Extending Network Slice Management to the End-host

Alexander Rabitsch
alexander.rabitsch@kau.se
Karlstad University

George Xilouris
Themistoklis Anagnostopoulos
{xilouris,thmanagnostopoulos}@iit.demokritos.gr
NCSR Demokritos

Karl-Johan Grinnemo
karl-johan.grinnemo@kau.se
Karlstad University

Thanos Sarlas
tsarlas@iit.demokritos.gr
NCSR Demokritos

Anna Brunstrom
anna.brunstrom@kau.se
Karlstad University

Özgü Alay[1,2],
Giuseppe Caso[2]
{ozgu,giuseppe}@simula.no
University of Oslo[1], Simula Metropolitan[2]

ABSTRACT

The network slicing concept of 5G aims to provide the flexibility and scalability required to support a wide array of vertical services. To coordinate the coexistence of network slices, and to guarantee that the required resources are available for each one of them, the 5G core employs a slicing management entity, a slice manager. In this paper, we propose an architecture where the network slicing concept is extended beyond the core and access networks to also include the configuration of the UE's network stack. We exploit the slice manager's global view on the network to feed fine-grained information on slice configuration, health, and status to the UE. This information, together with local policies on the UE, is then used to dynamically create services tailored to the requirements of individual applications. We implement this architecture in a 5G testbed, and show how it can be leveraged in order to enable optimized services through dynamic network protocol configuration, application-to-slice mapping, and network protocol selection.

CCS CONCEPTS

• **Networks** → **Network architectures**; **Transport protocols**; *Mobile networks*; *Network management*.

KEYWORDS

5G, Network slicing, Transport Services, Transport Layer, User Equipment

ACM Reference Format:
Alexander Rabitsch, George Xilouris, Themistoklis Anagnostopoulos, Karl-Johan Grinnemo, Thanos Sarlas, Anna Brunstrom, Özgü Alay, Giuseppe Caso. 2021. Extending Network Slice Management to the End-host. In *Workshop on 5G Measurements, Modeling, and Use Cases (5G-MeMU '21), August 23, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3472771.3472775

5G-MeMU '21, August 23, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8636-4/21/08…$15.00
https://doi.org/10.1145/3472771.3472775

1 INTRODUCTION

The network slicing concept was introduced by the 3rd Generation Partnership Project (3GPP) in order to provide optimized support for the wide set of different communication services, traffic loads, and end users [5] that the fifth generation of cellular networks (5G) is envisioned to support. Network slicing leverages Software Defined Networking (SDN), Network Function Virtualization (NFV), and the upcoming radio slicing capabilities [6] for the creation of separate, isolated end-to-end Network Slice Instances (NSIs) whose properties can be tailored to the specific application requirements. These kinds of NSIs are defined within a single Public Land Mobile Network (PLMN) and include the Core Network Control Plane and User Plane Network Functions as well as the Next Generation Radio Access Network (NG-RAN) [1].

In this context, a User Equipment (UE) that is connected to several differently configured slices may utilize them to receive different services. However, the only information a UE has on a network slice is, currently, very coarse-grained, and the details of the service offered by a network slice are not visible to the UE. We argue that the configuration of the network stack of a UE, such as the selection and configuration of network protocols, is crucial to the end-to-end performance of its applications. Still, at present, the optimization of the UE network stack is a challenge due to the lack of information provided by a 5G system on slice and RAN connectivity configurations.

To address this issue, we therefore propose an architecture that extends the network slicing concept beyond the core and access networks to also include the configuration of UE network stacks. We exploit the transport services (TAPS) architecture [21], which decouples applications from the underlying transport protocols and network properties to facilitate the optimization of the network stack on the basis of application requirements and network conditions at run-time. Together with fine-grained NSI information provided by the network, the system can, for example, intelligently choose an appropriate transport protocol, dynamically configure the protocol parameters, and a dynamically map flows to network slices, in order to fulfill the requirements of individual applications running on the UE.

The contribution of this work is threefold: (i) We define an interface between the UE and the network, where the network can assist the UE in dynamically fine-tuning the network stack of individual

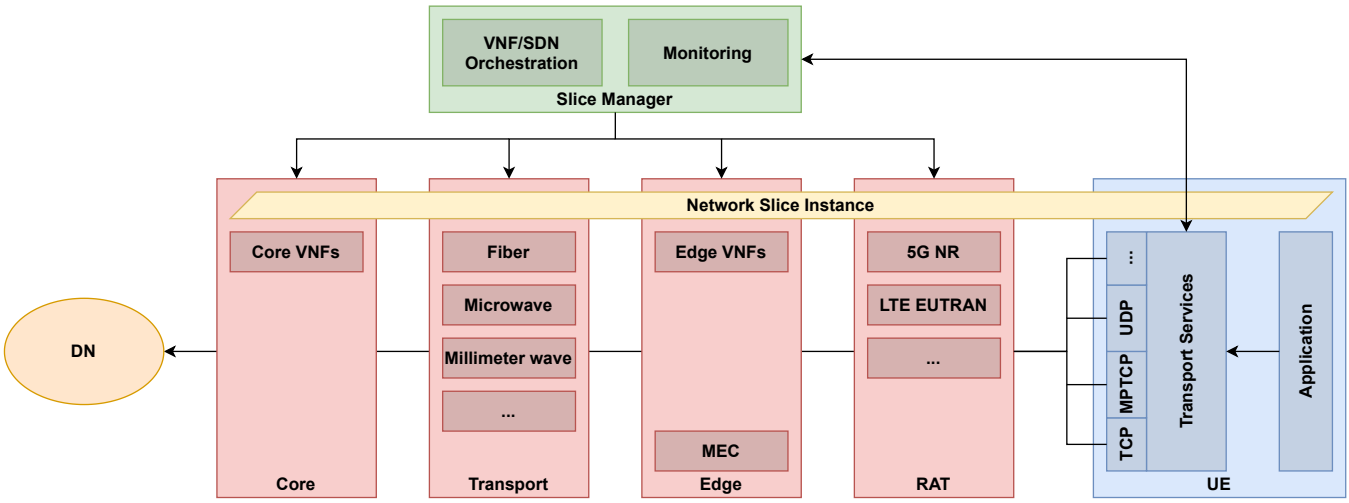

Figure 1: An overview of the proposed architecture. The network slice manager is shown in green, the 5G core and RAN networks in pink, and the UE in blue color.

applications by exposing slice information to the UE, or by deploying policies to enforce specific behaviors, (ii) we implement and test our proposed architecture in a 5G testbed, (iii) we highlight in three separate experiments how the proposed architecture can be used to create optimized services based on application requirements.

The remainder of this paper is structured as follows: Section 2 introduces the main concepts of our proposed architecture and its overarching design. In Section 3, we show an example implementation of our proposed architecture, and in Section 4 we employ a 5G testbed in the city of Athens, part of the 5GENESIS experimentation facility [23], to demonstrate in three experiments how our architecture can benefit the aforementioned use cases. Section 5 reviews related work and discusses directions for future improvements. Finally, Section 6 concludes the paper. This work does not raise any ethical issues.

2 KEY CONCEPT AND SYSTEM DESIGN

With network slicing being an important piece of 5G, we believe it is feasible to create an architecture that extends the network slicing concept beyond the core and access networks, and encompasses the applications running on top of the UEs. Our proposed architecture enables applications to dynamically configure their network stack, not only on the basis of their requirements, but also on fine-grained slice information. As the pivotal component for managing network slices, the *slice manager* has a complete view of the existing NSI. To this end, the UE interfaces with the slice manager to obtain up-to-date information on the NSI, i.e., its characteristics, health, and status. The slice manager may also push policies to enforce a specific behavior in the UE. This interface does not necessarily need to go directly from the UE to the slice manager; the UE may alternatively interface with the slice manager via another network function, such as the Network Exposure Function (NEF) [2].

Such an architecture enables, for example, (i) intelligent *protocol configuration*: e.g., by configuring TCP's Initial Congestion Window (IW), based on the network slice characteristics to better utilize the available resources. Traditionally, the recommended value for the IW is set quite conservatively [9, 20, 22], to avoid overwhelming the network. However, if the sender already knows that a certain amount of bandwidth is reserved for the sender, the IW can be set to a larger value, (ii) *application-to-slice mapping*: e.g., latency sensitive flows can be mapped onto a slice optimized for low-latency, whereas other flows are transmitted over a best-effort slice, and (iii) *slice aggregation*: where applications can be configured to use multiple slices *simultaneously*, in those cases the slices use different paths. In such a scenario, the network could either push policies to the UE that enforce the use of Multipath TCP (MPTCP), or otherwise inform the UE that multiple paths are available so that the UE can autonomously decide on the policy to adopt. We highlight the three above mentioned use cases in more detail in Sections 4.2, 4.3, and 4.4, respectively.

Figure 1 gives an overview of our architecture. The key idea is to let the network assist the UE in its selection of a *transport service*, i.e., the configuration of a suitable network protocol stack. Normally, the only information a UE has on a network slice is limited to what is contained in the Single – Network Slice Selection Assistance Information (S-NSSAI), i.e., very coarse-grained information on the type of service that the slice is meant to support. The goal of our proposed architecture is to feed fine-grained NSI information to the UE, and in that way make it possible to compute the "best" protocol stack configuration for an application running on the UE. The applications in our proposed architecture never concern themselves with the underlying network mechanisms. Instead, the applications convey their high-level requirements to a system that implements the TAPS architecture. TAPS offers a flexible, protocol-independent API, where the most suitable protocol stack configuration can be decided at run-time based on application requirements, current network conditions, and hardware capabilities. An application may for example express demands such as a minimum throughput, maximum latency, and other QoS requirements, or even specific features such as congestion control, reliability, and security. Such requests

are then combined together with local policies and information on network characteristics to select the best available transport protocol, configuration, and network interface. In our proposed architecture, the requirements of the applications are combined with the NSI information supplied by the network to dynamically configure a suitable protocol stack for the applications.

3 IMPLEMENTATION

In this section, we provide details of the three main components in our implementation of the proposed architecture: (i) the Slice Manager, (ii) the Transport Service (TAPS) Module, and (iii) the UE Slice Configuration Interface (USCI) that sits at the UE side to orchestrate the communication between the first two components. Figure 2 provides an overview of the system.

3.1 Slice Manager

The slice manager used in this work is the open source Katana Slice Manager.[1] Katana provides an interface for creating, modifying, monitoring, and deleting end-to-end network slices [8]. Through its North Bound Interface (NBI), Katana interacts with the network operator to enable the management and monitoring of network slices, and through its South Bound Interface (SBI), Katana communicates with the components comprising the Management and Orchestration (MANO) layer, namely: the NFV Orchestrator (NFVO), the Element Management System (EMS), and the Wide-Area Network Infrastructure Management (WIM). Through the use of the Katana SBI, resource provisioning and sub-network slice instantiation is achieved across different technologies and domains, as depicted in Figure 1.

Katana is implemented as a collection of micro-services, each of which is running in a Docker container, thus comprising a platform-agnostic, highly modular architecture that enables a rapid, frequent, and reliable deployment of slices. In the service mesh, requests are routed between micro-services through a common message bus system, implemented by an Apache Kafka container. The main components of the Katana software stack are (i) the Katana NBI, a module that implements RESTful APIs that can be consumed by the network operator; (ii) the Katana manager, the brain of the slice manager, which is responsible for service placement, resource provisioning, service activation and configuration, and the decommissioning of network slices, (iii) the Katana monitoring module, a module that monitors the health and the status of the deployed slices, and finally, (iv) an adaptation layer, which provides a level of abstraction to the technology of the underlying layer, making it feasible to operate over any MANO layer component without any modifications to its core functionality.

In order to create a slice, a descriptor file containing a NEtwork Slice Template (NEST) is sent together with a slice creation request to Katana via the NBI. The NEST is generated by filling in the values in the Generic Slice Template (GST) (based on the GSMA GST [13]), which is a set of attributes that can be used to characterise a type of network slice/service. The NEST is then parsed by Katana through a slice mapping process, which, in combination with the supported network functions by the underlying infrastructure, creates the slice.

[1]Katana Repository, https://github.com/medianetlab/katana-slice_manager. Accessed on: May 2021.

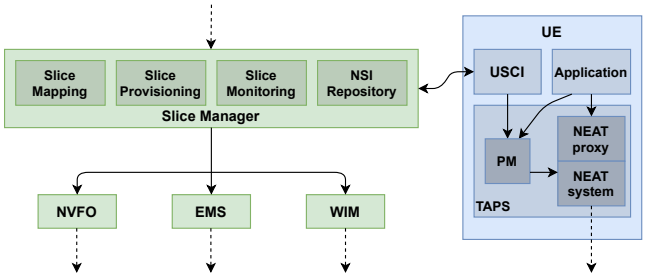

Figure 2: Implementation details, including the three main components: the network slice manager, the Transport Service (TAPS) module, and the UE Slice Configuration Interface (USCI).

3.2 Transport Service Module

In the implementation of our proposed architecture, we use the NEAT TAPS implementation [18] for UE configuration and policy enforcement. Applications access the NEAT system via a user API that offers functionality similar to that offered by the socket API, *with the distinction that the API is protocol-agnostic.* Besides the API and its associated diagnostics and statistics interface, the NEAT system also includes the Policy Manager (PM), that is responsible for the selection of the transport service, i.e., the selection and configuration of a suitable transport protocol and network interface. On the basis of the information provided by the user API, the PM, and optional probing/signaling mechanisms, the NEAT system computes a suitable configuration of the network protocol stack.

To take advantage of NEAT's capabilities, an application must use the NEAT user API. As it is not always feasible to rewrite existing applications so that they make use of this API, we employ a NEAT proxy [11] deployed on the UE, to provide applications with transparent access to the communication services of NEAT. The applications running on top of the UE are able to provide the NEAT system with their requirements by interfacing via a REST API with the PM. Through the use of Linux transparent proxy support, traffic is routed through the proxy, which terminates outgoing connections and, with the help of the NEAT system, establishes a new connection to the original destination.

3.3 The UE Slice Configuration Interface

The communication between the Katana slice manager and the UE is done through the *UE Slice Configuration Interface* (USCI) on the UE. When joining the network, the UE obtains the details required for contacting the slice manager, and also retains a unique identifier (e.g., the S-NSSAI) of each slice it belongs to. This information could for example be acquired during the registration procedure between the UE and the Access and Mobility Management Function (AMF). With this information, the USCI can then register itself at the slice manager via the slice manager's SBI. The USCI is also responsible for de-registering before the UE leaves the network.

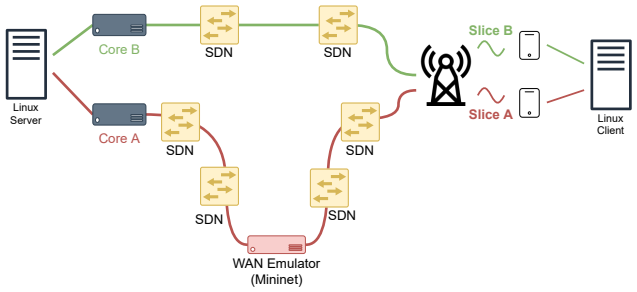

Figure 3: Network topology and testbed setup.

Once registered at the slice manager, the USCI begins to periodically poll for information on the slices identified by the retained slice identifiers. The information provided by the slice manager includes the attributes in the Katana GST; i.e., information on throughput guarantees, latency, 3GPP 5G QoS Identifier (5QI) [1] and other QoS characteristics, as well as information such as mobility support, RAN and core isolation levels, and more. In addition, the slice manager can provide information about the topology and the underlying network functions that are part of the slice. This information is then prepared by the USCI for use in the NEAT module, which is then added to the PM via a REST API. The same API can also be used by the slice manager directly to push policies to the UE.

4 CASE STUDIES

In order to illustrate the benefits of our proposed architecture, we show three different case studies, highlighting different use cases: protocol configuration, application-to-slice mapping, and slice aggregation via protocol selection.

4.1 Experiment setup

We have implemented the proposed architecture in the 5GENESIS Athens platform. It consists of a Linux client machine, representing the UE, which is connected to a Linux server over two different network slices. Both the client and server machines run Ubuntu 18.04.2 LTS. For the first two experiments, we use version 5.4.0 of the Linux kernel. For the third experiment, we use the MPTCP version 0.95 (Linux version 4.19.126) kernel.

Figure 3 depicts an overview of the network topology. The two network slices that are instantiated are *Slice A*, which is the default slice that allows interconnection of all UEs with the Internet, and *Slice B* which is destined for services sensitive to latency. The 5G RAN is common for both slices, as radio slicing capabilities are currently not available in the testbed. *The slice configuration is the same for all experiments.* Both core and radio components are compliant with Release 15, deployed in 5G standalone mode. QoS and priority policy is applied to each slice through different QoS Class Identifier (QCI)/5QI radio bearers. The radio component is configured to use New Radio (NR) band n78 (3500 MHz) in Time Division Duplex (TDD) mode, with 50 MHz bandwidth and up to 256 Quadrature Amplitude Modulation (256-QAM). The average Round-Trip Time (RTT) measured between each core and the attached UE is 29.8 ms for *Slice A* and 12.0 ms for *Slice B*. With the given radio configuration, the maximum achieved throughput by one UE attached to the mobile network is of about 362 Mbps. For the experiments, two mobile phones provide network connection to the client machine via tethering. This setup is required because, to our knowledge, there are currently no devices capable of connecting to multiple slices concurrently. We note that while this setup introduces latency and reduces the maximum bitrate, due to the use of USB interfaces, it does not affect the conceptual demonstration of our proposed architecture.

4.2 Experiment 1 - Protocol configuration

The first experiment demonstrates the dynamic tuning of TCP's IW at the UE for improved performance. Our strategy for setting the tuned IW, denoted IW' in the following, is provided in Equation (1):

$$IW' = \alpha * \frac{T * RTT}{S}, \tag{1}$$

where T is the throughput reserved for the UE in the uplink, in bytes/second, RTT is the upper bound for the round-trip time, in seconds, of the network slice, and S is the maximum packet size in bytes. The size of IW' can be fine-tuned by the weight α ($0 < \alpha \leq 1$), where a larger value leads to a more aggressive utilization of the slice resources.

For this experiment, we upload files of various sizes from the UE to a server, while dynamically tuning the IW based on information from the slice manager, and measure the completion times. To investigate how the dynamic IW tuning impacts different flow sizes, we upload files from 10 kB to 10000 kB. We look at four different α values, and compare the performance against the Linux default IW of 10. In order to avoid large bursts due to the increased IW, we enable packet pacing. The details on the experiment parameters are provided in Table 1.

Table 1: IW Tuning Experiment Parameters

Parameter	Values
File size (kB)	10, 50, 100, 500, 1000, 5000, 10000
Weight (α)	0.25, 0.50, 0.75, 1.00

Figure 4 shows the average upload times of the 10 kB, 1000 kB and 5000 kB files, normalized against the results obtained by adopting $IW = 10$. For the 10 kB flows, we see no significant differences between the different IWs, as the upload size is small enough to fit in the default window of ten packets. For all other flows, we see that the upload time can be decreased by tuning the IW. Overall, we see the largest improvement for medium sized flows (50 kB - 1000 kB), with a reduced median upload time of up to 40%. For the longer flows, the percentage decrease in the average upload time is smaller (≈20%). We also see that using $\alpha = 1.00$ can quickly congest the bottleneck link, as we observe that, due to queuing delay, the average RTT can become significantly higher (up to 30%) than for other values of α. Note that we do not necessarily recommend using the presented approach for setting the IW in the wild, as we have not investigated the potential negative impact it may have on the network. How to do this safely is a question we leave for future

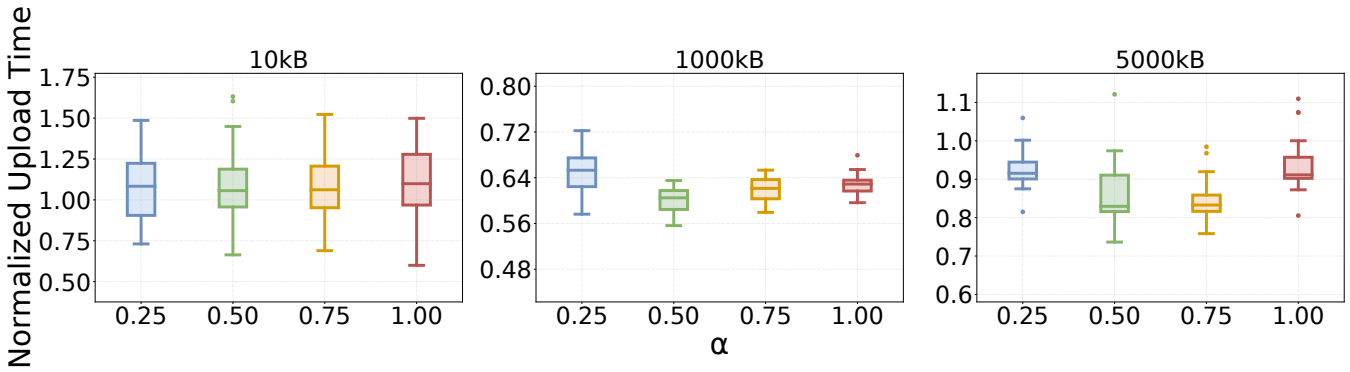

Figure 4: Protocol configuration test: TCP performance comparison for different α weights, for short flows (left), medium flows (middle), and long flows (right), normalized against IW = 10.

work. Rather, we see this result as a simple demonstration of the possibilities of our proposed architecture.

4.3 Experiment 2 - Application-to-slice mapping

For our second experiment, we highlight another important use case: dynamic application-to-slice mapping, in the case that a UE is served by more than one slice. To illustrate how the UE can use our architecture to dynamically select an appropriate slice for individual data flows, we consider a scenario where a remote controlled drone communicates with a controller over Slice A and Slice B. We emulate the traffic of two flows with distinct traffic patterns; a latency sensitive control flow between the drone and a controller, and an uplink video stream from the drone to the same controller. In the default case, both flows would contend for resources over Slice A, whereas the traffic could be split such that the latency sensitive flow is instead directed over Slice B. For the low-latency flow, we replay a packet trace captured from an actual drone and controller over TCP. The traffic was captured on-board the drone PixHawk 4.0[2] autopilot. The autopilot exploited UE tethering in order to connect via a 5G radio interface with the 5G RAN. The traffic capture comprised of MAVLink[3] (a lightweight messaging protocol for communicating with drones) packets sent over the 5G RAN with navigation commands. For the video flow, we use the Real-Time Messaging Protocol (RTMP) to stream a video file via ffmpeg to an nginx server on the controller. We use policies on the drone's TAPS module to distinguish between the two different flows, and NSI information from the slice manager to make an appropriate selection of the network slice.

Figure 5 shows the Empirical Cumulative Distribution Function (ECDF) of the packet delivery times for the low-latency control flow, counted from the transmission of a packet to the arrival of the corresponding ACK. The results illustrate a clear benefit of directing the latency sensitive traffic over *Slice B* rather than over *Slice A*. For the video flow, we see no significant difference in the achieved throughput in either of the two scenarios.

[2]PixHawk 4.0 https://pixhawk.org
[3]MAVLink https://mavlink.io/en/

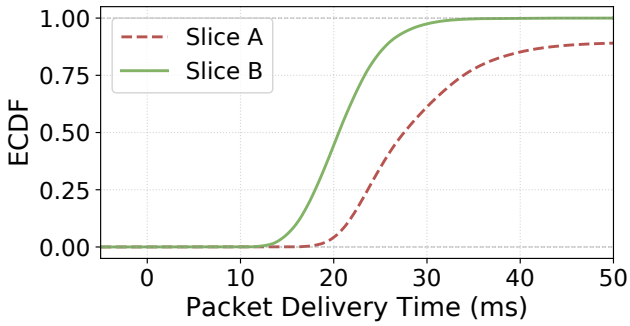

Figure 5: Slice Mapping test: Packet delivery times for the latency sensitive control flow over Slice A (best effort), and Slice B (low-latency).

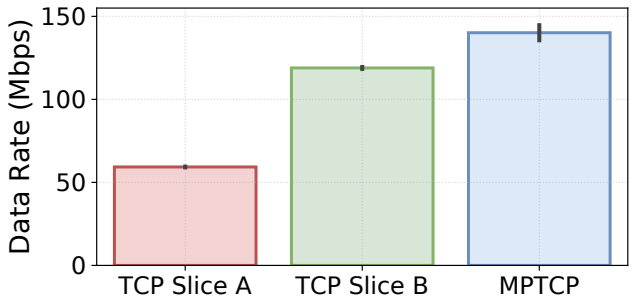

Figure 6: Aggregation test: Average data rates for single-path TCP over Slice A and Slice B, and MPTCP over both.

4.4 Experiment 3 - Multipath aggregation

In our final experiment, we highlight how the proposed architecture can be used for protocol selection to better be able to meet application requirements. In this scenario, the UE is again served by Slice A and Slice B. These two slices take two distinct paths over the network, as shown in Figure 3. Based on the knowledge of

the topology of the two slices, the slice manager provides information to the UE to let it know that multiple paths are available. An application on the UE requests a service with a specific downlink throughput requirement to the TAPS module. If the throughput requirement is below the guaranteed downlink throughput of at least one of the two slices, the TAPS module can conclude that throughput aggregation is not needed. Data sent over MPTCP often arrive out-of-order due to asymmetries in the path latency, giving rise to issues such as *head-of-line blocking* and *receive window blocking* [19]. These issues can degrade the performance of the protocol enough to make single-path TCP preferable. We therefore avoid using MPTCP by default, and only enable MPTCP for the application if the throughput aggregation is required to be able to fulfill the application's requirements.

For this experiment, the application on the UE requests a minimum data rate of 130 Mbps, which is above the advertised throughput of either of the two slices. However, by aggregating the capacity of the two slices using MPTCP, this requirement can be fulfilled, as shown in Figure 6. The results were taken from data generated by the *iperf3* network test tool. We note that MPTCP is unable to achieve the ideal aggregation (i.e the sum of the capacity of both paths). This is due to to the head-of-line blocking and receive window blocking issues mentioned previously.

5 RELATED AND FUTURE WORK

Several standardization and research activities consider the issue of extending the visibility of network configurations in order for applications to take advantage of a refined control and an increased Quality of Service (QoS). Among others, the Application-Layer Traffic Optimization (ALTO) protocol [17] provides network information, e.g., abstract network topology information and preferences of network paths, with the goal of improving the performance of applications. With the aim to optimize the Quality of Experience (QoE) of current and emerging communication paradigms, Gardikis *et al.* propose a framework that calculates a *network cost* between two or more network nodes, and provides QoE estimation for specific applications.

Mechanisms such as Mobile Throughput Guidance (MTG) exposure [14, 15] and the Mobile and Wireless Information Exposure (MoWIE) framework enable cellular networks to convey information to network-aware senders in order to improve TCP and application performance. These efforts have primarily focused on enhancing the network visibility for application providers in Multi-access Edge Computing (MEC) contexts. Our proposed architecture differs from these efforts by considering what can be done at the UE, and by exploiting end-to-end information on network slices.

Multiple works take advantage of the global view of SDN controllers to expose network information to applications, e.g. [10, 12, 16]. The work presented by Bozakov *et al.* [7] is perhaps closest to our work. They use TAPS in a policy-driven SDN framework, where network controllers and end-hosts cooperate to optimize the performance of individual applications as well as of the overall network. The aforementioned works focus primarily on software defined data center networks. Our work can be seen as an extension of these works, as it applies similar concepts to the context of network slicing.

3GPP Rel.17 specifications define APIs designed for exposing 5G system (5GS) information such as Network Exposure Function (NEF) [2] which provides means to securely expose the services and capabilities provided by 3GPP network functions; CAPIF [3] which provides for a unified north bound API framework across several 3GPP functions and Service Enabler Architecture Layer for Verticals (SEAL) [4] which provides signaling and application planes for application-enabling services that can be reused across vertical applications. The above specification all cater for releasing 5GS information and management capabilities towards vertical applications and are considered also as a possible integration for TAPS signaling requirements.

There are still a number of unanswered questions that we intend to answer in future work. The experiments presented in this paper are intended as a conceptual demonstration of our proposed architecture. In the future, we intend to examine how the system performs in more complex scenarios, such as when there are multiple competing flows. We also aim to address the overhead induced by the slice information exchange between the UE and the network. In the implementation presented in this paper, a message from the slice manager can have a size of up to several kB. The size of these messages, and the rate at which they are exchanged must be balanced such that the overhead does not outweigh the benefits. This is especially relevant when fetching very dynamic information from the slice manager, lest the UE acts on outdated information. In addition, we have not addressed any potential security or privacy concerns that could result from interfacing the UE with the network in this way.

Furthermore, we intend to extend the proposed architecture into a wider framework, where policy engines deployed on both the UEs *and* in the MANO layer can exchange information to not only optimize the performance for end-host applications, but to also allow information on application requirements to influence the slice manager's decisions in regards to the allocation of network resources to slices.

6 CONCLUSIONS

In this paper, we have proposed an architecture that extends the network slice concept to the UE. The main idea is to establish a communication channel between the UE and the slice manager, through which fine-grained information on network slices can be exchanged. We use this information on the UE together with the transport-services architecture to dynamically tailor the network stack to suit the requirements of individual applications. We have implemented the proposed architecture in a 5G testbed, and showcased through three different examples how such an architecture could be used to create optimized services through dynamic protocol configuration, application-to-slice mapping, and network protocol selection.

7 ACKNOWLEDGEMENTS

Partially funded by EU H2020 5GENESIS (no. 815178), and Region Värmland through DigitalWell Arena project (Dnr RV2018-678).

REFERENCES

[1] 3GPP. 2020. *Technical Specification Group Services and System Aspects; System architecture for the 5G System (5GS) (Release 15).* Technical Specification (TS)

23.501. 3rd Generation Partnership Project (3GPP). https://www.3gpp.org/ftp//Specs/archive/23_series/23.501/23501-g60.zip Version 16.6.0.

[2] 3GPP. 2021. *; Network Exposure Function Northbound APIs; Stage 3, (Release 17).* Technical Specification (TS) 29.522. 3rd Generation Partnership Project (3GPP). https://www.3gpp.org/ftp/Specs/archive/22_series/22.261/22261-h60.zip Version 17.2.0.

[3] 3GPP. 2021. *Functional architecture and information flows to support Common API Framework for 3GPP Northbound APIs; Stage 2,(Release 17).* Technical Specification (TS) 23.222. 3rd Generation Partnership Project (3GPP). https://www.3gpp.org/ftp/Specs/archive/29_series/29.522/29522-h20.zip Version 17.5.0.

[4] 3GPP. 2021. *Service Enabler Architecture Layer for Verticals (SEAL);Functional architecture and information flows; (Release 17).* Technical Specification (TS) 23.434. 3rd Generation Partnership Project (3GPP). https://www.3gpp.org/ftp/Specs/archive/23_series/23.434/23434-h20.zip Version 17.2.0.

[5] 3GPP. 2021. *Service requirements for the 5G system; Stage 1* . Technical Specification (TS) 22.61. 3rd Generation Partnership Project (3GPP). https://www.3gpp.org/ftp/Specs/archive/22_series/22.261/22261-h60.zip Version 17.6.0.

[6] 3GPP. 2021. *Study on enhancement of Radio Access Network (RAN) slicing, Rel.17* . Technical Report (TR) 38.832. 3rd Generation Partnership Project (3GPP). https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3726 Version 1.0.0.

[7] Zdravko Bozakov, Simone Mangiante, Cristian Hernandez Benet, Anna Brunstrom, Ricardo Santos, Andreas Kassler, and Donagh Buckley. 2017. A NEAT framework for enhanced end-host integration in SDN environments. In *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN).* IEEE, Berlin, Germany, 1–7. https://doi.org/10.1109/NFV-SDN.2017.8169828

[8] Maria Christopoulou, Georgios Xilouris, Athanasios Sarlas, Harilaos Koumaras, Michail-Alexandros Kourtis, and Themistoklis Anagnostopoulos. 2021. 5G Experimentation: The Experience of the Athens 5GENESIS Facility. In *Proceedings of the 2021 IFIP/IEEE International Symposium on Integrated Network Management (IM '21).* IEEE, Bordeaux, France, to–appear.

[9] Jerry Chu, Nandita Dukkipati, Yuchung Cheng, and Matt Mathis. 2013. Increasing TCP's Initial Window. RFC 6928. https://doi.org/10.17487/RFC6928

[10] Monia Ghobadi, Soheil Hassas Yeganeh, and Yashar Ganjali. 2012. Rethinking End-to-End Congestion Control in Software-Defined Networks. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks (HotNets-XI).* Association for Computing Machinery, New York, NY, USA, 61–66. https://doi.org/10.1145/2390231.2390242

[11] Karl-Johan Grinnemo, Zdravko Bozakov, Anna Brunstrom, Maria Isabel Sanchez Bueno, Thomas Dreibholz, Kristian Rikter Evensen, Gorry Fairhurst, Audun Fosselie Hansen, David Hayes, Per Hurtig, Mohammad Rajiullah, Tom Jones, David Ros, Tomasz Rozensztrauch, Michael Tüxen, and Eric Vyncke. 2017. *Deliverable D3.3 - Extended Transport System and Transparent Support of Non-NEAT Applications.* Technical Report. NEAT. http://kau.diva-portal.org/smash/record.jsf?pid=diva2%3A1273851

[12] Jochen Gruen, Michael Karl, and Thorsten Herfet. 2013. Network supported congestion avoidance in software-defined networks. In *2013 19th IEEE International Conference on Networks (ICON).* IEEE, Singapore, 1–6. https://doi.org/10.1109/ICON.2013.6781970

[13] GSM Association (GSMA). 2020. Generic Network Slice Template. https://www.gsma.com/newsroom/wp-content/uploads//NG.116-v3.0-1.pdf

[14] Ankur Jain, Andreas Terzis, Hannu Flinck, Nurit Sprecher, Swaminathan Arunachalam, Kevin Smith, Vijay Devarapalli, and Roni Bar Yanai. 2017. *Mobile Throughput Guidance Inband Signaling Protocol.* Internet-Draft draft-flinck-mobile-throughput-guidance-04. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-flinck-mobile-throughput-guidance-04 Work in Progress.

[15] Ankur Jain, Andreas Terzis, Nurit Sprecher, Swaminathan Arunachalam, Kevin Smith, and Guenter Klas. 2017. *Requirements and reference architecture for Mobile Throughput Guidance Exposure.* Internet-Draft draft-sprecher-mobile-tg-exposure-req-arch-03. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-sprecher-mobile-tg-exposure-req-arch-03 Work in Progress.

[16] Simon Jouet, Colin Perkins, and Dimitrios Pezaros. 2016. OTCP: SDN-managed congestion control for data center networks. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium.* IEEE, Istanbul, Turkey, 171–179. https://doi.org/10.1109/NOMS.2016.7502810

[17] Sebastian Kiesel, Wendy Roome, Richard Woundy, Stefano Previdi, Stanislav Shalunov, Richard Alimi, Reinaldo Penno, and Y. Richard Yang. 2014. Application-Layer Traffic Optimization (ALTO) Protocol. RFC 7285. https://doi.org/10.17487/RFC7285

[18] NEAT. 2015. A New, Evolutive API and Transport-Layer Architecture for the Internet. https://www.neat-project.org

[19] Christoph Paasch, Simone Ferlin, Ozgu Alay, and Olivier Bonaventure. 2014. Experimental Evaluation of Multipath TCP Schedulers. In *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop (CSWS '14).* Association for Computing Machinery, New York, NY, USA, 27–32. https://doi.org/10.1145/2630088.2631977

[20] Dr. Craig Partridge, Mark Allman, and Sally Floyd. 2002. Increasing TCP's Initial Window. RFC 3390. https://doi.org/10.17487/RFC3390

[21] Tommy Pauly, Brian Trammell, Anna Brunstrom, Gorry Fairhurst, Colin Perkins, Philipp S. Tiesel, and Christopher A. Wood. 2021. *An Architecture for Transport Services.* Internet-Draft draft-ietf-taps-arch-10. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-taps-arch-10 Work in Progress.

[22] Dr. Vern Paxson, Mark Allman, and W. Richard Stevens. 1999. TCP Congestion Control. RFC 2581. https://doi.org/10.17487/RFC2581

[23] George Xilouris, Themistoklis Anagnostopoulos, Thanos Sarlas, Maria Christopoulou, Harilaos Koumaras, Stavros Kolometsos, Fotini Setaki, Ioanna Mesogiti, Dimitris Tsolkas, Florian Kaltenberger, Panagiotis Matzakos, Daniele Munaretto, Fabio Giust, and Israel Koffman. 2020. *Deliverable 4.2: The Athens Platform (Release B).* Technical Report. 5GENESIS. https://5genesis.eu/wp-content/uploads/2020/02/5GENESIS_D4.2_v1.0.pdf