Towards High Performance Packet Processing for 5G

Balázs Pinczel, Dániel Géhberger, Zoltán Turányi and Bence Formanek
Ericsson Research, Hungary

***Abstract*—Network Functions Virtualization aims to implement networking functions in the cloud – even the ones working on the user plane. This is envisioned by decomposing today's monolithic devices into smaller functions and re-composing them real-time. This enables flexibility and independent innovation, but may come at a performance price. In this paper we investigate the performance aspects of composing user plane functions. We have developed a prototype for mobile service chaining to experiment with the function call based composition method provided by the Click modular router. The prototype allows flexible management of feature sets, including the support for relocating user chains using context transfer. We provide a thorough performance evaluation, draw conclusions and propose a versatile execution environment, which combines various composition and processing methods in order to achieve high performance packet processing for 5G networks.**

I. Introduction

Mobile core networks provide important contribution to the mobile communication service. They are responsible for managing connectivity sessions, handovers, idle mode and paging, security, policy and charging. These functions have been traditionally standardized to ensure interoperability between the network and the terminals, but also among networking devices. The standards govern not only functions, protocols and procedures, but also a grouping of the functions into nodes connected by standardized interfaces. The traditional approach to core networks is to implement these nodes in separate devices. This in turn, resulted in low feature velocity, since any new feature that impacts more than one node, has to be first standardized, then implemented by vendors (often as a new release of a monolithic device), and finally carefully rolled out – a process that can easily take years. This is very far from the DevOps model and the methods of innovation established in other technology domains. Any evolution of mobile core networks must significantly improve this problem. Modern IT software technologies and the cloud enable new ways to implement telecommunications equipment and represent good sources of inspiration. We adopt the argument for decomposing node functionality into smaller units. This argument is fairly common and forms one of the principles of the Network Function Virtualization (NFV) initiative [1]. The subject of this paper is real-time re-composition of network functions, focusing on the user plane. Even though some network functions are possible to implement in purpose-built silicon, in our prototype we use general-purpose CPUs, since we believe this is where rapid innovation can happen. Some of our results are also applicable to programmable NPUs.

Our concept consists of implementing core network user plane functionality in small User Plane Functions (UPFs) that can then be chained by a platform both within a physical server and between servers. The control plane of the platform is free to mix and match any set of UPF to a packet flow in any order running at any location. The platform must take mobility into account, which may result in the relocation of certain functions – a generic context transfer facility similar to the context transfer during X2 LTE handovers [2]. This concept, called Mobile Service Chaining [3] has the following abilities.

- Customization: specify which flow gets what processing
- Simplification: omit network functions not needed
- Deployment flexibility: control where functions execute
- Extensibility: easy to deploy and apply new functions

Deployment flexibility not only aids in efficient resource management, but also allows use cases, such as local breakout, which required extensive standardization before [4]. We also note that due to the extensibility of the concept radio and service functions can also be included and managed under a single platform. That is, some of the virtualized radio functions, like PDCP or transport encryption employed in LTE can be handled just as any other core network function. Also, functions that today are deployed above the (S)Gi interface can also be made part of a single (mobile) service chain tying these service functions to the same control structure making all core network information (such as user identity, policies, charging, location, etc.) available to them in a uniform manner.

User Plane Functions may come in various complexity, state, resource and security requirements. Composition must be performance efficient, so that the flexibility enabled by the system does not come at a prohibitive cost. Small, simple UPFs, such as token bucket policing, packet marking or encapsulation may be invoked all in the same thread, while more complex UPFs require more isolation or the use of a different operating system. Our ultimate goal is to address this diversity in a mobile network context.

We have developed a limited prototype of mobile service chaining to experiment with a certain kind of composition method. In particular, we have implemented a chain compositor composing UPFs inside a Click modular router [5]. We have also implemented forwarding between nodes (carrying metadata between sites) and support (lossless and reordering-free) handovers via context transfer. In Section III we describe the details of our prototyping environment establishing a baseline for performance. Then, in Section IV we describe

the prototype and report our findings about the kind of performance price one pays for flexible composability. In Section V we demonstrate handover performance, which is an important aspect in a mobile environment often overlooked in cloud-related discussions. Finally, in Section VI we conclude the paper by discussing how the composition method evaluated by our prototype fits among a wider set of composition methods. In addition, we outline our vision for a distributed packet processing execution environment for 5G.

II. Related work

A. Mobile Service Chaining preliminaries

Since our prototype follows the considerations of the Mobile Service Chaining 5G Core Network Architecture [3], we provide a brief overview here. The architecture is essentially an evolution of service chaining, with added support for mobility and a formalized set of metadata. It caters for the specifics of the mobile core environment, such as bearers, requirements on grouped flow processing (e.g., to provide aggregated maximum bit-rate policing) and the associated scalability requirements.

The Control Plane (CP) of this architecture includes the functions of today's Mobility Management Entity, Policy and Charging Rules Function and a chain controller. We focus on the realization of the user plane, however, as we also evaluate lossless handovers and chain forwarding, a simple Controller is necessary for the coordination. The architecture defines two types of functional elements for the actual packet processing in the User Plane (UP): Forwarding Elements (FE) and User Plane Functions (UPF). A FE forwards packets between its ports based on the rules, which are configured by the CP. A port is either connected to a UPF or to a network interface. The actual packet processing is done in the UPFs based on the locally stored per-user contexts. It is not expected that UPFs know any detail about the higher level topology.

Packets are sent through the system according to which service chain they need to traverse. Such information is added to the packets as tags by classifiers, which are also UPFs managed by the CP. Simple classification can be done based on header fields, packet direction (uplink or downlink) but it is also possible to re-use tags assigned by prior UPFs (like Deep Packet Inspection). The rules in the FEs may apply on the assigned tags or directly on the header fields of the packet. It is important that the assigned tags must be carried with the packets through the network. For handover support UP nodes have to be able to export and import the rules and UPF contexts related to the user handing over.

B. Other related work

Service chaining is usually referred to as a combination of OpenFlow and Virtualized Network Functions (VNF) realized in virtual machines (VMs) or Linux containers [6]. However, it is concluded in studies that the packet transfer from the OpenFlow switch to the VMs represents a significant performance overhead [7]. This is especially troublesome with simple VNFs where the actual processing time can be comparable with this overhead. While the overhead is very small for Linux containers [8], for a granular service chaining architecture – with a lot of small VNFs – the network stack of the Linux kernel itself can cause bottlenecks. The overhead of virtualization has been also addressed with configurations where the VNFs are simple processes on the host computer [9], [10]. Nevertheless, the inter-VNF communication is still not lightweight enough, since it is done via Ethernet softswitches, and requires packet copies along the way.

There are relatively few performance evaluations for service chaining. Cerrato et al. [9], [10] present results with identical network functions, which calculate a hash on the first bytes of the packets. This simplistic approach is also followed by papers addressing the performance capabilities of OpenFlow switches. Being the most popular, the literature mostly focuses on the Open vSwitch. For the measurements, the switch is usually operated with simple forwarding rules where higher performance is easier to achieve [7], [11]. After the basic one-rule cases, some papers also show that the number of rules affects the performance significantly. There has been impressive improvements during the last years [12], [13].

The Click modular router [5] provides a flexible packet processing environment at the cost of some performance overhead. The overhead is also measurable with high I/O rate extensions [14]. Click can be used to construct middleboxes and – besides general feasibility – Dobrescu et al. [15] show that it is possible to predict the performance with the CPU cache miss rates. Martins et al. [16] also present results for several middleboxes, where the complexity varies from simple forwarding to an intrusion detection system.

Regarding cloud based mobility, it is possible to perform live migration of virtual machines and Linux containers [17]. However, this is not a solution for handovers in 5G, because only the context of a single user needs to be migrated, not the whole user plane node which possibly handles thousands of users. Wubin et al. [18] describe a solution which provides high availability for containers by exporting the internal states, which is a significant step towards our needs, but it is still missing parts to support the handover of a single user.

We can conclude that in most of the cases packet processing and service chaining is evaluated involving simple rules or functions. The papers that evaluate more complex middleboxes do not consider the composition of these neither chained with an OpenFlow switch nor inside the execution environment itself. Furthermore, to the best of our knowledge there are no public results available about VNFs, which support per-user context migration.

III. Prototyping environment

We have selected the Click modular router environment to investigate the performance of packet processing composition. In this section we briefly overview Click and select one of its environments to build our prototype on. We seek to provide the prototype directly as a virtual machine or a Linux container to ease the cloud based deployment since this is is expected for 5G systems.

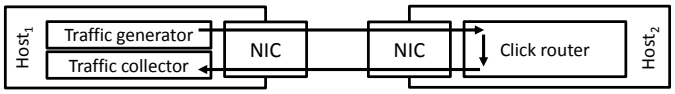


Fig. 1. Measurement setup.

The Click modular router enables dynamic composition of software routers [5]. A router is assembled in run-time – based on a configuration file – from simple packet processing elements. Elements can have multiple input and output ports, and as the internal logic can be written in pure C++, the packet processing is not limited to the restricted set of actions as in the case of OpenFlow switches [19].

Packets traverse the chain of elements as sequences of function calls initiated by the active elements of the actual configuration (ingress interfaces or elements with buffered packets, ready to send). Click has multithread support, which enables the scheduling of active elements to different cores in parallel – in the prototype, though, we focus on the performance of one CPU core, using a single Click thread.

The Click framework supports three different environments, namely, it can run in Linux user space, kernel space and over the Xen hypervisor as a library operating system called ClickOS [16]. More recent versions of Click implement support for netmap – a fast packet I/O solution available as a kernel module [20] – in user space and for ClickOS. As a result, a user space deployment can outperform the kernel based counterparts [14]. It is also possible to have netmap access in Docker containers with mounting the related device file and granting privileged access for the container. The netmap framework also contains a software switch called VALE, which provides high packet rates between netmap accelerated interfaces [21].

Before discussing the performance of the software environments, we provide a brief overview about the hardware components used. For all of our measurements we used Linux (Ubuntu 14.04 LTS) servers, each containing 3.5 GHz 6-core Intel Xeon E5-1650 CPUs and having 32 GiB RAM. For user plane traffic we used dedicated 10-Gigabit point-to-point links between the machines with Intel X540 network interface cards (NICs). The traffic on these links was directly handled by either a Click instance or a traffic generator/collector as it can be seen in Figure 1, without being processed by the OS network stack. An exception was the ClickOS where a VALE switch also had to be used. Control plane traffic used a separate 1-Gigabit network. The Click version used for the measurements is a recent (April, 2015) commit from the repository [22], as is netmap [23]. Our ClickOS-based evaluation was done on Xen 4.4.0. For traffic generation we use the Pkt-gen tool from the netmap framework to achieve high packet per second rates.

We tested the multiple different software environments of Click to select the best for our prototype. In these cases the Click router was used with a simple configuration, which only rewrote the Ethernet addresses, thus realizing a simple low

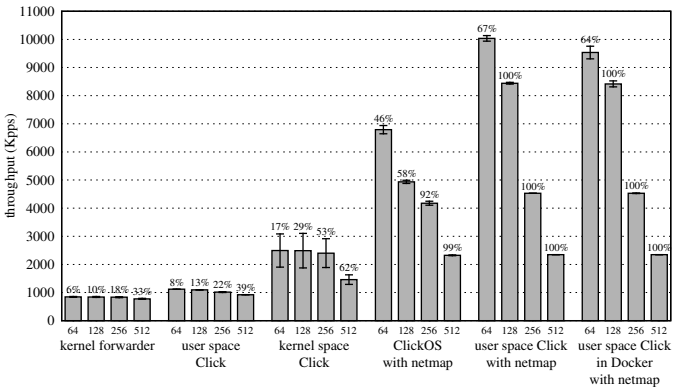


Fig. 2. Throughput and link utilization in different environments.

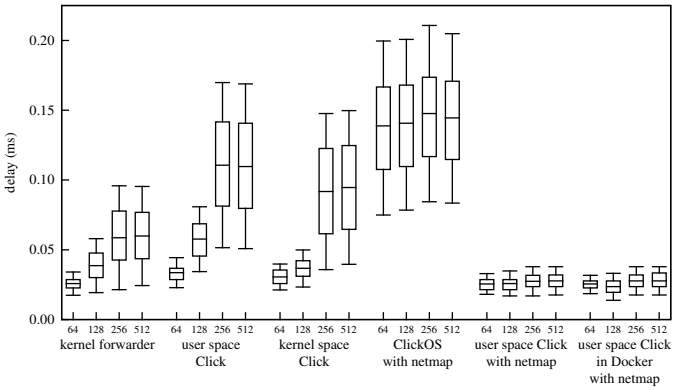


Fig. 3. Delay of different environments.

level forwarder. Figure 2 shows the averaged values of the throughput results extended with the standard deviation and the link utilization percentage on the top of the bars. Figure 3 shows the delay for the different environments as boxplots where the top and bottom 5 percentiles are defined as outliers. We used various packet sizes from 64 bytes up to 512 bytes in all the cases.

In order to provide an easily reproducible reference which can be used to estimate the peak performance of other machines, we measured packet forwarding in the Linux kernel as well. This method provides even lower throughput than the simple user space Click configuration. This is because in Click we only rewrite the Ethernet addresses without going through the Linux protocol stack. As the figure shows, the kernel space based Click router improves the throughput but adds variability.

The last three cases are accelerated with the netmap framework, which – as expected – improves the performance significantly. While ClickOS shows a decent improvement, we reached the best results with the user space Click router accelerated with the netmap framework. Here the processing rate goes up until 10 Mpps for the smallest packets, and above 128 bytes the 10 Gigabit link gets saturated. As we noted before, it is possible to place this software combination into a Docker container, which is shown as the last environment on

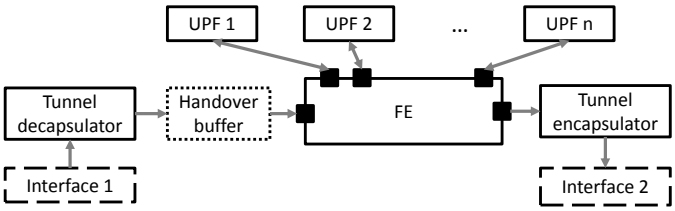


Fig. 4. Mobile Service Chaining 5G Core Network Architecture.

the figure. The container slightly limits the peak performance for small packets and adds some variation, but the link still gets saturated quickly as the packet size increases. Based on the above results, we selected the Click router running in a Docker container because despite the slightly lower performance, this setup seamlessly fits into the envisioned cloud based deployment of 5G systems.

IV. The 5G core network prototype

In our prototype we mapped the proposed logical 5G Core Network Architecture to nodes on physical computers each running a Click router instance. Besides a more detailed description, we provide a performance evaluation in this section.

A. Implementation details

The FEs and the UPFs are realized as Click elements, and the UPFs are connected to the ports of the single FE as shown in Figure 4. To carry the packet tags between Click instances, we implemented a simple tunneling protocol, which uses UDP encapsulation and carries the tag/value pairs besides payload. When an encapsulated packet arrives to a Click node, the extra headers are removed and the tags are placed to the annotation area defined by the Click for each packet, which provides efficient tag handling. The implementation is capable of per-user handovers, utilizing a special buffering element in each node. This element does nothing for users not in a handover and will be further described in Section V.

Both the input and the output ports of the UPFs are connected to the FE, which applies its rules for all the incoming packets. The actual rules can be defined with a rule language from the Control Plane where currently it is possible to match against UPF names and packet tags. Obviously, the performance of the rule matching determines the performance of the system, especially for a large rule set. In order to address this, we implemented a hash table based caching component to speed up the lookup.

For the evaluation, we implemented the following fully functional, per-user handover capable UPFs.

a) Counter: This UPF maintains a separate packet counter for each user, and writes the current value into the payload at an offset, which is chosen so as to avoid overwriting any previously placed counter values. While this is a relatively simple function, it still requires payload parsing, modification and the recalculation of the UDP checksum.

b) Marker: The marker element uses token buckets to check whether the traffic conforms to a per-user defined bandwidth limit. Non-conformant packets are marked with a tag to be dropped later.

c) Header compressor (IPHC): This element implements IP/UDP compression and decompression as specified in RFC2507 [24]. It is able to handle multiple users with multiple flows per user, assigning context identifiers automatically.

B. Evaluation of the prototype

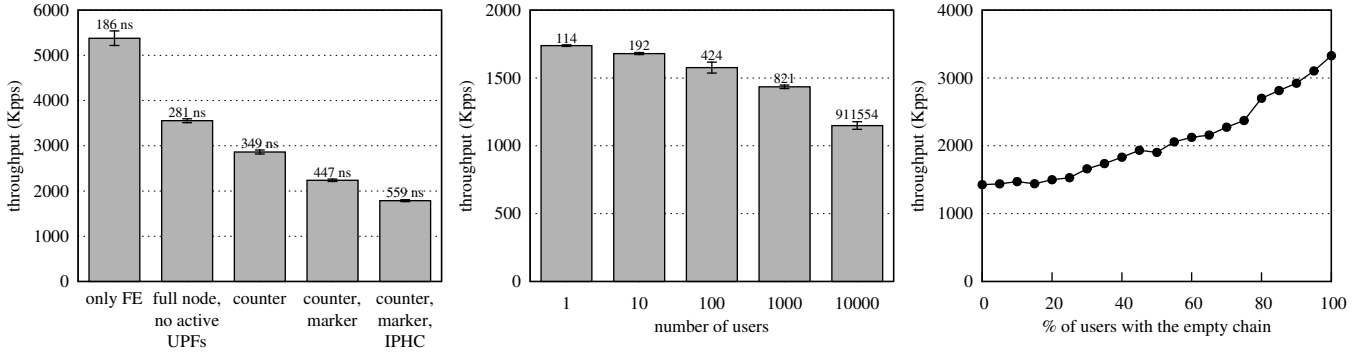
We evaluate the performance of the prototype running on a single node, assuming that it is an intermediate element of a longer chain, and that the incoming packets are already classified and tagged. To enable this, we modified the Pktgen tool to be able to generate tunneled traffic, tagged with, e.g., user-ID tag. As these packets are larger than 64 bytes, we used 128 byte packet size for the rest of the paper.

1) UPF Overhead: In each step of the following experiment, the configuration is extended with additional components. Figure 5a shows the average packet rate with the standard deviation. We also displayed the nominal time required to process each packet (simply taking the inverse of the average packet rate). The first setup uses no encapsulation, just one FE with one rule to forward. Compared to the baseline measurements of the previous section, this configuration also removes the Ethernet header and validates the IP header before entering the FE. After the rule matching, the IP addresses are rewritten and a new Ethernet header is placed to the packet.

The second setup includes tunneling and handover elements as shown in Figure 4. The encapsulation step also involves a configurable next hop selection based on tags. In the rest of the cases the actual UPFs are added to the chain one by one, creating more and more complex configurations. As these functions handle contexts, parse, validate and copy fields for each incoming packet, it is expected that these CPU and memory intensive tasks affect the performance. Still, as it can be seen on the figure, the throughput is stable and remains in the millions of packets per second range. The per-packet processing time shows that each component adds around 70–110 ns overhead, but due to the inverse proportionality, this effect appears to influence higher packet rates more.

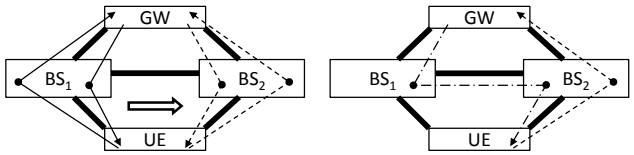
2) Impact of additional users: Current custom-made core network nodes handle 10–100 k users. Per component load is expected to be lower in cloud-based deployments of future 5G systems, but still in the thousands. The following measurement addresses this by taking the most advanced configuration with 3 UPFs, and increasing the number of users from 1 to 10 k. The average packet rates with the standard deviation and the recorded L3 cache load misses are shown in Figure 5b. Even though the hash table based lookups we used are near constant time, the increase in the number of users causes some performance degradation partly due to the L3 cache misses.

3) Handling multiple chains: Flexibility is one of the strengths of the proposed 5G architecture. The last throughput related evaluation addresses this, by taking the previous setup one more step forward. For this measurement we used 1000



(a) Throughput of different configurations (b) Throughput with multiple users (c) Throughput with a mix of chains

Fig. 5. Prototype performance with various workloads.



(a) Packet paths (b) Intermediate step

Fig. 6. Handover from BS_1 to BS_2.

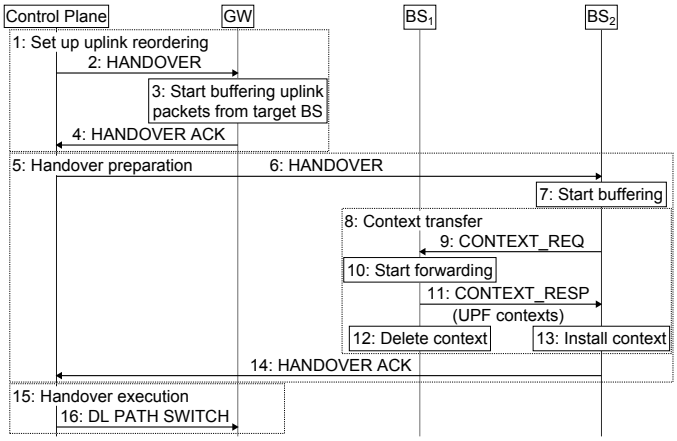


Fig. 7. Interaction of core network entities during a handover.

users and defined 2 chains, having one with three UPFs and one without any. In terms of FE rules, this requires only one extra rule, which defines the shortcut for packets arriving on the less complex chain. We generated the traffic with a set ratio of users taking the more and the less complex chains. The leftmost point of Figure 5c is the case when all the traffic goes through the more complex chain, and with each further point 5% of the users are moved to the empty one, which scales up the performance.

This demonstrates that while the prototype – as a general-purpose packet processing environment – supports more complex feature sets as well, the aggregated performance of serving multiple chains of various complexity remains predictable. The results also show that this kind of flexibility does not sacrifice the scalability: the same compute resources can be saturated either with more complex services at lower throughput, or with less complex ones at higher throughput.

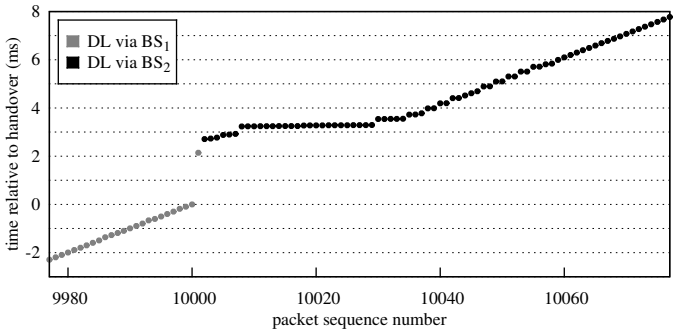
V. Handover

As a core requirement for mobile service chaining solutions, the proposed architecture supports transfer of a user's UPF contexts from a source node to a target node. To avoid packet reordering, the prototype uses buffering in nodes where traffic arrives on the new (post-handover) path until all the packets arriving on the old path are processed.

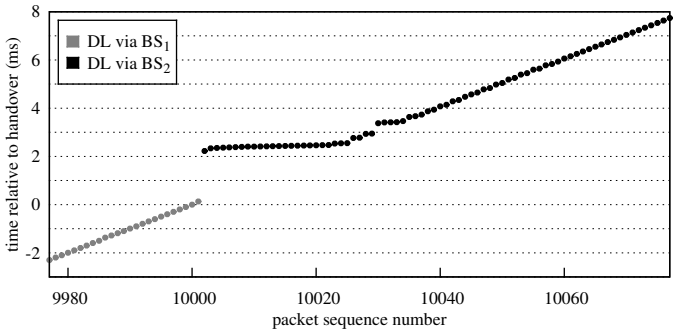
To describe the process of handing over, we consider an example scenario depicted in Figure 6a, where a gateway node (GW) has two-way communication with an emulated UE through either one of the two base stations ($BS_{1,2}$), with the possibility of switching between the paths when instructed to do so by the Control Plane. Figure 7 shows the procedure of a handover from BS_1 to BS_2. There is a buffer in the GW for the uplink packets arriving from BS_2, as well as one in BS_2 for packets already arriving on the new path. Also in BS_2, there is a separate buffer for the packets arriving on the temporary path shown in Figure 6b. This buffer stores incoming packets until the migration of contexts completes. When all the preparatory steps are completed, the path of downlink packets is switched in the GW, and after some time – since no new packets take the old path – the buffered packets can be consumed, until eventually the buffers get empty, at which point the node switches to processing incoming packets directly.

In the prototype the bulk of the above functionality is realized in a special element chained before the FE, which manages the lifecycle of per-user handover buffers, and handles packet forwarding (see Figure 4). The rest of the functionality (e.g., control-related communication or context transfer) is spread among the FE, the UPFs, the daemon of the user plane nodes and the Control Plane.

To test handovers the UE emulated the behavior of a radio device by replying to the same base station it received the packet from – the reply being the same 128 byte packet with only one modified tag. Traffic was generated in the GW at the rate of 10 kpps, and was captured both in the UE, and upon



(a) Captured at the GW



(b) Captured at the UE

Fig. 8. Packet arrival times during a handover.

returning to the GW. To examine the effect of a handover procedure, we look at the disturbance in packet arrival times. Figure 8 shows these for a single handover selected from our measurements (all other measurements show highly similar characteristics).

In normal conditions, the packet arrival times form a diagonal line, the slope of which is determined by the rate of the packets. When a handover is in progress, packets cease to arrive for a short period (approximately 2 ms), caused by the buffering during, which the context is migrated. When packet processing is resumed, the nodes receive larger batches of packets at a time, because the buffers can be processed faster than the packet arrival rate. This eventually leads to the buffers being empty, from which point the arrival times follow their usual linear pattern. The pattern is already very close to linear 3.5 ms after the beginning of the handover, and is fully restored after 6 ms. During this period the curve approaches its normal slope in gradually shortening steps, which is likely a scheduling artifact.

The figures also show a single packet that arrives to the UE through BS_1, during the exact time of the context transfer. The UE sends the reply still to BS_1, where the forwarding causes it to be sent through BS_2, before arriving to the GW suffering the delay of the context transfer. This is normal – even if rare – behavior, and the continuity in packet sequence numbers shows that even in this case the handover is lossless and reordering-free.

VI. Discussion

In the previous sections we have investigated the performance aspects of a certain composition model. This model is characterized by the following attributes

- individual processing functions (UPFs) are invoked through function calls;
- a (cached) rule database is consulted before selecting the next UPF to execute;
- UPFs are free to process, duplicate, buffer, drop packets or even generate new packets; and
- packets can carry metadata between the UPFs.

The main advantage of this composition model is its flexibility. It allows single- or multithreaded operations and has its internal scheduler to initiate elements that have work to do. It has some limitations, however. Since all Click elements run in the same process, very little can be done to realize isolation – either for resource control or security. Another issue is that in order to manage relatively high performance and in order to run in a variety of situations, Click represents a confined programming environment. There are a lot of rules on how to handle packets, allocate memory or do I/O. As a result third party libraries in general, cannot be used, not even the STL. This makes the development of complex UPFs, such as a full DPI suite or transparent HTTP proxies quite cumbersome. These would best run in a separate container or VM, since we think similar limitations would appear in any system using function calls as composition method.

At the other end of the scale, even the overhead represented by function calls and rule lookups may be significant for very small UPFs. For example, flow counters, bandwidth policers, simple encapsulation modules (and in general most actions of an OpenFlow switch) complete in very few cycles. For these an even more lightweight composition model is better. It is simply more efficient to copy their code one after another in as many combinations as needed [25].

Since a versatile programmable packet processing execution environment may contain a mix of small and large UPFs it is best, if several composition methods are available to the programmers. The figure below summarizes our proposal for composition methods needed. The just-in-time (JIT) linker method dynamically copies small UPFs one after another. This method has been shown to achieve the fastest software switch for simple actions [25]. The function call method has been the subject of our investigations in this paper and is suitable for moderately sized, trusted code. The separate container/VM model passes packets between different containers or VMs on the same server, and enables the use of any third party library or even OS (in the case of VMs) at the expense of passing the packets between contexts [7]. Finally, the last method, chain forwarding, enables the functions to reside on different hosts and uses encapsulation to carry chain ID and other metadata on the wire [26].

The first two methods may be used to support a run-to-completion mode of operation, whereas the second and third can split the processing of a single packet among multiple

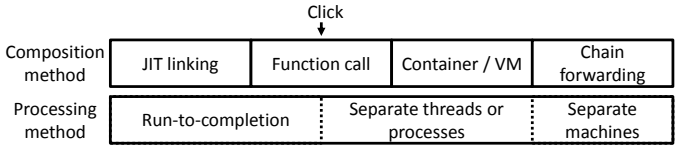


Fig. 9. Composition methods and the corresponding processing methods.

threads or processes (latter of which can even reside in different containers/VMs). Presumably a UPF needs to be developed with its composition method in mind. E.g., for JIT linked components small fragments of code are required with linker metadata. Function call components can be provided as shared libraries, while the rest as apps or VM images.

In general, it is best if the controller of this execution environment has no knowledge of the type of each UPF. This leaves the implementation and optimization of the user plane to the developer of the UPFs and the execution environment. It is especially good if the distinction is not hard coded in a standard. For example, an OpenFlow softswitch combined with VMs could implement a wide variety of packet processing applications, but it creates an artificial distinctions between things that can be done in the OF switch and things that must be done in a separate VM. Even if a particular function is small enough to fit into the programming environment of the softswitch, it must be placed in a VM, if the OpenFlow standard does not include it.

And finally, we note that the presented user plane model, that is, individual UPFs chained together inside and between machines is fairly future proof. Controllers have an abstract view of the user plane, can mix and match standard or de-facto components. The ability to add components regardless of their size enables continuous feature innovation, while the abstraction itself enables continuous performance innovation (hidden from controllers). Due to the range of composition models employed high performance can be provided even for small functions without the controller to understand the details of the user plane implementation. This level of abstraction (composable primitives) can fully utilize hardware innovation without requiring the controller to understand the intricacies of the underlying user plane node.

REFERENCES

[1] ETSI NFV Industry Specification Group. Network Functions Virtualisation – White Paper. Visited Apr 2015. [Online]. Available: https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper3.pdf

[2] "Evolved Universal Terrestrial Radio Access Network (E-UTRAN); X2 general aspects and principles," (3GPP TS 36.420).

[3] D. Roeland and Z. Fu, "Mobile Service Chaining 5G Core Network Architecture," *Submitted to IEEE Commun. Mag.*, 2015.

[4] "Local IP Access and Selected IP Traffic Offload (LIPA-SIPTO)," (3GPP TR 23.829).

[5] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click Modular Router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[6] J. Blendin, J. Ruckert, N. Leymann, G. Schyguda, and D. Hausheer, "Position Paper: Software-Defined Network Service Chaining," in *2014 Third European Workshop on Software Defined Networks (EWSDN)*, Sept 2014, pp. 109–114.

[7] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, "Performance characteristics of virtual switching," in *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, Oct 2014, pp. 120–125.

[8] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. Lightweight Virtualization: A Performance Comparison," in *2015 IEEE International Conference on Cloud Engineering (IC2E)*, March 2015, pp. 386–393.

[9] I. Cerrato, G. Marchetto, F. Risso, R. Sisto, and M. Virgilio, "An efficient data exchange algorithm for chained network functions," in *2014 IEEE 15th International Conference on High Performance Switching and Routing (HPSR)*, July 2014, pp. 98–105.

[10] I. Cerrato, M. Annarumma, and F. Risso, "Supporting Fine-Grained Network Functions through Intel DPDK," in *2014 Third European Workshop on Software Defined Networks (EWSDN)*, Sept 2014, pp. 1–6.

[11] A. Mian, A. Mamoon, R. Khan, and A. Anjum, "Effects of Virtualization on Network and Processor Performance Using Open vSwitch and Xen Server," in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC)*, Dec 2014, pp. 762–767.

[12] G. Pongrácz, L. Molnár, and Z. Kis, "Removing Roadblocks from SDN: OpenFlow Software Switch Performance on Intel DPDK," in *2013 Second European Workshop on Software Defined Networks (EWSDN)*, Oct 2013, pp. 62–67.

[13] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, "Assessing Soft- and Hardware Bottlenecks in PC-based Packet Forwarding Systems," in *ICN 2015: The Fourteenth International Conference on Networks*, Apr 2015, pp. 78–83.

[14] L. Rizzo, M. Carbone, and G. Catalli, "Transparent acceleration of software packet forwarding using netmap," in *INFOCOM, 2012 Proceedings IEEE*, Mar. 2012, pp. 2471–2479.

[15] M. Dobrescu, K. Argyraki, and S. Ratnasamy, "Toward Predictable Performance in Software Packet-Processing Platforms," in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. San Jose, CA: USENIX, 2012, pp. 141–154.

[16] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the Art of Network Function Virtualization," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, Apr. 2014, pp. 459–473.

[17] W. Li and A. Kanso, "Comparing Containers versus Virtual Machines for Achieving High Availability," in *2015 IEEE International Conference on Cloud Engineering (IC2E)*, March 2015, pp. 353–358.

[18] W. Li, A. Kanso, and A. Gherbi, "Leveraging Linux Containers to Achieve High Availability for Cloud Services," in *2015 IEEE International Conference on Cloud Engineering (IC2E)*, March 2015, pp. 76–83.

[19] O. N. Foundation, "OpenFlow Switch Specification Version 1.5.0," Dec 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf

[20] L. Rizzo, "netmap: a novel framework for fast packet I/O," in *Proceedings of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*. Bellevue, WA: USENIX Association, Aug. 2012, pp. 101–112.

[21] L. Rizzo and G. Lettieri, "VALE, a Switched Ethernet for Virtual Machines," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. New York, NY, USA: ACM, 2012, pp. 61–72.

[22] Click modular router source code repository. Visited Apr 2015. [Online]. Available: https://github.com/kohler/click/

[23] Netmap source code repository. Visited Apr 2015. [Online]. Available: https://code.google.com/p/netmap/

[24] M. Degermark, B. Nordgren, and S. Pink. IP Header Compression. Visited Apr 2015. [Online]. Available: https://tools.ietf.org/html/rfc2507

[25] "Universal Node Initial Benchmarking Documentation," D5.4 deliverable of EU FP7 project UNIFY (will be publicly available at https://www.fp7-unify.eu/index.php/results.html#Deliverables).

[26] P. Quinn and T. Nadeau. Problem Statement for Service Function Chaining. Visited Apr 2015. [Online]. Available: https://tools.ietf.org/html/rfc7498