SDN Implementation of Slicing and Fast Failover in 5G Transport Networks

Dimitris Giatsios, Kostas Choumas, Paris Flegkas, Thanasis Korakis
School of Electrical and Computer Engineering
University of Thessaly

Daniel Camps-Mur
Mobile and Wireless Internet Group
i2CAT Foundation

***Abstract*—Software-defined networking is at the root of future 5G transport network design. Among others, it allows for automated network reconfiguration and network slicing support. In this paper we present an OpenFlow-based implementation of a control plane area in the transport network architecture envisioned by the 5G-XHaul project. We analyze the implementation of the slicing mechanism at the network edge. Furthermore, we propose a simple low-overhead fast proactive failover scheme for recovering from single link failures, without the delays and packet drops associated with reaching a remote controller entity.**

I. Introduction

Transport networks for 5G cellular services are expected to accommodate multiple tenants through virtualization and to satisfy stringent QoS requirements, such as those related to the transport of fronthaul and backhaul interfaces for supporting centralized and distributed radio access network (RAN) deployments. In addition, the increased fluctuations in demand patterns due to network densification call for flexible transport architectures. Considering these challenges, the 5G-PPP Architecture Working Group has recognized that virtualization and softwarization will shape the future 5G architecture [1].

5G-Xhaul [2] is a project under the umbrella of the 5G-PPP initiative, focusing on challenges in the design of 5G transport networks. Its scope is to provide a flexible transport network consisting of multiple technologies, including wireless and optical segments, which will facilitate creation of virtual and isolated overlay networks belonging to different tenants.

In this paper, we present an Openflow-based implementation of functionalities in a control plane area of the 5G-XHaul architecture. We provide details on our flow-table design for mapping tenant slice traffic to transport tunnels. We also propose a proactive failover scheme which, given a link failure in the primary path of a tunnel, reroutes traffic to alternative paths, without the delay involved with contacting a remote controller. We measure the impact of this proactive approach in terms of the number of additional flow table rules that have to be installed in the transport nodes and examine how this scales with the size of the area. Our scheme works in parallel with the reactive approach where a remote controller with area-wide view reconfigures the network topology as needed to recover from arbitrary numbers of failures.

The remainder of this paper is organized as follows. In section II, we summarize the data plane abstraction and control plane architecture of the 5G-XHaul transport network. In section III, we provide more details on the Openflow-based implementation of transport node functionalities. In section IV, we present a proactive failover mechanism guaranteeing recovery from single link failures. We mention related work in section V and conclude the paper in section VI.

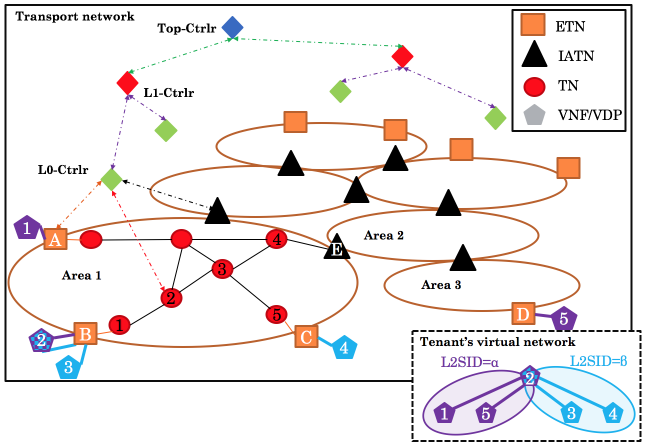


Fig. 1: Example of design and deployment of a tenant's virtual network.

II. 5G-XHaul Dataplane Abstraction and Hierarchical Control Plane Architecture

In terms of the dataplane abstraction, the 5G-XHaul transport network can be seen as a collection of isolated virtual networks. Each of these virtual networks is controlled by a tenant, which brings its virtual entities, namely Virtual Network Functions (VNFs) and virtual DataPaths (vDPs). VNFs are the end points and the vDPs are the tenant controlled datapaths.

The dataplane is based on three types of transport nodes which represent dataplane functions and are clustered into areas, as depicted in Figure 1. First, *Edge Transport Nodes (ETNs)* lie at the edge of the areas and are responsible for hosting the tenant's virtual entities. Second, *Inter-Area Transport Nodes (IATNs)* support the necessary functions to connect different areas, possibly implemented using different transport technologies. Finally, regular *Transport Nodes (TNs)* support an area specific transport technology, and provide forwarding services between ETNs and IATNs of that area. Specifically, the ETNs and IATNs are connected through transport tunnels, based on the forwarding services of the TNs. Figure 1 illustrates an example implementation of a virtual network over the 5G-XHaul transport network. More information on potential physical implementations of these nodes can be found in [3].

Each virtual network is a slice of the transport network.

Different local *Transport Slice IDs* may be used in each 5G-XHaul area. Having a notion of slice ID at the data plane is useful in order to easily deploy policies at the tenant or slice level. A transport slice is composed of i) virtual layer two segments and ii) virtual entities (VNFs/vDPs). A virtual layer two segment emulates a broadcast domain over a set of tenant virtual entities and is identified by a *Layer 2 Segment Identifier (L2SID)*. L2SIDs are unique system wide, meaning that L2SIDs cannot be reused within or across slices. VNFs are identified by a MAC address scoped to a single layer two segment. vDPs contain custom network control logic defined by the tenant, for instance they may correspond to a virtual switch. Unlike a VNF, a vDP may have several interfaces, each one connected to a different virtual layer two segment. Each interface of a vDP is identified again by a MAC address scoped to the L2SID where it is attached.

All transport nodes embed one major function, the *Forwarding Information Base (FIB)*. FIB is in charge of forwarding packets between VNFs/vDPs, which are either colocated in a single ETN or bound to different ETNs. In the second case, packets are inserted into pre-instantiated transport tunnels, implemented with use of encapsulation. Traffic from multiple slices can be combined into a single tunnel. The *Transport network Adaptation Function (TAF)* is responsible for pushing (or popping) the corresponding transport header, which signals at least three major pieces of information: i) the address of the destination ETN (which might be located in the same or in another area), ii) the Transport Slice Id and iii) the *tunnel ID*. Only ETNs/IATNs feature a TAF corresponding to the transport technology used in the area. For instance, in an Ethernet-based area, Provider Backbone Bridging (PBB) [4] can be used to encapsulate the packets.

The control plane of 5G-XHaul transport network is composed of a hierarchy of logical controllers, as illustrated in Figure 1. The top level controller, referred to as *Top controller*, is responsible for provisioning per tenant slices and orchestrating the required connectivity across areas and domains (e.g. optical transport domain, wireless transport domain). At the lowest level of the hierarchy we find the *Level-0 Area controller (AC)* that is responsible for the provisioning and maintenance of transport tunnels between ETNs and IATNs of a given area; a Level-0 controller operates at the level of individual network elements. A set of Level-0 controllers, which are technology-specific, are logically organized under a Level-1 controller. The latter is technology-agnostic, is in charge of maintaining connectivity between the corresponding areas, and operates with a higher level of abstraction, namely maintains state at the area level. Finally, all transport nodes (ETNs/IATNs/TNs) are directly controlled by *Local Agents (LA)* which are the glue between their datapaths and the ACs. The reader is referred to [5], [6] for a more detailed presentation of the overall control plane architecture of 5G-XHaul.

In this work, we focus on the area-level control plane and introduce a two-level approach where the AC interacts with the LAs residing inside (or in machines attached to) the transport nodes of the area. We have adopted an explicit tunnel setup approach in the transport network where the control plane functions proactively. Rather than reacting to an incoming packet, the SDN controller(s) populates the flow tables (tunnel setup) ahead of time for all traffic matches that could reach a transport node, thereby eliminating any latency induced by consulting a controller on every new flow when a packet-in event occurs. The result is all packets are forwarded at line rate, by merely doing a flow table lookup in the switches as it happens today where the forwarding tables of the network nodes are populated by the routing protocols.

The two-level SDN control architecture we propose in this work, combines a longer-term, off-line, centralised control implemented in Level-0 Area Controller with a more dynamic, on-line, distributed handling of traffic fluctuations and events implemented at the Local Agents in each transport node. The AC is responsible for proactively setting up all tunnels and allocating the available resources in the area it controls, including alternatives used for load balancing or failure recovery for a given provisioning period. The LAs are responsible for reacting to network events, perform dynamic allocation of resources and map or reroute incoming traffic to the tunnels provided by the AC. With this approach, LAs may be able to deal locally with certain events that may occur during network operation without triggering a reconfiguration from a remote controller, which might incur significant delays. Stability of the network configurations is guaranteed since all LAs, despite taking local decisions, are following the directives provided by the AC which has a network-wide view of the area. When LAs are unable to deal with unexpected events, they might issue alarms to the AC which will then trigger a whole area reconfiguration. As an example of this approach, we describe a proactive failover mechanism in section IV.

III. Details on the implementation of the 5G-XHaul transport nodes

Our transport node implementation requires support for PBB headers and multiple flow tables, hence it works for OpenFlow version 1.3 or newer. It has been tested on Ryu, one of the few frameworks supporting newer OpenFlow versions.

A. ETN

The main function of an ETN is to host virtual entities from several tenants, and to offer a datapath abstraction connecting them to the transport network. In Figure 1, ETN B hosts VNF 3 and both interfaces of vDP 2, and acts as their gateway to entities in other ETNs, e.g. VNFs 1, 4 and 5. Apart from the ports where local virtual entities are connected, it also features a port which connects it to all the other ETNs/IATNs in its area through transport tunnels; we will refer to it as the ETN's *external port*.

Each ETN maintains a set of five mappings with responsibility of its LA: i) *port⇒L2SID*, ii) *<L2SID, dst. MAC>⇒port*, iii) *<L2SID, dst. MAC>⇒dst. ETN*, iv) *dst. ETN⇒tunnel ID* and v) *L2SID⇒transport slice ID*. The LA is responsible for updating these mappings accordingly after instructions received by higher level controllers. Note that the ETN needs to be aware of any addition, removal or migration of virtual entities attached to an L2SID present in at least one of the ETN's own entities, no matter where this takes place in the transport network. This is implemented with the hierarchical control topology of 5G-Xhaul. The updating procedure is proactive. Whenever a tenant requests a modification in its virtual network (e.g. VNF migration to another ETN), the 5G-Xhaul operator first makes sure that the rules for all potential

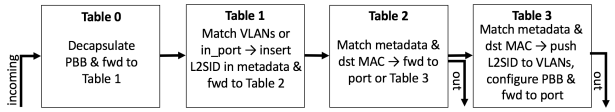


Fig. 2: ETN as a datapath.

flows from/to the affected virtual entities are installed before it allows the tenant to start sending flows through them.

ETN implements FIB and TAF and features four flow tables (cf. Figure 2):

- Table 0: packet decapsulation
- Table 1: packet classification for layer 2 segmentation
- Table 2: packet forwarding (FIB)
- Table 3: L2SID tagging and packet encapsulation

The utilization of four flow tables, instead of using the most typical approach of a single flow table, enabled the decrease of the total number of flow entries in each ETN. From now on, without loss of generality, we assume that all packets are Ethernet frames and PBB is used for their encapsulation.

Packets arriving at the ETN datapath first enter Table 0, which decapsulates them if they have been received through the external port. Packets received from local virtual entities are not modified. For example, in Figure 1, Table 0 of ETN B decapsulates the packets received from ETN C, but not packets coming from the local VNF 3. In both cases, packets are moved to Table 1 without a PBB header.

Table 1 marks the metadata for each packet it handles with the L2SID of the layer two segment the source virtual entity belongs to, and moves the packet to Table 2. In the case that the packet has been received from a local virtual entity, the *port⇒L2SID* mapping is used to determine the L2SID. For example, in Figure 1, ETN B marks with L2SID=β the metadata of all packets coming from the port connecting to VNF 3. If the packet has been received from entities hosted in other ETNs, then their L2SID is defined by their VLAN tags (one VLAN tag for maximum 4096 globally unique L2SIDs or two VLAN tags for even more). The VLAN tags are pushed before packet encapsulation in the source ETN. For example, in Figure 1, ETN B checks the VLAN tags of the packets coming from VNF 4 and marks their metadata with L2SID=β.

Table 2 forwards the packet to the virtual entity that i) is identified by the packet's destination MAC address and ii) belongs to the segment identified by the L2SID of the packet's metadata. In this way, layer 2 segmentation is achieved, that is, packets are forwarded only between entities belonging to the same segment. There is also full address space virtualization, meaning that different slices and segments can reuse MAC and upper layer addresses. The forwarding process utilizes the mapping *<L2SID, dst. MAC>⇒port*. If the port pointed to is the external port, then the packet is forwarded to Table 3. Otherwise, it is forwarded to the designated local port.

Table 3 first pushes the L2SID of the packets it receives (visible in their metadata) in their VLAN tags. Then it needs to push the PBB header. It puts its own address in the Backbone Source Address (B-SA) field. It resolves the destination ETN address from the mapping *<L2SID, dst. MAC>⇒dst. ETN* and puts it in the Backbone Destination Address (B-DA) field.

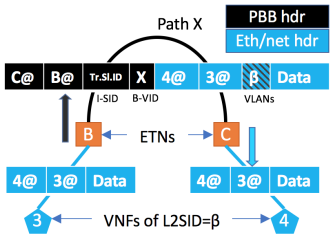


Fig. 3: Packet progress.

Then, it resolves the Transport Slice ID from the mapping *L2SID⇒transport slice ID* and puts it in the Backbone Service Instance Identifier field (I-SID). Recall that this ID is used for enforcing slice specific policies in the transport network, so we need it to be visible in the PBB header. Finally, the table uses the mapping *dst. ETN⇒tunnel ID* to determine the tunnel ID, puts it in the Backbone VLAN ID field (B-VID) and forwards the packet to the external port of the ETN.

As an example, Figure 3 illustrates the progress of a packet going from VNF 3 to VNF 4. In the bottom row we can see the tenant view of the Ethernet frames exchanged, which is agnostic of transport-specific details. In the top row, we see the form in which the packet traverses the transport nodes, with its inner VLAN tag set with its L2SID, and encapsulated with the PBB header.

The tunnel ID minimally defines the primary path from the source ETN towards the destination ETN/IATN, but can also be used to provision multiple backup paths, activated at TN-level. In Figure 1, ETN B encapsulates packets destined to ETN C with the ID of tunnel BC, while for packets destined to ETN D it pushes the ID of tunnel BE, as IATN E is the one connecting the areas of the source and destination ETNs.

B. IATN

Each IATN acts like a bridge between two areas, since it is in charge of moving packets from one area to the other, forwarding them to the appropriate tunnels. Tunnel selection for packet forwarding is based on the destination ETN. IATN does not host virtual entities, thus it handles only encapsulated packets. For each incoming packet, an IATN determines which tunnel will be used for forwarding it, changes its tunnel ID and forwards it through the appropriate port. For this purpose, IATN maintains the mappings *dst. ETN⇒port* and *dst. ETN⇒tunnel ID*. In addition, since different local Transport Slice IDs can be used in each area, the IATN also needs to account for relevant ID translations, hence it needs to maintain an additional mapping *Area X Tr. Sl. ID⇒Area Y Tr. Sl. ID* for any pair of areas X, Y it connects. For example, in Figure 1, IATN E receives through Area 2 a packet from VNF 5 that is destined to vDP 2, updates the B-VID with the ID of tunnel EB, translates the I-SID if necessary, and forwards it to TN4.

C. TN

TN's datapath implements FIB by forwarding packets based on their tunnel ID. AC is responsible for determining which tunnels are required for connecting the ETNs/IATNs of its area, and for establishing primary and backup paths for these tunnels. For example, in Figure 1, the path $B \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow C$ connecting ETN B and ETN C might

correspond to the primary path used for this connection. We get into more detail regarding the TN's datapath in the next section, after presenting our policy for fast failover.

IV. A SIMPLE APPROACH FOR PROACTIVE FAILOVER AT TN LEVEL

Fast recovery from failures is a crucial feature for any transport network. In the context of a 5G-Xhaul control plane area, the latency caused by waiting for a remote AC to take action (i.e. reactive failover) might be unacceptable. Another concern is related to the possibility of temporary loss of connection with the AC. Thus, proactive approaches incorporating alternative paths directly at the data plane or, in the worst case, stored at LA memory, appear attractive.

Our approach builds on the notion of the tunnel ID, which is part of the PBB header of packets traversing the TNs and determines the routing decisions. The tunnel ID in our design is not tied to a single node sequence, but also involves alternative routes towards the destination. We will be using the following terminology in this section, scoped to a given tunnel:

- *Path*: Any node sequence beginning and terminating at the source and destination ETN/IATN of the tunnel, respectively (and comprising only TNs in-between).
- *Primary path*: The default path used by the tunnel in the absence of link failures affecting it.
- *Backup subpath*: A node sequence beginning and terminating at nodes in the primary path (not necessarily the source/destination ETN/IATN), and featuring at least one TN not present in the primary path.
- *Backup path*: A path including a backup subpath in its node sequence.

Essentially, in the proactive failover procedure, whenever a link in the primary path fails, a backup path allows the flows using the tunnel to be rerouted through a backup subpath beginning at the node which detected the failed link or at a node preceding it in the primary path node sequence. This process is transparent to tenants utilizing these tunnels and even to the ETNs/IATNs communicating with them, and minimizes delays and dropped packets.

One thing to note is that, in principle, load balancing can take place for packets of a given tunnel ID, irrespective of potential link failures. This can be seen as a form of implicit failover. However, it does not deal with dropped packets until the load balancing mechanism completely avoids the malfunctioning path. Furthermore, centralized planning of load balancing solutions is based on different assumptions and guarantees with respect to planning failover mechanisms. In what follows, we assume that, at any given instant, traffic of a specific tunnel ID follows a unique path to the destination.

A. Proactive redundancy planning at the area controller

In general, the AC plans primary and backup paths for its tunnels based on a combination of considerations, reflecting its view of the network status and expected tunnel utilization. In this work, we do not study the specifics of the AC's policy for selecting these paths. Instead, we make the simplifying assumption that, for a given tunnel ID, the AC has already calculated the primary path and the subset of TNs which will be utilized for backup paths. Given this prior information, we focus on the problem of defining generic rules which the AC's path planning policy should follow, in order to:

- Guarantee recovery from an arbitrary failure of any single link in the primary path (given that all ETN/IATN pairs in the area still remain connected).
- Prevent loops irrespective of the number of failures that might take place.

To this end, we have identified two basic rules, which are sufficient conditions for a policy to meet the above requirements, while incurring a relatively small increase in the number of flow entries to be added in the transport nodes. To best illustrate these rules, consider the example topology of Figure 4, scoped to a single tunnel which transfers packets from ETN A to ETN B. In this topology assume that the primary path for the tunnel is the sequence $A \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow B$. The other TNs are only used for backup purposes.

The first rule is to only provision backup subpaths for failures of links whose *transmitting end* is a node lying in the primary path of the tunnel, and not a TN in backup paths. For instance, if the link $2 \rightarrow 3$ fails, the subpath $2 \rightarrow 6 \rightarrow 4$ is used to circumvent the failure. If, in addition, the link $2 \rightarrow 6$ is found not to be working, then the subpath $2 \rightarrow 1 \rightarrow 7 \rightarrow 8 \rightarrow 5$ is used. However, no backup paths exist for this tunnel in case, for instance, links $6 \rightarrow 4$ or $7 \rightarrow 8$ fail. Note that only providing a single backup solution ($2 \rightarrow 6 \rightarrow 4$) would still satisfy the single link failure recovery guarantee. The programming of the second failover subpath is thus not strictly required; however, we have included such subpaths in our design to make it even more robust.

The second rule is that whenever a backup subpath used to circumvent a failure diverges from the primary path, it may only remerge with it at a TN *closer* to the destination ETN/IATN with respect to the transmitting end of the failed link. For instance, if link $4 \rightarrow 5$ fails, TN 4 will not use subpaths $4 \rightarrow 6 \rightarrow 2$ or $4 \rightarrow 3 \rightarrow 2 \rightarrow 6 \rightarrow 4$, because after those diverge from the primary path they return to it at TNs 2 and 4 respectively, which are not closer to ETN B. Instead, the subpath $4 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 7 \rightarrow 8 \rightarrow 5$ will be used.

In order to support the full range of failover subpaths complying with the above rules, there is a small extension that needs to be made in the data plane. In some subpaths packets may move backwards along the primary path, such as in the subpath $4 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 7 \rightarrow 8 \rightarrow 5$ in the example. In order for this to work without potential loops, the packet must be tagged with an identifier related to the node that detected the failure, which we call failure identifier, or F-ID for short. The header field holding the F-ID will only be set to some nonzero value for backward moving packets. It is tunnel-specific, thus its maximum value is the number of hops in the primary path (e.g. only 3 bits are needed for 8-hop paths). It could be part of the B-VID, for instance. The F-ID will be matched by at most a single TN in the primary path. For instance, in Figure 4, if link $4 \rightarrow 5$ fails, the F-ID makes sure that the subpath $2 \rightarrow 6 \rightarrow 4$ is not used to redirect these packets (something that would create a loop).

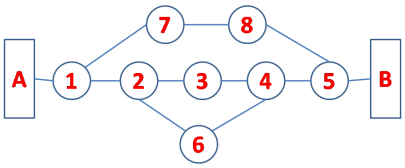


Fig. 4: Example topology to illustrate proactive failover rules.

B. Runtime failover procedure

In this subsection, we elaborate a little further on the required TN operations for supporting our failover approach, and describe the structure of the TN flow tables. To facilitate comprehension, we will describe the process for a single tunnel in an area, the one illustrated in Figure 4. In Figures 5-8 we depict the flow table structure for TNs 1 and 2 for this tunnel.

Whenever a TN in the primary path of a tunnel receives a packet, it checks whether the F-ID is set. If not, then pipeline processing moves to the group table, and specifically to a group of fast failover (FF) type whose first action is to forward the packet to the next TN along the primary path. The FF table has a built-in mechanism to check the state of the incident links associated with the actions in each action bucket. Thus, it may detect that its default outgoing link for packets of that tunnel ID is malfunctioning. In this case, if a subpath leading to a TN closer to the destination ETN exists, then a corresponding rule installed in this table redirects the packet to a backup TN. Otherwise, the TN sets the F-ID of packets belonging to this tunnel to a value declaring its own identity in the tunnel context, and forwards the packets back to the incoming port.

Now assume a TN in the primary path receives a packet with the F-ID set to a nonzero value. If the AC has installed a rule in this TN for it to redirect packets with this F-ID towards a backup node, then the F-ID is cleared and the forwarding action is applied. Otherwise, the packets are forwarded to the preceding TN in the primary path, retaining the F-ID.

Finally, consider a TN not in the primary path. This is the simplest case, as this TN will simply forward packets towards the next hop according to a flow entry installed by the AC. If such a TN detects a failure of the outgoing link for this tunnel, no backup path exists, and the packet is dropped.

There is a subtlety related to efficiency when packets are moving backwards. Ideally, whenever a TN receives a packet with a nonzero F-ID and it has a suitable subpath to reroute packets with this F-ID, we would also like this TN to reroute untagged packets of the same tunnel through this subpath, in order for them to avoid an unnecessary forth and back trip. However, this cannot be simply implemented with the OpenFlow fast failover group table. It requires retaining state at TN level (not packet tags). In this case, the above procedure should be extended with the TN sending the first tagged packet it matches also towards its LA. The latter will update the flow entry for untagged packets, in order to match the one with tagged packets. This incurs an added delay for the untagged packets received in the meantime, which is however small compared to waiting for the reaction of a remote AC.

C. Assessment of the amount of added flow entries

The number of rules installed in the TNs affects the FIB delay. Also, if this number grows excessively, low-end switching elements might not be able to store such amounts. Our proactive approach requires new flow entries to be stored at TNs which are not in the primary path of a tunnel, but have been selected as parts of backup subpaths. In addition, failover flow entries have to be installed in TNs along the primary path. The maximum number of flow entries depends on the size of the area, the proportion between ETNs/IATNs and TNs, the area topology, and the AC's path planning policy.

Match	Action
tun=AB, F-ID=0	Group AB.1
tun=AB, F-ID=2	Group AB.2
tun=AB, F-ID=4	Group AB.2
tun=AB	Fwd A

Fig. 5: Rule table for TN 1

Group	Type	Action Buckets
AB.1	FF	Fwd TN2 Fwd TN7 F-ID←1, Fwd A
AB.2	FF	Fwd TN7 Fwd A

Fig. 6: Group table for TN 1

Match	Action
tun=AB, F-ID=0	Group AB.1
tun=AB, F-ID=3	Group AB.2
tun=AB	Fwd TN1

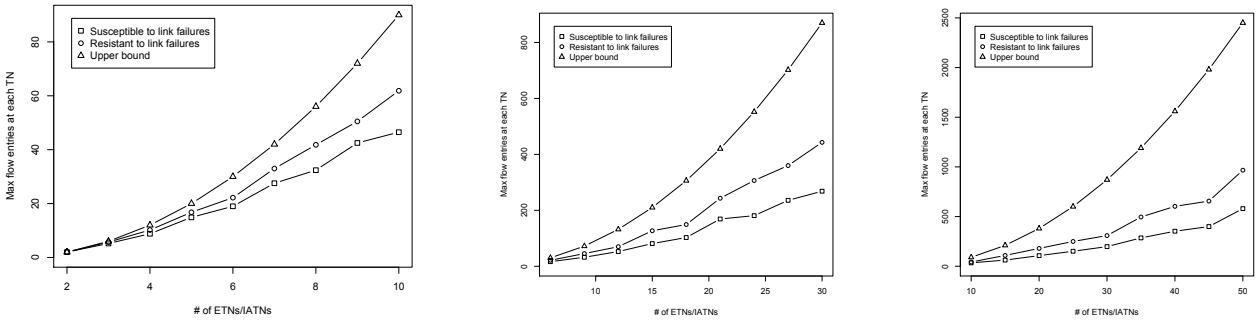
Fig. 7: Rule table for TN 2

Group	Type	Action Buckets
AB.1	FF	Fwd TN3 Fwd TN6 F-ID←2, Fwd TN1
AB.2	FF	Fwd TN6 Fwd TN1

Fig. 8: Group table for TN 2

In order to assess these numbers, we consider random network "Small-world" topologies of TNs and ETNs/IATNs, where each TN is connected in average with 4 other TNs, and vary both their total number and their proportion. We are also interested in assessing the relative increase with respect to the pure reactive failover case, where no redundant flow entries are used. The assessment has been done with use of the R toolkit, and we have used shortest path routing. The simulation results are depicted in Figure 9. We show the maximum number of flow entries required in each TN for different network sizes (10, 30 and 50 TNs). The number of ETNs/IATNs varies between 20% and 100% of the number of TNs.

The "Upper bound" plot corresponds to the maximum theoretical number of flow entries in an TN, if every flow were using all network TNs (ignoring backup entries). In this case, if N is the number of ETNs/IATNs in the network, each TN would have $N \cdot (N-1)$ flow entries. This bound is loose in practice; our simulations show that the real number of flow entries required is much lower, indicated by the plot "Susceptible to link failures". Our proposed policy for resilience to single-link failures increases the number of flow entries in the TNs. However, the fact that we do not provide guarantees for multiple failures (which would require backups of backups and so on) allows us to keep this increase in moderate levels, as we can see in the figure. Also note that even for the most demanding scenario we examine, the maximum number of entries remains below typical capacities of low-end switching elements, estimated at around 2000 entries [7].



(a) "Small-world" network with 10 TNs. (b) "Small-world" network with 30 TNs. (c) "Small-world" network with 50 TNs.

Fig. 9: Analysis on the maximum number of flow entries required in the TNs.

V. Related Work

Virtualization and tunneling techniques have traditionally been explored in the context of data centers. VL2, an early approach proposed in [8], does not provide full address virtualization. Instead, the full IP address space is partitioned into Locator Addresses, used by physical infrastructure switches, and Application Addresses, used by tenant VMs. Unlike VL2, Netlord [9] is a system that provides full L2 and L3 address virtualisation, meaning that tenants can use overlapping L2 and L3 address spaces, as we also do in our approach. Our scheme is similar to VXLAN, which is a standardized encapsulation mechanism [10] that enables virtualization of layer two segments. Similar to the MAC address space ID (MACASID) in Netlord, VXLAN defines a VXLAN Network ID (VNI) representing a L2 broadcast domain.

The topic of fast failover has gained new interest in recent years, following the SDN trend, and especially the capabilities offered by recent versions of the OpenFlow protocol. In [11] a new language for fault-tolerant networks programs is presented, called FatTire, which allows SDN programmers to specify failure-recovery policies. The FatTire compiler takes programs specified in terms of paths and translates them to OpenFlow switch configurations that automatically respond to link failures without controller intervention. In [12] some failover implementations for OpenFlow are proposed which ensure connectivity as long as the underlying physical network remains connected. The solutions involve tagging, without which fast failover mechanisms are severely limited, as it has been shown in [13]. The price to be paid for the ultimate connectivity guarantee provided by [12] is a large overhead in terms of tag bits. In contrast, in our work we provide more moderate guarantees, namely recovery of any single link failure, but manage to keep the overhead at very low levels.

VI. Conclusion

We presented an Openflow-based implementation for the basic functionalities of a control plane area in the 5G-XHaul architecture. Our design targets in providing robustness and efficiency. We have thus tried to minimize the amount of flow entries to support the required operations. Future extensions of our work will focus on QoS support at the transport nodes and its implications in designing efficient policies.

Acknowledgment

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671551. The European Union and its agencies are not liable or otherwise responsible for the contents of this document; its content reflects the view of its authors only.

References

[1] "5GPPP architecture working group, View on 5G architecture," https://5g-ppp.eu/wp-content/uploads/2014/02/5G-PPP-5G-Architecture-WP-For-public-consultation.pdf.

[2] "5G-XHaul project," http://www.5g-xhaul-project.eu/.

[3] "5G-Xhaul project, deliverable 2.4: Network topology definition," January 2017.

[4] S. Salam and A. Sajassi, "Provider backbone bridging and MPLS: complementary technologies for next-generation carrier ethernet transport," *IEEE Communications Magazine*, vol. 46, no. 3, 2008.

[5] D. C. Mur, P. Flegkas, D. Syrivelis, Q. Wei, and J. Gutiérrez, "5g-xhaul: Enabling scalable virtualization for future 5g transport networks," in *Ubiquitous Computing and Communications and 2016 International Symposium on Cyberspace and Security (IUCC-CSS), International Conference on*. IEEE, 2016, pp. 173–180.

[6] "5G-Xhaul project, deliverable 3.1: Analysis of state of the art on scalable control plane design and techniques for user mobility awareness. definition of 5G-XHaul control plane requirements," March 2016.

[7] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "Past: Scalable ethernet for data centers," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM, 2012, pp. 49–60.

[8] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, and S. Sengupta, "VL2: a scalable and flexible data center network." *ACM SIGCOMM computer communication review*, vol. 39, no. 4, pp. 51–62, 2009.

[9] "NetLord: a scalable multitenant network architecture for virtualized datacenters." *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 62–73, 2011.

[10] "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks, RFC 7348."

[11] M. Reitblatt, M. Canini, A. Guha, and N. Foster, "Fattire: Declarative fault tolerance for software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 109–114.

[12] M. Borokhovich, L. Schiff, and S. Schmid, "Provable data plane connectivity with local fast failover: Introducing openflow graph algorithms," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 121–126.

[13] M. Borokhovich and S. Schmid, "How (Not) to Shoot in Your Foot with SDN Local Fast Failover," in *International Conference On Principles Of Distributed Systems*. Springer, 2013, pp. 68–82.