FSA: Fronthaul Slicing Architecture for 5G using dataplane programmable switches

Nishant Budhdev
National University of Singapore

Raj Joshi
National University of Singapore

Pravein Govindan Kannan
IBM Research - India

Mun Choon Chan
National University of Singapore

Tulika Mitra
National University of Singapore

ABSTRACT

5G networks are gaining pace in development and deployment in recent years. One of 5G's key objective is to support a variety of use cases with different Service Level Objectives (SLOs). Slicing is a key part of 5G that allows operators to provide a tailored set of resources to different use cases in order to meet their SLOs. Existing works focus on slicing in the frontend or the C-RAN. However, slicing is missing in the fronthaul network that connects the frontend to the C-RAN. This leads to over-provisioning in the fronthaul and the C-RAN, and also limits the scalability of the network.

In this paper, we design and implement Fronthaul Slicing Architecture (FSA), which to the best of our knowledge, is the first slicing architecture for the fronthaul network. FSA runs in the switch dataplane and uses information from the wireless schedule to identify the slice of a fronthaul data packet at line-rate. It enables multipoint-to-multipoint routing as well as packet prioritization to provide multiplexing gains in the fronthaul and the C-RAN, making the system more scalable. Our testbed evaluation using scaled-up LTE traces shows that FSA can support accurate multipoint-to-multipoint routing for 80 Gbps of fronthaul traffic. Further, the slice-aware packet scheduling enabled by FSA's packet prioritization reduces the 95th percentile Flowlet Completion Times (FCT) of latency-sensitive traffic by up to 4 times.

CCS CONCEPTS

• **Networks → Wireless access points, base stations and infrastructure**; **Programmable networks**; **Wireless access points, base stations and infrastructure**; **Programmable networks**.

KEYWORDS

5G Cellular Networks, Slicing, Programmable Switches

ACM Reference Format:
Nishant Budhdev, Raj Joshi, Pravein Govindan Kannan, Mun Choon Chan, and Tulika Mitra. 2021. FSA: Fronthaul Slicing Architecture for 5G using dataplane programmable switches. In *The 27th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '21), October 25–29, 2021, New Orleans, LA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3447993.3483247

ACM MobiCom '21, October 25–29, 2021, New Orleans, LA, USA

ACM ISBN 978-1-4503-8342-4/21/10.
https://doi.org/10.1145/3447993.3483247

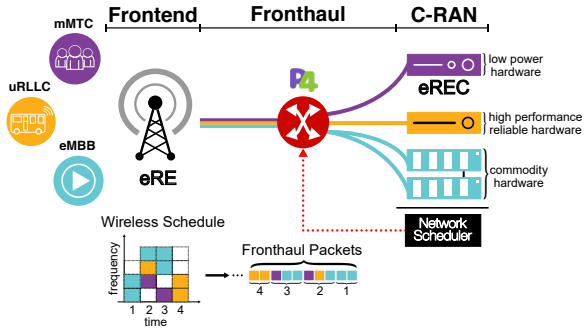


Figure 1: Overview of Fronthaul Slicing Architecture (FSA)

1 INTRODUCTION

5G networks aim to support use cases in diverse domains such as Internet-of-Things, intelligent transport systems, telemedicine, industrial control systems, etc., beyond the basic use cases of voice and data supported by previous cellular network technologies. Consequently, 5G networks need to provide a diverse set of Service Level Objectives (SLOs) for these new use cases. At the same time, 5G networks have explored the use of the Cloud Radio Access Network (C-RAN) architecture as it allows operators to reduce costs by multiplexing resources and to enable new features with centralized and coordinated decision making [1–3]. Fig. 1 shows a simplified view of a C-RAN architecture. It consists of (1) a radio frontend responsible for transmitting and receiving radio signals consisting of enhanced Radio Equipment (eRE), (2) a RAN with the compute elements responsible for processing the radio signals called enhanced Radio Equipment Controller (eREC), and (3) a fronthaul network that enables communication of digitized radio signals between the eREs and the eRECs.

To support different SLOs, 5G introduces a critical technology called slicing, which allows operators to support multiple virtual networks on top of shared infrastructure [4]. This allows network operators to deploy a tailored set of resources for specific use cases. For example, low power hardware along with fine-grained scheduling for massive Machine Type Communication (mMTC) [5], high performance reliable hardware along with low latency access to the wireless channel for ultra-Reliable Low Latency (uRLLC) use cases [6, 7], and shared pool of commodity hardware for enhanced Mobile Broadband (eMBB). To this end, there exists a large body of work that explores slicing in the radio frontend (wireless spectrum) [8], the RAN [9–12], and the Core Network [13, 14].

However, these proposals lack slicing in the fronthaul network which leads to two major problems. First, without fronthaul slicing, there is a point-to-point connection between an eRE and an

eREC [15]. This implies that users from slices with different SLOs connected to an eRE, are processed together on the assigned eREC. As a result, each eREC may have to provision for low power hardware for mMTC users, high performance reliable hardware for uRLLC, and commodity hardware for eMBB. This leads to over-provisioning of each eREC with a variety of hardware to support peak workloads for each use case, while in practice cellular network traffic is known to have bursty peaks and low average utilization [16]. Additionally, a point-to-point connection between an eRE and an eREC limits the scalability of the system as the number of 5G users per eRE increases. This is because the maximum number of users per eRE is limited by the processing capacity of the assigned eREC. Second, without fronthaul slicing, operators lack the ability to provide differentiated services in the fronthaul network. Services such as packet prioritization can help operators satisfy different SLOs without over-provisioning the fronthaul network. In summary, lack of fronthaul slicing leads to over-provisioning in the fronthaul and the C-RAN as well as limits the scalability of the 5G network.

In this paper, we address the above issues by enabling slicing in the fronthaul network. Fronthaul slicing enables multipoint-to-multipoint routing which allows the digitized radio signals to be routed from any eRE to any eREC and vice versa. For example, in Fig. 1, we see that the uRLLC traffic (yellow) from an eRE could be routed to an eREC with high performance hardware while mMTC traffic (violet) from the same eRE could be routed to another eREC with low power hardware. This way each eREC can be provisioned with only a singular hardware type. This also improves resource efficiency in the C-RAN, as traffic from several eREs can be multiplexed across the combined pool of eRECs. Such slice-based routing also enables recently proposed C-RAN architectures [17–21] that distribute baseband processing across multiple eRECs by taking into account the different slice SLOs. Also, multipoint-to-multipoint routing makes the system scalable by allowing operators to add new eRECs to support increased traffic loads, without needing to upgrade each eREC. Other than multipoint-to-multipoint routing, fronthaul slicing also enables packet prioritization. With packet prioritization, traffic with stricter SLOs (uRLLC) could be prioritized over other traffic (mMTC) in the fronthaul network. This allows operators to satisfy different SLOs efficiently without over-provisioning the fronthaul network.

However, slicing the fronthaul network to enable multipoint-to-multipoint routing and packet prioritization is not straightforward. The first challenge lies in ***identifying*** the slice/user to which a fronthaul network packet belongs. This challenge originates from the *functional split* deployed in the C-RAN architecture. The functional split defines how the processing of traffic is split between the eRE and the eREC. Due to the need to lower the cost of eREs, which are expected to be deployed in large numbers in 5G, the commonly chosen functional splits minimize the amount of processing at the eRE [22, 23]. As a result, the slice identifier information available in the Access Layer (MAC) is missing from the fronthaul packets in the uplink direction, since the eREs do not perform MAC layer processing in the most common functional splits [24]. In other words, it is not possible to look at a packet in the fronthaul network and identify the slice/user that it belongs to. The second challenge is to correctly route the fronthaul data packets after the user has been identified. In cellular networks, nearly 70% of the user sessions are very short in duration (active for ∼1 millisecond) [25]. As a result, the information required to perform multipoint-to-multipoint routing – the slice/user identifier and the destination eREC – changes at a very high rate. Conventional switches which update their routing tables via the control plane do not support such high frequency routing updates due to the high latency of the control plane [26].

Our solution called the Fronthaul Slicing Architecture (FSA) addresses the two challenges as follows. First, we observe that while slice/user information may not be available in the fronthaul packets, the information is available in the wireless schedule since all transmissions in a cellular network are scheduled by the network scheduler in advance. As a result, slice information for all packets in the fronthaul can be precisely identified using the wireless schedule which is generated in advance. Based on this observation, our key idea is to transform the wireless schedule into a sequence of expected uplink packets and use this sequence in the fronthaul to identify the slice/user. Fig. 1 shows a simple representation of the relationship between the wireless schedule and the corresponding sequence of uplink packets in the fronthaul. To address the second challenge and enable multipoint-to-multipoint routing, FSA introduces a *high frequency dynamic forwarding* scheme in which the routing table of a fronthaul switch is continuously updated based on the wireless schedule known in advance. This enables FSA to route the fronthaul packets to the correct eREC with the wireless schedule being generated every millisecond independently for each eRE. FSA leverages programmable switches for implementing both slice identification and high frequency dynamic forwarding. FSA runs entirely in the dataplane and implements an SRAM-based dynamic routing table that can be updated in the switch dataplane at line rate. We also build a slice-aware packet scheduler as an example use case of the packet prioritization enabled by FSA. The packet scheduler prioritizes the fronthaul packets based on the slice a packet belongs to, and the user's expected baseband processing time.

In summary, we make the following contributions:

- To the best of our knowledge, FSA is the first work to enable slicing in the fronthaul by identifying the slice/user for each packet at line rate in the switch dataplane.
- FSA enables multipoint-to-multipoint routing through a *high-frequency dynamic forwarding scheme* that allows fronthaul packets to be routed from any eRE to any eREC dynamically. The design of the forwarding scheme can handle packet drops and reordering with minimal overhead.
- FSA also enables packet prioritization in the fronthaul network. As an example use case, we design and implement a slice-aware packet scheduler that minimizes latency for delay-sensitive slices by using Least Slack Time First (LSTF) scheduling in the switch dataplane.

We implement FSA and the two functions of multipoint-to-multipoint routing and packet prioritization (enabled by it) on an Intel Tofino [27] programmable switch. We evaluate FSA on a hardware testbed using 5G traces obtained by upscaling real LTE traces. Our results show that the forwarding schedule can be updated in the switch dataplane at line rate and with a worst-case latency of less than 6 μs, which is less than the smallest scheduling slot of 62.5 μs in 5G [28]. Further, our evaluations show that FSA can

support accurate multipoint-to-multipoint dynamic routing with an aggregate fronthaul traffic of ~80 Gbps sent from 8 base stations. With the slice-aware packet scheduling (Least-Slack-Time-First) implemented on FSA, we observe 4x reduction in the 95th percentile of the flowlet completion time for latency-sensitive traffic as compared to a FIFO-based scheduler. Also, FSA's implementation consumes very little extra resources in the switch dataplane, requiring an additional SRAM usage of only 10.42%.

The remaining paper is organized as follows. In §2, we present the background and related work. §3 and §4 present the design and implementation of FSA's high-frequency dynamic forwarding scheme. We detail the slice-aware packet scheduling in §5 and evaluate FSA in §6. In §7, we discuss how FSA can be modified to support other functional splits and finally conclude in §8.

2 BACKGROUND AND RELATED WORK

Functional split. With the introduction of C-RAN, network operators can now distribute the radio processing between the eRE and eREC. This distribution is known as the *functional split*. In one extreme, nearly all radio processing is done on the eREs (traditional RAN design). These are called the *higher* functional splits. The other extreme is where nearly all radio processing is performed in the cloud (eREC). These are called the *lower* functional splits. Between these two extremes, there are a variety of functional split options with each split providing a unique trade-off between cost for fronthaul equipment and the CAPEX/OPEX for the RAN. As the amount of radio processing performed in the eRE increases, the cost of the eRE also increases; while the amount of traffic in fronthaul network reduces. Conversely, centralized radio processing can significantly reduces cost of eREs [22], but these gains come with the need to provide high throughout and low latency for fronthaul traffic. The remaining discussion in this paper considers the 7-2 split which allocates majority of the baseband processing functions to the eREC [29]. This split reduces fronthaul throughput significantly by removing IQ samples corresponding to unallocated parts of the wireless network schedule. In §7, we discuss how FSA can enable fronthaul slicing for other functional splits.

2.1 Related Work

Slicing in cellular networks has received substantial attention due to its benefits for enabling wide-variety of SLOs, and ensuring efficient resource usage. Owing to the ease of slicing compute resources and the Core Network, much of the research on slicing focuses on RAN [9–12, 30] and the Core Network [13, 14]. PRAN [30] introduces flexible RAN processing for users associated with an eRE. This flexibility allows users from each slice to have customized dataplane processing with specialized functions to meet their individual SLOs. To extend this flexibility to the wireless channel Orion [8] proposes a RAN slicing architecture that introduces slicing in the wireless channel, to support smaller scheduling intervals for low latency and low-power use cases.

Unlike the RAN and the Core Network, slicing in the fronthaul is challenging due to the high throughput and low-latency nature of fronthaul traffic. Larsen et al. [31] show that while slicing a packet-switched fronthaul network can bring great advantages, fronthaul slicing in 5G needs to be flexible to support a wide variety of functional. Fronthaul slicing can also be used to support Multiple Network Operators with different functional splits and SLOs on the same physical network [32].

In the past decade, multiple protocols have been developed for fronthaul networks such as Common Public Radio Interface (CPRI) [15], IEEE 802.1CM [33], Open Base Station Architecture Initiative (OBSAI) [34] etc. Of these protocols, CPRI is the most widely adopted protocol for fronthaul networks. CPRI uses a serial interface that transmits digitized radio samples at a constant bit-rate in the fronthaul. However, due its serial interface, CPRI can only support point-to-point transmissions as it does not use Ethernet frame/packet-based transmission. This along with integrated protocol design and high throughput makes CPRI highly inefficient for 5G networks. Thus, a successor to the protocol was developed, known as enhanced CPRI (eCPRI) [35], which is built on top of Ethernet/IP protocols. eCPRI also supports a wide variety functional splits to enable operators to optimize their fronthaul networks. Additionally, eCPRI uses a modular protocol stack which enables cellular operators to use standard protocols for control messaging and time synchronization, while still using eCPRI for transporting digitized radio samples.

Similar to eCPRI, IEEE 802.1CM [33] also uses a packet-based transmission for fronthaul networks. 802.1CM also provides operators the ability to categorize fronthaul traffic into 8 different priority classes using a 3-bit Priority Code Point field [36], similar to the QoS Class Identifiers available in LTE [37]. However, classifying and prioritizing the fronthaul traffic into just 8 unique classes limits the ability to support a wide variety of use cases with different SLOs. Additionally, this information is insufficient for supporting a flexible multipoint-to-multipoint fronthaul network critical for scaling in 5G. Also, for lower functional splits, the slice/user information is unavailable at the eRE, and hence marking the priority class for the uplink traffic requires a non-trivial solution. In contrast, FSA which uses eCPRI, can support lower functional splits, provide per-user slicing and prioritization as well as multipoint-to-multipoint routing.

3 DESIGN SPACE AND CHALLENGES

In order to provide multipoint-to-multipoint routing and packet prioritization in the fronthaul network, the first step is to identify the slice for each packet in the fronthaul. Recall that slice information is not available for most functional splits. As mentioned before, our key insight is to use the wireless schedule (which is known in advance) to identify the slice. Given this key insight, in this section, we describe the various ways in which it can be realized in practice. Note that, while traffic flow is bi-directional, in this paper, we focus on the uplink direction (eRE to eREC) as the slice/user identification information is unavailable at the eRE for most functional splits.

Identification at the eRE. The uplink wireless transmissions from users are received by the eREs which are then digitized for transmission to the eREC. If the eREs are provided with the wireless schedule, they need to tag the packets with both the slice/user identifier and the routing information before transmission. Implementing such a function on the eREs is non-trivial as most eREs use specialized hardware for processing which do not easily support the addition of new features. Additionally, supporting such a feature

on new eREs will increase the cost for 5G, as the number of eREs will increase significantly to compensate for smaller coverage [16]. Finally, coordinating the wireless schedule with a large number of eREs within a tight timing constraint entails high coordination complexity.

Identification at the eREC. A diametrically opposite solution would be to implement slice identification in the eREC as a network function running on servers. In such a scenario, eRECs would be made aware of the uplink wireless schedule and they would then tag the incoming fronthaul data packets with slice and routing information. However, since each base station in 5G is capable of generating over 300 Gbps [35] of fronthaul traffic, and each C-RAN can manage ~1000 base stations, we would need to *scale out* the fronthaul slicing operation across 100's of servers similar to traditional L4 load balancing [38]. This would dramatically increase both the CAPEX and OPEX of the C-RAN.

Identification at fronthaul switches. In FSA, we take a middle ground to realize fronthaul slicing on switches in the fronthaul network. In such a deployment, the wireless scheduler sends the schedule to the network switches in the fronthaul network. Compared to slicing at the eREs, schedule coordination in this scenario is much simple and has no scalability concerns since the number of network switches involved is relatively small. Each switch serves a large number of base stations simultaneously, processing up to several Tbps of traffic. Prior works have shows that network functions such as L4 load balancing when performed by a switch can replace hundreds of servers [39]. Thus it is natural to do identification on the switch to support multipoint-to-multipoint routing and packet prioritization.

While the high-level insight of using a wireless schedule to identify a packet's slice may sound simple, there are several challenges to realize the same on a network switch. First, the uplink wireless network schedule can be thought of as representing the number of frequency resource blocks allocated to different users within a time slot of 1 ms. It is not straightforward to use this information for slice identification. Further, the schedule is generated only 4 ms in advance and a new schedule is generated every 1 ms. As a result, FSA has to identify the slice for each packet at a high rate and with low latency.

Key idea. Our key idea is to translate the uplink wireless schedule of an eRE into a sequence of forwarding actions that correspond to the sequence in which packets would be sent on the uplink by the eRE (see Fig. 1). Then we install this sequence of forwarding actions in the dataplane of the switch where they get applied to the arriving uplink packets in sequence. We keep updating the sequence of forwarding actions in the switch dataplane as the wireless schedule keeps changing every 1 ms.

It is not possible to realize this idea on traditional network switches because the "match-action" paradigm limits these switches to perform packet forwarding based on only the contents of the packet being forwarded. Instead, we use an emerging programmable switch [27, 40, 41] which allows us to do packet forwarding and scheduling more flexibly. Programmable switches provide flexible packet parsing (custom headers) and header manipulation through reconfigurable match-action pipelines. Most importantly, they provide transactional stateful memory that allows stateful processing across packets at line rate. This allows us to use additional information such as timing and byte count to make forwarding and scheduling decisions. As the transactional stateful memory can be updated in the dataplane at line rate, we can provide sub-millisecond timing guarantees on updating the sequence of forwarding actions to keep up with changes in the wireless schedule.

4 DESIGN & IMPLEMENTATION

Our design takes advantage of two key observations: (i) even though there is no explicit user identifier in the fronthaul (eCPRI) data packet, each packet contains a unique sequence number (Fig. 2) that increases monotonically, and (ii) packet arrival sequence in the fronthaul can be determined from the uplink transmission schedule which is generated by the wireless network scheduler. We leverage the uplink transmission schedule coupled with the monotonically increasing sequence number to identify the user.

High Level Workflow. In FSA, after a transmission schedule is generated by the network scheduler, it is first converted into a series of [sequence number, destination server ID] pairs (§4.1). This information is then sent to the switch via special "Schedule Packets" or *s-packets*. Each *s-packet* contains [sequence number, destination server ID] pairs corresponding to each user in the wireless schedule. To help differentiate packets from users in the wireless schedule, the sequence number in the pair refers to the sequence number of the last fronthaul data packet for the user. To store this information in the switch, FSA uses a "ring buffer" implemented using SRAM-based register arrays in the switch dataplane. To manipulate this "ring buffer" in the high-speed network data-plane, FSA additionally maintains a read and a write index. When the switch receives the *s-packet*, it adds the entries from the packet into the "ring buffer" starting at the location pointed to by the write index and increments the write index correspondingly. When the switch receives a fronthaul data packet, it reads the entry in the ring buffer at the location specified by the read index. Upon receiving all the packets of a particular user (based on the schedule), the FSA increments the read index. Finally, since the network schedule is generated for each base station separately, FSA maintains a unique ring buffer for each base station along with separate read and write indices for it. FSA uses the 48-bit source MAC address of the base-stations to redirect the processing to the right "ring buffer". In summary, FSA implements a high-speed dynamic forwarding scheme that runs entirely in the switch dataplane.

4.1 Wireless Schedule Conversion

FSA uses the wireless schedule to identify the slice/user for each packet in the fronthaul. This is feasible as all transmissions in the cellular network are scheduled by the wireless scheduler which dynamically allocates the resources in the wireless channel. The digitized radio samples of these scheduled users are then transmitted over the fronthaul. The number of samples generated in each time interval is dependent on base station attributes such as sampling frequency, wireless bandwidth, sample size, etc., as well as the functional split, which are all known a priori and are rarely changed. Using these attributes combined with the wireless allocation provided by the wireless scheduler, we can calculate the size of the digitized radio samples, as well as the number of fronthaul

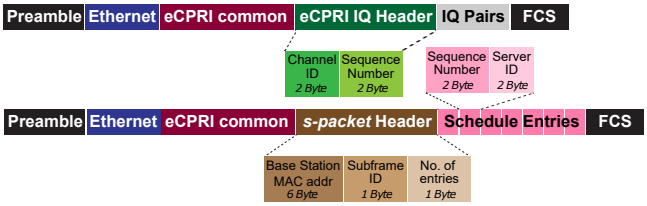


Figure 2: eCPRI header for two message types

packets for each user in the wireless schedule. FSA then uses this information together with the packet sequence number field in the eCPRI IQ Header (see Fig. 2) to identify the slice/user for each packet in the fronthaul at line rate.

Generation of *s-packets*. When a wireless schedule is generated, the scheduler also crafts the *s-packet*, which is a regular eCPRI packet with a custom message type and header fields (*"s-packet Header" and "Schedule Entries"* as shown in Fig. 2. The *s-packet header* contains the base-station MAC address which is used to identify the appropriate base-station to whom the schedule entries should be added. The field "Number of entries" in the header corresponds to the number of entries in the *s-packet header*. The subsequent headers consist of the list of schedule entries. Each entry corresponds to a user that is allocated resources in the wireless schedule and contains information that is written to the ring buffer to identify and route the user accurately.

4.2 Forwarding Information

Once the *s-packet* is received by the switch, FSA first identifies the base station ID for the schedule packet by using the "Base Station MAC address" field in the *s-packet* header. After identifying the base station ID, FSA:

(1) writes the [16-bit sequence number, 16-bit destination server ID] in the ring buffer at the index indicated by the write index
(2) increments the write index for the ring buffer
(3) decrements the header field "Number of entries" in the *s-packet*
(4) removes the corresponding schedule entry from the *s-packet*
(5) Recirculates the packet back if the "Number of entries" is not zero, thus implementing an updation loop
(6) Drops the packet if "Number of entries" becomes zero

Fig. 3 summarizes this process. Each entry in the ring buffer consists of a 16-bit sequence number and a 16-bit destination server ID, which is stored together in a 32-bit SRAM register. The 16-bit sequence number corresponds to the sequence number for the last fronthaul packet belonging to the user. Note that, we store a 16-bit server ID instead of the physical server destination MAC/IP address to reduce storage overhead in the ring buffer. To convert a server ID to its destination MAC/IP address, we use a standard match-action table, as the total number of servers in the C-RAN is mostly constant. This table is updated only when servers are added/removed in the C-RAN. Thus for each base station, we can support up to 65K unique destination addresses.

The ring buffer only maintains entries corresponding to users that have been allocated wireless resources in the next few milliseconds. As a result, the amount of SRAM memory required for storing the [sequence number, destination server ID] pairs is reduced significantly. However, it is crucial that such a compact representation can deal with loss and/or reordering of data/eCPRI packets as well as *s-packets* since the routing scheme relies solely on the sequencing of fronthaul packets. We address these issues in §4.4.

4.3 Ring Buffer

At the core of our solution is the design of the ring buffer data structure which enables high frequency dynamic forwarding in the switch dataplane. The switch dataplane provides high-speed transactional stateful memory in the switch dataplane is available in the form of SRAM-based register arrays. Implementing and maintaining the ring buffer data structure using such a memory requires handling the following constraints: (i) Due to the limited size of SRAM memory available (10's of MBs) in the switch dataplane, the ring buffer has to be implemented within strict memory constraints in order to scale to several hundreds/thousands of base stations and users. (ii) SRAM memory blocks in the switch dataplane are single-ported (for power and cost reasons) due to which the register arrays built using them can only read/write a single entry in one memory access. Further, to maintain line-rate processing in the dataplane, only one memory access is allowed in a single dataplane pass of a packet [42]. In this section, we describe how the ring buffer is designed while handling these SRAM memory constraints.

FSA maintains one ring buffer for each base station connected to the switch. Each ring buffer has at least 40 entries, as the wireless schedule is generated at least 4 ms in advance and each schedule contains no more than 10 users [43]. We also maintain a read and a write index for each ring buffer. The write index indicates the position where new schedule entries will be added by the *s-packets*. The read index indicates the position that contains the slice/user identity and the routing information for the next incoming fronthaul data packet. The range of sequence numbers between the read and write index indicates the sequence numbers of fronthaul packets scheduled for transmission in the network by the network scheduler in the next few milliseconds. By keeping packet forwarding information for only a short time window, we reduce the amount of SRAM required for the ring buffer.

To address the second constraint, we implement a ring buffer across multiple register arrays. Therefore, even though the programmable switch hardware allows access to a single index in a register array each pass, we can now read/write multiple entries by accessing data from each register array in parallel. We use the term *splitting ratio* to indicate the number of register arrays used for implementing the ring buffer. For a splitting ratio of n:1, the ring buffer is divided into n register arrays. Clearly, n is bounded by the number of register arrays that could be formed using the SRAM memory blocks available in the switch dataplane architecture. In our implementation, the splitting ratio is 4:1 (see Fig. 3).

4.4 Handling Reordering & Drops

As FSA uses packet ordering to build and continuously update a schedule-based routing table in the dataplane, it is extremely imperative to handle reordering and packet drops to maintain consistent routing. Fig. 4 shows examples of packet sequences that can lead to erroneous user identification (packets highlighted in red). Without proper care, the routing context loses synchronization and can route packets to the wrong destination in case of mis-identification.

One way to handle reordering/drops is to compare the received packet's sequence number to multiple entries in the ring buffer.

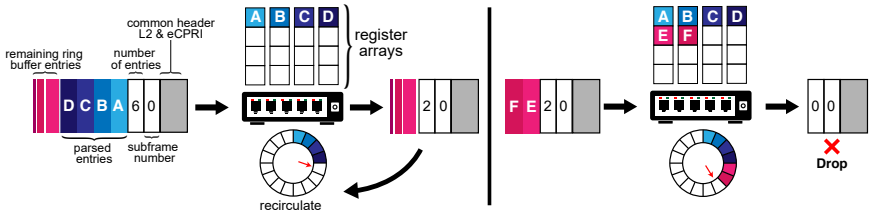


Figure 3: Adding entries in the ring buffer using the *s-packet*. The small red arrow denotes the write index. The *s-packet* is recirculated if it has remaining buffer entries (case on the left) and is dropped otherwise (case on the right).

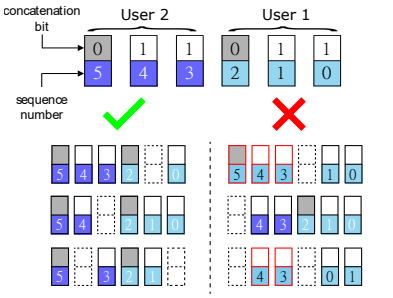


Figure 4: Valid and invalid packet sequences

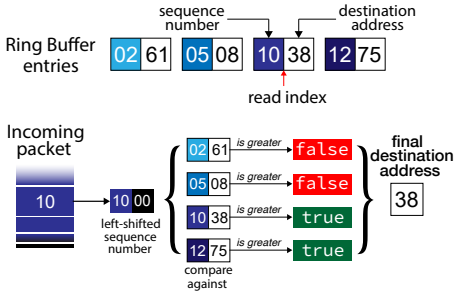


Figure 5: Handling reordering in the dataplane.

However, due to the limitations of the SRAM-based register memory, only one entry from each register array can be accessed in a single packet pass. However, as mentioned earlier(§4.3), since, we implement the ring buffer using n register-arrays, we can now look-up n consecutive entries for comparison to identify the correct slice/user in the presence of packet reordering and drops.

4.4.1 Reordering. The eCPRI protocol header includes a sequence number that allows us to detect message reordering [35]. In FSA, we do not guarantee ordering for packets that are received out-of-order at the switch. Instead, we detect the potential reordering and ensure that it does not cause mis-identification and cause routing errors. The logic to identify the reordering is provided below.

Upon an incoming data packet, we read n subsequent entries from the schedule to be compared against the sequence number incoming packet's header. The comparison can lead to the following subsequent steps:

(1) First, if the packet sequence number is less than all n values in the array registers, the packet has arrived too late. We can either drop the packet, multicast the packet to all servers, or forward the packet to a server that can determine the packet's destination address. We leave this decision to the cellular operator.
(2) If the packet sequence number is greater than all n values, the packet has arrived too early. In this case, we recirculate the packet within the switch to delay its processing.
(3) Finally, for cases where the packet sequence number matches one of the n entries, FSA simply routes the packet to the correct destination using the corresponding server ID (shown in Fig. 5).

4.4.2 Data Packet drops. Recall that there is no mechanism for data packet retransmission in the fronthaul due to the real-time nature of cellular networks. Therefore, a data packet drop directly impacts FSA's routing consistency when a user's last packet in the schedule is dropped. This packet can be identified using a specific bit (concatenation bit set to 0) in the eCPRI header as shown in Fig. 4. Such a drop causes the read index to not move forward correctly to the next user. It is important to note since FSA reads n forwarding entries in a single pass, forwarding can still continue correctly even when the read index lags behind (we call this drift). However, if this drift grows beyond n due to multiple packet drops, it can cause FSA to misidentify users and route them incorrectly.

To remedy the situation, we keep track of the drift, i.e. the difference between the read index and the index of the user identified by comparing the sequence numbers. When a packet with concatenation bit 0 is dropped, this drift value will increase with every subsequent packet. When the drift value increases beyond a threshold, we generate a "Drift Packet" which will adjust the read index based on the value of the drift. For example, if the drift value is negative, it implies that the read index has consistently lagged behind the actual user index identified by using the sequence number. Once the value reaches a threshold, we generate a "Drift Packet" and send it to the recirculation port. To generate the "Drift Packet", we clone any incoming packet and add a custom header before sending it to the recirculation port. When the packet recirculates, it will increment the read index to fix the error.

4.4.3 Schedule Packet drops. For handling *s-packet* drops we track the subframe ID (refer Fig. 2) of the most recent *s-packet* received for each base station. In case of a drop, when the switch receives an *s-packet* whose subframe ID is not the immediate next value, it generates a NACK packet in the dataplane to inform the network scheduler to retransmit the packet. Since a network schedule is generated 4 millisecond in advance and the round trip time in the fronthaul between the eREC and the switch is less than 10μs (§6.5), retransmission of an *s-packet* can be completed in time well before data packets corresponding to the schedule are received at the switch.

4.5 Putting it all together

Fig. 6 illustrates the key components of FSA on the switch dataplane and how eCPRI packets are processed. When the switch receives an eCPRI packet, first it checks whether the packet is a fronthaul data packet (IQ) or an *s-packet* from the scheduler. Then, FSA gets the base station ID of the packet using the source MAC address or base station MAC address field respectively. If it is a fronthaul data packet, it accesses the read index for the base station ID and reads

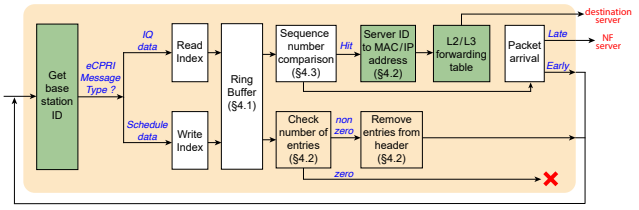


Figure 6: Key components of FSA

4 entries from the ring buffer. It then generates the 32-bit sequence number by left shifting the 16-bit sequence number in the packet header and compares it with the values read from the ring buffer. If there is a match against any of the entries, it directly accesses the MAC/IP address of the destination server using the server ID.

On the other hand, if the packet is an *s-packet*, FSA accesses the write index and writes up to 4 entries in the ring buffer. It then updates the number of entries in the header and recirculates the packet till all the entries are written in the ring buffer. Once all entries are written, the packet is dropped.

Overhead of FSA. In FSA, storing forwarding information in the ring buffer consumes SRAM-based memory resources in the dataplane. Considering that the network schedule is generated 4 subframes/slots in advance and there is uplink data for a maximum of 10 users in each subframe/slot, FSA requires 40 entries in the ring buffer per base station. Also, updating this forwarding information using *s-packets* incurs some amount of latency and bandwidth overheads. From our evaluation, we show that it takes an *s-packet* less than 2μs to write up to 10 entries in the dataplane as the packet needs to be recirculated multiple times since FSA can write 4 entries in each pass. For bandwidth, since FSA needs additional *s-packets* which are also recirculated, it affects both the switch pipeline processing and the link bandwidth consumption. This overhead scales linearly with the number of base stations supported by the switch.

FSA also consumes link bandwidth for transmitting *s-packets* from the network scheduler in the C-RAN to the switch in the fronthaul. In the evaluation (§6), we show that both the switch pipeline and the link bandwidth overheads are negligible for real network traces.

5 FRONTHAUL PACKET SCHEDULING

In §3, we described how FSA identifies the user (and thus slice) for a fronthaul packet in order to enable multipoint-to-multipoint routing. This information can be also used by the cellular operator to provide differentiated services such as packet prioritization in the fronthaul. In this section, we describe how FSA provides packet prioritization through scheduling by designing the Least Slack Time First (LSTF) scheduler in the switch dataplane.

5.1 LSTF

LSTF [44, 45] schedules packets in increasing order of slack time, i.e., the time remaining until each packet's deadline. The slack time for each packet is the amount of excess time left after subtracting the current waiting time and total processing time from its deadline.

$$slack\_time = (deadline - processing\_time) - waiting\_time\_experienced \quad (1)$$

In practice, LSTF can minimize average flow completion times, minimize tail latency and achieve per-flow fairness. All of these are desirable properties for a packet prioritization/scheduling algorithm in the 5G fronthaul to support use cases with different deadlines such as industrial automation (uRLLC), smart sensor networks (mMTC) and AR/VR, 4K streaming video (eMBB). In addition, a user can be allocated varying amounts of wireless spectrum in each schedule consequently requiring different processing times in the C-RAN. Therefore, it is beneficial to have a packet prioritization/scheduling algorithm in the fronthaul that takes into account the different packet deadlines such that packets with shorter deadlines spend little time queuing in the fronthaul.

Strawman solution. One approach is to include the packet deadline in the packet (via an additional header) at the source (eRE). The slack time can then be calculated by subtracting the wait time at each of the switch's priority queues. The rank of each packet will then be the sum of the slack time and the arrival time [46]. However, this requires the eRE to be aware of the user type, calculate the corresponding slack time and add it in the header.

Our approach. Our approach does not require any additional header. We calculate each component of the slack time as follows:

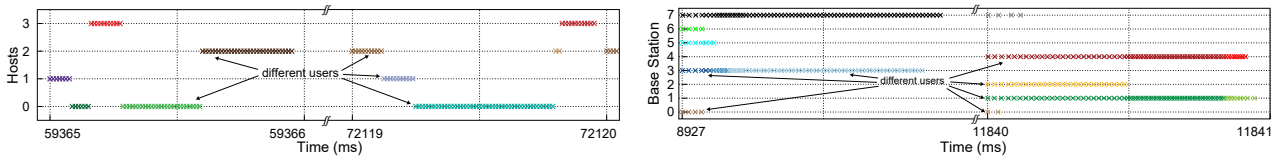
(1) Deadline: We create multiple partitions in the virtual server ID address space to address servers belonging to different use cases/MVNOs. This approach allows us to identify the user type and the corresponding deadline. We use a deadline of 0.6 ms, 2.5 ms and 2.5 ms for uRLLC, mMTC, and eMBB users respectively [43]. These deadlines can be changed according to the deployed network configuration.

(2) Processing Time: The processing time for each user depends on multiple factors such as allocated bandwidth, modulation scheme, number of spatial layers, etc. However, we know that the number of fronthaul packets for each user is also proportional to the allocated bandwidth and the number of spatial layers. Hence, we can use the packet sequence number stored in the ring buffers to calculate the relative processing time for each user. Note that the modulation scheme used does not affect the number of fronthaul packets for a user and has a limited impact on the baseband processing time required [47].

(3) Waiting Time Experienced: For each packet, we need to measure the expected one-way delay and add/subtract any additional delay/advance. This can be calculated as:

$$waiting\_time\_experienced = one\_way\_delay + (arrival\_time - previous\_packet\_arrival\_time - inter\_packet\_gap) \quad (2)$$

We use the one-way delay measurement mechanism available in eCPRI, which sends a packet periodically with message type *5*, to measure the one-way delay. This value can be stored either in a register in the dataplane or in a match-action table and updated via the control plane. To calculate the delay experienced by each packet in the fronthaul, we take advantage of the fact that the fronthaul traffic is constant-bit rate in nature and hence the inter-packet arrival gap is nearly constant. Therefore, if a packet experiences additional delay in upstream switches, the inter-packet arrival gap will be larger for the delayed packet. Subtracting the expected inter-packet gap from the calculated inter-packet gap gives us the amount



(a) Packet arrivals from the same base station

(b) Packet arrivals at Host 1 from different base stations

Figure 7: Snapshot of packet arrival sequences validating FSA's ability to enable multipoint-to-multipoint routing

of delay experienced by the packet till the current time. A more accurate measurement of the waiting time experienced using time synchronization is beyond the scope of this paper.

While traditional packet schedulers do not explicitly support LSTF, Sharma et al. [48] propose a novel way for implementing LSTF (approximated) on the programmable switch. Using this approach we are able to assign appropriate priority values to each fronthaul packet based on the slack time calculated in the dataplane. All uRLLC users are assigned value 7 which corresponds to the highest priority value (for 8 priority queues). The remaining users are classified into 7 different groups based on their user transmission size and experienced delay. Overall, the implementation of LSTF consists of only 20 lines of P4 code in the dataplane.

6 EVALUATION

FSA is implemented on an Intel Tofino switch in ~1000 lines of P4 code [49]. We use Intel P4 Studio 9.2.0 [50] to compile and load FSA's P4 code into the switch dataplane. We evaluate FSA on a hardware testbed using real and synthetic cellular network packet traces. Our key findings are:

- FSA is able to support multipoint-to-multipoint routing for 10 Gbps synthetic traffic from 8 base stations to 4 hosts.
- FSA is able to update routing information entirely in the dataplane with low latency (0.5 μs - 2 μs).
- With LSTF, FSA reduces the 95th percentile flowlet completion time for uRLLC users by 4x as compared to FIFO.
- FSA uses less than 10% and 6% of the SRAM and TCAM resources respectively in the switch dataplane.

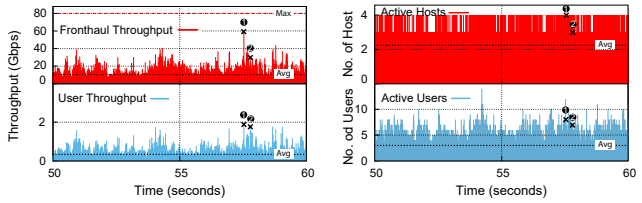
Experimental setup. Our testbed consists of 2 Wedge100BF-32X switches [51] with the Intel Tofino programmable ASIC and commodity servers. The first switch runs FSA and is responsible for slicing fronthaul traffic. The second switch generates the fronthaul traffic for multiple eREs using a packet generator. Additionally, the second switch is also responsible for receiving the fronthaul traffic after the eCPRI packets are processed and routed by the switch running FSA. For each packet that leaves the switch, we also obtain its egress timestamp from the port [52] and conversely for each packet we receive, we capture its ingress MAC timestamp. As the same switch is used for generating and receiving traffic, these timestamps are used to evaluate the correctness of FSA in §6.1

We use network schedules captured from multiple LTE eRE in a real cellular network across multiple days during the day. Users in the captured schedule are classified into different use cases based on their traffic patterns [53]. Each eRE has a bandwidth of 20 MHz and the sampling frequency for each eRE is 30.84 MHz. We scale the captured schedule by 8x to simulate a 5G eRE with 40 MHz bandwidth, and 4 layers. We assume that each sample consists of 15+15 bits of in-phase and quadrature (IQ) data. These samples are transmitted over the fronthaul using the eCPRI protocol [35]. Using the base station parameters (1200 subcarriers) and the network schedule, the packet generator switch generates fronthaul packets with appropriate header fields corresponding to each network schedule. We generate eight 1400 byte packets for each Resource Block which consists of 12 subcarriers resulting in a maximum bandwidth consumption of 9.6 Gbps per 5G eRE. The size of the packet accounts for the Ethernet, eCPRI common, and message headers along with additional parity bits added to the payload to ensure error-free decoding at the receiver.

6.1 Functional Verification

Slicing fronthaul traffic from a single base station. First, we verify that FSA performs fronthaul slicing correctly from a single base station whose users are allocated across 4 different hosts. As an illustration, Fig. 7(a) shows a snapshot of network schedules at 59365 ms, 72119 ms and 72120 ms. Each color in Fig. 7(a) corresponds to a unique user or Cell Radio Network Temporary Identifier (CRNTI). At 59356 ms, we observe that 5 users are allocated the wireless channel. Users in the network schedule are allocated to Hosts 1,0,3,0 and 2 respectively. Additionally, the packet arrivals across and within hosts are also equally spaced out due to the constant bit-rate nature of fronthaul traffic generated by the eRE. Similarly, for network schedules at time 72119 ms and 72120 ms, we can clearly observe that FSA is able to route packets for a different set of users with different host allocations. Overall, in this experiment, FSA processed 1.27 million eCPRI packets over a duration of 120 s with users distributed across 4 hosts.

Slicing fronthaul traffic from multiple base stations. Next, we verify that slicing is performed correctly when 8 base stations are transmitting to 4 hosts with each base station generating its network schedule independently. As an illustration, we plot two such network schedules at times 8927 ms and 11840 ms in Fig. 7(b). The plot shows a snapshot of two unique time-frames with a large number of active user transmissions all allocated to Host 1. Similar to Fig. 7(a), each color in Fig. 7(b) corresponds to a unique user. At 8927 ms, we observe there are 9 active users from 6 base stations that are mapped to Host 1. Although Host 1 receives a similar number of fronthaul packets from base station ID 3 and 7, there are three active users for base station ID 3 as compared to only one active user for base station ID 7. Similarly, for base station ID 4 and 5 which have similar amounts of fronthaul traffic, the number of active users are 2 and 1 respectively. Note that in the interest of simplicity, Fig.7(b) does not show the active users during these timeframes mapped to other hosts. Overall, we run the experiment for 120 secs, forwarding a total of nearly 10.5 million eCPRI packets from 8 base stations and verify that FSA is able to correctly deliver all eCPRI packets to the correct host.



(a) Fronthaul and User throughput

(b) Active Hosts and users

Figure 8: Fronthaul and Wireless statistics for network trace with 8 base stations

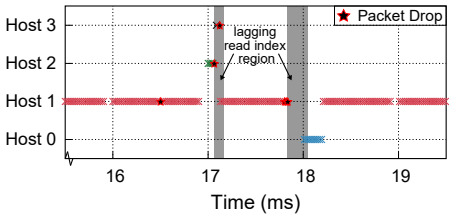


Figure 9: Routing in the presence of packet drops

6.2 Network and System Statistics

We also measure statistics such as fronthaul throughput, user throughput, active hosts, and the number of active users in our experiments. User throughput refers to the aggregate throughput observed by the end users within the cellular network for a given network schedule. Similarly, the number of active users refers to the number of users scheduled for transmission. Active hosts refer to the number of hosts which receive data within a particular network schedule.

Fig. 8(a) shows the fronthaul and user throughput for 8 base stations for 10 s of the trace starting from 50 s. Two points of interest are marked in the graph as points 1 and 2. We observe that fronthaul throughput at point 1 is twice as high as it is at point 2. However, the aggregate end-user throughput at both these times is almost the same. This can be explained by the fact that fronthaul throughput is independent of the modulation schemes used during wireless transmission. This is because fronthaul data consists of sampled radio signals which haven't been processed. Thus, even though the fronthaul throughput is higher at the earlier point, most users transmit using QPSK modulation and hence the effective user throughput achieved is much lower. Note that although high fronthaul throughput does not necessarily imply high user throughput, it does imply larger wireless channel allocation since fronthaul traffic is proportional to the amount of allocated wireless channel.

Fig. 8(b) shows the number of active users and hosts for the same time frame between 50 and 60 seconds. We observe that the average number of active users is ~3 for each network schedule for a combined traffic of 8 base stations. Thus, even in the worst case where each active user belongs to a unique base station, at least 5 base stations on average do not contain any active users and hence incur minimal overhead when slicing resource is utilized according to active user traffic. We also note that the average aggregate fronthaul throughput is around 10 Gbps while the maximum aggregate fronthaul throughput reaches 80 Gbps. Thus, the average bandwidth needed from the aggregate switch to the C-RAN servers is much lower than 80 Gbps. Hence, the multipoint-to-multipoint switching provided by FSA achieves significant savings in terms of bandwidth required to the C-RAN.

6.3 Handling Packet Drops

To evaluate how well FSA handles packet drops, we use the same setup as used in the previous section's experiments with 1 base station. During the experiment, we randomly drop fronthaul and schedule packets to observe the behavior of FSA. In the case of fronthaul packet drops, we expect FSA to continue routing packets to the correct destinations by correcting the read index in the dataplane itself. For schedule packet drops we expect FSA to generate a packet that is sent back to the network scheduler to ask it to retransmit the dropped packet.

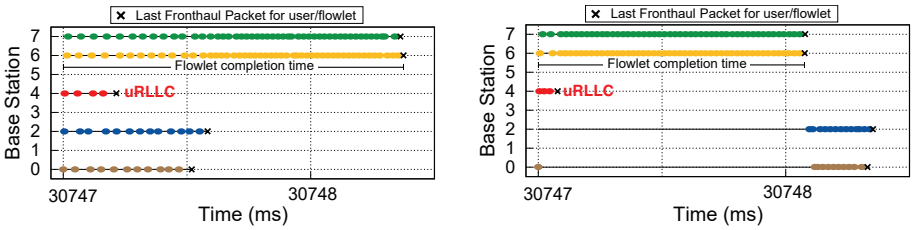
Fronthaul packet drops. Fig. 9 depicts a snapshot of the host packet arrivals when fronthaul packets are dropped. The first packet which is dropped is one of the fronthaul packets among many others belonging to the same user. As the dropped packet does not mark the end of the user's fronthaul packets, the read index in FSA is unchanged and packets continue to be routed to correct destination hosts. Note that since the drift value for read index is maintained separately for each ring buffer, drift for a particular base station does not impact the routing for users from other base stations.

Next, we demonstrate a case where the last packet (indicated by the concatenation bit 0) for a user is dropped. Here, we immediately see (in Fig. 9) that the read index starts lagging and the current user identified using the sequence number is at read index + 1. This is because the read index did not get incremented as the last packet for the user was dropped. As detailed in §4.4, once FSA observes the read index lagging for multiple consecutive packets, it generates a packet in the dataplane to fix the drift in the read index. The threshold for the number of consecutive packets can be set and updated by the cellular operator via the control plane at runtime. For our experiments, we set the threshold to 4. This implies that when we see 4 consecutive packets where the current user is identified at read index ± 1, FSA will automatically fix the drift to ensure that the current user always maps to the read index. Fig. 9 also shows the scenario where the last packet for consecutive users is dropped. We see that FSA continues to route packets accurately despite such an event, demonstrating the robustness and speedy nature of our dataplane solution.

Finally, Fig. 9 also shows a scenario where multiple consecutive fronthaul packets along with the last packet from the same user are dropped. Note that packets other than the last packet do not cause the read index to lag. Additionally, we observe that the time taken to fix the drift in the read index is dependent on the number of packets it receives from the base station, 4 in this case. This implies that in the absence of any packets from the base station the dataplane algorithm is not triggered. However, since no packet is sent by the base station, the drift in the read index does not affect the operation of the switch. As soon as the switch receives new packets, the solution is triggered and drift in the read index is fixed. Overall, when we introduce up to 7.3% of random drops in the transmission of 2717 eCPRI packets, there is no error observed in the multipoint-to-multipoint routing.

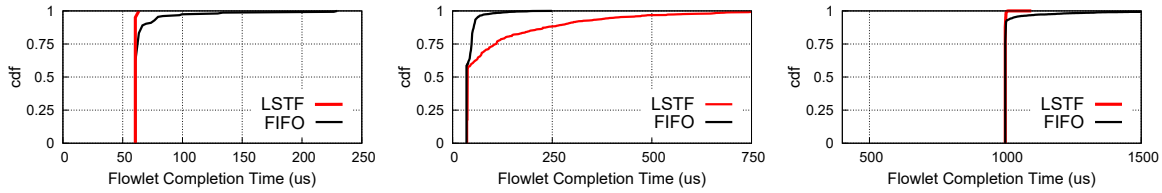
6.4 Packet Scheduling with FSA

To demonstrate the utility of packet prioritization enabled by FSA, we compare the completion time for each user in the network schedule when using a FSA-enabled user-aware LSTF scheduling policy



(a) FIFO (b) LSTF

Figure 10: Snapshot of packet departure sequence at switch under different packet scheduling algorithms



(a) uRLLC users (b) small eMBB flowlets (c) large eMBB flowlets

Figure 11: 95thpercentile Flowlet Completion Time between LSTF and FIFO scheduling for different user types

versus a simple FIFO scheduling policy without user-awareness. To understand why packet scheduling can have an impact on the slack, we note that in a single (1 ms) wireless allocation schedule, a user can be allocated multiple resource blocks which can translate to multiple eCPRI packets. However, in order to start processing the user in the C-RAN, all packets of the user have to be received by the server. In order to capture this behavior, we define a *flowlet* as the set of packets belonging to a user in a given network schedule. The metric we use is thus the time taken to transmit all the packets for a user. We call this the flowlet completion time (FCT).

Fig.10 depicts the impact of LSTF and FIFO scheduling policy on flowlet completion time for a given schedule. Since FSA enables us to identify each user's packets in the fronthaul, we observe that uRLLC user from base station 4 encounters minimal queuing in the switch as it is allocated the highest priority under LSTF. The flowlet completion for the uRLLC user under FSA is nearly 50% shorter compared to the completion time in FIFO. Additionally, we also observe that since LSTF favors large users with significant processing time over small users since the former has less slack time, the FCT for large users from base station's 6 and 7 is ~15% lower when using an LSTF policy instead of a FIFO policy.

Fig. 11 depicts the 95$^{\text{th}}$ percentile FCT for different user types and sizes. From Fig. 11(a), we observe that FIFO leads to increased FCT for uRLLC users. This increase of over 100 μs in for some users is significant and accounts for nearly 10% of the 1 ms E2E latency required for uRLLC users. Similarly, for large eMBB users, LSTF reduces flowlet completion by up to 250 μs which accounts for approximately 6% of the 4 ms E2E latency required for eMBB users. This is because users with large wireless resource allocation require much larger processing duration and therefore have lower slack time.

Naturally, there is a trade-off. We plot the FCT of small eMBB users in Fig. 11(b). We observe that since LSTF prioritizes critical and/or large users over small users, the FCT for small users increases significantly versus using a FIFO policy. However, since these users require significantly less baseband processing in the C-RAN, the increase in the network latency does not impact the overall performance for these users.

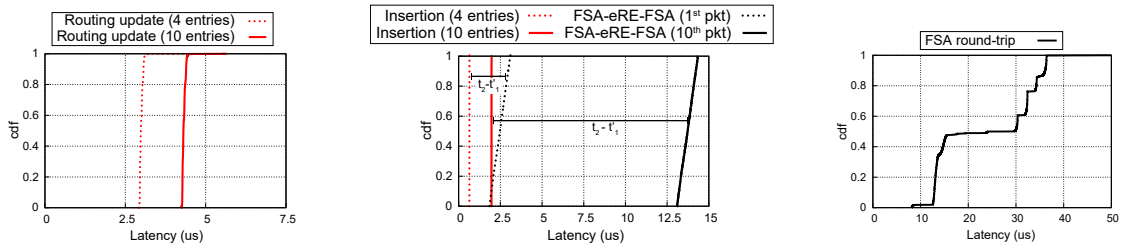
6.5 Latency in FSA

In this section, we show the latency incurred in various components of FSA. The setup is shown in Fig. 13 where 8 eREs send aggregated traffic of up to 80Gbps to 5 eRECs through the FSA switch. Each eRE is connected to the FSA switch via a dedicated 10G link and each eREC is connected to the FSA switch using a 25G link. 5 timestamps are recorded for control and the associated data packet transfers

- t_0**:** timestamp when an *s-packet* is transmitted by eREC.
- t_1**:** timestamp when an *s-packet* arrives at FSA switch.
- t_1'**:** timestamp when processing of *s-packet* is completed and all the routing entries are from the *s-packet* have been added.
- t_2**:** timestamp when the fronthaul packet corresponding to the *s-packet* arrives at the FSA switch.
- t_3**:** timestamp when the fronthaul packet corresponding to the *s-packet* arrives at the eREC.

Based on these timestamps, we compute the CDF of the following latencies. Fig. 12(a) shows the latency between the transmission of the *s-packet* (t_0) to the update of the routing entries t_1'. Since the local clocks on the eREC and the switch are not synchronized, we estimate the one-way latency (t_1 - t_0) as half of the round trip time between eREC and the switch. The results show that this *insertion latency* varies between 2.5μs to 6μs. When the *s-packet* contains 4 or 10 entries, the average latencies are about 2.9μs or 4.5μs respectively.

In order for routing to work correctly, processing of the *s-packet* has to be completed (t_1') before the arrival of the associated data packet (t_2). In Fig. 12(b), we show the CDF of ($t_1' - t_1$) the time it takes for the switch to insert the routing entries after receiving the *s-packet* and, ($t_2 - t_1$) the time it takes for the data packet to arrive at the switch after the associated *s-packet* is received. Since t_1, t_1', and t_2 are recorded at the FSA switch, all these timestamps are obtained from the internal clock in the FSA switch. The result shows there is a minimum time gap separation of 1.19μs between



(a) Routing update latency ($t_1' - t_0$)

(b) Insertion latency ($t_1' - t_1$) and packet arrival latency ($t_2 - t_1$)

(c) FSA round-trip latency ($t_3 - t_0$)

Figure 12: Latencies involved in FSA's end-to-end operation. Refer to Fig. 13 for timestamp labels.

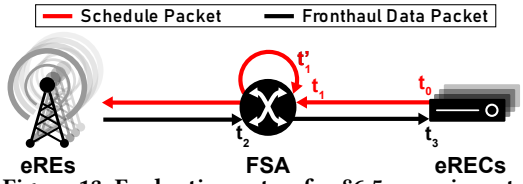


Figure 13: Evaluation setup for §6.5 experiments.

Table 1: Overhead for transmitting Schedule Packets

	Link Capacity	Pipeline
1 eRE	236.67 Kbps	1005.11 pps
Total for 1000 eREs	0.007%	0.10%
Max. for 1000 eREs	0.018%	0.33%

Table 2: Switch dataplane resources required FSA

Resource	switch.p4[54]	FSA
Exact Crossbar	29.36%	6.12%
Hash Bits	34.74%	6.51%
SRAM	29.58%	10.42%
Stateful ALU	14.58%	11.46%
TCAM	32.29%	5.60%
Ternary Crossbar	43.18%	4.55%
VLIW Instructions	36.72%	13.80%

($t_2 - t_1$) and ($t_1' - t_1$) for the first routing entry update to the first data packet arrival. For the subsequent updates and packets, the gap is even larger since the routing update takes even less. The result shows the minimum time gap separation between ($t_2 - t_1$) and ($t_1' - t_1$) for the 10 routing entry update and the 10^{th} packet is 11.11μs.

Finally, we look at the latency incurred in the fronthaul network running FSA, estimated as (t_3 - t_0). This result provides a measure of the additional latency added by FSA. Fig. 12(c) shows that this latency varies between 10μs to 40μs. The overall latency overhead added by FSA is thus small. In all the above evaluations, we assume that the eRE generates traffic immediately after receiving the *s-packet*. However, in LTE the gap between the schedule transmission and the corresponding uplink data reception is 4 ms [43]. During this 4 ms the eRE transmits the schedule to the user, the user processes the received schedule, the user processes the uplink data based on the received schedule, and finally transmits the processed uplink data back to the eRE.

6.6 FSA Overhead and Resource Usage

Overhead. We evaluate the overhead due to additional packets generated by the system to update network schedule information at the switch. In Table 1, we note that on average each base station's schedule can be updated at the FSA-switch with a usage of only 236.67 Kbps of link capacity and consumes only 1005 pps of pipeline capacity. Overall, for supporting 1000 eREs, the Schedule Packets will incur an additional overhead of 0.007% and 0.1% of switching and pipeline capacity in the FSA-switch respectively. We also estimate the maximum usage for 1000 eREs where each network schedule contains the maximum number of entries, i.e. 12. The maximum link capacity and pipeline overhead is 0.018% and 0.33% of the switch's total switching and pipeline capacity respectively.

Dataplane Resource Usage. As explained in §4.5, for each base station about 40 schedule entries are required to be stored in the SRAM-based ring buffer. However, considering shorter scheduling intervals by future 5G deployments, in this evaluation we consider 128 schedule entries per base station. Our prototype implementation supports 1000 base stations and therefore has a ring buffer of 128K schedule entries. Table 2 shows the various switch dataplane resources required by FSA as well as those required by the baseline switch.p4 [50, 54]. switch.p4 is a baseline P4 program that represents a standard L2/L3 switch with all the common networking features as well as support for VXLAN. We see that compared to switch.p4, FSA consumes very little resources in the switch dataplane. In particular, FSA requires only about 10% of the total SRAM, even while supporting 1000 base stations. Since the sum of values in both the columns is less than 100%, this implies both switch.p4 and FSA can easily fit within the resource constraints of the switch dataplane. This is especially useful for backward compatibility, which requires supporting legacy LTE/3G eREs on the fronthaul that requires standard L2/L3 forwarding support since all the processing for them is done at the eRE. In practice, due to the high throughput nature of the fronthaul traffic, the bottleneck for a fronthaul switch to support more number of base stations is likely to be the switch's maximum throughput capacity, rather than the switch's hardware resources.

7 DISCUSSION

So far we have discussed the design and implementation of FSA using the 7-2 functional split. In this section, we discuss how FSA can support the other functional splits. A survey of the different functional splits can be found in [55, 56]. The functional split options range from Split 1, where nearly all of the baseband processing is performed at the eRE, to Split 8, where only the RF processing is performed at the eRE and the raw IQ data is sent over the fronthaul to be processed at the eREC.

Table 3: Study on FSA support for different functional splits.

Protocol stack	Split	Max. fronthaul throughput (Gbps)	FSA-supported	Details
RLC & PDCP	1/2/3	3	Yes	Significant processing done at eRE and hence need FSA for multipoint-to-multipoint routing only.
RLC-MAC	4	4.5	Yes	FSA can also match on slice/user ID along with sequence numbers to identify and route packets.
Split MAC/MAC-PHY	5/6	7.1/7.1	Yes	Schedule information needs to be sent for each layer separately.
High PHY	7-3	15.2	Yes	Similar to 5/6.
Low PHY	7-2*	15.2	Yes	Detailed explanation given in this paper.
Low PHY	7-2x	15.2	Yes	Similar to 7-2 but requires duplication of packets for subframes containing SRS symbols.
Low PHY	7-1	60.4	Yes	Similar to 7-2x but schedule information needs to be sent for each antenna port separately.
RF-PHY	8	157.4	N.A.	FSA cannot separate user's as fronthaul data consists of IQ pairs in the time domain.

Table 3 summarizes the available split options and how FSA can support them. Each split represents a specific trade-off between the cost of the hardware (eRE and eREC) and the amount of fronthaul traffic required. For example, Split 8 requires the least amount of processing at the eRE but incurs the highest amount of fronthaul traffic while the reverse is true for Split 1. From Table 3, we can see that FSA can support split options 1-7 with little modification. For splits 1/2/3 the slice identifiers are available at the eRE and hence FSA needs to only provide high frequency dynamic routing to enable multipoint-to-multipoint fronthaul networks. Since the majority of the processing is done at the eRE in these splits, any packet prioritization in the fronthaul would only provide minimal gains. For splits options 4/5/6/7-3, FSA can enable fronthaul slicing by using ring buffers to store schedule information for different layers. Note that, in split 4, even though the slice/user information can be added to the fronthaul packet in the form of a VLAN tag to the eCPRI packet, operators cannot route packets simply on the basis of the VLAN tag using conventional L2/L3 switches. This is because the slice/user identifier information keeps changing frequently as a result of a large number of short user sessions [25]. Conventional L2/L3 switches cannot support the required high-frequency routing updates because of the high latency of the control plane [26] which is used for updates. Therefore, even when slice/user information is available in the packet, we still require FSA's SRAM-based high frequency dynamic forwarding scheme in order to enable multipoint-to-multipoint routing. For splits 7-2x/7-1, FSA would be required to duplicate some packets since fronthaul data packets can contain uplink data belonging to multiple users. FSA cannot support user-level slicing for networks using split 8, since the fronthaul data needs to be converted from the time domain to the frequency domain in order to separate different user's data. In practice, split 8 is rarely chosen since it is difficult to deploy eREs with fronthaul network links supporting the required throughput (~100's of Gbps).

We note that any implementation of fronthaul slicing which supports lower functional splits would need to provide additional support beyond simple slice/user identification. For example, supporting multipoint-to-multipoint routing requires marking each fronthaul packet with the destination eREC's address. Supporting packet prioritization requires the ability to implement packet scheduling schemes in the fronthaul switch. Accordingly, FSA also addresses issues beyond simply identifying the slice/user for fronthaul packets. It looks at how features such as multipoint-to-multipoint routing and slice-based resource allocation can be supported in the fronthaul network.

Clearly, FSA's cross-layer design has its limitations in the sense that more processing and state management is needed in the fronthaul switches. However, as we show in the evaluation, FSA's resource requirement in the switch dataplane is much smaller than the commodity L2/L3 switches (Table 2), and that it adds negligible overhead to the switch's dataplane performance (Table 1). We note that any solution for fronthaul slicing which supports lower functional splits would require a cross-layer design because slice/user identifiers available in the MAC layer are missing from the fronthaul packets as the eREs do not perform MAC layer processing. Such cross-layer designs have become easier to implement in recent years since Software Defined Networking has allowed operators to separate dataplane functions such as baseband processing from control plane functions such as network scheduling. Also, given that the industry is already using programmable switches in the 5G core network [57], we believe that FSA's reliance on programmable switches is in line with the current technology trends.

8 CONCLUSION

In this paper, we design and implement FSA, which provides slicing in the fronthaul network by using the wireless schedule to identify the slice/user for each fronthaul packet. FSA enables multipoint-to-multipoint routing through a high-frequency dynamic routing scheme. It also enables packet prioritization in the fronthaul network. FSA runs entirely in the dataplane of a programmable switch at line rate. Our testbed evaluation shows that FSA can update routing entries with a worst case latency of 6 μs and support accurate multipoint-to-multipoint routing for ~80 Gbps of fronthaul traffic. The slice-aware packet scheduling enabled by FSA's packet prioritization reduces the 95th percentile flow completion times by 4x for latency-sensitive traffic. We believe that FSA's benefits go beyond multipoint-to-multipoint routing and packet prioritization. FSA can also be used to support and enhance other features such as broadcast/multicast (MBSFN), coordinated multi-point, joint transmission, etc.

Acknowledgement. We thank our shepherd and anonymous reviewers for their valuable feedback on previous drafts of this paper. We also thank Changhoon Kim (Stanford University) and Jeongkeun Lee (Intel Barefoot Switch Division) for their helpful suggestions. This research was supported by the Singapore Ministry of Education Academic Research Fund Tier 1 (T1 251RES1910) and Tier 2 (MOE2019-T2-2-134).

REFERENCES

[1] SKT. SK Telecom's 5G Architecture Design & Implementation Guidelines, 2015.
[2] Huawei. 5G Network Architecture: A High-Level Perspective, 2016.
[3] Ericsson. The four key components of Cloud RAN, 2020. https://www.ericsson.com/en/blog/2020/8/the-four-components-of-cloud-ran.
[4] 3GPP. TS 28.500: Management Concept, Architecture and Requirements for Mobile Network that include Virtualized Network Functions. 2016.
[5] 3GPP TR 38.913. Study on Scenarios and Requirements for Next Generation Access Technologies. 2017.
[6] Thilina N Weerasinghe, Indika AM Balapuwaduge, and Frank Y Li. Priority-based initial access for URLLC traffic in massive IoT networks: Schemes and performance analysis. *Elsevier Computer Networks*, 2020.
[7] Eunkyung Kim and Heesoo Lee. Low-latency random access in wireless networks. *Elsevier ICT Express*, 2021.
[8] Xenofon Foukas, Mahesh K Marina, and Kimon Kontovasilis. Orion: RAN slicing for a flexible and cost-effective multi-service mobile network architecture. In *ACM MOBICOM*, 2017.
[9] Yasir Zaki, Liang Zhao, Carmelita Goerg, and Andreas Timm-Giel. LTE Mobile Network Virtualization. *Springer Mobile Networks and Applications*, 2011.
[10] Aditya Gudipati, Li Erran Li, and Sachin Katti. Radiovisor: A slicing plane for radio access networks. In *ACM HotSDN*, 2014.
[11] Ian F Akyildiz, Pu Wang, and Shih-Chun Lin. SoftAir: A software defined networking architecture for 5G wireless systems. *Elsevier Computer Networks*, 2015.
[12] Chengchao Liang and F Richard Yu. Wireless Virtualization for Next Generation Mobile Cellular Networks. *IEEE Wireless Communications*, 2015.
[13] Navid Nikaein, Eryk Schiller, Romain Favraud, Kostas Katsalis, Donatos Stavropoulos, Islam Alyafawi, Zhongliang Zhao, Torsten Braun, and Thanasis Korakis. Network store: Exploring slicing in future 5G networks. In *ACM MobiArch*, 2015.
[14] Peter Rost, Albert Banchs, Ignacio Berberana, Markus Breitbach, Mark Doll, Heinz Droste, Christian Mannweiler, Miguel A Puente, Konstantinos Samdanis, and Bessem Sayadi. Mobile network architecture evolution toward 5G. *IEEE Communications Magazine*, 2016.
[15] CPRI Consortium et al. CPRI Specification V7.0, 2015.
[16] Dongzhu Xu, Anfu Zhou, Xinyu Zhang, Guixian Wang, Xi Liu, Congkai An, Yiming Shi, Liang Liu, and Huadong Ma. Understanding Operational 5G: A First Measurement Study on Its Coverage, Performance and Energy Consumption. In *ACM SIGCOMM*, 2020.
[17] Open Networking Foundation. Aether, 2020. https://www.opennetworking.org/aether/.
[18] Nokia. AirScale Cloud RAN, 2019. https://www.nokia.com/networks/solutions/airscale-cloud-ran/#overview.
[19] Huawei. 5G Network Architecture: A High Level Perspective, 2016.
[20] ITU. 5G networks and 3GPP Release 15, 2019.
[21] 5G-PPP Architecture Working Group. 5G Architecture v3.0, 2019.
[22] Uwe Dötsch, Mark Doll, Hans-Peter Mayer, Frank Schaich, Jonathan Segel, and Philippe Sehier. Quantitative analysis of split base station processing and determination of advantageous architectures for LTE. *Bell Labs Technical Journal*, 2013.
[23] Gabriel Otero Pérez, José Alberto Hernández, and David Larrabeiti. Fronthaul network modeling and dimensioning meeting ultra-low latency requirements for 5G. *IEEE/OSA Journal of Optical Communications and Networking*, 2018.
[24] Anil Umesh Tatsuro Yajima, Toru Uchino, and Suguru Okuyama. Overview of O-RAN Fronthaul Specifications. 2019.
[25] Yaxiong Xie, Fan Yi, and Kyle Jamieson. PBE-CC: Congestion Control via Endpoint-Centric, Physical-Layer Bandwidth Measurements. *ACM SIGCOMM*, 2020.
[26] Maciej Kuźniar, Peter Perešíni, and Dejan Kostić. What you need to know about SDN flow tables. In *Springer PAM*, 2015.
[27] Intel Tofino. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series/tofino.html.
[28] 3GPP. TS 38.211 v15.2.0 : 5G; NR; Physical channels and modulation. 2018.
[29] Anil Umesh, Tatsuro Yajima, Toru Uchino, and Suguru Okuyama. Overview of O-RAN Fronthaul Specification. *NTT Docomo Technical Journal*, 2019.
[30] Wenfei Wu, Li Erran Li, Aurojit Panda, and Scott Shenker. PRAN: Programmable Radio Access Networks. In *ACM HotNets*, 2014.
[31] Line MP Larsen, Michael S Berger, and Henrik L Christiansen. Fronthaul for Cloud-RAN enabling network slicing in 5G mobile networks. *Hindawi WCNC*, 2018.
[32] MEF White Paper. Slicing for Shared 5G Fronthaul and Backhaul, 2020.
[33] IEEE. IEEE 802.1CM – Time-Sensitive Networking for Fronthaul, 2018.
[34] OBSAI. Reference Point 3 Specification V4.0, 2010.
[35] CPRI Consortium et al. eCPRI Specification V2.0, 2019.
[36] IEEE. 802.1Q-2014 - Bridges and Bridged Networks, 2014.
[37] 3GPP. 3GPP TS 23.203 Policy and Control Charging Architecture, 2015.
[38] Daniel E Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. Maglev: A fast and reliable software network load balancer. In *USENIX NSDI*, 2016.
[39] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. Silkroad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *ACM SIGCOMM*, 2017.
[40] Cavium. Xpliant ethernet switch product family, 2018.
[41] Intel. Flexpipe, 2018.
[42] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. *ACM SIGCOMM Computer Communication Review*, 2013.
[43] 3GPP. TS 36.213 v14.2.0 : E-UTRA Physical Layer Procedures. 2017.
[44] Joseph Y-T Leung. A new algorithm for scheduling periodic, real-time tasks. *Springer Algorithmica*, 1989.
[45] Radhika Mittal, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. Universal packet scheduling. In *USENIX NSDI*, 2016.
[46] Anirudh Sivaraman, Suvinay Subramanian, Mohammad Alizadeh, Sharad Chole, Shang-Tse Chuang, Anurag Agrawal, Hari Balakrishnan, Tom Edsall, Sachin Katti, and Nick McKeown. Programmable packet scheduling at line rate. In *ACM SIGCOMM*, 2016.
[47] Nishant Budhdev, Mun Choon Chan, and Tulika Mitra. PR^3: Power Efficient and Low Latency Baseband Processing for LTE Femtocells. In *IEEE INFOCOM*, 2018.
[48] Naveen Kr Sharma, Chenxingyu Zhao, Ming Liu, Pravein G Kannan, Changhoon Kim, Arvind Krishnamurthy, and Anirudh Sivaraman. Programmable Calendar Queues for High-speed Packet Scheduling. In *USENIX NSDI*, 2020.
[49] Github. FSA. https://github.com/NUS-CIR/FSA.
[50] Intel P4Studio. https://www.intel.com/content/www/xa/en/products/network-io/programmable-ethernet-switch/p4-suite/p4-studio.html.
[51] EdgeCore Wedge 100BF-32X, 2019.
[52] Pravein Govindan Kannan, Raj Joshi, and Mun Choon Chan. Precise time-synchronization in the data-plane using programmable switching ASICs. In *ACM SOSR*, 2019.
[53] Nishant Budhdev, Mun Choon Chan, and Tulika Mitra. Poster: IsoRAN: Isolation and Scaling for 5G RAN via User-Level Data Plane Virtualization. In *IEEE IFIP Networking*, 2020.
[54] Pravein Govindan Kannan, Nishant Budhdev, Raj Joshi, and Mun Choon Chan. Debugging Transient Faults in Data Centers using Synchronized Network-wide Packet Histories. In *USENIX NSDI*, 2021.
[55] 5G Fundamentals: Functional Split Overview, 2019. https://www.hubersuhner.com/en/documents-repository/technologies/pdf/fiber-optics-documents/5g-fundamentals-functional-split-overview.
[56] Line MP Larsen, Aleksandra Checko, and Henrik L Christiansen. A survey of the functional splits proposed for 5G mobile crosshaul networks. *IEEE Communications Surveys & Tutorials*, 2018.
[57] Kaloom. Cloud Edge Fabric. https://www.kaloom.com/products/cloud-edge-fabric.