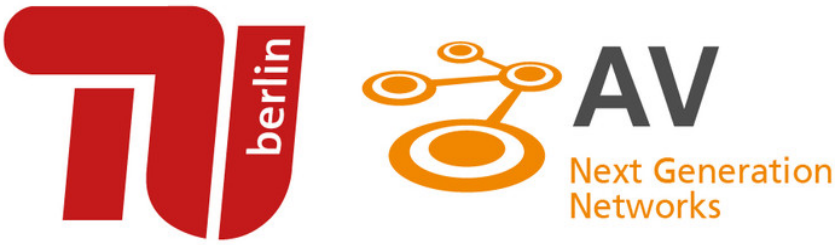


Design and Implementation of a Deterministic Multipath Solution for Optimizing Backhaul Performance in Geographically Distributed 5G Campus Networks

Nicolas Zunker

n.zunker@campus.tu-berlin.de

December 28, 2023

MASTER'S THESIS

Telecommunication Systems Institute

Technische Universität Berlin

Examiner 1: Prof. Dr. Thomas Magedanz

Examiner 2: Prof. Dr. Axel Küpper

Advisor: Dr. Ing. Marius Corici

Declaration

I hereby declare that I have created the present work independently and by my own without illicit assistance and only utilizing the listed sources and tools.

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschliesslich unter Verwendung der aufgefährten Quellen und Hilfsmittel angefertigt habe.

Die selbständige und eigenständige Anfertigung versichert an Eides statt:

Berlin, December 28, 2023

Nicolas Zunker

Acknowledgment

I would like to thank my unofficial advisor Hauke Buhr for his help, advice and support. I also thank my advisor Marius for his insights. Both of them provided excellent constructive feedback to me, for which I am grateful. I am also indebted to the rest team at Fraunhofer FOKUS for all their help, especially Hemant Zope, Eric Troudt, Christian Scheich and Aleksandar Yonchev. Lastly I would like to thank Prof. Dr.-Ing. habil. Falko Dressler and Prof. Dr. Zubow whose courses I enjoyed immensely during my studies, and from whom I learned a lot.

NZ

Abstract

TODO

Zusammenfassung

TODO

Table of Contents

List of Figures

List of Listings

Chapter 1

Introduction

1.1 Motivation and Problem Description

One of the aims for the fifth generation of mobile networks (5G) and it's successors will be a greater diversification of the classes of service. As the use cases for these networks evolve, there is a greater need for quality of service (QoS) tailored to each use case. For example, in the Industrial Internet of Things (IIoT) the requirements on latency, jitter, and reliability may be extremely stringent. Supporting these kinds of classes of service can be a challenge for mobile network operators (MNOs) and will require novel approaches to familiar problems, such as backhaul.

As there are more heterogeneous edge deployments and more campus networks, backhaul becomes more challenging, since many sites may not have access to optical fibre, and may be forced into using other solutions such as satellite links, mmWave backhaul, or pre-existing on-site ISP connections. Providing the kind of deterministic quality of service that these sites may require can be a very difficult challenge.

Particularly with the rise of satellite backhaul options, fuelled by the new space race, many

remote deployments may choose to integrate satellite backhaul because fibre is infeasible or its rollout is too slow. These connections are usually being added in addition to existing one, and thus network operators may choose to utilize more than one backhaul connection at the same time. Either in order to increase the available bandwidth or to utilize the different qualities of the backhaul links. This bears the question whether multipathing could then be used to provide deterministic backhaul by intelligently selecting on which links to forward packets. This approach bears similarity to multihoming as well as to multi-path routing in Wireless Sensor Networks (WSNs), and can take inspiration from the substantial body of research in these fields which already exists, and which has demonstrated that QoS can be improved by using multiple links or paths simultaneously [1, 2, 17, 19, 22, 35].

1.2 Goal

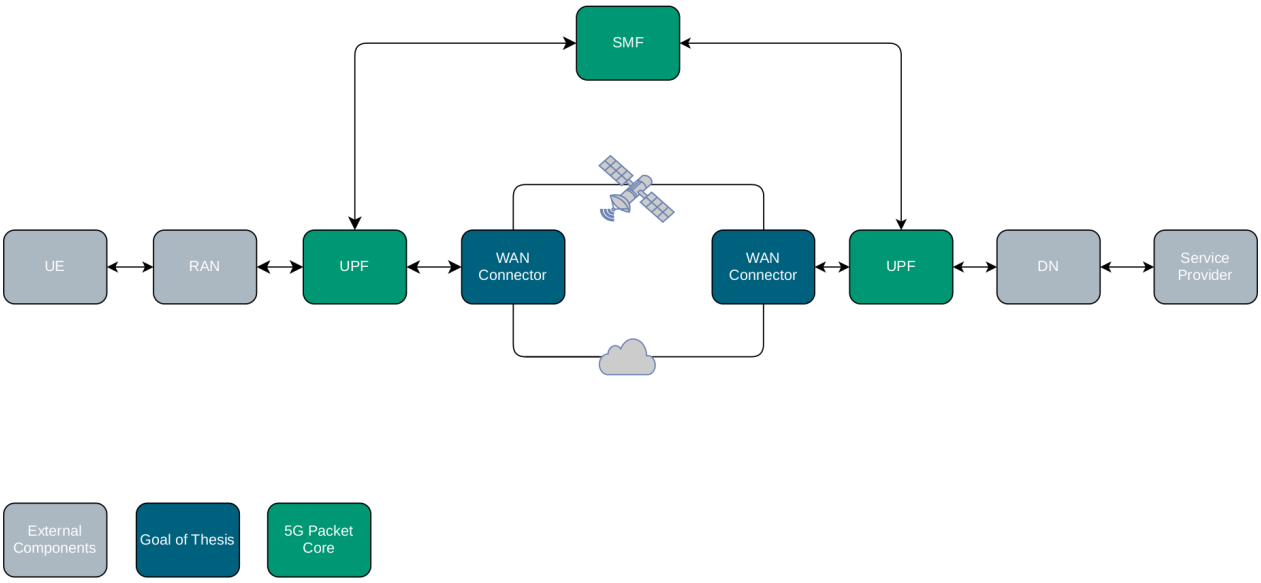


Figure 1.1: 5G Deployment with 2 UPFs

The goal of this thesis is to design and proivde an implementation of a Wide Area Network (WAN) connector, that can be placed at the ingress and egress point of two or more locations, and utilizes multipathing in order to provide deterministic backhaul between the two sites.

The performance of this approach will then be quantitatively analyzed in experiments.

Looking at Figure 1.1 we can see how this is envisioned to work. WAN connectors are deployed both in the geographically distributed 5G campus network, which has more than one egress link, and in the core. Then, using the multiple outgoing links, the traffic is backhauled to the other site, while respecting it's QoS requirements. This can be especially beneficial for critical applications (e.g. at industrial sites), which are in locations that do not have access to optical fibre for backhaul.

For a 5G deployment the proposed WAN connector could be deployed in between two User Plane Functions (UPF) , in order to provide deterministic backhaul. The architecture for such a deployment is shown in Figure 1.1. Further, the on-site UPF is not a strict requirement; it is also feasible to connect the RAN directly to the WAN Connector.

1.3 Structure

This thesis will follow a simple 5 chapter structure. This section concludes the introduction chapter, what follows will be one chapter to provide both basic background information as well as to highlight the existing literature which is of relevance to the problem statement. Next will be the approach and evaluation chapters, where, respectively, the design of the implementation will be presented and discussed, and then it's performance will be analyzed. The conclusion chapter will review the relevant findings, the successes and failures of the approach, and the possibilities for future research in this area.

Chapter 2

Background Information and Related Work

2.1 Mobile Networks and Backhaul

2.1.1 Backhaul in 5G

2.1.2 5G and Campus Networks

2.2 Multipathing and Multihoming

2.2.1 Collecting Per-Link Metrics

Measurement-based Metrics

In [2] the authors collected both passive metrics (looking at response times for outgoing packets), and active measurements (sending ICMP ping, or TCP SYN messages and measuring

the response time). Using the passive measurements enabled their multihomed approach to perform well, but when using the active measurements the performance was better. Crucially, the passive measurements worked better over larger sampling periods, because it took longer to get a full overview of all the possible routes. Whereas the active sampling approach acquired it's measurements faster and was thus more effective over smaller sampling intervals.

Considering these results, it is proposed to utilize both active and passive measurements. All three metrics- packet loss, latency, and jitter- will be periodically measured in an active manner. The period over which to perform these measurements is an important design decision for the WAN connector, and it will be met later.

Beyond this, these metrics will also be monitored on a passive basis wherever possible. In order to measure either the time needed (for latency and jitter), or to ascertain that a packet has been lost, a response is required for each outgoing message. This may only be possible for TCP's SYN and SYN ACK messages as well as other protocols which are guaranteed to contain request-response handshakes, and thus complicates the process of passive measurement.

For classical wired links in multihomed scenarios, [34] have observed that one link will generally dominate with regards to latency, but with brief periods where other links' performance is superior. These same authors also note that with regards to packet loss there is far less prevalence of a "dominant" link, and generally the links will perform comparably. Packet loss is also a particularly difficult metric to measure, since most links are highly reliable and when they do experience packet loss it is in bursts [34]. Wireless connections are usually less reliable and may experience more consistent rates of packet loss at the data link layer, however it is opaque from the perspective of the higher layers, which may only perceive it through the jitter and/or latency.

Ultimately, many of the best performing approaches for predicting packet loss, e.g. Hidden Markov Models [8, 34], are still somewhat imprecise and inaccurate. These models assume

the link is in one of two states, good or bad, and each state has a different probability for packet loss, and there is a transition matrix which represents the probabilities of switching from one state to another. This thesis will also use such a model to try to predict packet loss.

2.2.2 How to Guarantee QoS

Limiting Jitter

The design of an approach which guarantees QoS is also an interesting challenge. One idea to improve jitter when backhauling across multiple links is to duplicate packets and forward them on multiple links, and have the WAN connector on the other end buffer incoming packets and release them at a constant rate. This way, in the event of a packet being lost on one link, the other link is still able to receive it and the delay caused by retransmission is avoided. The downside of this approach is that it guarantees that the latency will always be as slow as the slowest link.

Low Delays

For reducing latency it would appear likely that the simplest approach may be a greedy method (as in [17], in the online case) which always selects the lowest latency connection. However there is room for nuance here since the connection must not be overloaded and also because certain traffic may have very relaxed latency requirements but use up more bandwidth. This means monitoring the load on any one link will be important. Finally there are also more intelligent approaches, i.e. integer linear programming (used in [22], and used for the offline case in [17]) which find an optimal solution satisfying the given requirements.

The timescale over which to use a chosen link is also of interest. In [19] the time for which a link should be used is varied based on the predicted qualities of the link. These predictions

are made based on past performance.

Error Rates

Reliability presents yet another challenge. However in a multihomed scenario it becomes easier to guarantee this via duplication, and/or forward error correction (FEC). For example if a packet flow requires 99% reliability this can be guaranteed by duplicating packets across two links which are both only 90% reliable. Alternatively, in such a situation, an FEC configured for 10% packet loss could be used to pre-code the packets sent across one of the links, and thus increase the reliability to the required level.

Although duplication uses a lot of bandwidth, in order to support 5G's ultra-reliable low latency (URRLC) QoS requirements, which are especially relevant for IIoT applications, it may be the only option for certain traffic flows. Forward Error Correction is an excellent protection against consistently lossy links, however it may fail to be reliable when there are concentrated bursts of dropped packets, which is a more common occurrence in packet switched networks. Either way, the effectivity of FEC in a deterministic backhaul unit is an interesting question which this thesis may also explore.

2.3 Determinism in Computer Networking

2.3.1 Quality of Service (QoS)

2.3.2 Deterministic Networking Specification

2.3.3 Traffic Shaping

Chapter 3

Approach

To discuss the solution implemented for this thesis requires a review of, firstly, the requirements contained in the problem statement, and, secondly, a description of both the decisions made as well as the process behind making those decisions. To that extent, this chapter will discuss, the components needed in order to address the problem statement. It will try to discuss not only what decisions were made, but also, crucially, what decisions were *not* made, and why. The first section will begin with the fundamental nature of the solution's architecture, thereafter a review of the design requirements which arise from the Deterministic Networking specification will follow, finally the precise nature of the solution's internal workings will be presented.

3.1 Requirements

Concretely, a multipath backhaul solution, as has been described in the introduction, will need to be able to transport the 5G traffic from the edge deployment to the core, while respecting the traffic's QoS requirements. This breaks down into a short list of hard requirements.

1. Forward only the appropriate traffic (as is defined in the 5G flow descriptions)

2. Manage multiple outgoing paths

3. Co-ordinate flows and their requirements with the Control Plane

4. Select the best path to enable the flow's requirements and achieve determinism

3.2 Skeleton Structure

At the outset of this thesis it was decided to use a Control User Plane split. The packet processing and forwarding is performed in the user plane, and the control plane makes the high level decisions about which packets to send where, and in this case, on which outgoing interface to send them. The data plane is kept simple and only performs the time critical packet processing, and blindly follows the instructions of the control plane. This type of architecture is very common in modern software-based networks, for example OpenFlow, the Evolved Packet Core (from LTE), and the 5G core all implement this kind of Control-Data-Plane split.

However, this decision immediately locks one into certain positions. For example, this requires a method of communication between the data plane and the control plane, it also means the state machine that represents the WAN Connector becomes more complex, and complexity always brings with it a greater danger for hidden mistakes and fallacies. The benefit is that the user plane function does not need to perform any of its own decisions, thus simplifying its internal architecture, and these components can be deployed at different locations, and scaled up or out with greater ease.

3.3 Architectural Components Required for Deterministic Multipath Backhaul in a 5G Campus Environment

Determinism, in a networking context, refers to the ability of a network to supply specific QoS requirements with extremely high degrees of certainty. IP flows and applications running in these networks are assured they will experience bounded latency, jitter, and packet loss. To this extent, the Determinsitic Networking specification (also called DetNet) is a specification with recommendations for how to achieve determinism. However it does not define determinism. A network which does not fully comply to these specifications may still call itself determinstic (but it may not say it is a DetNet). Since this thesis aims to provide determinstic backhaul, it is not required to follow the DetNet specifications in any form. Though, it does make sense to consider what solutions they have proposed for determinism, since ultimately the goals are the same. In this section, therefore, the DetNet specifications will be used as a source of inspiration to gleam ideas, but not as an absolute source of truth.

Deterministic networks have been discussed in the previous chapter. In order to guarantee determinism the IETF DetNet working group has proposed an architecture for determinism over IP networks [**detnet-arch**]. Their specification identifies four key mechanisms for guaranteeing the determinism of a flow: 1) elimination of contention loss, 2) jitter reduction, 3) service protection, and 4) explicit routes. Since the 5G campus environment may need to use the infrastructure of other operators this rules out the ability to use explicit routes. A key difference between deterministic networks and 5G campus backhaul networks is that the operator may not control all the links between nodes, and that there are only two nodes in the network that are definitely under the administrator's control- the WAN Connectors at the core and the edge. In such a scenario the problem is less like a network and more like a client - server application with multiple paths between the client and the server.

3.3.1 Elimination of Contention Loss and Bufferbloat

In order to prevent packet loss due to buffer overflows, and, just as importantly, to prevent high latencies due to bufferbloat [**bufferbloat**]or excessive queuing, contention between flows must be carefully managed. A greedy flow, which uses too much bandwidth, or the presence of too many flows, whose cumulative bandwidth demands exceed the link's capabilities, will have negative effects on all the flows running over the link.

Elimination of contention loss can be achieved by using a traffic shaper and/or rate limiter, and the ingress to any DetNet domain **must** apply such a function. For the WAN Connector therefore a traffic shaper must also be applied to the traffic on ingress, from the RAN, before it is sent across the backhaul links. Since the implementation of a traffic shaper is beyond the scope of this thesis, an existing solution must be used. The traffic control subsystem in the linux kernel (TC) provides several implementations of different algorithms for both rate limiting and traffic shaping. Some of these algorithms have already been discussed in the previous section. For the purposes of this solution several algorithms were considered: Hierarchical Token Bucket (HTB), Hierarchical Fair Service Curve (HFSC) [**hfsc**], Time Aware Prioriy Queueing (taprio), Common Applications Kept Enhanced (CAKE) [**cake**], and Fair Queuing with Controlled Delay (FQ_CoDel) [**fq-codel**].

For HTB and HFSC, due to the implementation of these algorithms in the linux tc subsystem, integrating them within the context of this solution would require filters for each flow. This is not necessarily a hindrance yet, however it would introduce additional overhead upon the creation and/or addition of each new flow. Perhaps more importantly, the implementation of the HTB algorithm provides no means for fair queuing. This means that while it is not possible for greedy and/or malicious flows to use up the bandwidth of other flows because HTB guarantees bandwidth, the greedy flows can cause other flows to experience higher latency because it does not guarantee delay. The HFSC algorithm improves on this by offering both bandwidth and delay guarantees. However this makes it far more difficult to

configure, and would require precise calculation of the service curve for each new flow.

Another feasible option available in linux TC would be the time aware priority queuing (taprio). This queuing discipline provides scheduled gates for specific traffic classes. It could be used to schedule flows according to their requirements and/or priorities, and thus provide a form of prioritization, and, to an extent, traffic shaping. Flows can be assigned different windows and congestion due to a greedy flow is avoided if it is scheduled for a different window. However one issue is that taprio fails to provide fair queuing, and thus would need to be configured and possibly adapted as new flows are added and removed, depending on their priority. Greedy flows of the same priority would have to fight each other to use the bandwidth available in their assigned window.

Lastly there are FQ_CoDel and CAKE. Both provide fair queuing as well as active queue management, which means flows are less likely to experience loss due to congestion, particularly congestion caused by one particularly greedy flow. Since the CAKE algorithm was originally designed for routers in WLAN networks, and these are loosely analogous to a 5G campus network (consider the User Equipment as the Stations, and the Base Station as the Access Point), it makes it an attractive choice. CAKE also allows the use of 8 different priority tins, which fits well with the nature of 5G traffic, where different flows have different priorities. Finally, on a practical level, the implementation of CAKE in the linux kernel uses the kernel's "skb_flow_dissector" which exposes a hook point for eBPF [**eBPF**] programs. Specifically within a 5G context, this could allow one to attach an eBPF program to allow CAKE to peek inside of GTP tunnels, and differentiate the flows within them. For all the other algorithms mentioned so far, all of the GTP packets would appear to belong to the same flow, and thus none of these algorithms would work at all if the traffic is arriving in a GTP tunnel. This is ultimately the strongest case that can be made for any of the options listed so far. TC does have one other algorithm which hooks into the "skb_flow_dissector", namely the Choose and Keep Scheduler (CHOKe) [**choke**], which does perform active queue management, however does not provide fair queuing. This means that while it manages buffers in order to prevent

them from filling up, it does not guarantee different flows a fair share of the bandwidth, which can still lead to flows experiencing increased delay due to other greedy flows.

3.3.2 Jitter Reduction and Latency Guarantees

In the Deterministic Networking Architecture specification it is recommended to adopt time synchronization as well as sending "Time of Execution" fields in the application packets, in order to achieve jitter reduction. For time synchronization, the PTP protocol may serve well, especially since there is an existing implementation- the linuxptp project - which extends it to work over IP networks. As for the time of execution fields, it may be possible to place these in a GTP header, or to combine them with timestamping.

Latency guarantees are not mentioned within the specification but they are important for the 5G campus setting. Specifically when backhaul options may include satellite links it becomes important to consider latencies. Additionally, in a multipath setting, latency can be a useful criteria for selecting between links, and guaranteeing latency becomes easier to do when it is possible to switch links should one of them start to experience greater latency than before.

3.3.3 Service Protection

In order to protect against equipment failure, it may be recommendable to perform packet replication and/or encoding. To address this the DetNet specification speaks of both duplication of flows, and network coding [**network-coding**]. This can be a clever solution, however Network Coding requires control over and co-ordination between all the nodes in the network (like other parts of the Deterministic Networking specifications). Though to guard against equipment failure and/or packet loss, duplication does provide an option. So too, does forward error correction (FEC). Many FEC schemes work on a continuous stream of bytes, providing correction for bit errors, as opposed to correcting the loss of entire packets. There

are schemes which address this, fountain codes - such as Raptor [**raptor**], which encode the data as being made up of multiple discrete symbols, and these symbols can be mapped directly to packets, thus allowing the Raptor FEC algorithm to protect against packet loss as opposed to just bit erros. Unfortunately the open source ecosystem surrounding fountain codes is not as developed as that around other FEC schemes, and, because of the complexity involved in implementing them efficiently, there are relatively few freely available projects or libraries which can be used to encode packets using such a scheme. For reference, [1] mainaints a list of C/C++ FEC libraries and none of them support the Raptor family of fountain codes (all other fountain codes incur high overhead). Even if it were more feasible to integrate FEC into the backhaul solution it bears questioning how much benefit it brings. FEC is very powerful in situations with consistently lossy links, however paths over the internet tend to experience loss in bursts, as opposed to at a consistent rate. To this extent it may make more sense to duplicate those flows which require a high degree of reliability.

If one does choose to duplicate packets in order to achieve service protection, that means an elimination function is also required, as well as a packet ordering function. This is because multiple packets may arrive on one link before they arrive on the backup path. If a packet got lost on the faster path, it may still show up on the slower one and thus the packet forwarding needs to pause until the missing packet arrives. The DetNet specifications provide both a basic and an advanced Packet Ordering Function (POF) algorithm. This algorithm can be very easily adjusted to also perform the elimination of duplicate packets. The DetNet specification also clarifies how long one should wait to see if the missing packet arrives (or doesn't) - this value "cannot be smaller than the delay difference of the paths used by the flow" and is called the $POFMaxDelay$.

[1] https://aff3ct.github.io/fec_libraries.html

```
   /* initialization                                                    */
1  foreach flow f do POFLastSent_f ← 0;
   /* Start POF logic                                                  */
2  while true do
3      receive packet p, with sequence number seq_num for flow f;
4      if seq_num <= POFLastSent_f + 1 then
           /* eliminate duplicates                                     */
5          if seq_num == POFLastSent_f + 1 then
6              forward p;
7              POFLastSent_f = seq_num;
8          end
9      else
           /* in a separate thread                                     */
10         buffer p until seq_num = POFLastSent_f + 1 or POFMaxDelay_f elapses;
11         forward p;
12         POFLastSent_f = seq_num;
13     end
14 end
```

Algorithm 1: Basic POF Algorithm Adjusted for De-Duplication

The algorithm shown in 1 is the Basic POF algorithm, with a single adjustment made on line 5 to make sure that an incoming packet's sequence number is not lower than the next one we are expecting, in order to properly eliminate duplicates. The algorithm's **if/else** logic in this representation could be unified in a cleaner way, but it was done this way specifically because it thus requires just one extra line (line 5) to be changed from the DetNet specifications POF algorithm, and that one change suffices to make it a Packet Ordering and Elimination Function.

3.3.4 Multipath Considerations

While the previous subsections have all considered requirements for determinism in an IP network, the issue of multipathing still remains. For a component which is backhauling over multiple links in an IP networks this means link and/or path selection is required. Traditionally, routers select links primarily based on their ability to route the packet to the destination, and secondarily based on various metrics, which can be defined by the administrator. For the WAN Connector's scenario routing considerations are not made- it's purpose is to backhaul the traffic from the RAN / Edge to the Core, where the network's own internet gateway can perform the routing.

At this point it is crucial to differentiate, again, between multihoming and multipathing. Multihomed hosts are able to select one of many links for their outgoing traffic's destination. In multipathing the same link may lead to multiple paths to *different* destinations. When two multihomed hosts communicate with eachother multiple different paths are opened up. Multipathing over the internet requires complex data collection about all the routers in between the source and the destination, see [**multipath**] for an example of such an implementation. This is because multipathing over the internet needs to choose paths with maximally edge-disjoint paths to achieve the best reliability, as well as maximally vertex-disjoint paths in order to reduce the load across all nodes on the path.

The goal of this thesis is of course to use the link and/or path selection to provide determinism for the flows being backhauled. To this extent then, the link selection algorithm must attempt to forward flows over paths which can provide at worst the maximum allowed latency and jitter, as well as meeting the minimum reliability. Here it is worth noting that while duplication cannot really be used to guarantee latency, it can be used to improve reliability since a flow which is duplicated across two paths is far less likely to experience packet loss.

3.3.5 Summary of Necessary Features

Looking back on the previous subsections, the following points (in no particular order of priority) were identified as mandatory for a deterministic backhaul solution: 1) traffic shaping, 2) path selection according to jitter and latency requirements, 3) service protection (i.e. via packet duplication), 4) packet ordering and de-duplication, and 5) time synchronization. The ways in which these can be addressed, or quite simply the way in which they were implemented, will be discussed in the following section

3.4 Overview of the WAN Connector's Features and Components

At this point, one can take the skeleton structure proposed in the first section, and superimpose the other requirements which were determined in the second section. Combining these ideas yielded the final implementation, which will be now be presented.

3.4.1 Path Selection Algorithm

Meeting the various requirements - jitter, latency, and delay - of a flow can be formulated as a multi-constrained QoS problem. Solving such a multi-constrained QoS problem via path selection is a binary optimization problem, and the key to deterministically meeting the flows requirements. The problem can be posed as: "select those paths on which to forward packets while making sure to satisfy the latency, jitter and reliability requirements of the given flow, and minimizing the overall weight of the paths used". The mathematical definition is as follows:

$$\text{Minimize } \sum_{i=1}^{P} w(x_i) \tag{3.1}$$

$$\text{Where, } d(i) * x_i \leq D \tag{3.2}$$

$$j(i) * x_i \leq J \tag{3.3}$$

$$1 - \prod_{i=1}^{P} (1 - r(i) * x_i) \geq R \tag{3.4}$$

$$\text{for } x_i \in \{0, 1\} \tag{3.5}$$

Here the variables D, J, and R are the flow's delay, jitter, and reliability requirements, while the functions $d(i)$, $j(i)$, $r(i)$, $w(i)$ are the estimated delay, jitter, reliability, and weight of link i. The predicted values will usually just be the latest measurement, as recommended in [2], however there is room here to use more advanced metrics to predict the future link quality and thus perform preemptive path switching in future work. The total number of paths is P. The x_i variable indicates whether or not link i shall be used. If a solution is found, then the flow's packets will be forwarded on each link i where $x_i = 1$, and if no solution can be found which satisfies these conditions then the flow is rejected because its QoS cannot be guaranteed.

It is worth noting that solving such problems is NP-Hard [**tsp-np-hard**]. However this hardness arises primarily because of equation 3.4, the equation for reliability (also the only non-linear equation). Due to this equation one must consider every possible combination of paths on which to forward, and the complexity is $O(2^n)$. This means, even though akella et. al [1] have shown that multihomed approaches experience diminishing returns after more than 4 links, attempting to brute force the solution by limiting the number of outgoing interfaces to 4 still yields a very large problem space - in the worst case both WAN connectors could have 4 outgoing paths, leading to 16 possible paths and thus 65536 possible combinations to consider.

In order to further avoid the combinatorial explosion the problem needs to be parameterized even more. The first logical parameterization has already taken place by limiting the number of interfaces to 4. This can be expanded on by limiting duplication to only take place over disjoint interfaces. The reasoning behind this is that in a geographically distributed Campus 5G environment, and especially for environments featuring wireless backhaul (e.g. satellite), it is more likely that a link's reliability is most affected by the over the air transmission on the first hop. By performing this parameterization the problem space shrinks considerably, to just 2^4 possible combinations of outgoing paths, but, because certain paths going out on the same interface are ignored, the optimality of the overall solution is gone. This is an acceptable trade off for quick computation, and reasonably strong guarantees on reliability. When choosing which path to include from a set of paths using the same outgoing link, only the path with the greatest reliability is taken into consideration so that the likelihood of rejecting a potentially viable flow is as low as possible.

3.4.2 Packet Ordering and Elimination Function

Since it is possible for flows to be duplicated across multiple paths, it becomes a necessity to have a packet ordering and elimination function. To be able to re-order and de-duplicate packets means that their sequence numbers need to be tracked. This thesis' implementation will track the sequence number on a per flow basis, using the GTP "sequence number" field. This thesis uses it's variation of the DetNet POF specification's Basic POF algorithm [**detnet-pof**], as presented in 1.

3.4.3 Internal Architecture

Control - Data Plane Communication

A protocol is desired which can provide reliable communication over multiple paths in order to communicate between the data plane and the control plane which is where the statistics are collected, flows are added or removed, and the multipath decisions are made. This is crucial, since in a geographically distributed campus 5G deployment it is unlikely that there will be a separate management or control network which uses different underlying network infrastructure than the data plane. Therefore link failure, as well as packet loss, on the link being used by the control plane must either be avoided or protected against. To this extent there are only two feasible options - either multipath TCP or SCTP. SCTP does support multihoming and intelligent failover, despite not being explicitly designed for multipathing, and is slightly easier to manage and configure than multipath TCP, which is why it was chosen for the data plane control plane interactions.

GTP Tunneling and Custom Header Extension

The tunneling protocol used between the two data plane instances of the WAN Connector will be GTP version 1 [**GTP-spec**]. The specification for GTP allows for the use of sequence numbers, as well as the use of extension headers. For the data plane communication there will always be an additional custom header sent which includes a timestamp taken by the sender. The timestamp is taken in microseconds. For the timestamp there are several options available within linux. The TAI clock in linux (representing the Atomic International Time) is used in this implementation because it does not have leap seconds, and so is a monotonic function, which is an important guarantee for time sensitive applications. An example of a packet capture of the WAN Connector using the custom header is included in the appendix C.

Normally, in 64 bit linux systems, a timestamp takes up 16 bytes and consists of 8 bytes for the time in seconds since the epoch, and another 8 bytes for the nanoseconds. To reduce the footprint of the timestamp in the header, the seconds are represented with just 8 bytes, thus wrapping around the interval $[0, 255]$. The receiver needs to take this into account, but is programmed to do so, and since the path delays are not realistically expected to ever exceed 3 seconds this is more than enough. The microseconds have a maximum value of 10^6 and can be represented with just 24 bits. For Wide Area Networks and internet connections usually the latencies are on the order of milliseconds and as such microsecond precision is deemed to be sufficient.

Metric Collection

To be able to intelligently switch flows between paths, and to be able to know when this is required, it is necessary to collect the relevant metrics about latency, jitter, bandwidth usage, as well as packet loss. Packet loss can be detected via the sequence numbers, while delay and jitter can be calculated using the timestamps passed along in the GTP headers. These metrics are periodically reported to the control plane so that it can make its decisions on up to date data. Keeping a healthy overview over the state of each path requires periodic probing on these paths. This is especially important for detecting when paths become viable again, since these paths will not have any traffic on them while they are considered down, or if they have experienced high latency and/or jitter recently. The probes are sent once per period of reporting so that they have a minimal impact on the bandwidth usage.

Flow Descriptions, Flows, and Hashing

Incoming packets need to be quickly mapped to their respective flows. It is common in network environments to perform flow hashing, in order to quickly lookup which flow incoming packets belong to. This approach makes sense here too. However, since 5G flow descriptions

can apply to various IP and port ranges as well as matching based on the transport layer protocol it is not possible to hash an incoming flow and obtain the same hash as the flow description. Furthermore it is possible for a flow to match multiple different flow descriptions. In these cases the first matching flow description is taken. This implies some sort of ordered storage of flow descriptions will be required, as well as a method to quickly match packets to flows. The solution used here is to maintain a simple linked list of the known flow descriptions, and match packets to their flow descriptions, if the packet is unknown. For "known" packets, which have been matched to a flow description, these are hashed and stored in a table, alongside their flow descriptor and a list of paths on which the flow is supposed to be forwarded. This allows quick lookup for every subsequent packet, as well as making it simple to change which paths flows are meant to be forwarded on. One alternative to storing the flow descriptions in a linked list would be to store them in a tree, such as in [**flow-lookup-trees-phd**], sorted either by priority or ID or even by the hash value of the flow description. This method would provide a faster lookup of new, "unknown" packets, whose hashes don't yet match to a flow, but since new flows are not such a frequent event the additional complexity of implementing such a data structure may not be worth the gain in performance.

It is important to make a good choice when selecting a hash function. Since the hashes used are only for the purpose of lookups it would not be recommendable to pick a hashing algorithm designed for use in cryptography. These algorithms are designed so that it is very difficult to reverse engineer the original value from the hash, and this may often make the computation of the hash more computationally intensive than when computing hash algorithms designed for looking up table entries. Lastly, for hashing algorithms it is desirable that they provide a healthy distribution, to reduce collisions. Collision reduction can also be affected by choice of hash table size- and in general it is usually recommended to use a prime number. In this implementation, the MurmurHash algorithm was chosen due to its strong performance for lookup-based hashing. Specifically, the MurmurHash3 [**murmurhash3**] version was chosen, and since it can generate 32 or 128 bit values - the exact sizes of IPv4 and IPv6 addresses -

this simplifies the hashing implementation for IP flows. When a packet comes into the WAN connector it is hashed based on it's destination address, source address, destination port, source port, and finally the transport layer protocol number. If the packet was tunneled with a GTP header, the header is removed and the hash is performed on the tunneled packet, not on the GTP packet. If the hash does not match, the packets is compared to the list of existing flow descriptions. If the flow belongs to one of these descriptions then that decision is stored in the hash table.

3.4.4 Overview

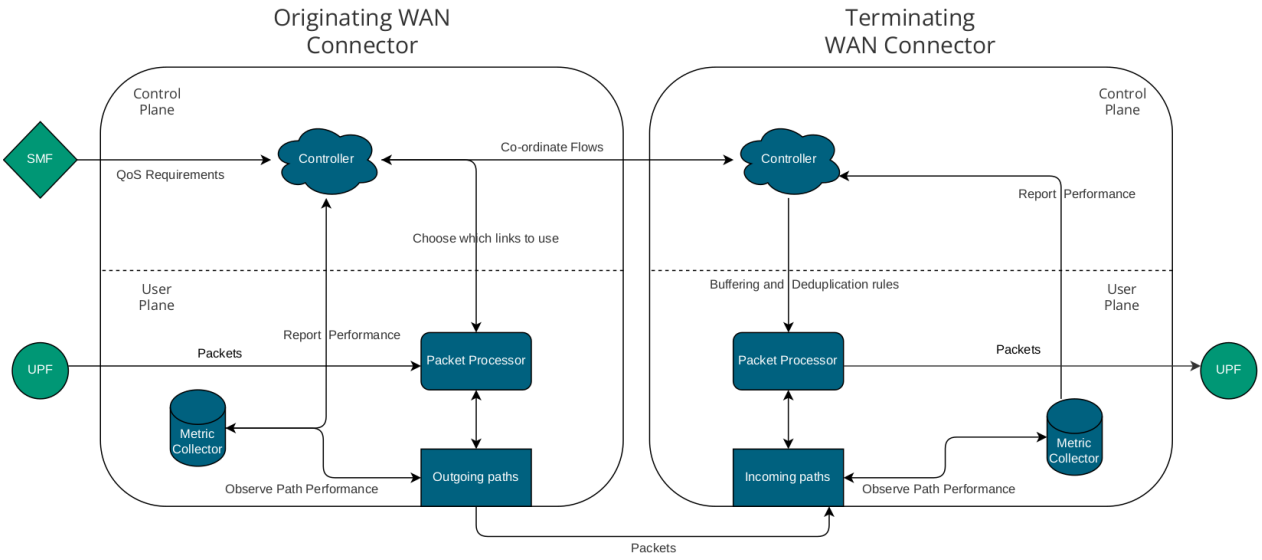


Figure 3.1: Internal Architecture of the WAN Connector

The internal architecture of the WAN connector is shown in Figure 3.1. The connector is split into a control plane, which uses the algorithm in 3.1 to choose which interfaces to forward on, and a data plane, which performs the forwarding, as well as any necessary buffering, re-ordering and/or de-duplication. The WAN connector also performs different tasks depending on whether it is the origin or termination of a flow. For example, the terminating node may be receiving duplicate packets on the other paths, and it must know to drop these. Additionally, the sender, can only store statistics about how many bytes it has sent, and on what paths.

The receiver is able to use the information contained in the packets it has received to construct a complete picture about the nature of the path - its jitter, latency and packet loss. The link selector in the control plane can then use this information to make its decisions.

One part which is missing from this diagram is the traffic shaper. That is because it is not connected to the control and the data plane, rather it is meant to be installed alongside them. Furthermore, it is important that the traffic shaper resides in the part of the network before the packets enter the WAN connector. If the deployment features a UPF in the edge network, it may make sense to place the traffic shaper before this as well. For the core, the traffic shaper will similarly need to be placed at the ingress point at which packets reach the WAN connector.

Chapter 4

Evaluation

This chapter reviews how the approach from the previous chapter was tested, and discusses the observed results. This chapter will explain the setup of the testbed, as well as the motivation behind the types of tests which were performed. Finally a discussion of the results and their origins will take place.

4.1 Approach to Evaluating Performance

In order to evaluate the success of the proposed approach a multipath scenario with four links will be emulated and investigated. Each test will consist of two WAN connectors, and the four emulated links going between them. The characteristics of the links will be changed during the tests, each time with respect to different properties: latency, packet loss, and jitter. The evaluations will be performed on a testbed consisted of four seperate host servers. Two of these servers will act as the originating and terminating WAN Connectors. One server will generate and measure traffic, in between the two WAn Connectors, a server will act as the link emulator. This architecture is shown in figure 4.1

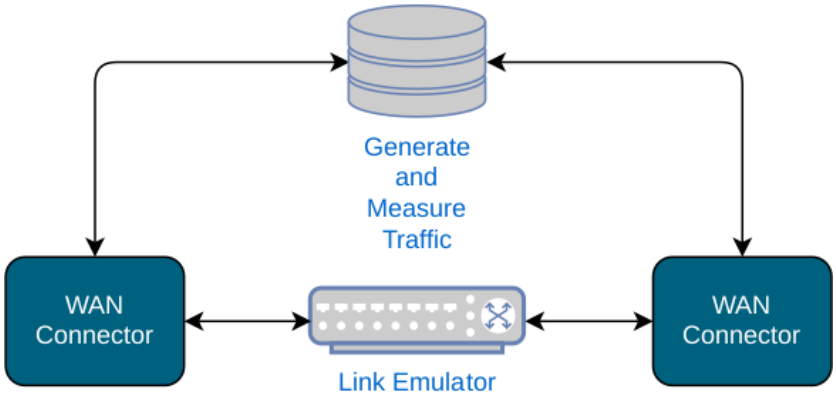


Figure 4.1: Testbed Setup

In figure 4.1, the links between the servers are 10 GB ethernet cables. The hosts are connected as is shown. In the "traffic generator and measurer" two linux namespaces are used to separate the traffic generation and the measurement. The packets generated in one namespace are routed via the ethernet connection to the next host (the terminating or "edge" WAN Connector), because of the linux namespace they are not aware that their destination is technically the same host. The packets are then are backhauled by the WAN Connector over the emulated links in the next host, and via the terminating WAN Connector, back to the original host, but now in the "measurer" namespace.

The link emulation is done using the tc subsystem. First a root Heirarchical Token Bucket qdisc is established on the "downlink" and "uplink" interfaces (the ethernet cables between the WAN Connectors). In this case, the link emulation is performed before the outoing packet is queued on the ethernet cable, so that is also where the qdisc is placed. Secondly, the qdisc is given children classes, one class per link. This allows one to establish bandwidth caps for each emulated interface. Within these HTB classes the root qdisc is a netem qdisc. These allow the installation of link characteristics such as latency, packet loss, and jitter.

In order to make the emulation more realistic, the netem qdiscs will periodically be adjusted, this allows one to degrade or improve links over time, for example by increasing the latency or packet loss in frequent intervals, up to a large value. It also more closely mimics the real-life behavior of WAN connections, which do experience changes in their characteristics

over time.

For the purposes of evaluating the WAN connector, three series of experiments will be performed to isolate and investigate it's link switching capabilities. First, purely latency based link selection will be investigated- a flow will be defined with specific latency requirements and then the emulated links will have their latency repeatedly changed. The same will be done once with packet loss and with jitter. In each of these scenarios the other two parameters will be held constant, so that the WAN Connector can be judged based on its ability to meet one of the latency, jitter, or packet loss requirements, without the other two requirements interfering. In order to simulate realistic scenarios these experiments will also be repeated once with background traffic which aims to saturate the link's capabilities.

The data for the emulation used in all of these scenarios was obtained using the Satellite Constellation Network Emulator (SCNE) [1] from the European Space Agency. The default scenario is 24 hours long and consists of 288 satellites, 10 gateways (GW), and 50 user terminals (UT). The reason this data is used is because LEO satellite links are expected to be the most common type of backhaul link for geographically distributed 5G Campus Network deployments. Furthermore, since LEO satellite connections are exceptionally more volatile than "regular" links [**leo-links-characteristics1**] [**leo-links-characteristics2**]. By choosing to emulate more volatile links in the testbed, the WAN Connector experiences a more difficult test, and one can be assured that the real life performance in "normal" scenarios, is highly unlikely to be worse.

Since the WAN Connector is defined to work on at most four outgoing links, each investigated scenario will feature four links. These links will all be based on data from the SCNE emulation.

[1] https://connectivity.esa.int/projects/scne

4.1.1 Accuracy of the Emulation

To make sure that the testbed emulation also matches the values from the SCNE, a brief comparison was done, where the testbed's latencies were compared with those recorded in the SCNE.

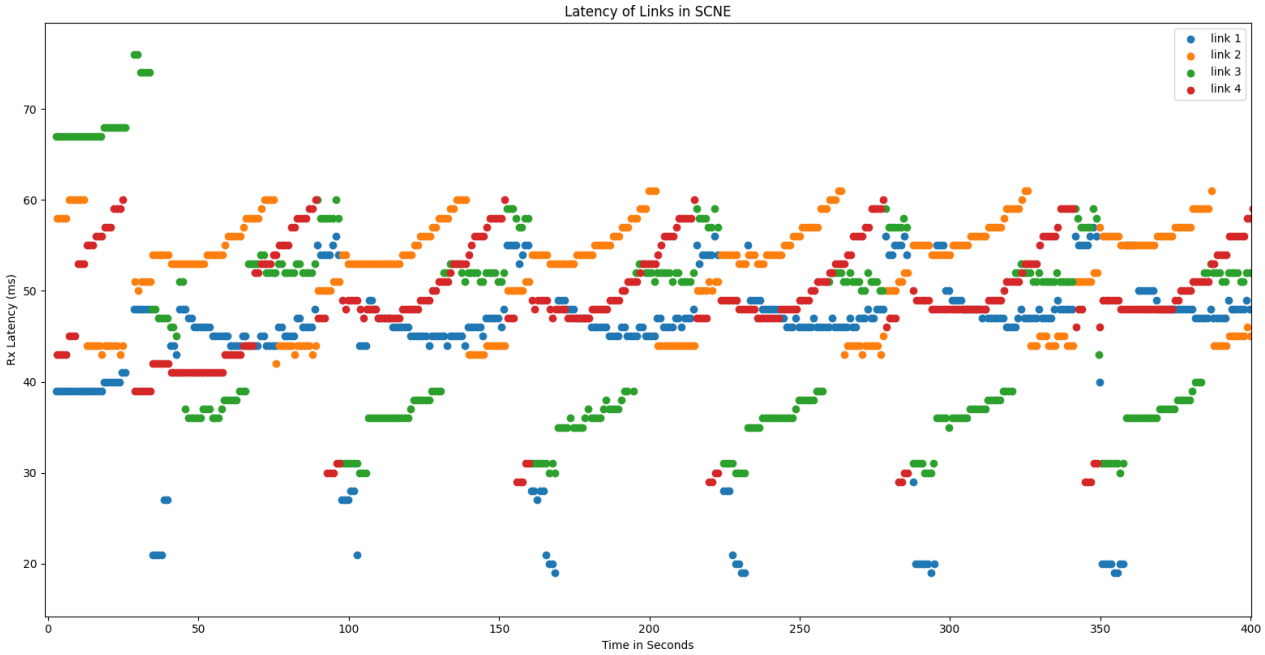
Figure 4.2 shows the comparison between the data observed in the testbed, and the SCNE. The testbed data in the graphic is measured by the packet flows running over the individual links. The emulation data comes from the values recorded during withh the SCNE. Since the testbed data is collected at a rate of 100 packets per second, but runs at 10 times the speed of the SCNE emulation, there are 10 times as many data points. This explains why the testbed data appears blurrier.

As can be seen in the figure **??**, the latencies recorded on the testbed closely match those from the SCNE, which it was seeking to replicate.

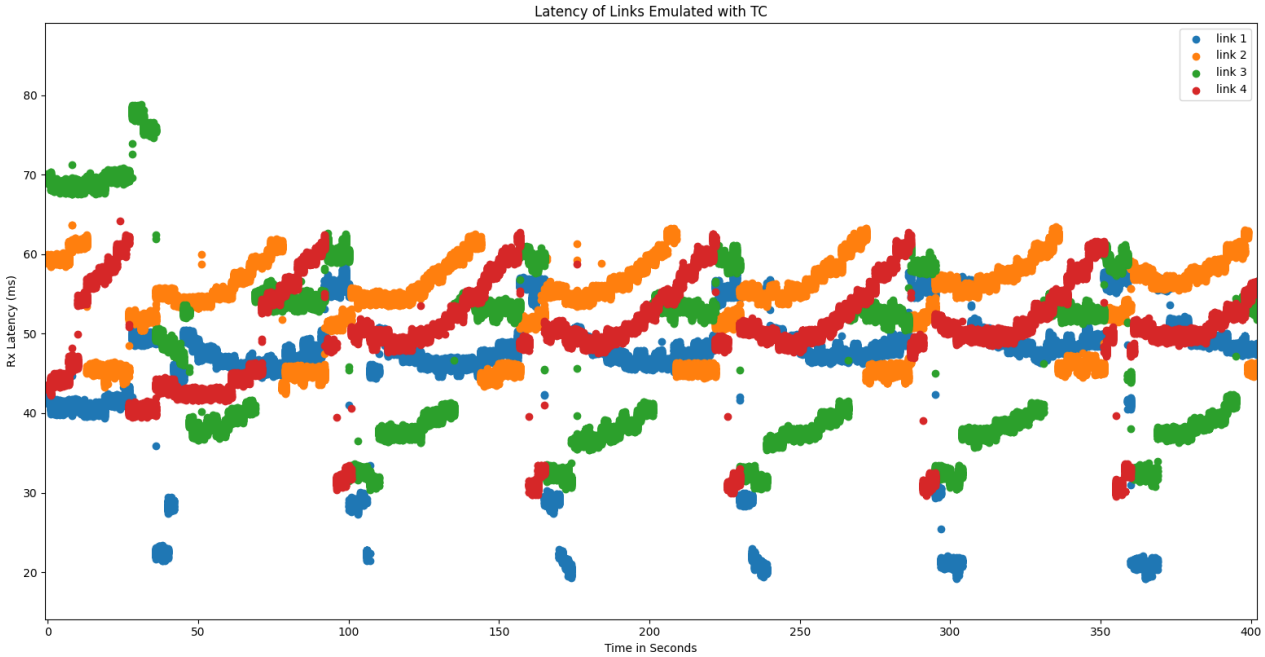
4.2 Latency Based Path Switching

First is the scenario investigating latency. During the emulation, the simulation data is used to adjust the tc netem qdisc on the "link emulator" machine periodically. Because the SCNE simulator provides latency data over a 24 hour period, in 10 second intervals, the simulation data is "sped up". That is the emulation is changed each second, based on the next datapoint from the simulation. This means the 24 hour window can be run over a 2 hours test. Since real links are usually steadier and less volatile than the sped up emulation one can expect results in a real situation to be better.

In the experiment 100 packets are sent per second across each link, as well as 100 packets per second from the traffic generator, through the WAN Connector. This is done to have a comparison between the performance of the WAN Connector and the baseline performance



(a) Latencies Recorded in SCNE



(b) Emulated Latencies Measured in the Testbed

Figure 4.2: Comparison of the Emulators Latencies and the Testbed

level, which would be the best performing link. In any scenario the minimum acceptable performance of a path switching application would be to provide a better performance than the best single link. Otherwise it would be better to just select the best link, and not use the application at all.

4.2.1 Performance Compared to Single Links

For the experiment a flow is defined in the WAN Connector with a minimum round trip time (RTT) of 56 milliseconds. This value is chosen because it is the mean of the mean values of the latencies of the four links from the emulation. For the evaluation of the latency based path switching a single flow matching the definition stored in the WAN Connector is backhauled through the WAN Connector at a rate of 100 packets per second. In order to serve as a comparison, identical 100 packet per second flows are sent across the four other links. Finally their latencies are compared, and the latencies measured by the four links are used to calculate the Oracle approach.

The graphics 4.3 and 4.4 show a cumulative distribution of the different latencies experienced by the packet flows running over the individual links and the WAN Connector, as well as the performance of the Oracle approach. The Oracle results are purely theoretical. For each packet, it is able to select the best link to forward on, based on that links future characteristics. This approach is calculated by comparing each packet sent by the WAN Connector with the latency that packet *would* experience on any of the other links, and if the latency of the WAN Connector's link would be greater than the minimum value (56 ms), selecting a different link (if one exits) with a lower latency. Since it always picks an optimal path, the Oracle approach provides an upper bound on the best possible performance. In practice it is not possible because it has knowledge of the future, but nonetheless it acts as a very valuable comparison.

The CDF shown in figure 4.3 and zoomed in on in figure 4.4 tracks the probability that the

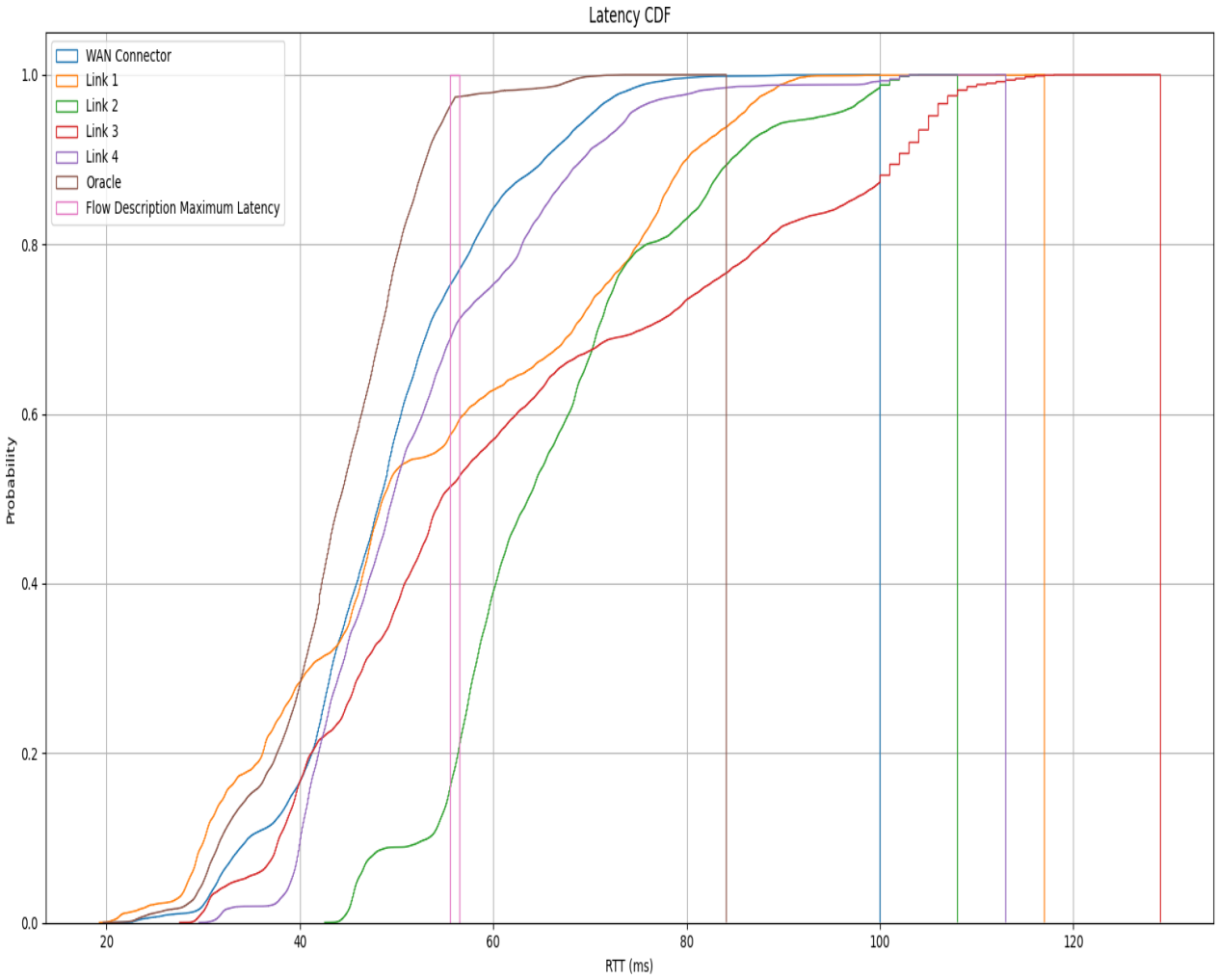


Figure 4.3: Cumulative Distribution of the Latencies

flow experiences a latency less than or equal to the latency value on the y axis. The pink bar labelled "Flow Description Maximum Latency" is set at 56 milliseconds and represents latency bound for the flow being measured, which was communicated to the WAN Connector.

The results in the figures show that the WAN Connector is able to achieve a superior performance to any of the individual links on their own. This was the baseline requirement and it is important that it is able to achieve this. However the performance does not significantly improve on the next best link, which is is able to achieve the maximum allowed latency 72% of the time, while the WAN Connector does so 77.5% of the time. The Oracle

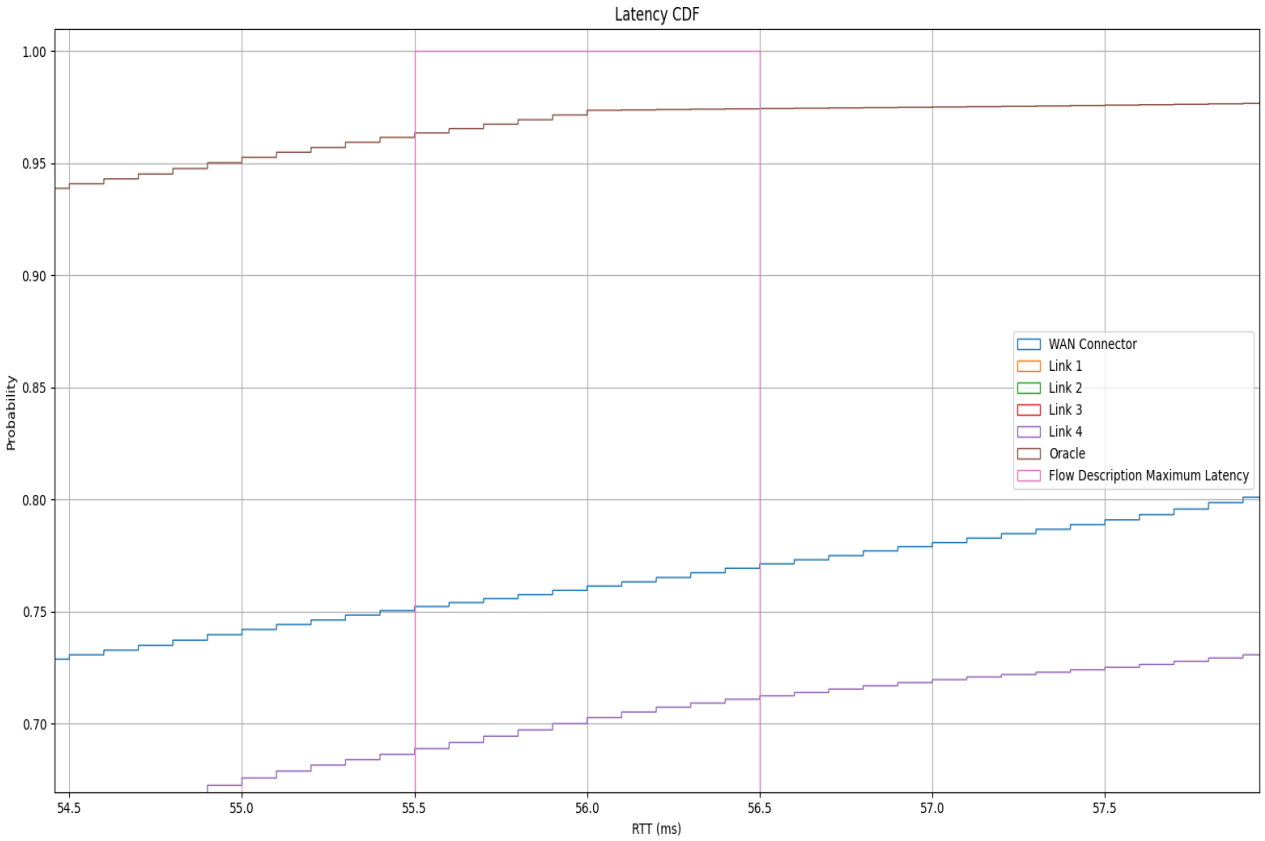


Figure 4.4: Closeup of the Latency CDF

approach shows that in theory one could, at best, have maintained the minimum latency required by the flow in 95% of the cases. This means the WAN Connector's performance is only 80% optimal.

4.2.2 Latency Performance Under Load and the Importance of Traffic Shaping

During this experiment, the same 100 packet per second flow from before is repeated, but background traffic is added. An iperf3 test is run across the WAN Connector at the same time as the latency critical flow, and the background traffic is given the same latency requirements, so that the WAN Connector always schedules both flows to the same link. This ensures that

whatever link is being used will be saturated. Latency performance during congestion and close to congested scenarios is a crucial element of a deterministic backhaul solution. The WAN Connector's control plane only makes decision about which link to forward on- it does not perform load balancing or prioritization. The data plane does not do this either, it implements a purely First In First Out (FIFO) approach to packet queuing. This is done because the burden of implementing an appropriate packet shaper, in addition to the other multipath calculations, is too significant for the purposes of this thesis, and because very powerful traffic shaping implementations, which are easy to integrate into this solution, already exist.

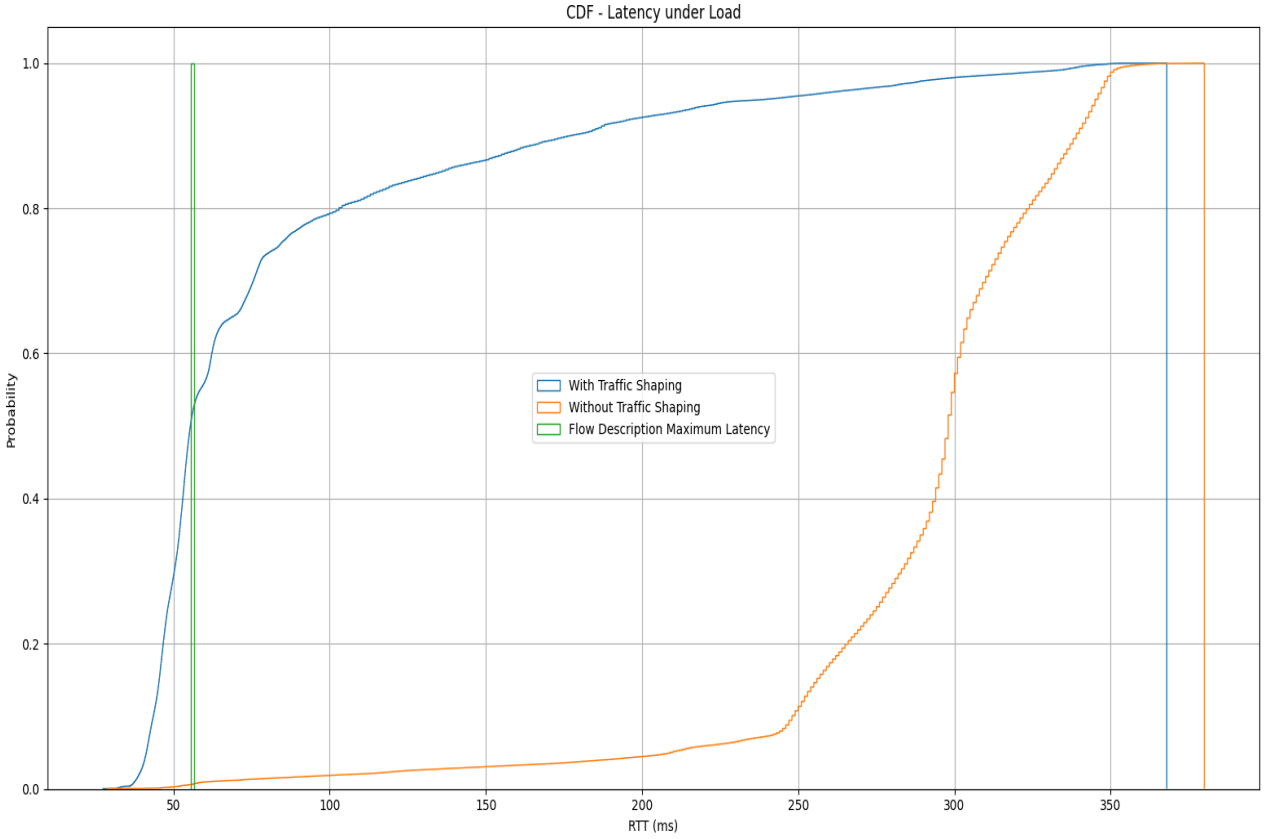


Figure 4.5: Latency with Background Traffic

The figure 4.5 neatly demonstrates why traffic shaping is an essential element of any deterministic network. In the case in which there is no traffic shaper the latency critical flow experiences significant additional latency due to the network being overloaded. The TCP algorithm fills

the link's capacity, and the latency critical packets spend too much time buffering and arrive late. It should be noted that the WAN Connector's performance with the traffic shaper is worse than in the previous scenario, where there was no background traffic, and only 50% of the packets are under the required latency of the critical flow. However the approach without the traffic shaping is at just 1%.

4.3 Reliability Based Path Switching

In the next scenario, to analyze the ability of the WAN Connector to achieve the required reliability of a given flow, the same test flows from before are run over the WAN Connector and the four outgoing links. However this time instead of varying the latency, the packet loss ratio of the link is changed. The SCNE emulation's data is used for the testbed once again. However since the emulation does not provide packet loss on a per second basis, the latencies of all the different links in the simulation are used. For the emulation, these 51 values are looped over in 10 second increments. Each link is given a shuffled set of these 51 values. This means that the cumulative value of the packet loss experienced by all of the links is the same (2%), but the instantaneous values will differ. To appropriately analyze this scenario the flow being backhauled over the WAN Connector is given a reliability threshold of 0.1% this is because in this scenario, where the existing links all provide an average of 2% reliability, it becomes necessary to duplicate at some points, in order to achieve the required reliability. Afterward, the experiment is repeated under load, like in the latency scenario, where the test flows across the links themselves are not run, and a second iperf3 flow is run across the WAN Connector in order to saturate the link. Like before, this second flow is given the exact same requirements as the test flow, in order to ensure that they are always running across the same link.

The results of this experiment are shown in figure 4.6. The packet loss recorded by each of the flows over the period of the experiment is plotted in a bar graph, as well as the theoretical

packet loss which would have been experienced by the Oracle approach.

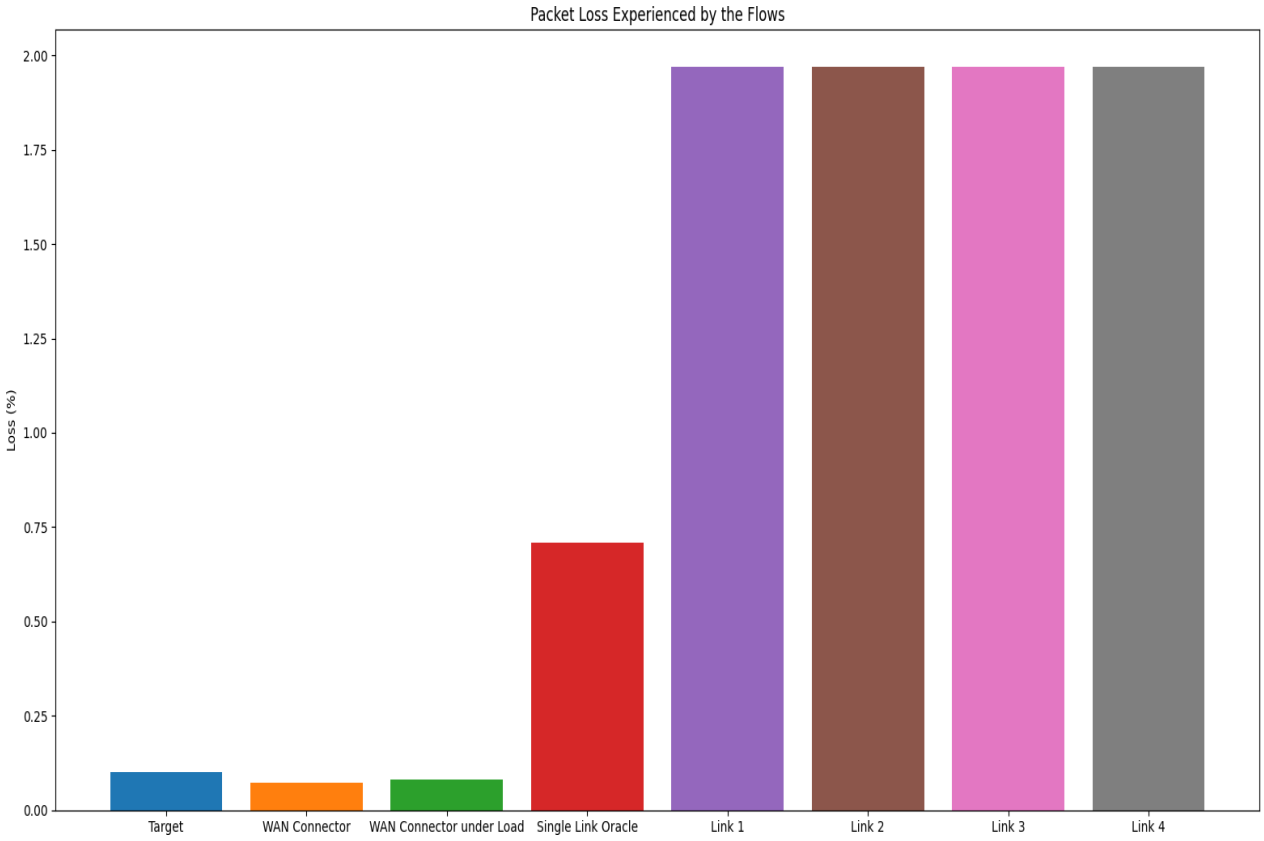


Figure 4.6: Reliability

For the analysis of the performance this time the Oracle approach is defined as choosing, before forwarding any packet, the link with the lowest packet loss ratio. But, crucially, the Oracle is not allowed to duplicate flows across different links. This explains why in figure 4.6 the Oracle is outperformed by the WAN Connector. Indeed, as the graphic shows, it is not possible to achieve the critical flow's required reliability with any of the available links, nor with the Oracle approach. But the WAN Connector is able to achieve it. This speaks to the potency of the flow duplication approach, which is made possible in the optimization equation used to decided on which flows to forward.

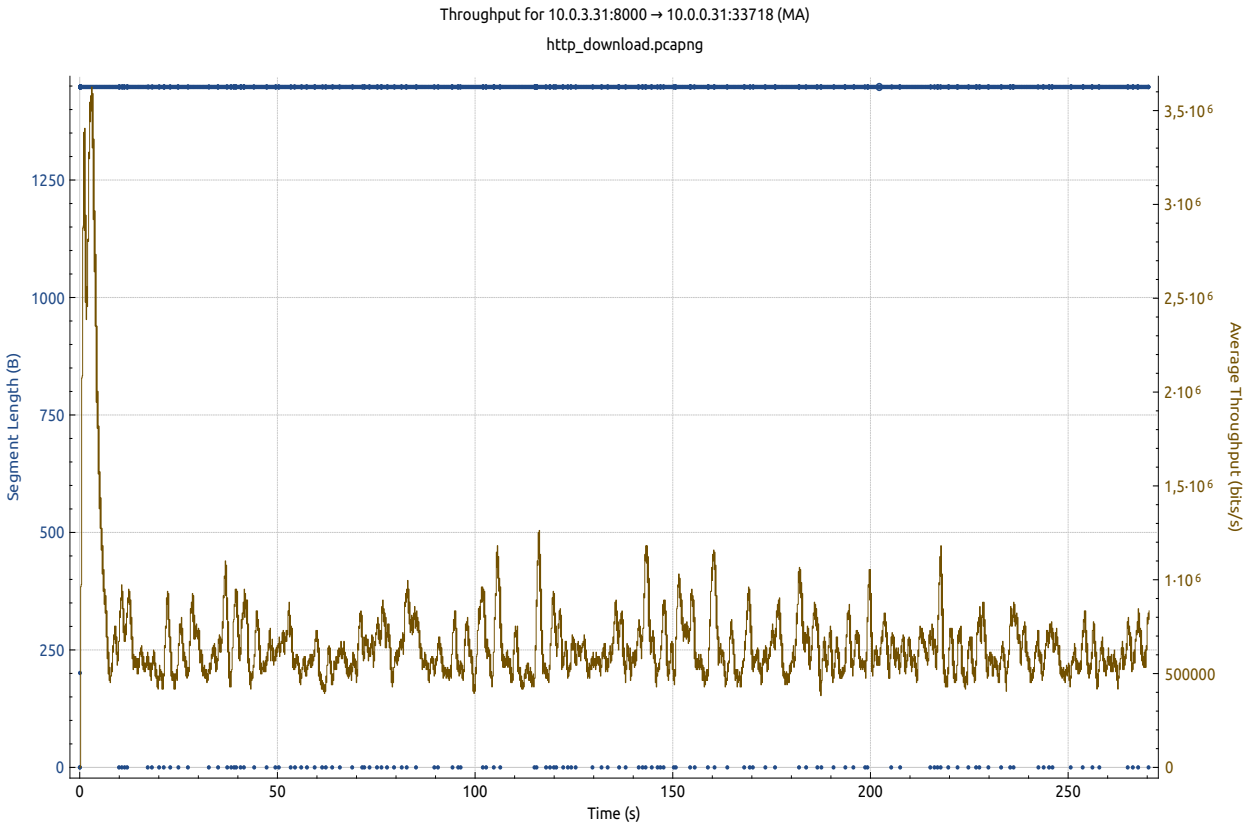


Figure 4.7: Throughput of a Duplicated Flow

4.3.1 Impact on Throughput

One of the unfortunate impacts of packet duplication is that it reduces the available bandwidth. It represents a waste of up to 50% of the available resources, in the worst case, where every flow must be duplicated. Additionally, since flows which are being duplicated need to be re-ordered, this means that the FIFO approach for packets cannot be used and instead the packets must pass through a Packet Ordering and Elimination Function (POEF). When a packet arrives out of order the POEF starts to incur significant additional overhead because it must store all packets until the missing packet arrives, or a point is reached at which the stored packets must be dequeued. This point is either the point at which the packets expire, or the queue of out of order packets becomes full. Beyond this, the storing and eventual mass de-queuing of out of order packets can also interfere with the TCP algorithm- thus leading

to reduced throughput.

These two effects - the POEF overhead and it's effects on TCP- lead to a very heavily reduced throughput, as can be seen in figure **??**. The graph in the figure shows the throughput experienced by a routine TCP download, running over the WAN Connector, and experiencing replication, as well as the segment length of the TCP stream. As can be seen in the graph, the throughput is very low, and is not able to saturate the link's bandwidth despite being the only flow running at the time.

4.4 Jitter Based Path Switching

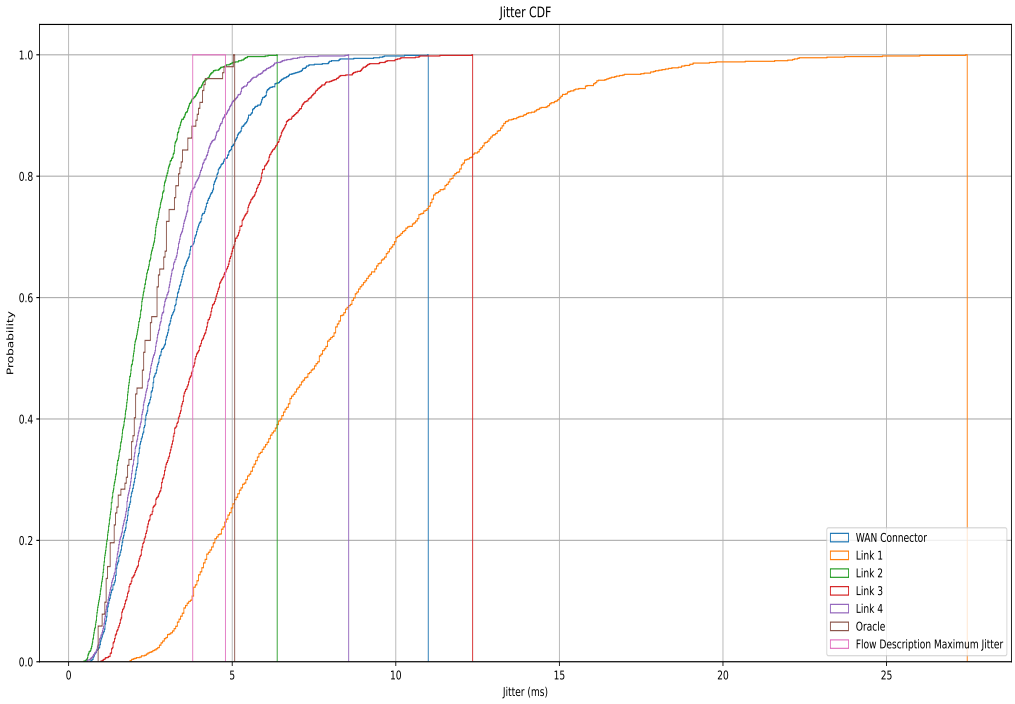


Figure 4.8: Performance of Jitter Based Path Switching

For the final analysis, the jitter must be investigated. For this purpose the simulation data

was once again adapted. This time the standard deviation of the latency experienced was calculated for each of the 102 connections in the simulation. Then these standard deviations were scaled down by diving them by one, two, three, and four, respectively, in order to acquire four different set of jitter characteristics. Finally the scaled values were shuffled. This resulted in 4 links with varying jitter characteristics, which suffices for the emulation. Each different jitter value is emulated for 10 seconds on the respective link. During the emulation scenario a UDP iperf3 stream is ran at a rate of 3/4ths of the link's capability for 1000 seconds. iperf3 records jitter each second, and this data is used to then analyse the performance. Since more than half of the link capacity is being used by the flows being sent on the links (and not through the WAN Connector), the WAN Connector cannot be tested at the same time. Thus, to test the WAN Connector the emulation is repeated but with the flow only being sent over the WAN Connector. Lastly, the Oracle approach is calculated based on the simulation data. The Oracle approach, like before, displays the theoretical optimal performance. It chooses, at each point in time, the link with the lowest jitter. The cumulative distribution of the jitter experienced by the flows is plotted in figure 4.8.

As can be seen from the figure, the WAN Connector gave a very disappointing performance. It displays an inferior performance to two of the links. This means the path switching is actually causing a worse performance. This is most likely down to insufficient statistics causing poor path selection, because jitter statistics can't be gathered about other paths in this single flow scenario. The probe packets which are periodically sent out do not provide enough data points to accurately determine jitter. While they can give a sufficient baseline for latency calculations, as well as helping to determine if a link is dead or alive, their jitter measurements are not likely to provide a strong insight into the true nature of the link's jitter.

A final note about the plot in figure 4.8. In the figure the first link actually performs better than the Oracle for most jitter values, before ultimately proving to have the greater maximum jitter values. This occurs because the Oracle is based on the simulation data and not the

measurements. Therefore it is likely that the better performing link experienced, for the most part, very low jitter, and thus because there are more data points available than for the Oracle (and because no doubt there was some small amount of luck with the random values used by tc's netem qdisc), the percentage of packets below a certain level of jitter is greater. However ultimately the maximum values experienced by the flow, during those times when the link experiences it's worst jitter, are worse than the Oracle approach's worst cases, as one would expect.

4.5 Discussion

In light of these results one can conclude that the approach presented in the previous chapter is not without its flaws, however the results have also highlighted some of the functional aspects that were important for good performance. For latency based path switching it was able to outperform the other available links and achieve 80% optimality, and the inclusion of a traffic shaper proved vital for preventing the latency critical flow from experiencing higher buffering times in the presence of a competing flow. For reliability, the WAN Connector showed that it was able to guarantee a critical flow a lower level of packet loss than would have been possible using any single link. Furthermore, it was even able to outperform the single link Oracle, and in a situation where the single link Oracle is unable to provide the required minimum level of packet loss for the desired flow, the WAN Connector was able to deliver the desired reliability. This is a hugely important result because it shows that the WAN Connector can provide greater reliability than even an optimal single link path selection algorithm, via the packet replication. This can enable critical applications to be run in environments where the backhaul options would usually be too unreliable to enable the critical applications.

This section, however, also demonstrates the downside of the packet replication approach, as it greatly reduces the overall throughput. Beyond this, during the jitter experiments, it is

seen that the WAN Connector provides a poorer performance than simply using a single link for the entire experiment. This is fundamentally a bad result. The baseline by which any path selection algorithm should be judged is its ability to outperform the individual paths, since it can always select a superior option when one exists. Since the WAN Connector fails in this instance it must be concluded that it cannot provide an overall reduction in jitter for any jitter sensitive applications. The author would suggest, however, that the problem in this respect is not due to the design but rather the implementation. Ultimately there was not enough up to date information about the available links' jitter, so that the path selection algorithm could choose the right one.

4.6 Improvements

There are several possible improvements that immediately spring to mind. These could be applied both to the actual implementation as well as to the high level design.

Measuring and Predicting Path Characteristics

Firstly, the path estimation could be adjusted to not just report previous statistics but also attempt to infer what the path might look like in the near future, for example as a path begins to experience increased latency a predictive algorithm/approach might be able to preemptively move flows off of that path, before their latency requirements are violated. This approach could be based on machine learning or AI, or it could use analytical methods.

Furthermore, it would be prudent to use an exponentially weighted averaging function when calculating a path's current characteristics, based on previous measurements. This would strike a nice balance between re-actively adjusting the predictions based on previous measurements and keeping in mind a path's performance across its entire history.

Also, as the authors in [**adaptive**] did, this thesis' approach could benefit from adaptive windows of reporting. Instead of collecting statistics at a constant periodic rate, the rate can be adjusted to that period which performs best. This way there is not additional overhead with overly frequent statistical updates, as well as avoiding the reverse situation, where the reporting is too infrequent for a rapidly changing path.

Lastly, the probe packets which are sent out periodically to measure each path could be adjust to periodically be sent in large bursts, this would serve to provide a more accurate idea of the jitter and the packet loss, since these cannot be measured purely by sending a small number of probe packets. For accurate jitter and reliability measurements more packets are required. This may incur larger overhead on the link due to the bursts of extra packets, however one idea to counteract this would be to adjust the probing so that it is only performed on those links with insufficient traffic currently running through them. The links which are currently backhauling large amounts of traffic will be able to make their own measurements based purely on the backhaul traffic, and there may be no need at all for probe packets. Conversely, the underutilized links will not be able to make good estimations, and sending bursts of probe packets will not interfere with the actual traffic. This would present a solution to one of the biggest issues with the approach presented in this thesis.

Adjustments for Jitter

Another potential improvement would be to pass a "Time of Execution" field in the GTP header of packets of jitter-sensitive applications. This allows the receiving WAN connector to store packets if they have arrived too early, and, conversely, the packet ordering function may use this field to determine that it is more important to forward the current packet now, than to wait for a missing packet. This store and forward approach, with the time of execution field, is one of the solutions mentioned in the Deterministic Networking specification and would be a very sensible addition to this solution.

Additionally, specifically for the Lower Earth Orbit (LEO) satellite case, the path selection could potentially be adjusted to account for the periodic increases in latency as the current satellite leaves the range of the ground station, and the next one comes into range. For example during this phase it might make sense to temporarily forward packets on a different path, or to enable the store and forward mechanism, before disabling it once the satellite handover has passed.

Forward Error Correction

One possible improvement is the addition of Forward Error Correction (FEC). While this would be difficult to integrate into the equation to select paths, FEC could be used to increase the resilience of consistently lossy links, which is a big benefit for links which commonly exhibit this characteristic, such as wireless links. Especially LEO satellite links tend to present around 1% packet loss [**leo-links-characteristics1**], which makes the use of FEC much more reasonable since one would only need to add a 2 to 5% overhead, using a fountain code scheme, and thus wind up with a much more reliable link. However as was observed in the evaluation of the WAN Connector's throughput when using the Packet Ordering and Elimination Function, additional computation when receiving packets can lead to reduced overhead. To this extent, the addition of FEC would require efficient decoders and encoders on the sending and receiving side, so that the packet processing speed is not slowed down.

Traffic Prioritization

Perhaps the greatest theoretical flaw in the presented approach is that the WAN Connector does not perform packet prioritization, even though the 5G specifications explicitly mention that flows may have different priorities and should be prioritized accordingly. This is a significant flaw, and very difficult to account for. The most reasonable approach to integrating this is to add it to the traffic shaper. In the experiments performed here, the CAKE traffic

shaper was used. It's implementation in the linux kernel does allow for up to 8 different priority "tins" for traffic, however that is not enough for the 5G requirements which allow for a great deal more priority levels than just 8. One option, therefore, would be to extend the CAKE implementation to allow a greater number of priority tins. Alternatively, the a different traffic shaper could be chosen, or developed, which incorporates prioritization as well as fairness and active queue management at the level of individual IP flows.

Load Balancing

In instances where more than one path is viable, the algorithm used for path selection does not account for the current load of the paths, nor does it try to perform load balancing when selecting paths. This is because the algorithm attempts to find the minimum weight path. Even in cases where two paths have the same properties and the same weighting, the algorithm does not attempt to shuffle or rotate between them. All flows with similar requirements will wind up on the same path. This can lead to under utilization of other paths and congestion problems on the "primary" path. This could be partially addressed by integrating a degree of random selection for equal paths. However a potentially more interesting approach would be to dynamically update the weights of different paths, based on their current load. This would keep randomness out of the algorithm, for reproducible results, and act as a form of load balancing. However great care would have to be taken with respect to how the weights are adjusted, so that paths are still appropriately weighted relative to each other. i.e. a path which is deemed twice as expensive as another path should not have more than half as much traffic as the other path, for flows which could go be forwarded on either path.

Chapter 5

Conclusion and Outlook

5.1 Summary

In review, this thesis has addressed the usage of multiple backhaul paths in a campus 5G setting to achieve determinism. A review of the topic and relevant literature was done in the background chapter, then an approach was developed based on that information and based on the requirements of the problem. Finally this approach was evaluated in a testbed which emulated multiple outgoing links, to see how it performed. The implementation provided was able to achieve some of the goals set for latency and packet loss, but struggled with jitter reduction, as well as utilizing all of the available bandwidth when it is performing packet replication to improve reliability. In the discussion of these results, their potential sources and improvements were addressed.

Placed in a wider context, this thesis presents a foray into an area which will become more and more relevant as 5G campus deployments increase and mature, and as 5G applications which require greater degrees of determinism become more commonplace. On the whole, greater degrees of bespoke packet processing are required for traffic on the internet, as application's evolve and their requirements become more strict. Especially in geographically distributed

settings with multiple backhaul paths, the ability to intelligently select among these paths will be crucial to enabling these types of applications.

5.2 Implications and Further Areas of Research

The results obtained here imply the presented approach is worthy of further investigation, however only in conjunction with some of the presented improvements. The implementation provided by this thesis cannot achieve determinism without certain adjustments, but nor can it be concluded that it will be definitively unable to do so.

Since the results were only verified in a testbed setup it would make sense to now test the WAN Connector in a real campus 5G deployment where there actually are multiple outgoing paths. Furthermore many of the improvements suggested in the previous section would all be worthy of implementing and evaluating in similar experiments.

Research should also be conducted on evaluating the performance of specific applications performance. For example, the quality of VoIP calls does'nt depend just on latency and jitter [**voip-measurement**], but rather how they interact together, and they have their own suites of evaluation criteria. Another specific application to consider has to be interactive video. Mission critical as well as control systems could also be considered in the test suites for evaluation.

It would be interesting to see what benefit AI and machine learning approaches may bring to this problem since they can act more dynamically, and perhaps learn the characteristics of a given link over time. Perhaps they can discover what characteristics the link exhibits right before total failure and thus perform pre-emptive path switching.

Appendix A

Patch for BPF Flow Dissector

As was mentioned in the "Approach" chapter, in a true 5G Campus deployment the messages between either the RAN nodes or the UPF and the WAN Connector will be carrying GTP packets. In order for the tc-cake traffic shaper in the linux kernel to be able to detect the packets being tunneled, a custom BPF flow dissector must be used [1].

The linux kernel provides a very simple example of a set of dissectors which remove the tunnel headers for various protocols. Unfortunately the GTP protocol is not included in this set, but it is a very simple patch to add the required functionality to the kernel.

The listing A.1 shows a patch for the example file given in the linux kernel [2]. This patch constitutes an important part of this thesis' contribution because without it, a realistic deployment of the proposed system would not be possible, because the traffic shaper would not function.

[1] https://www.kernel.org/doc/html/v5.1/networking/bpf_flow_dissector.html#overview
[2] https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/tools/testing/selftests/bpf/progs/bpf_flow.c?h=

```
1
2  diff --git a/bpf_flow.c b/bpf_flow.c
3  index ed437e1..70a08a8 100644
4  --- a/bpf_flow.c
5  +++ b/bpf_flow.c
6  @@ -161,6 +161,8 @@ int _dissect(struct __sk_buff *skb)
7          return parse_eth_proto(skb, keys->n_proto);
8   }
9
10 +#define GTP_ENCAP_OFFSET 24
11 +
12  /* Parses on IPPROTO_* */
13  static __always_inline int parse_ip_proto(struct __sk_buff *skb
       , __u8 proto)
14  {
15 @@ -171,6 +173,10 @@ static __always_inline int parse_ip_proto(
       struct __sk_buff *skb, __u8 proto)
16          struct ethhdr *eth, _eth;
17          struct tcphdr *tcp, _tcp;
18          struct udphdr *udp, _udp;
19 +        struct udphdr *encap_udp, _encap_udp;
20 +        struct iphdr *iph, _iph;
21 +        __u16 srcport;
22 +        __u16 destport;
23
24          switch (proto) {
25          case IPPROTO_ICMP:
26 @@ -242,7 +248,26 @@ static __always_inline int parse_ip_proto(
       struct __sk_buff *skb, __u8 proto)
27                  udp = bpf_flow_dissect_get_header(skb, sizeof(*
                       udp), &_udp);
28                  if (!udp)
29                          return export_flow_keys(keys, BPF_DROP);
30 +                srcport = bpf_ntohs(udp->source);
31 +                destport = bpf_ntohs(udp->dest);
32 +                if (srcport == 2152 && destport == 2152) {
33 +                        keys->thoff += sizeof(*udp) +
       GTP_ENCAP_OFFSET;
34 +                        iph = bpf_flow_dissect_get_header(skb,
       sizeof (*iph), &_iph);
35 +                        if (iph) {
36 +                                keys->thoff += (iph->ihl << 2);
37 +                                encap_udp =
       bpf_flow_dissect_get_header(skb, sizeof(*encap_udp), &
       _encap_udp);
38 +                                if (encap_udp) {
39 +                                        keys->thoff -= (iph->ihl
        << 2);
```

```
40 +                                            keys->is_encap = true;
41 +                                            return parse_eth_proto(
   skb, bpf_htons(ETH_P_IP));
42 +                                    } else {
43 +                                            bpf_printk("Couldnt
   dissect encapsulated udp header");
44 +                                    }
45 +                            } else {
46 +                                            bpf_printk("Couldnt
   dissect encapsulated ip header at GTP offset (%d bytes)",
   keys->thoff);
47 +                            }
48
49 +                }
50                  keys->sport = udp->source;
51                  keys->dport = udp->dest;
52                  return export_flow_keys(keys, BPF_OK);
```

Listing A.1: Patch to remove GTP Headers for the BPF Flow Dissector Example in the Linux Kernel

Appendix B

JSON Formats Used in Control and Data-Plane Messages

This appendix contains excerpts of the JSON messages passed between the control plane and the data plane during operation. Listing B.1 shows how the control plane sends updates to the data plane nodes. Listing B.2 shows the format of the messages used for reporting statistics.

These are the only two message formats used between the Control Plane and the Data Plane. These messages are sent over SCTP, UTF-8 encoded. They are the only data in the SCTP messages' body. By using JSON there are three distinct advantages- firstly, no specific protocol must be developed for the messaging between control and data plane, secondly the messages are both human and machine readable which makes for easier debugging, and lastly because it is a hugely popular data format there are lots supporting libraries.

The messages which add new flows to the Data Plane contain the flow Description as well as its priority, and the Packet Ordering Function (POF) maximum delay, which was mentioned in the description of the POF algorithm in the approach.

In the reverse direction, when the Data Plane nodes report to the Control Plane, the statistics reported include firstly the paths description (source and destination IP address) as well as the number of packets and bytes received on that path, and the average jitter and latency experienced, both over the entire reporting period, and since the last message. Finally, the message are also sent with a list of the "sending" statistics. That is, the number of bytes and packets sent out on the WAN Connector's interfaces. Both the path and interface statistics are sent with three timestamps. One timestamp to describe the first packet sent or received during this reporting window, one to describe the last packet sent or received during that same reporting window, and a timestamp taken before the message itself was sent.

By including the "sending" statistics, the WAN Connector is able to inform it's own Control Plane if an interface is being actively used. On the receiving side, if an interface has not received packets from the respective source in too long a time, then the Control Plane can infer complete link failure. This is crucial for the ability of the WAN Connector to switch all user traffic off of dead links. Once a link works again, the WAN Connector can detect this because the periodic probe packets will be received.

```
1
2  {
3    "flowsToAdd": [
4      {
5        "flowID": 1,
6        "priority": 1,
7        "flowDescription": {
8          "src": "10.0.0.31",
9          "srcSubnet": 24,
10         "srcPortStart": 0,
11         "srcPortEnd": 0,
12         "dst": "10.0.3.31",
13         "dstSubnet": 24,
14         "dstPortStart": 0,
15         "dstPortEnd": 0,
16         "protocol": 1,
17         "direction": "in"
18       },
19       "POFmaxDelayMicros": 0,
20       "nForwarding": 1,
21       "decisions": [
22         {
23           "interfaceName": "link3",
24           "remoteEndpoint": {
25             "ip": "10.1.22.21",
26             "port": 2152
27           }
28         }
29       ]
30     },
31     {
32       "flowID": 2,
33       "priority": 1,
34       "flowDescription": {
35         "src": "10.0.0.31",
36         "srcSubnet": 24,
37         "srcPortStart": 0,
38         "srcPortEnd": 0,
39         "dst": "10.0.3.31",
40         "dstSubnet": 24,
41         "dstPortStart": 0,
42         "dstPortEnd": 0,
43         "protocol": 17,
44         "direction": "in"
45       },
```

```
46        "POFmaxDelayMicros": 0,
47        "nForwarding": 1,
48        "decisions": [
49          {
50            "interfaceName": "link3",
51            "remoteEndpoint": {
52              "ip": "10.1.22.21",
53              "port": 2152
54            }
55          }
56        ]
57      },
58      {
59        "flowID": 3,
60        "priority": 1,
61        "flowDescription": {
62          "src": "10.0.0.31",
63          "srcSubnet": 24,
64          "srcPortStart": 0,
65          "srcPortEnd": 0,
66          "dst": "10.0.3.31",
67          "dstSubnet": 24,
68          "dstPortStart": 0,
69          "dstPortEnd": 0,
70          "protocol": 6,
71          "direction": "in"
72        },
73        "POFmaxDelayMicros": 0,
74        "nForwarding": 1,
75        "decisions": [
76          {
77            "interfaceName": "link3",
78            "remoteEndpoint": {
79              "ip": "10.1.22.21",
80              "port": 2152
81            }
82          }
83        ]
84      }
85    ]
86  }
```

Listing B.1: Format of Flow Forwarding Decisions from Control Plane

```
{
 "pathStats": [{
   "src": "10.1.33.31",
   "dst": "10.0.33.32",
   "nbytes": 6272,
   "npackets": 224,
   "ndropped": 0,
   "avgLatencyMicros": 30655,
   "avgJitterMicros": 1835,
   "bytesSince": 56,
   "packetsSince": 2,
   "droppedSince": 0,
   "latencySinceMicros": 29276,
   "jitterSinceMicros": 1431,
   "earliestRecvdMillis": 1703448057713,
   "lastRecvdMillis": 1703448057716,
   "heartbeatTimestampMillis": 1703448058929
  }, {
   "src": "10.1.44.41",
   "dst": "10.0.44.42",
   "nbytes": 6272,
   "npackets": 224,
   "ndropped": 0,
   "avgLatencyMicros": 28648,
   "avgJitterMicros": 4182,
   "bytesSince": 56,
   "packetsSince": 2,
   "droppedSince": 0,
   "latencySinceMicros": 29540,
   "jitterSinceMicros": 5351,
   "earliestRecvdMillis": 1703448057711,
   "lastRecvdMillis": 1703448057718,
   "heartbeatTimestampMillis": 1703448058929
  }, {
   "src": "10.1.99.91",
   "dst": "10.0.99.92",
   "nbytes": 6912,
   "npackets": 239,
   "ndropped": 6,
   "avgLatencyMicros": 23577,
   "avgJitterMicros": 2592,
   "bytesSince": 56,
   "packetsSince": 2,
   "droppedSince": 0,
```

```
    "latencySinceMicros": 20983,
    "jitterSinceMicros": 3116,
    "earliestRecvdMillis": 1703448057704,
    "lastRecvdMillis": 1703448057708,
    "heartbeatTimestampMillis": 1703448058929
   }, {
    "src": "10.1.22.21",
    "dst": "10.0.22.22",
    "nbytes": 6272,
    "npackets": 224,
    "ndropped": 0,
    "avgLatencyMicros": 23387,
    "avgJitterMicros": 8052,
    "bytesSince": 56,
    "packetsSince": 2,
    "droppedSince": 0,
    "latencySinceMicros": 16977,
    "jitterSinceMicros": 9769,
    "earliestRecvdMillis": 1703448057701,
    "lastRecvdMillis": 1703448057704,
    "heartbeatTimestampMillis": 1703448058929
   }],
  "interfaceStats": [{
    "name": "link1",
    "src": "10.0.99.92",
    "nbytes": 170863822,
    "npackets": 114442,
    "bytesSince": 5812016,
    "packetsSince": 3887,
    "earliestSentMillis": 1703448055928,
    "lastSentMillis": 1703448058928,
    "heartbeatTimestampMillis": 1703448058929
   }, {
    "name": "link2",
    "src": "10.0.33.32",
    "nbytes": 162768024,
    "npackets": 109020,
    "bytesSince": 56,
    "packetsSince": 2,
    "earliestSentMillis": 1703448055928,
    "lastSentMillis": 1703448055928,
    "heartbeatTimestampMillis": 1703448058929
   }, {
    "name": "link3",
    "src": "10.0.22.22",
```

```
91      "nbytes": 87215536,
92      "npackets": 58517,
93      "bytesSince": 56,
94      "packetsSince": 2,
95      "earliestSentMillis": 1703448055928,
96      "lastSentMillis": 1703448055928,
97      "heartbeatTimestampMillis": 1703448058929
98    }, {
99      "name": "link4",
100     "src": "10.0.44.42",
101     "nbytes": 174418872,
102     "npackets": 116808,
103     "bytesSince": 56,
104     "packetsSince": 2,
105     "earliestSentMillis": 1703448055928,
106     "lastSentMillis": 1703448055928,
107     "heartbeatTimestampMillis": 1703448058929
108   }]
109 }
```

Listing B.2: Format of Statistics from Data Plane

Appendix C

Example of Custom GTP Header Format

This appendix contains a screenshot of a packet captured during normal operation of the WAN Connector (figure C.1), being displayed using wireshark [1]. The packet dissection shows the GTP format being used, along with the additional custom header, which is 8 bytes long. The custom header consists of 2 bytes of regular overhead from the GTP protocol as well as a 4 byte timestamp and 2 spare bytes. This packet was captured during the jitter experiment.

[1] https://www.wireshark.org/

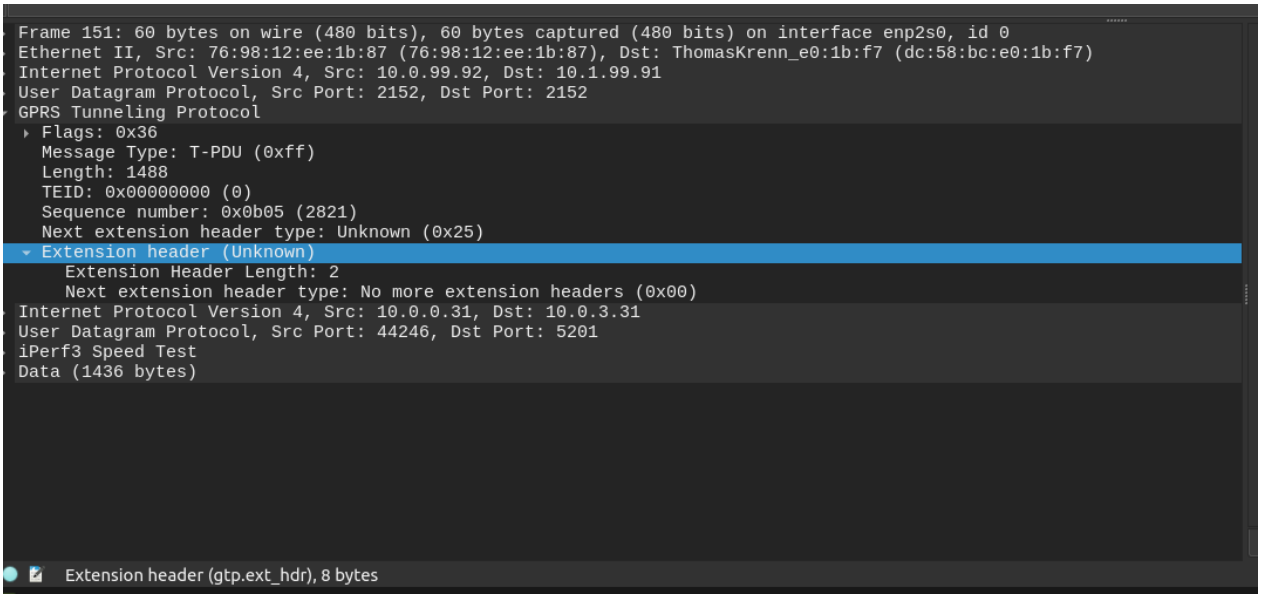


Figure C.1: Screenshot of Wireshark Dissection

Bibliography

[1] Aditya Akella et al. "A measurement-based analysis of multihoming". In: *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. 2003, pp. 353–364.

[2] Aditya Akella et al. "On the performance benefits of multihoming route control". In: *IEEE/ACM Transactions on Networking* 16.1 (2008), pp. 91–104.

[3] Hind Alwan and Anjali Agarwal. "Multi-objective reliable multipath routing for wireless sensor networks". In: *2010 IEEE Globecom Workshops*. IEEE. 2010, pp. 1227–1231.

[4] Jennifer Andreoli-Fang and John T Chapman. "Mobile-aware scheduling for low latency backhaul over DOCSIS". In: *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. IEEE. 2017, pp. 1–6.

[5] Maria Apostolaki, Ankit Singla, and Laurent Vanbever. "Performance-driven internet path selection". In: *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*. 2021, pp. 41–53.

[6] Austin Appleby. "MurmurHash3, 2012". In: *URL: https://github. com/aappleby/smhasher/blob/m cpp* (2012).

[7] Xavier Artiga et al. "Terrestrial-satellite integration in dynamic 5G backhaul networks". In: *2016 8th advanced satellite multimedia systems conference and the 14th signal processing for space communications workshop (ASMS/SPSC)*. IEEE. 2016, pp. 1–6.

[8] Anat Bremler-Barr et al. “Predicting and bypassing end-to-end Internet service degradations”. In: *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment*. 2002, pp. 307–320.

[9] Nelson Capela and Susana Sargento. “Multihoming and network coding: A new approach to optimize the network performance”. In: *Computer Networks* 75 (2014), pp. 18–36.

[10] Kameswari Chebrolu and Ramesh R Rao. “Bandwidth aggregation for real-time applications in heterogeneous wireless networks”. In: *IEEE Transactions on Mobile Computing* 5.4 (2006), pp. 388–403.

[11] Fuqiao Chen and Chenyang Yin. “A Collaborative Traffic Scheduling Mechanism for Multi-homed Networks”. In: *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*. IEEE. 2020, pp. 177–181.

[12] Joerg Deutschmann, Kai-Steffen Hielscher, and Reinhard German. “Broadband internet access via satellite: Performance measurements with different operators and applications”. In: *Broadband Coverage in Germany; 16th ITG-Symposium*. VDE. 2022, pp. 1–7.

[13] Doğanalp Ergenç and Mathias Fischer. “On the reliability of ieee 802.1 cb frer”. In: *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE. 2021, pp. 1–10.

[14] Norman Finn and Pascal Thubert. *Deterministic networking problem statement*. Tech. rep. 2019.

[15] Norman Finn et al. “Deterministic networking architecture”. In: *RFC 8655*. IETF, 2019.

[16] Igor Ganichev et al. “YAMR: Yet another multipath routing protocol”. In: *ACM SIGCOMM Computer Communication Review* 40.5 (2010), pp. 13–19.

[17] David K Goldenberg et al. “Optimizing cost and performance for multihoming”. In: *ACM SIGCOMM Computer Communication Review* 34.4 (2004), pp. 79–92.

[18] Fanglu Guo et al. “Experiences in building a multihoming load balancing system”. In: *IEEE INFOCOM 2004*. Vol. 2. IEEE. 2004, pp. 1241–1251.

[19] Ahsan Habib and John Chuang. “Improving application QoS with residential multihoming”. In: *Computer Networks* 51.12 (2007), pp. 3323–3337.

[20] Toke Høiland-Jørgensen, Dave Täht, and Jonathan Morton. “Piece of CAKE: a comprehensive queue management solution for home gateways”. In: *2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. IEEE. 2018, pp. 37–42.

[21] Chung-Ming Huang and Ching-Hsien Tsai. “WiMP-SCTP: Multi-path transmission using stream control transmission protocol (SCTP) in wireless networks”. In: *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*. Vol. 1. IEEE. 2007, pp. 209–214.

[22] Xiaoxia Huang and Yuguang Fang. “Multiconstrained QoS multipath routing in wireless sensor networks”. In: *Wireless Networks* 14.4 (2008), pp. 465–478.

[23] Mona Jaber et al. “5G backhaul challenges and emerging research directions: A survey”. In: *IEEE access* 4 (2016), pp. 1743–1766.

[24] Ralf Kundel et al. “User Plane Hardware Acceleration in Access Networks: Experiences in Offloading Network Functions in Real 5G Deployments.” In: *HICSS*. 2022, pp. 1–10.

[25] Stanislav Lange et al. “Performance benchmarking of a software-based LTE SGW”. In: *2015 11th International Conference on Network and Service Management (CNSM)*. IEEE. 2015, pp. 378–383.

[26] Ming Li et al. “Multipath transmission for the internet: A survey”. In: *IEEE Communications Surveys & Tutorials* 18.4 (2016), pp. 2887–2925.

[27] Sami Ma et al. “Network characteristics of LEO satellite constellations: A Starlink-based measurement from end users”. In: *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE. 2023, pp. 1–10.

[28] Thinh Nguyen and Avideh Zakhor. "Path diversity with forward error correction (pdf) system for packet switched networks". In: *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428)*. Vol. 1. IEEE. 2003, pp. 663–672.

[29] Jonathan Prados-Garzon and Tarik Taleb. "Asynchronous time-sensitive networking for 5G backhauling". In: *IEEE Network* 35.2 (2021), pp. 144–151.

[30] Pablo Rodriguez et al. "Mar: A commuter router infrastructure for the mobile internet". In: *Proceedings of the 2nd international conference on Mobile systems, applications, and services*. 2004, pp. 217–230.

[31] Reza Sheyibani et al. "A reliable and qos aware multi-path routing algorithm in wsns". In: *2012 Third International Conference on Emerging Intelligent Data and Web Technologies*. IEEE. 2012, pp. 125–132.

[32] Ion Stoica, Hui Zhang, and TS Eugene Ng. "A hierarchical fair service curve algorithm for link-sharing, real-time and priority services". In: *ACM SIGCOMM Computer Communication Review* 27.4 (1997), pp. 249–262.

[33] Shu Tao and Roch Guérin. "Application-specific path switching: A case study for streaming video". In: *Proceedings of the 12th annual ACM international conference on Multimedia*. 2004, pp. 136–143.

[34] Shu Tao et al. "Exploring the performance benefits of end-to-end path switching". In: *Proceedings of the joint international conference on Measurement and modeling of computer systems*. 2004, pp. 418–419.

[35] Shu Tao et al. "Improving VoIP quality through path switching". In: *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.* Vol. 4. IEEE. 2005, pp. 2268–2278.

[36] Mohammed Tarique et al. "Survey of multipath routing protocols for mobile ad hoc networks". In: *Journal of network and computer applications* 32.6 (2009), pp. 1125–1143.

[37] Jack Tsai and Tim Moors. "A review of multipath routing protocols: From wireless ad hoc to mesh networks". In: *ACoRN early career researcher workshop on wireless multihop networking*. Vol. 30. Citeseer. 2006.

[38] Simon Tschöke et al. "Time-sensitive networking over metropolitan area networks for remote industrial control". In: *2021 IEEE/ACM 25th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE. 2021, pp. 1–4.

[39] Zhiqiang Xiong et al. "A lightweight FEC algorithm for fault tolerant routing in wireless sensor networks". In: *2006 International Conference on Wireless Communications, Networking and Mobile Computing*. IEEE. 2006, pp. 1–4.

[40] Pouria Zand et al. "Wireless industrial monitoring and control networks: The journey so far and the road ahead". In: *Journal of sensor and actuator networks* 1.2 (2012), pp. 123–152.

[41] Gongzheng Zhang et al. "Fundamentals of heterogeneous backhaul design—Analysis and optimization". In: *IEEE Transactions on Communications* 64.2 (2016), pp. 876–889.