WizHaul: An Automated Solution for vRAN Deployments Optimization

Josep Xavier Salvat, Andres Garcia-Saavedra, Xi Li, Xavier Costa-Perez
NEC Laboratories Europe, Germany.
E-mail: {josep.xavier.salvat, andres.garcia.saavedra, xi.li, xavier.costa}@neclab.eu

***Abstract*—Future 5G deployments will support a flexible split of Base Station (BS) functions, i.e., it will be possible to decide which atomic operations will be co-located on the edge and which ones will be processed on a Central Unit (CU). Thus, network owners will be able to decide how much centralization they would like to retain in different deployments. However, deciding which BS components should be offloaded to a CU becomes a challenge because routing and BS function placement choices are coupled. We present WizHaul, a software framework enabling the implementation of a centralized functional split decision-making engine for future 5G networks. The purpose of WizHaul is twofold. First, it may be used in a network planning phase to settle the optimal amount of centralization. Second, it may also be used to support network automation/adaptation scenarios where network failures or congestion in the cloud may draw the current configuration infeasible.**

I. Introduction

Classic mobile network architecture features Base Stations (BSs) geographically distributed and connected to an Evolved Packet Core (EPC) through a backhaul network [1]. This distributed architecture has several limitations such as high cost and difficult operation, among others [2]. However, 5G will leverage on centralized architectures such as Cloud Radio Access Networks (C-RAN). C-RAN places all functions of each BSs into a centralized cloud computing platform. In this way, each BS is decoupled into two main building blocks: (i) Radio Units (RUs) and (ii) a Central Unit (CU). The CU receives data from the EPC, performs centralized baseband processing and exchanges digitized radio samples with the RUs through high-capacity fronthaul (FH) links [3]. Centralization of BS functions has shown major benefits such as grater spectrum efficiency due to joint signal processing, simplified network operations and management and lower capital and operating expenditures, among others [4], [5].

Due to its benefits, C-RAN architecture has stood out as a key technology for future 5G networks. Nevertheless, current C-RAN deployments come at a high cost that hinders its development for 5G. Namely, fronthaul links need to exchange raw radio information in a timely manner, within a tiny time window of a few microseconds [6]. Furthermore, the bandwith requirement on these links not only is substantially higher than the user data rates, but it also grows linearly with the number of antennas [7]. This renders the use of technologies such as massive MIMO infeasible. Therefore, fronthaul networks exhibit tight network bandwidth and delay requirements which can only be met using high-cost fiber links employing serial line interfaces [8]. As a result, current backhaul deployments are not suitable for *fronthauling* [7], [9].

To overcome the aforementioned limitations, both industry and academia are pushing for a re-design of the fronthaul interface that could steer the adoption of centralized architectures on 5G deployments [9], [10]. Two key ideas are driving the fronthaul re-design: the future fronthaul interface will be based on a (i) packet-based network rather than on a point-to-point architecture and (ii) will support a flexible functional split of BS functions [7], [11], [12]. The former idea will fuel the use of statistical multiplexing, infrastructure reuse and higher degrees of freedom for routing. The latter idea will hand operators the ability to retain as much centralization as possible. The vision behind a flexible functional split is to reach a balance between centralization and reasonable network requirements. As a result, network designers will have the ability of deciding which BS functions (e.g. PDCP, RLC, MAC, etc.) can be processed on the edge, i.e. co-located with the RU and which ones could be offloaded into a CU. As BS functions are not completely moved away from the edge, network requirements are softened while retaining some of the advantages of centralization [13]. Note that a flexible functional split blurs the separation between fronthaul and backhaul traffic. In fact, the vision for 5G is the convergence of both types of interfaces on a packet-based network. We will referer to this as *Crosshaul*. A detailed study of the benefits and network requirements on centralizing the different LTE layers is presented on [7]. We summarize the requirements for different functional splits on Table I.

In this paper, we aim to study the implications that placing several BS functions on a CU impose on a packet-based crosshaul network with high path diversity. One the one hand, choosing which BS functions are offloaded into a CU depends of the transport capabilities of the underlying network. On the other hand, a reasonable routing on the transport network requires knowledge on which BS functions will be placed to a CU as transport requirements are dependent on the amount of offloaded functions. Thus, we confront a coupled problem where the path computation between different RUs and CUs and functional splits must be optimized jointly. We have prototyped **WizHaul**, a software framework enabling the implementation of a centralized decision-making algorithms that find the best functional splits according to the transport capabilities of the underlying transport network. Our Wizhaul framework may serve two purposes:

TABLE I: Functional splits analysis in [7]. LTE scenario: 1 user/TTI, 20 MHz bandwidth; Downlink: MCS (modulation and coding scheme) index 28, 2x2 MIMO, 100 Resource Blocks (RBs), 2 transport blocks of 75376 bits/subframe; Uplink: MCS 23, 1x2 SIMO, 96 RBs, 1 transport block of 48936 bits/subframe.

Split #	**LTE BS Functional decomposition**	**DL/UL BW req. (Mb/s)**	**Delay req. (μs)**	**Gains**
A	RRC - PDCP	151/48	$30e3$	• Enables L3 functionality for multiple small cells to use the same HW; • Enhanced mobility across nodes w/o inter-small cell data forwarding/signaling; • Reduced mobility-related signaling to the mobile core segment; • No X2 endpoints between small cells and macro eNBs; • Control plane and user plane separation.
B	PDCP - RLC	151/48	$30e3$	• Enables L3 and some L2 functionality to use the same HW.
C	RLC - MAC	151/48	$6e3$	• Resource sharing benefits for both storage and processor utilization.
D	MAC I - MAC II	151/49	$6e3$	• Synchronized coordination and control of multiple cells; • Coordination across cells enables CA, CoMP, eICIC or cross carrier scheduling.
E	MAC - PHY	152/49	250	• Enhancements to CoMP with RU frame alignment and centralized HARQ.
F	PHY split I	173/452	250	• More opportunities to disable parts of the CU at quiet times to save power; • Central L1 CU can be scaled based on average utilisation across all cells; • Smaller CU results in less processing resource and power saving; • Enhancements to joint reception CoMP with uplink PHY level combining.
G	PHY split II	933/903	250	
H	PHY split III	1075/922	250	
I	PHY split IIIb	1966/1966	250	
J	PHY split IV	2457.6/2457.6	250	

- **Network planning**: WizHaul may serve as a key support tool to any network planning phase as it searches for the optimal functional splits of the RUs. Network operators may leverage on Wizhaul to decide how much centralization they would like to retain;
- **Network automation/adaptation**: WizHaul can, not only adapt to link failures by means of re-configuration of functional splits and/or re-routing, but also can support re-location/placement of BS functions in different CUs in case of resource (computing and/or networking) congestion in the cloud or the network. It is implemented as a centralized software framework based on the SDN/NFV to provide the network automation and adaptations during the operation period.

Our goal is to profile the WizHaul software framework showing its performance for different experiments. We refer to [14], [15] for a mathematical analysis on jointly optimizing functional splits and paths from each RU to a CU for future centralized deployment on a 5G mobile transport network.

II. WizHaul

We have built a proof-of-concept scenario to carry out different experiments on top of our WizHaul framework. We present how it is possible to leverage WizHaul features to decide the initial functional split configuration and adapt it in the presence of unexpected events such as link failures or network congestion. WizHaul is able to change the current paths and functional split configuration for each RU/CU pair. We experiment with different algorithms implemented on top of WizHaul that could find the best paths between RUs and CUs while retaining as much centralization as possible. We envision that future fronthaul and backhaul networks will leverage the use of the SDN architecture [16]. Thus, we implement WizHaul on Java on top of a REST client capable to communicate with the north-bound interface (NBI) of an SDN controller so that it can have full control over the control plane of a transport network.

TABLE II: Detailed HW components in our testbed.

Device type	Description	Ref.
vEPC	OpenEPC Rel. 6	[17]
μWave	56 MHz bandwidth @ 7GHz band Adaptive rate $\leq$ 1 Gb/s	[18]
mmWave	500 MHz bandwidth @ E-band Adaptive rate $\leq$ 3.2 Gb/s	[19]
Switch	OpenFlow switch 48 one-gigabit, 4 ten-gigabit ports	[20]
Small-cell	20 MHz channel @ band 3	[21]
RU	20 MHz BW @ band 3 Split 1 (PHY, MAC, RLC) and 3 (PHY)	[22]
CU	Virtual MAC, RLC, PDCP, RRM, RRC	[23]

A. Experimental setup

We have prototyped the scenario shown in Fig. 1a, which is deployed as shown in Fig. 1b. In this testbed, we present a proof-of-concept mobile communication network. The edge part embraces three RUs. RU1 is a fully-fledged LTE small-cell which may not offload any of its functions. RU2 and RU3 support moving some of its functions to a CU. In detail, these RUs support centralizing its PDCP layer. The core segment holds a baseline EPC. Between them, we have prototyped an SDN mobile transport network which contains a CU that processes the centralized functions from RU2 and RU3. We used `Floodlight`[1] as the SDN controller and a commercial OpenFlow (OF) switch [20] supporting OpenFlow 1.0 [24] to set up the network. Furthermore, the transport segment has a μWave wireless link and a mmWave wireless link. For demonstration purposes, the μWave radio link is wired with an SMA cable and the mmWave link is wired with a rigid wave-guide. Each wireless link has a variable attenuator in between the wired connection of the two ends. These reduce the SNR of the channel so that it forces the wireless links to change its MCS or ultimately make impossible any information transmission. Both wireless links feature different transport capabilities. The mmWave link have a maximum capacity of 3.2 Gb/s while the μWave link may reach up to 500 Mb/s. All other links are connected using Ethernet links which render a maximum capacity of 1Gb/s. Following, all end

[1]http://www.projectfloodlight.org/floodlight/

points are synchronized using precision time protocol (PTP[2]) so that we can measure latency accurately. Table II summarizes the details of the scenario. Finally, as we cannot support all the functional splits described in Table I we generate its traffic flow patterns based on its requirements. Table III lists the splits we support. We use `mgen`[3] to generate UDP flows accordingly and `trpr`[4] to process the `mgen` logs.

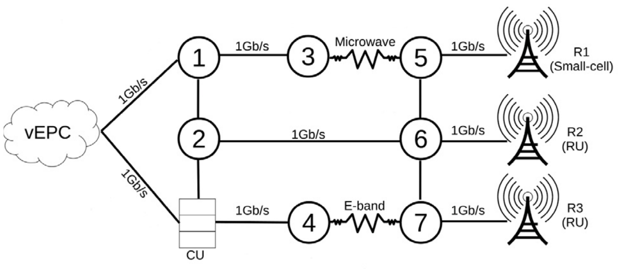
TABLE III: Supported splits

Split	LTE function
Split 1 (B in Table I)	PDCP - RLC
Split 2 (C in Table I)	RLC - MAC
Split 3 (E in Table I)	MAC - PHY
Split 4 (G in Table I)	PHY split II
Split 5 (J in Table I)	PHY split IV

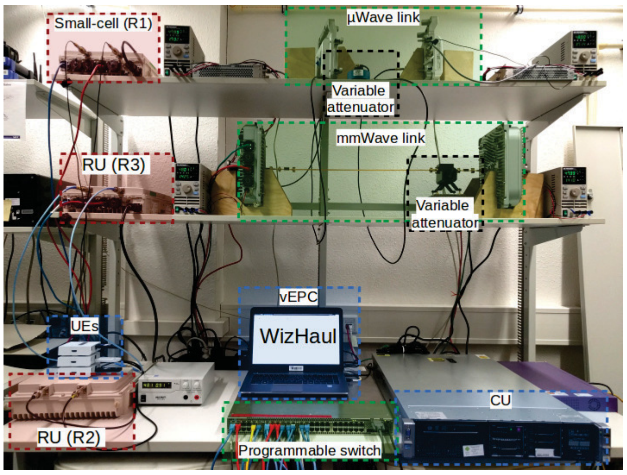
B. *Software architecture*

Fig. 2 depicts our software architecture. WizHaul is implemented on top of `Floodlight`, a Java-based Apache-licensed SDN controller. WizHaul communicates with `Floodlight` by means of its REST interface using JSON messages. Our framework uses the `CommunicationManager` class which receives HTTP objects from the `HTTPManager` class and sends them to the SDN controller. `Floodlight` in turn, communicates with the underlying switches by means of the OpenFlow protocol. On start-up, our application retrieves the underlying topology leveraging the `TopologyManager` class of the SDN controller which discovers the different links between switches using LLDP. Both wireless links convey status messages to the controller using SNMP. This messages are used to detect any MCS change on the wireless interfaces or a link failure. Further, the controller communicates with the CU and RUs by means of SSH protocol. WizHaul can push any path to the controller and change the functional splits. The `Manager` class is where different orchestration algorithms for fronthaul and backhaul traffic may be implemented according to its eclectic requirements. For our experiments, we programmed WizHaul so that it always aims to maximize the amount of centralization on a mobile network. That is, our strategy is to try to always offload as many BS functions as possible. Moreover, WizHaul also reacts to topology changes. Thus, on the presence of a network failure WizHaul checks if the current configuration, i.e., paths and functional splits, is still feasible. If they are not feasible, WizHaul will compute a new configuration and install it. If the link was not used by any RU WizHaul is not going to compute any new configuration. Similarly, if there is any new link on the topology, WizHaul will compute a new setup. However, the new paths and functional splits will only be set up if the new configuration improves the amount of centralization of the previous configuration.

[2]https://tools.ietf.org/html/rfc8173
[3]https://www.nrl.navy.mil/itd/ncs/products/mgen
[4]https://downloads.pf.itd.nrl.navy.mil/docs/proteantools/trpr.html



(a) Baseline PoC scenario



(b) Testbed

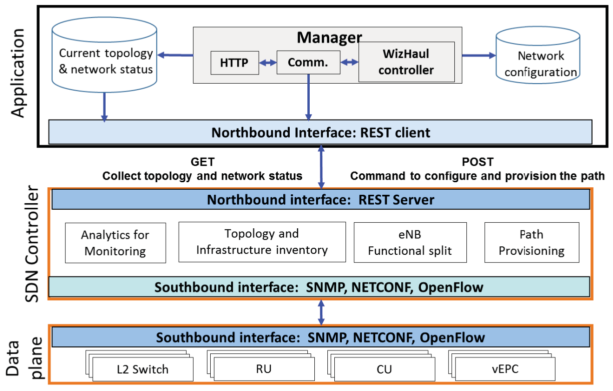
Fig. 1: Experimental setup

III. Experiments

We will carry out the following experiments on our proof-of-concept scenario.

1) **Provisioning path time**: The first experiment we present is the provisioning path time for each RU to its CU. We measure how much time it employs pushing a set of computed paths and functional splits plus the time the SDN controller employs installing the corresponding rules to each switch.
2) **Virtual Network Functions deployment time**: Whenever we configure a BS with a certain functional split, WizHaul has to deploy a set of Virtual Network Functions (VNFs) in the CU that will support the offloaded BS functions. We have measured how much time it takes to deploy the different VNFs in the CU for our BSs;
3) **Recovery time**: As mentioned before, we are able to decrease the SNR of both wireless channels manually using two variable attenuators. One is attached in between the μWave wireless link and the second one is in between the rigid waveguide of the the mmWave link. Lowering the SNR forces the wireless equipment to lower the MCS due to channel conditions or even make any packet transmission infeasible. Any change on channel conditions will be notified to the SDN controller via SNMP. This in turn will notify it to WizHaul. The algorithms implemented on top of our framework analyze any change on the transport

network and act consequently. We measure how much time WizHaul employs notifying a change and setting up a new network configuration after a topology change. We will consider failures on both the μWave and mmWave links.



Application
Current topology & network status
Manager
HTTP
Comm.
WizHaul controller
Network configuration
Northbound Interface: REST client
GET
Collect topology and network status
POST
Command to configure and provision the path
SDN Controller
Northbound interface: REST Server
Analytics for Monitoring
Topology and Infrastructure inventory
eNB Functional split
Path Provisioning
Southbound interface: SNMP, NETCONF, OpenFlow
Data plane
Southbound interface: SNMP, NETCONF, OpenFlow
L2 Switch
RU
CU
vEPC

Fig. 2: Software architecture

A. *Provisioning path time*

To start with, we have profiled the installation path time for each RU to its CU. We start pairing the OF switches with the SDN controller. Then, we launch our WizHaul engine and we measure how much time it employs pushing a set of paths and functional splits plus the time the controller employs installing the corresponding rules to the forwarding table of each switch. The controller has to install two rules in each switch for every hop of a path. One rule forwards the uplink traffic while the other one forwards the downlink traffic. To profile the time it takes to install a path, we have considered paths with n hops while taking into account m tenants. By tenants we mean entities that need to setup a path with another entity. Figure 3 shows the results by means of boxplots where the top and bottom bar represent the 1st and 4th percentile and the middle bar the median, respectively. The dots represent the different experiments we have performed for each case. As Figure 3 depicts, the time to install a path increases linearly with the number of tenants and hops. We note that this is expected as the more hops and tenants we have, the more rules we have to install. The time to install two forwarding rules in one switch is approximately 15 ms. However, there are some strategies to lower the total installation time. For example, it is possible to aggregate the traffic by destination in the the uplink so that fewer rules have to be installed.

We have implemented two different installation strategies in our WizHaul engine: (i) a complete path installation strategy and, (ii) a differential path installation strategy. The first approach erases all the previous rules from all the switches and installs all the new forwarding policies; while the second one erases and installs only the necessary forwarding rules when comparing a new set of paths with the current set of paths. The first strategy starts sending a *OF DELETE* command to all the switches to erase all the forwarding rules. Afterwards,

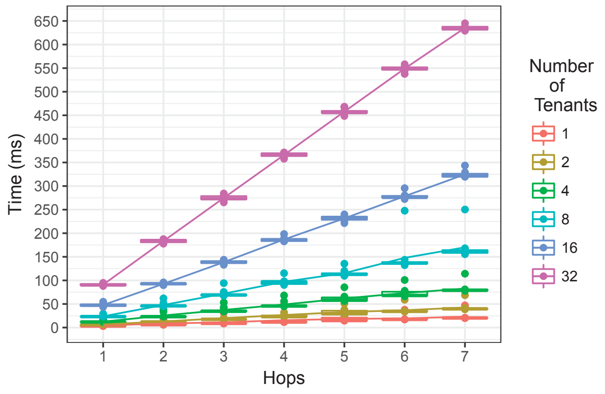


Fig. 3: Provisioning path time

it installs all the new forwarding policies. This strategy might be good for the first time WizHaul has to install the paths; however, if a setup is already running on the transport mobile network, it will disrupt every flow even though its new path is the same as the previous one. It may also be beneficial when a network owner may wants to change completely the current configuration. The second strategy starts comparing each new RU/CU path with its old path. WizHaul stores each path between each RU and CU as a set of links. Once a new path has been computed, it finds the common links between the new path and the old path for each RU. Once we obtain the common links, we compute two more sets of links. On the one hand, we compute the uncommon links between the old path and the common links previously obtained. On the other hand, we compute the uncommon links between the new path and common links. Then, WizHaul proceeds to install the new rules in two steps. First, it removes the rules that forward traffic through the links that are contained on the first set. Second it installs all the rules needed to forward traffic through the links that are in the second set. This strategy is particularly well suited for link failures as it allows for a very fast recovery.

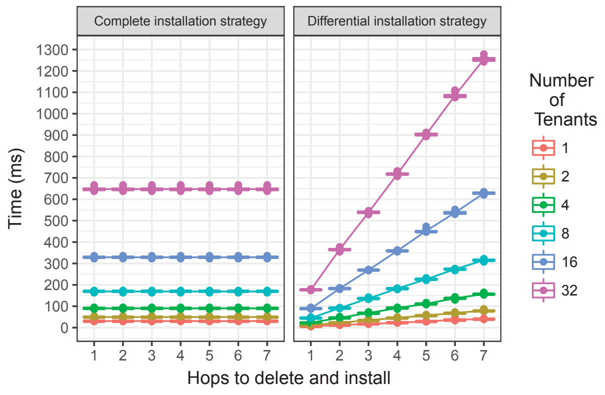


Fig. 4: Comparison between the path installing strategies

We have profiled the time it takes for both strategies to install different sets of paths. We have assumed that for each different tenant WizHaul has to install a path of length 7. Thus, in the case of the complete path installation for each number of tenants it has to send a message to each switch to delete all the rules and then install the new forwarding policies. To profile the differential installation strategy we have proceed a differently. We also considered different number of tenants and a path of length 7, but we assumed that each tenant has to delete n rules and install n. Figure 4 shows the profiling results using boxplots for both strategies. The first strategy always takes the same time to install regardless of the hops that have to be deleted and reinstalled as it does not try to find common hops to save the any time. The second strategy improves performance over the first one substantially when the new paths that we have to install have many common paths with the previous path. In a large mobile transport network, a failure in one link may disrupt some flows but, since the topology is not going to suffer major changes due to a simple link failure, we expect that the configuration will be very similar to the previous one. Therefore, comparing the new paths to the old paths will usually result in a much faster failure adaption.

B. VNF deployment time

In this experiment, we profiled the VNF deployment time for our C-RAN like equipment. Our RU2 and RU3 support the centralization of the PDCP function and abover via a set of VNFs running on a CU. It is key to profile the time it takes to deploy the set of VNFs that support the BS operations in order to understand how fast we can install new BSs, change their functional splits or react to server failures. In detail, our CU has to deploy three virtual machines (VMs) that support the different BS functionality. The first VM supports the S1AP, RRC and RRM functions and operation, administration and management tasks (OAM). The second VM supports the routing operation for the GTP tunnel and the third VM supports the operation for the PDCP layer. Note that this set of VMs has to be deployed in one CU to support the operations of all the RUs that centralize its functions using those VMs. It is not necessary to start a set of VMs for each RU. Therefore, we can benefit from managing a specific BS function using one virtual machine for a set of BS. Whenever we start the set of VMs we have to perform two steps. First we have to launch all the VMs. Second we have to copy different configuration files to each VM and setup its network addresses and routing tables.

Figure 5 shows the measured delay to perform each of the two steps mentioned before by means of boxplots. The VNF launching time is about 20 sec on average while the time to configure all the VMs is 103 seconds on average. Note that the VNF deployment time is mainly dominated by the configuration time. These times are much larger than hundreds of milliseconds. Thus, if any CU has a failure, the operations of any BS attached to the CU will be disrupted during a significant time. Thus, network owners should ensure that each CU has a mean to migrate its state to a different location or that the RUs can operate using a known backup CU. It is also worth noting, that it could be a good strategy to configure the

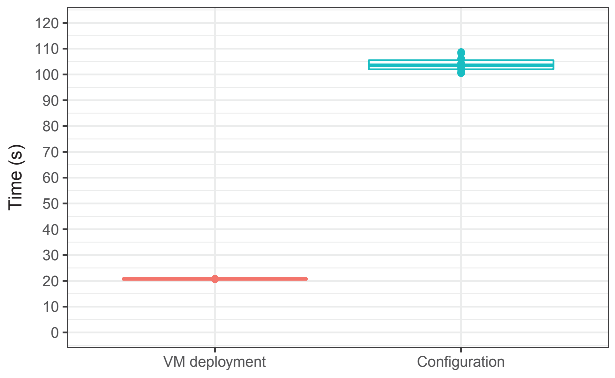


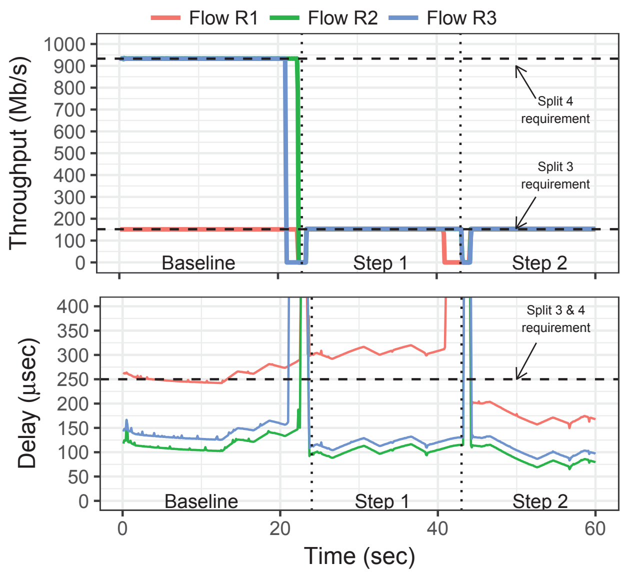
Fig. 5: VNFs Deployment time

set of VMs used to centralize other functions even though they are not used so that it is possible to support a fast functional split change.

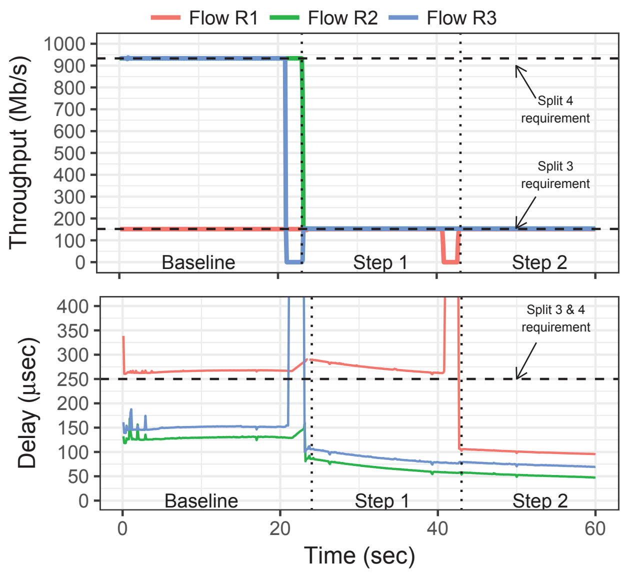
C. Recovery time

Following, we experiment with WizHaul in an scenario with possible link failures. We show how the WizHaul engine adapts the functional split configuration to the current transport capabilities of the underlying network. We start with the baseline scenario depicted in Figure 7a. WizHaul configures RU2 and RU3 with split 4 from Table III. Due to the limiting of 1Gb/s links, WizHaul does not settle a full C-RAN configuration (which would require the transmission of 2.5Gbps flows). WizHaul routes the backhaul traffic from RU1 through the μWave link while using the mmWave link for the RU3 traffic and using the central link for RU2. RU2 and RU3 are not directly connect to the EPC, but they have to connect with the CU first. The CU in turn will connect to the EPC. We constantly measure throughput and latency to verify that flows are compliant with the chosen split requirements.

Figure 6a and Figure 6b depicts downlink throughput and latency measurements for the two different paths installation strategies that we have previously explained. We can easily verify from these figures how throughput and latency requirements for split 4 are compliant with the network requirements. After 20 sec, we attenuate the mmWave wireless link until communication is fully disrupted. This triggers WizHaul to make two changes. On the one hand, WizHaul changes the current path of RU3 and routes the traffic through link 2-6. On the other hand, it lowers the split of base stations 2 and 3 to 3 in Table III so that both flows may fit into one link. The measurements on Figures 6a and 6b clearly show this behavior. We can see how the throughput changes for RU2 and RU3. Latency has a very high peak due to a significant gap between the packets received. Figure 7b shows the scenario after the first link failure. Furthermore, we can see from Figures 6a and 6b how the complete installation strategy causes a disruption in all the flows even though a path is not altered by the link failure. Finally, 20 sec later, we attenuate the μWave wireless link. WizHaul routes the RU1

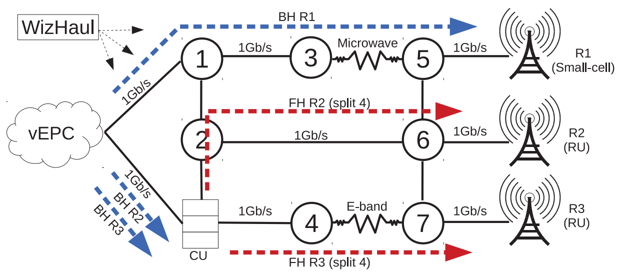


(a) Complete installation strategy

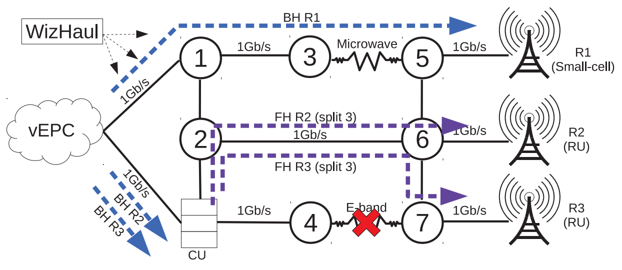


(b) Differential installation strategy

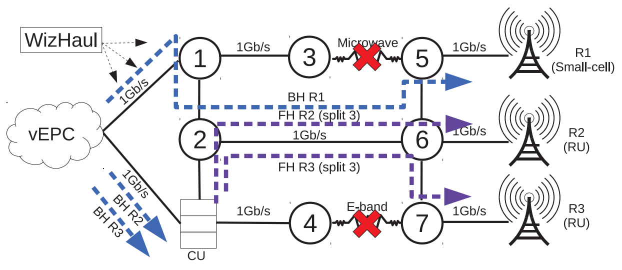
Fig. 6: Experimental validation.



(a) Baseline.



(b) Step 1



(c) Step 2

Fig. 7: Use case illustration

traffic trough the link 2-6 with RU2 and RU3. Figure 7c shows the scenario after the second link failure. There is no need to change the split as all flows fit into link 2-6. We observe in Figure 6a how the throughput for all RUs reaches drops to zero while we are deleting and installing paths. On Figure 6b only the throughput for RU1 is disrupted. Thus, the differential installation strategy is much better to handle unexpected events as it does not disrupt the operation of other BSs.

To provide further insights on our WizHaul engine, we have profiled the time taken to deploy a new configuration after a topology change. In this case we profile the recovery time for the differential installation strategy as we believe it is the best to handle these cases. We have profiled both the failure of the mmWave and μWave links. Figure 8 shows the different operations that constitute a recovery time period. The WizHaul engine goes through four operations. First, it receives a notification form the hardware that has failed (labelled "HW reaction"). In our case, this notification is received through SNMP. We note that other protocols could be used in different setups. Before doing any operation, WizHaul checks that this link is being used. If it is not used, then it concludes that no path was disrupted. If it was used by at least one RU, WizHaul computes a new set of paths and functional splits (labelled "Algorithm"). Afterwards, WizHaul installs (labelled "Path installation") them. We also depict the time employed by other functions such as processing messages of the REST interface (labelled "Others"), etc. The recovery time is mainly dominated by the hardware reaction. We see how in both cases we obtain the same value. On step 1, the installation path time is a bit higher than the one on step two as there are more uncommon links with the old paths.

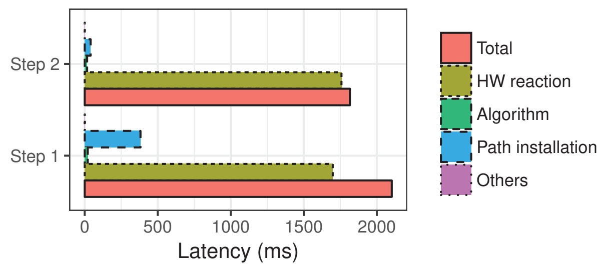


Fig. 8: Software reaction time

IV. Conclusions

The redesign of the fronthaul interface is a key building block to the adoption of centralized infrastructures in 5G. The main value proposition of the new design is the support of a flexible functional split. This flexible functional split will help to relax the stringent traffic requirements of C-RAN deployments while retaining as much centralization as possible. In this paper, we introduced WizHaul, a centralized decision-making engine that optimizes the functional split of each RU taking into account the capabilities of the transport network. Then, we presented our prototype implementation in a real testbed and carried out a thorough experimental profiling, validating WizHaul as a tool for optimal mobile network planning and fault recovery tool.

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 761536 (5G-Transformer project).

References

[1] E. Metsälä and J. Salmelin, *Mobile Backhaul*. John Wiley & Sons, 2012.

[2] A. Checko, H. L. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. S. Berger, and L. Dittmann, "Cloud RAN for Mobile Networks - A Technology Overview," *IEEE Communications Surveys Tutorials*, vol. 17, no. 1, pp. 405–426, Firstquarter 2015.

[3] Y. Lin *et al.*, "Wireless network cloud: architecture and system requirements," *IBM Journal of Research and Dev.*, vol. 54, Jan. 2010.

[4] N. Nikaein, "Processing radio access network functions in the cloud: Critical issues and modeling," in *Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services*, ser. MCS '15. New York, NY, USA: ACM, 2015, pp. 36–43. [Online]. Available: http://doi.acm.org/10.1145/2802130.2802136

[5] V. Suryaprakash, P. Rost, and G. Fettweis, "Are heterogeneous cloud-based radio access networks cost effective?" *IEEE J. Sel. Areas Commun.*, vol. 33, no. 10, pp. 2239–2251, Oct 2015.

[6] U. Dötsch, M. Doll, H.-P. Mayer, F. Schaich, J. Segel, and P. Sehier, "Quantitative analysis of split base station processing and determination of advantageous architectures for lte," *Bell Labs Technical Journal*, vol. 18, no. 1, pp. 105–128, 2013.

[7] "Small Cell Forum, R6.0. Small cell virtualization functional splits and use cases," Jan. 2016.

[8] A. Pizzinat, P. Chanclou, F. Saliou, and T. Diallo, "Things you should know about fronthaul," *Journal of Lightwave Technology*, vol. 33, no. 5, pp. 1077–1083, 2015.

[9] "Next Generation Fronthaul Interface," White paper, China Mobile, Alcatel-Lucent, Nokia, ZTE, Broadcom, Intel., June 2015.

[10] I. Chih-Lin *et al.*, "Rethink fronthaul for soft RAN," *IEEE Comm. Magazine*, vol. 53, no. 9, pp. 82–88, September 2015.

[11] IEEE, *IEEE 1914 WG "IEEE WG, Next Generation Fronthaul Interface"*.

[12] 3GPP, *TR 38.801: Study on New Radio Access Technology: Radio Access Architecture and Interfaces*, Dec. 2016.

[13] P. Rost *et al.*, "Cloud technologies for flexible 5G radio access networks," *IEEE Comm. Magazine*, vol. 52, no. 5, pp. 68–76, May 2014.

[14] A. Garcia-Saavedra, J. X. Salvat, X. Li, and X. Costa-Perez, "WizHaul: On the Centralization Degree of Cloud RAN Next Generation Fronthaul," *IEEE Transactions on Mobile Computing*, vol. PP, no. 99, pp. 1–1, 2018.

[15] A. Garcia-Saavedra, X. Costa-Perez, D. J. Leith, and G. Iosifidis, "FluidRAN: Optimal vRAN/MEC Orchestration," in *INFOCOM, 2018 Proceedings IEEE*, April 2018.

[16] J. Costa-Requena, "Sdn integration in lte mobile backhaul networks," in *Information Networking (ICOIN), 2014 International Conference on*. IEEE, 2014, pp. 264–269.

[17] OpenEPC. http://www.openepc.com/.

[18] NEC iPASOLINK VR4. http://www.nec.com/en/global/prod/nw/pasolink/products/ipasolink_VR4.html.

[19] NEC iPASOLINK EX. http://www.nec.com/en/global/prod/nw/pasolink/products/ipasolinkEX.html.

[20] NEC ProgrammableFlow. http://www.nec.com/en/global/prod/pflow/pf5240.html.

[21] NEC MB4420 small-cell. http://www.nec.com/en/global/solutions/nsp/sc2/prod/e-nodeb.html.

[22] NEC NFV C-RAN. http://www.nec.com/en/global/solutions/nsp/sc2/prod/c-ran.html.

[23] HP DL380p Gen8 server. https://www.hpe.com/h20195/v2/default.aspx?cc=za&lc=en&oid=5177957.

[24] O. S. Specification, "Version 1.0. 0 (wire protocol 0x01)," *Open Networking Foundation*, 2009.