100.1    OpenFlow   1522 Type: OFPT_PACKET_IN
6.492871000   192.168.100.1    192.168.100.5    OpenFlow    194 Type: OFPT_FLOW_MOD
6.503288000   192.168.100.3    192.168.100.1    OpenFlow   1522 Type: OFPT_PACKET_IN
6.505102000   192.168.100.1    192.168.100.3    OpenFlow    194 Type: OFPT_FLOW_MOD
6.538746000   192.168.100.2    192.168.100.1    OpenFlow   1522 Type: OFPT_PACKET_IN
6.540555000   192.168.100.1    192.168.100.2    OpenFlow    194 Type: OFPT_FLOW_MOD
```

```
UDP,TCP path:  ['28:b2:bd:36:89:4a', 9, 7, 6, 2, '00:23:54:46:39:ec'] 1234
ARP path:  ['00:23:54:46:39:ec', 2, 3, 5, 9, '28:b2:bd:36:89:4a']
UDP,TCP path:  ['28:b2:bd:36:89:4a', 9, 5, 3, 2, '00:23:54:46:39:ec'] 1235
ARP path:  ['28:b2:bd:36:89:4a', 9, 8, 4, 2, '00:23:54:46:39:ec']
```

Fig. 4. Load balancing: videos from the same source towards the same destination follow different disjoint paths.

V. Conclusions and Future Work

This paper presents Wisehaul, focusing on all-wireless backhauls managed with SDN. We conduct some experiments to illustrate the challenges that an all-wireless network may pose with a canonical SDN model based merely on OF protocol to configure forwarding tables of backhaul nodes. In particular, the time required to establish paths on a reactive way using different alternatives for the control plane, the time devoted to reconfigure the network, and the amount of overhead required to perform fine-grain load balancing decisions are some of those challenges. As future work, we plan to extend WiseHAUL functionalities by means of including distributed intelligence in backhaul nodes, using models such as that defined in [6].

Acknowledgment

This work has received funding from 5G-Crosshaul project (H2020-671598) and grant TEC2014-60491-R (5GNORM).

References

[1] ONF Wireless and Mobile Working Group (WMWG), available at: https://www.opennetworking.org/technical-communities/areas/specification

[2] Open Networking Foundation, OpenFlow Switch Specification (Version 1.3.0), June 2012

[3] M. Bjorklund, YANG - A Data Modeling Language for he Network Configuration Protocol (NETCONF), RFC 6020 (Proposed Standard), Internet Engineering Task Force, Oct. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc6020.txt

[4] Ryu SDN framework, available at: http://osrg.github.io/ryu/

[5] The eXtensible OpenFlow datapath daemon (xDPd), available at: http://xdpd.org/

[6] J. Núñez, J. Baranda, J. Mangues, *A Service-based model for the Hybrid Software Defined Wireless Mesh Backhaul of Small Cells* , in Proc. of the 2nd Int. Workshop on Management of SDN and NFV Systems (ManSDN/NFV), 09-13 November 2015, Barcelona (Spain)