# Reducing Latency in Virtual Machines enabling Tactile Internet for Human Machine Co-working

Zuo Xiang*, Frank Gabriel*, Elena Urbano*, Giang T. Nguyen*,
Martin Reisslein†, and Frank H.P. Fitzek*
*Deutsche Telekom Chair of Communication Networks
Technische Universität Dresden, 01062 Dresden, Germany
Email: {firstname}.{lastname}@tu-dresden.de
†School of Electrical, Computer, and Energy Engineering
Arizona State University, Tempe, AZ 85287-5706, USA
Email: reisslein@asu.edu

**Abstract**

Software Defined Networking (SDN) and Network Function Virtualization (NFV) have been proposed as a critical ingredient to fulfil the low latency requirements of the Tactile Internet. But while virtual network functions allow greater flexibility compared to hardware based solutions, the abstraction also introduces additional delay in the packet processing. In this paper, we investigate the practical feasibility of Network Function Virtualization with respect to the latency requirements of the Tactile Internet. We propose CALVIN , an approach for distributed service function chaining with strict low-latency requirements. With this architecture, we evaluate the latency overhead of different VNF implementations. Our results show, that it is possible to achieve the requirements of the Tactile Internet with current NFV implementations, even when the VNFs are distributed on multiple machines. Additionally, we evaluate the latency performance of network functions with high computational requirements. Further, our results show potential directions for latency optimizations.

# Reducing Latency in Virtual Machines enabling Tactile Internet for Human Machine Co-working

## I. Introduction and Motivation

In order to enable the tactile Internet for human-machine co-working, low latency communication is the central requirement. Both, humans and machines require latencies below one millisecond for a wide range of co-working scenarios. For instance, in order to let humans operate in a virtual world or to interact with robots or other machines the latencies for visual, audio, or tactile multi-sensoric feedback should be below 15 ms, 3 ms, or 1 ms, respectively [1]. Every machine based on control loops also requires low latencies in order to work efficiently or to operate in a stable manner [2], [3].

As a concrete example, consider a classical inverted pendulum whose controller is placed in the cloud. Closing the control loop through a communication network will likely introduce some delays and packet losses. Figure 1 shows the influence of the communication link delay on the stability of the control unit. For long delays (30 ms in the figure), the system becomes unstable, and the pendulum will never reach stability in the inverted position. In the case of shorter delays (25 ms), the system takes some time to achieve stability, which could mean lack of quality of service, and may affect other systems if the pendulum is part of a more complex environment with interconnected systems, or multiple pendulums coexisting in the same physical space.

The 5G communication standard for automation in vertical domains [4] defines the allowed latency requirement of one milisecond for an end-to-end communication. Based on this standard, suppose individual 5G communication network delay budget components as illustrated in Figure 2. Figure 2 assumes that 0.4 ms is consumed in the embedded systems and the wireless link on both ends. According to this, times will split into 0.1 ms for latency in communication and 0.1 ms for the embedded platform to evaluate the information and execute accordingly.

This leaves 0.6 ms for the wired domain. The wired domain has two main delay components. One delay component is the basic communication over fiber where we are bound to the speed of light (3.34 $\mu$s per kilometer) and the physical characteristics of the fiber itself (66%–90%). The second delay component is based on the communication nodes. In conventional "store and forward" communication networks, these communication nodes are routers or switches. But in upcoming future communication, there is a paradigm shift from "store and forward" to "compute and forward". Now the communication nodes can process and manipulate the incoming data. The idea of "compute and forward" is realized by new technologies, such as software defined networks (SDN) [5], network function virtualization (NFV) [6]–[10], and service function chain (SFC) [11]–[25] standardized by the IETF/IRTF.

These novel technologies enable the concept of the mobile edge cloud (MEC) [17], [26]–[31], which allows for local processing of data, which in turn will reduce the latency on the pure communication path. Assuming a maximum of 25 km between sensor and MEC as well as MEC and actuator, would incur 0.25 ms for the communication and leave 0.35 ms for the virtualized communication environment.

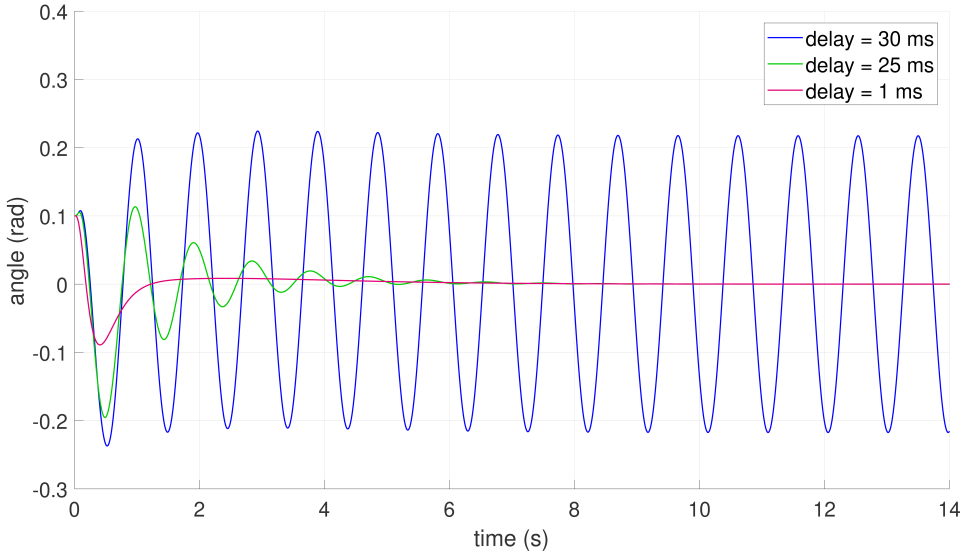



Fig. 1: Angle of an inverted pendulum trying to reach stability for different delays in the one-way communication link (30 ms, 25 ms, and 1 ms)

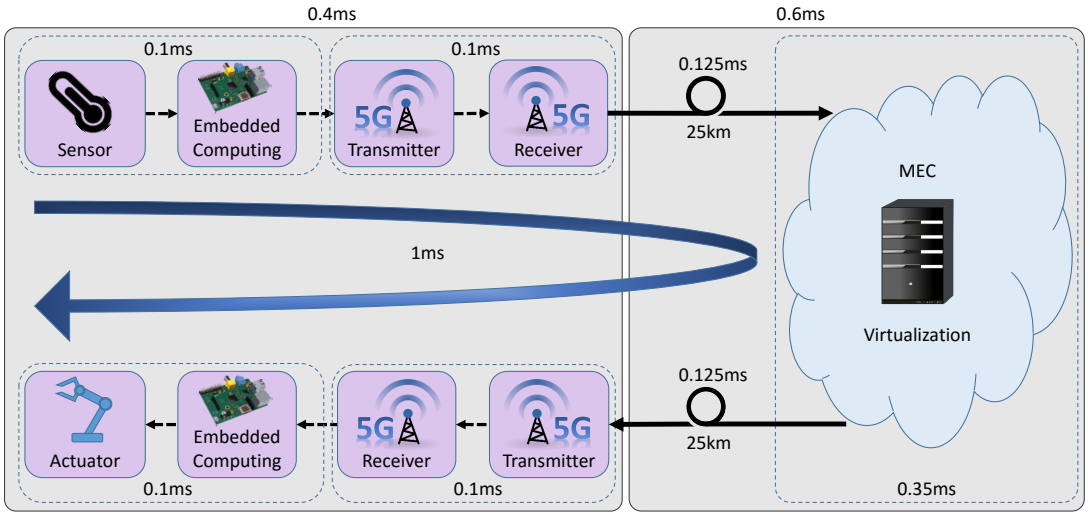



Fig. 2: Latency budget for sensor-to-machine control loop

Nowadays, the virtual switches are quite fast [32]–[34], but the virtual machines, specifically the packet IO and processing operations inside the VM, are slow. With the centralized approach proposed in [35], we see more than 2 ms inside the virtual environment for a single virtual machine running the elementary forwarding function, which is clearly above the 0.35 ms delay budget. In this paper, Chain bAsed Low latency VNF ImplemeNtation (CALVIN)

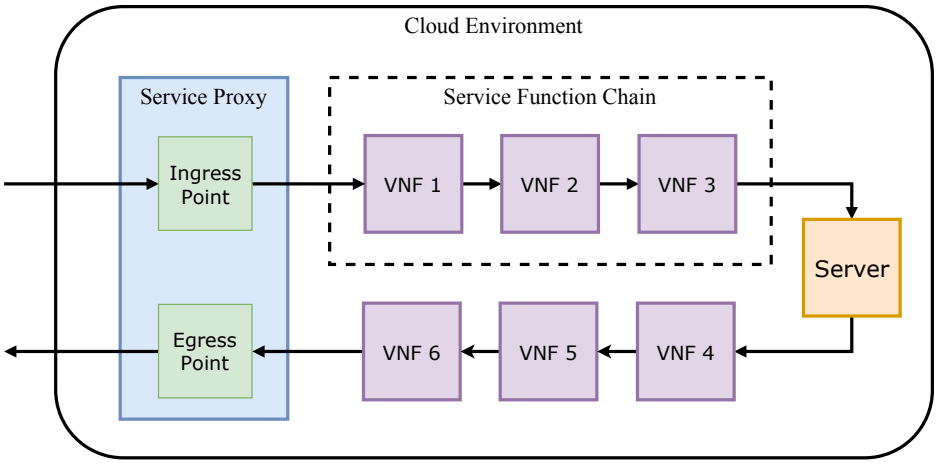



Fig. 3: Illustration of service loop in a cloud environment: Packets traverse an ingress service functions chain (SFC) consisting of an ordered sequence of virtual network functions (VNFs) to reach the server for cloud processing.

is proposed to achieve the strict latency requirement of 5G applications. Our evaluation measurements performed on practical cloud platform demonstrate that we can reduce the fundamental latency cost inside the virtualized communication environment to the order of 0.32 ms, which allows us for additional processing of data by using network coding and encryption technologies.

## II. Background and Related Work

### A. Background

The typical service loop inside the virtualization part of Fig. 2 is presented in Fig. 3. Each packet is received by the cloud through the ingress point of a service proxy, processed by a chain of virtual network functions (VNFs), i.e., by an ingress service function chain (SFC), transmitted to the requested server, processed by the server, and leave the cloud via the egress point of the service proxy (after optional egress SFC processing). The ingress and egress points are endpoints or instances that are exposed to the external network (outside of the cloud network, which is commonly a dedicated private network). For several reasons, including security considerations, not all internal components are exposed to the external network. Therefore, ingress and egress points are required for clients in the external network to access the cloud services. As illustrated in Fig. 3, a service chain consists of an ordered set of virtualized network functions (VNFs) that operate typically as a pipeline.

The virtualized service loop of Fig. 3 is typically implemented on a cloud computing infrastructure platform. A cloud computing infrastructure platform provides flexible management control over underlying physical computing resources, such as servers, networking facilities, and storage. All components of the virtualized service loop are implemented and run in the virtual machines (VMs) that are orchestrated by the cloud computing platform.

Fig 4 illustrates a typical cloud computing infrastructure scenario where multiple VMs are connected with the virtualized networking overlay. In order to enable multi-tenant networking with configurable networking resources and isolation, a virtualized networking overlay needs to be built on top of the physical networking infrastructure.

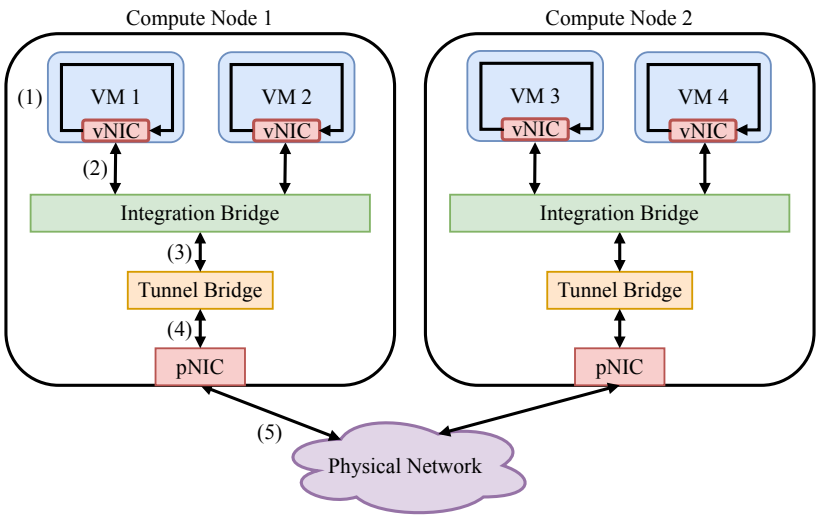



Fig. 4: Illustration of typical virtual network connection between two compute nodes, where by each compute node hosts multiple virtual machines (VMs). Each VM connects via a virtual network interface controller (vNIC) to the bridges and onwards via a physical NIC to the physical network.

For example, VM1 and VM3 can be allocated in the same broadcast domain in a virtual network (or named tenant network) even though they are hosted on different physical nodes that are connected by layer three routing entities.

In order to provide a virtual overlay network on top of the underlying heterogeneous physical network, two software bridges (or virtual switches) are used to connect the virtual network interfaces (vNIC) with the physical network interface (pNIC). The integration bridge connects all VMs running on the same physical node. These VMs can belong to different virtual tenant networks even if they are on the same physical node. The tunnel bridge is used to encapsulate and transfer tenant network data through a tunneling protocol(e.g., GRE or VXLAN).

The aforementioned networking components introduce different types of latency:

- Between VM and integration bridge (1, 2): This latency component includes mainly two parts:
  - Process packets inside VM (1): This latency component is the focus of this study. With the performance improvements of virtual bridges and interfaces, this latency component (1) becomes a bottleneck for low latency applications. The existing state of art approaches introduce latencies in range of several milliseconds. Our proposed approach significantly reduces this latency component (1) down to around 0.32 ms
  - Transfer packets between integration bridge and VM through vNIC (2): This latency depends on the technology used for the vNIC. By using Open vSwitch DPDK [36], the latency can be significantly reduced down to microseconds.
- Between integration bridge and the pNIC (3, 4): This latency component depends on the employed virtual bridges as well as the physical resource management and orchestration (or MANO) platform. Commonly cloud platform setups employ OVS-DPDK and OpenStack (which we also employ in our testbed).
- Between pNICs (5): This latency component depends on the physical network technologies.

**To be solved Problem:** How to implement VNFs such that the overall latency of the service loop is minimal.

- How to map VNFs to the VMs: 1) Pack several VNFs in a single VM (centralized approach) 2) Distributed VNFs over several VMs (chain-based approach)
- Where to implement the VNF: 1) In the kernel space or user space. 2) What are the pros and cons 3) What are the overhead of implementing VNFs in the user space in terms of latency. 4) What are the drawbacks of both scenarios and how to combine them.
- How to connect multiple VNFs.

### *B. Related Work*

This section surveys existing related frameworks **both in kernel and user space** and distinguishes our approach from the existing frameworks. Our approach is not based on full kernel-bypassing; instead, we utilize complementary strengths of both kernel and user space technologies to reduce the latency.

*1) Kernel Space:* Recent kernel space approaches have typically been based on the eXpress Datapath (XDP) framework [37]–[39] which builds on the extended Berkeley Packet Filter (eBPF). Since the XDP framework is relative new (it became available with the Linux kernel versions after 4.1X), few studies have been conducted with XDP.

Miano et al. [40] conducted quantitative characterizations of a range of eBPF aspects. Miano et al. documented several obstacles that are encountered with using eBPF to build complex network services. Due to these limitations we do not implement all functions in the kernel space in our approach. Our own preliminary evaluations of XDP, which are not included here due to space constraints, indicated that XDP achieves low latencies for elementary packet processing. However, due to main XDP limitations (e.g., limited number of instructions, not Turing complete since loops are not allowed) we do not implement advanced packet processing in the kernel space in our approach. Nevertheless, XDP has several benefits over kernel-bypassing technologies. XDP does not require large pages(The smallest unit or block for memory management in the operating system), nor dedicated CPUs. Also, XDP does not replace the kernel TCP/IP stack; rather, XDP works in concert with the kernel TCP/IP stack along with all benefits of eBPF. Moreover, XDP avoids the need to define a new security model by utilizing the security model implemented in the Linux kernel.

In-kernel processing with XDP based on the eBPF with a focus on network virtualization has recently been examined in the InKeV study [41]. InKeV employs XDP based on the eBPF, which can quickly execute simple functions; however, the implementation of advanced functions is very challenging. The InKeV latency measurements were performed on a single machine and thus do not capture the latencies incurred when traversing a physical network to complete a service chain consisting of physically distributed VMs. In contrast, we consider physically distributed VMs in our evaluations. InKeV has been compared with the OpenStack neutron [42] networking infrastructure, which we also employ in our evaluations. Thus, InKeV could be a good competitor for underlying virtual switches (OVSs) used by OpenStack (Part (3)-(5)).

But we focus on the latency introduced by the VM (Part (1)). So we focus on different latency parts.

For completeness, we note that relatively simple networking functions relating to security and virtual switching in the kernel space have recently been studied in [43]–[46].

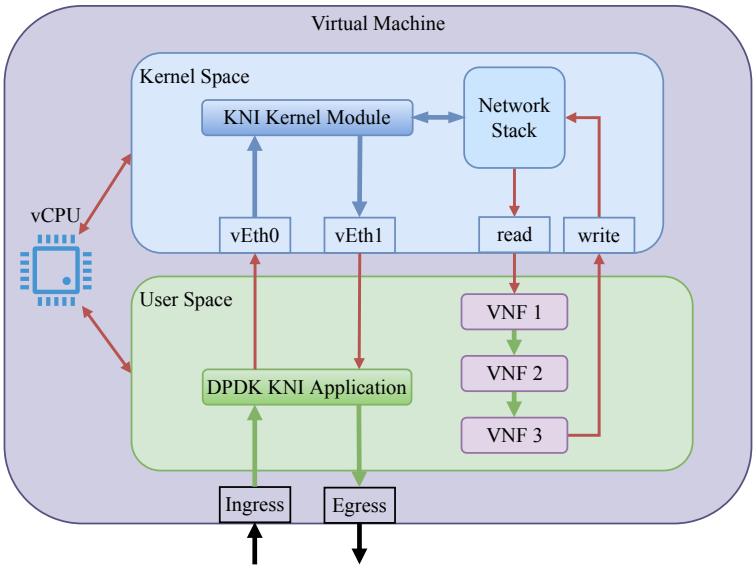



Fig. 5: Illustration of centralized combined kernel and user space approach [35].

*2) User Space:* User space frameworks have been studied more frequently than kernel space frameworks in the literature [47], [48]. However, most user space framework evaluations focus on throughput and do not consider latency performance. Generally, there is a trade-off between throughput and latency [49].

*3) Combined Kernel and User Space:* Closer related to our approach are recent frameworks that combine kernel and user space techniques. General architectural principles for building hybrid kernel-user space virtual network functions have been explored in [50]. With the combined kernel and user space approaches, the VNF applications can be programmed with the common socket interfaces; thus, some legacy applications can be deployed without modification. The combined approaches can utilize the scheduler provided by the guest operating system and can simultaneously utilize kernel and user space tools. Thus, the combined kernel and user space approaches allow for low complexity implementations.

Zhang et al. [35] have recently implemented network coding on typical virtual machines by employing DPDK with the kernel network interface (KNI) [51]. We refer to this approach as the "centralized approach" as it strives to pack all VNFs into a single VM so as to avoid the latency introduced by transmissions between VMs, see Fig 5.

In order to make multiple VNFs cooperate properly, the centralized approach employs both kernel and user space tools. As illustrated in Fig. 5, a packet needs to be transmitted between kernel space and user space at least four times (red lines in Fig. 5 indicate slow path or bottleneck). Additional copies of packets between kernel and user space introduce non-negligible latency, especially in a virtual guest operating system. Moreover, the resources, especially virtual CPU resources, need to be shared between multiple processes both in kernel and user space. These context switches also incur latencies. Furthermore, the cache behavior of the CPU can not be optimized because of this context switching; specifically, some processing related instructions cannot be fetched and consistently stored in the CPU cache.

One main drawback of the centralized approach is relatively high latency which cannot be readily scaled down.

For instance, utilizing two virtual CPU cores (one core to handle packet receptions and another core to handle packets transmissions), the latency cannot be reduced [35]. Moreover, the centralized approach is not flexible for dynamic resource allocations. Since several VNFs run the same VM, the VM should be provisioned with redundant resources to ensure QoS if more VNFs are added in the chain. To overcome the drawbacks of the centralized approach, the main strategy of CALVIN is to distribute VNFs over a chain of VMs that are optimized for running the network function ether completely in kernel or completely in the user space.

We briefly note that a specific combined kernel and user space approach, referred to as "Tuna", for a 5G wireless access point has been developed in [52]. The Tuna approach locates management frames in the user space for virtualization, while placing control and data frames in the kernel space to reduce packet processing delays. In contrast, our approach is suitable for general VM processing and not tied to a particular application.

## III. Proposed Approach: Chain bAsed Low latency VNF ImplemeNtation (CALVIN)

The proposed CALVIN approach aims to take advantage of the benefits of in-kernel and kernel bypass techniques while avoiding the overhead of context switching of the centralized approach. We describe in this section the overview, architectural design and workflow of our CALVIN approach.

### *A. Overview*

Compared to the centralized approach, the main strategy of proposed CALVIN is to distribute VNFs over a set of VMs which are chained with high performance service function forwarder (e.g. virtual switch). For the purpose of minimizing the context switching between spaces, VMs are optimized for running the main processing loop of VNF applications either completely in kernel or completely in user space.

The main advantages of this strategy is listed as follows:

- Reduce the cost of context switching inside VM: As quantified in [53], the context switching can produce direct and indirect costs. The direct cost are incurred mainly for storing the state of a process. The cache sharing between multiple processes creates an indirect cost that can exceed one millisecond for high system workloads. Running VNF application in a single space mitigates negative effects of both costs on latency.
- Avoid copying data structure of packets between spaces: For ultra low latency VNF implementation, the cost of data copying and format conversion at any location of the data path should be considered. As illustrated in Fig 5, the data exchange between spaces is one of the crucial bottlenecks for the processing delay of VNFs. Limiting all packet operations in a single space can minimize required instances of data copying.
- Increase the scalability and flexibility: In a centralized approach, such as [35], multiple VNFs run on the same VM, sharing the same resources and configurations. Because of the virtual resource contention, the latency performance is not scalable without resizing resources when new VNFs are added into the processing pipeline. In comparison, the performance of CALVIN can be scaled due to the flexible structure of the dynamic SFC. Each new VNF can be assigned a specific VM whose resource capacity is optimized for the current functional requirements.

Based on these design imperatives, the **first problem** to be solved by the CALVIN is: Which functions should be implemented in kernel space and which should be implemented in the user space?

The network functions are divided into three main categories in CALVIN :

- Elementary(or Skeleton) functions: This type is the fundamental functionalities for all VNFs: $i$) Retrieve packets from the ingress virtual interface. $ii$) Create data structures to store packets for operations. $iii$) Transmit processed packets through egress virtual egress interfaces. These functions can be implemented in both kernel and user spaces.
- Basic functions: The main characteristics of this class are listed as follows:
  - Operations are performed mainly on the header(or metadata) rather payload of the packet.
  - The computational intensity and complexity is trivial.
  - The implementation has no strict dependency on specific running environment or frameworks.

  With above listed properties, these functions are feasible and also suitable to be implemented in the kernel space: $i$) The size of header is typically small. Header operations can be performed without using iterative execution which are not fully supported in the current version of eBPF and XDP. $ii$) The processing delay is able to be limited to an acceptable range even without using acceleration mechanisms like Single Instruction, Multiple Data (SIMD), hugepages or CPU cache prefetching. Such mechanisms are not completely available in most in-kernel frameworks. $iii$) This type of functions can be implemented directly in the kernel running environment. Typical examples for basic functions including router, load balancer, NAT or packet filter.
- Advanced functions: Compared to basic functions, advanced function involve functions that perform complex and compute-intensive operations: $i$) Both header and payload parts need to be processed. $ii$) Acceleration mechanisms are essential to mitigates the overhead of processing in the user space. $iii$) The implementation of these functions requires typically particular runtime or execution environment. For example, one of the most used open source network coding library Kodo needs the C++ runtime which is not available in the kernel space [54]. Due to the features listed above, advanced functions are more beneficial to be implemented in the user space. Data encryption, compression and network coding are practical instances for this category.

### *B. Architecture Design*

The architectural design of the CALVIN is illustrated in the Fig 6. The CALVIN is built on top of the in [55] proposed research oriented SFC framework SFC-Ostack. Components of control and data plane are extended to enable latency optimization strategies:

- Control plane: The VNF classifier is an added module that is responsible for the translation between VNF and SFC description. VNFs are sorted into basic and advanced functions. The processing pipeline of these VNFs is then converted into a function chain of VMs with their networking configurations. The life-cycle management of all instances in the SFC description is then handled by the SFC manager that communicates with OpenStack services.
- Data plane: In the data plane, multiple VMs are launched to run service functions over several physical compute nodes. In each VM, the service function is positioned either in kernel or user space. Packets are received from the ingress interface, processed by the function program and sent out through the egress interface. VMs in the service chain are connected orderly by Open vSwitch with DPDK datapath (OVS-DPDK).

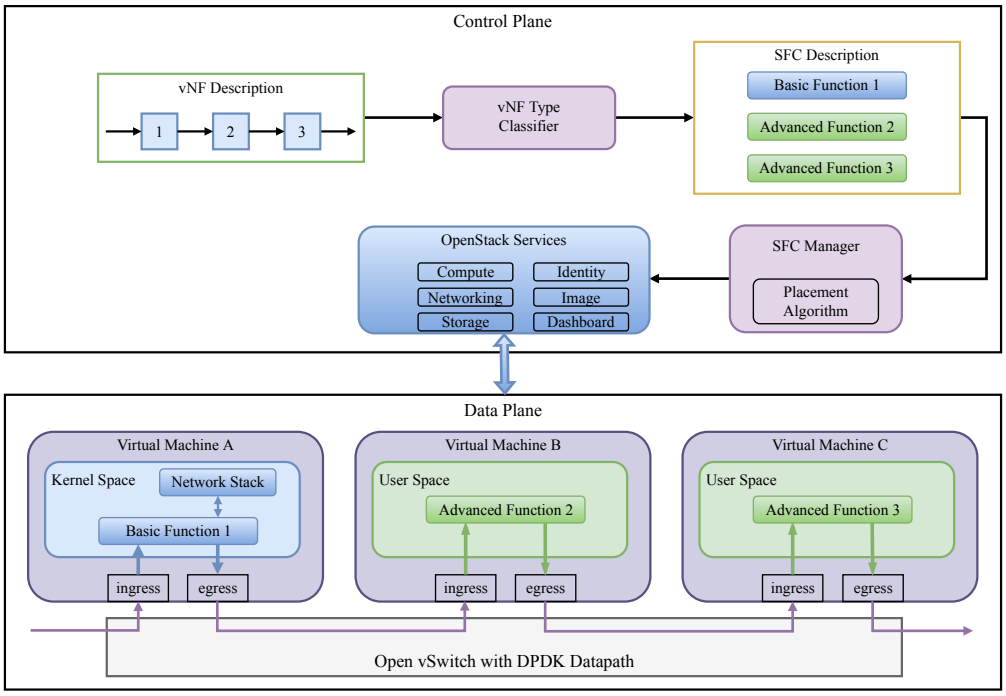



Fig. 6: Architectural design of the CALVIN including fundamental components in control and data plane. The SFC manager utilizes services provided by OpenStack to deploy classified VNFs into a chain of virtual machines (VMs). The functions are implemented either in kernel or user space. VMs in the service chain are connected by Open vSwitch with DPDK datapath (OVS-DPDK).

### *C. Setup and Workflow*

In order to deploy optimization strategies proposed by CALVIN , various settings are required for both hardware and software used by the OpenStack cloud platform.

$i$) Subsequent configurations are required to accelerate the virtual networking infrastructure:

- The physical NIC of each node must support DPDK to deploy the official OVS-DPDK plugin of OpenStack.
- Each compute node should have dedicated CPUs assigned only for OVS-DPDK: Current version of OVS-DPDK works in polling mode and any interruptions from other processes can cause performance mitigation and even lead to packet losses. Consequently, dedicated CPU cores must be configured for OVS-DPDK by using the CPU isolation tool provided by Linux.
- The Input/Output Memory Management Unit (IOMMU) should be enabled to allow guest VMs to directly use physical NICs through Direct Memory Access (DMA).
- Sufficient memory should be reserved to allocate hugepages for OVS-DPDK on all nodes. For compute nodes, additional memory need to be retained to enable allocating hugepages for guest OS of each VM instance.

$ii$) Following setups are essential to enable CALVIN 's optimization strategies inside VMs. Since the Kernel-based Virtual Machine (KVM) is the default hypervisor used by the computing service of OpenStack (until latest version: rocky) [56], following configurations are deployed for VMs launched by the KVM hypervisor.

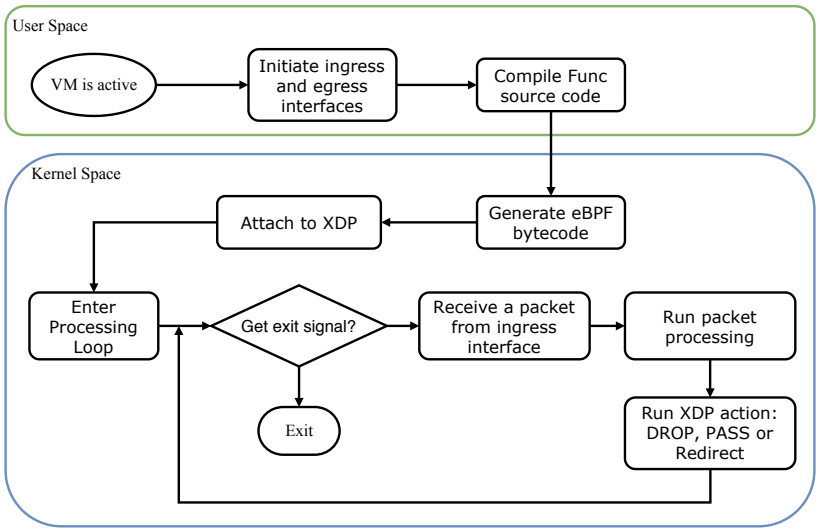



Fig. 7: Workflow for functions running in the kernel space

- Kernel Space:
  - Due to the requirement of the XDP [57], the virtual interface of each VM should enable supporting for allocating dedicated transmitting queue (TX queue). This can be done by deploying the this virtio_net patch [57] to OpenStack or extending the OpenStack Nova project.
  - The Linux kernel of the guest OS running inside the VM should be updated to at least 4.8 to run the XDP program [58].
  - The eBPF compiler framework BCC [58] should be installed to compile XDP program and also attach the compiled program to the virtual interfaces.
- User Space:
  - Virtual interfaces should be assigned with IOMMU for minimal latency overhead.
  - Sufficient amount of memory should be allocated for hugepages. These memories are used not only to initialize the running environment of each DPDK application but also to store packets that must be queued before sending out.
  - The DPDK kernel module igb_uio need to be loaded to enable the kernel Poll Mode Drivers (PMD) driver. This driver should be bound to the ingress and egress interfaces of the VM.

The Fig 7 and 8 presents the workflow of running basic functions in the kernel space and running advanced functions in the user space.

## IV. Performance Evaluation of the Elementary and Basic Functions

For the purpose of evaluating the feasibility of proposed CALVIN for the 1 ms control loop, the elementary and basic functions are firstly deployed to measure the latency performance with a trivial computational workload. The measurement results of this evaluation indicate the baselines of latencies.

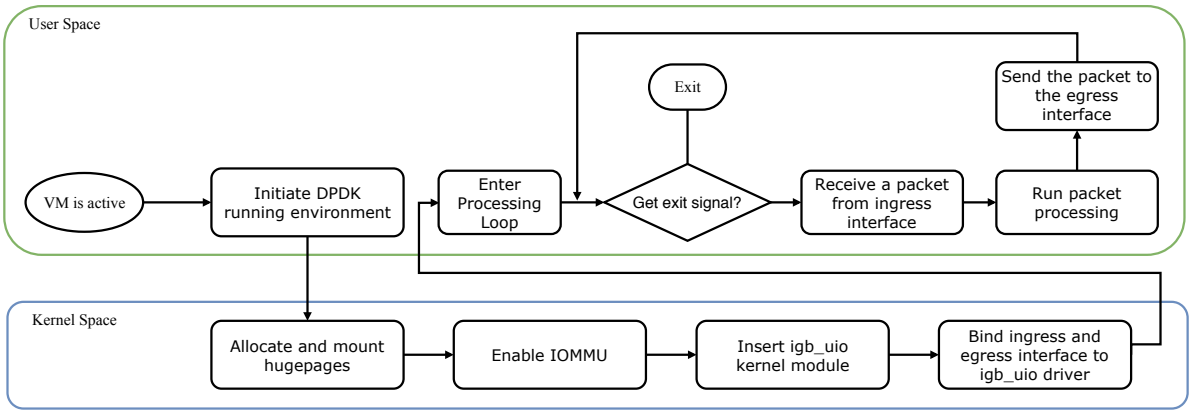



Fig. 8: Workflow for functions running in the user space

The measurement campaign, VNF Implementation, performance comparison and the measurement results are described and analyzed in this section in detail.

### A. *Measurement Campaign*

*1) Measurement Setup:* The illustration of the evaluation scheme is presented in Fig 9. Four VMs are launched on two compute nodes to evaluate the latency performance of a simplified service loop introduced in II-A: The active measurement strategies are used to measure the latency performance under UDP traffic. The UDP protocol is used for the probing traffic since it is used when low latency is a key requirement. Latency measurements for TCP are relative difficult due to its complex flow control and congestion control mechanism. The UDP traffic provides us a full control of both sending and receiving processes. The Methodology, metric and the selection of parameter are described as follows:

1) Methodology:

   The measurement setup is based on the service loop presented in Fig 3. The service proxy is used here to run the UDP client, which sends probing packets to the service server located on the same compute node. This server simply bounces all packets it receives back to the client as fast as possible. Probing packets can be forwarded directly to the server to measure the underlying network infrastructure's fundamental latency or operated by a chain of multiple VNFs that are launched on another compute node. Compared to the centralized measurement setup used in [41], components of our measurement are distributed over two different physical nodes. This distribution make our evaluation more practical: This setup takes all types of latency presented in Fig 4 into consideration. Therefore, the measurements results can demonstrate the feasibility of using our approach for low latency applications in the practical cloud environment. Time stamps (ts) and identification numbers (IDs) are added before the payload part of each probing packet. While time stamps are used for delay calculation, the IDs are used to avoid Out-Of-Order packets which is concluded as one KPI of performance characterization in [59].

2) Metric:

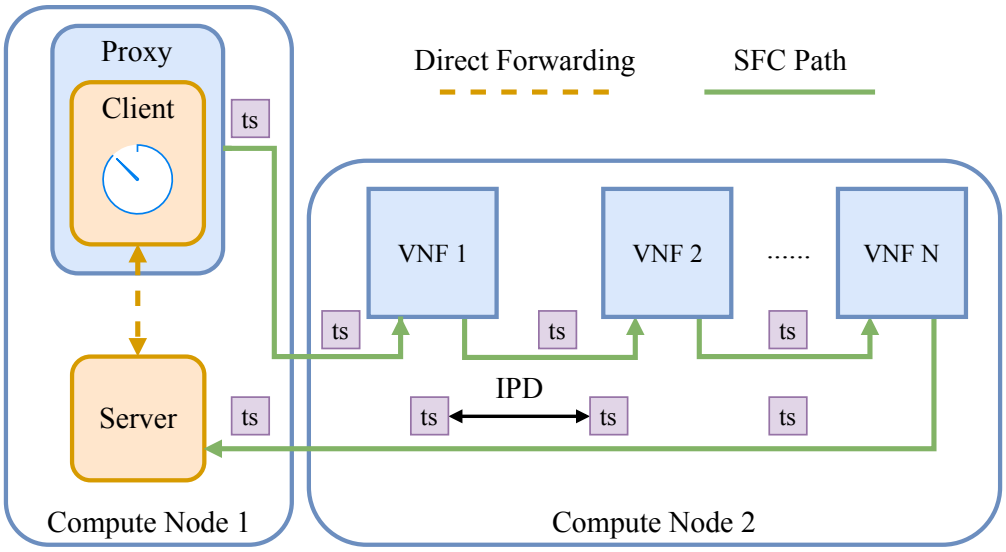



Fig. 9: Graphical representation of the evaluation scheme for the elementary forwarding function. The SFC proxy and the server are allocated on the same compute nodes. Multiple VNFs are launched on another compute node to handle the probing UDP traffic generated by the SFC proxy.

The Round-Trip-Time (RTT) of each UDP packet (namely per-packet delay) sent by the proxy is chosen as the latency metric in our evaluation for following reasons: $i$) The end-to-end service delay (the KPI we focus on in the evaluation) includes the delay introduced by the forward and backward paths. $ii$) The RTT measurement do not require the time synchronization between VMs. Depending on the experiments performed in [60], VM-level time synchronization is prone to errors due to the interference between VMs.

3) Measurement parameter:

Because of the usage of active measurements, it is necessary to estimate two important UDP traffic related parameters: the Inter-Packet-Delay(IPD) and the payload size. Based on our estimation measurements, relative small IPDs (within a few milliseconds) have little impact on the RTT values. The RTT values present a upward trend with the increase of the IPD. The reason of this trend is due to the potential queuing mechanism of the OS that is activated under low network traffic load. To avoid the additional latency introduced by queueing and ensure a certain level of stability, the IPD is configured to 5 ms in all RTT measurements. For the payload size, 256 and 1400 bytes are selected as the lower and upper limits. According to our estimation, UDP segments with the payload smaller than 256 bytes can not be processed properly by the XDP program. At the same time, the official SFC plugin of OpenStack [61] currently does not support handling jumbo frames. The maximal UDP payload size depends on the Maximum Transmission Unit (MTU) of the underlying physical network. For the default MTU of Ethernet for 1500 bytes, the maximal UDP payload size is limited to 1472 bytes (1500 - 20 (Minimal IPv4 header) - 8 (UDP header)). A payload size of 1400 bytes is chosen to provide enough free spaces reserved for IP header options or additional service function related headers.

To fully evaluate the latency performance of the proposed CALVIN , two different comparisons are performed

in the RTT measurements: $i$) Comparison among different technologies: Based on the literature research, the XDP and DPDK are finally chosen by CALVIN to implement VNFs in kernel and user space. To evaluate the latency performance of two selected technologies, two other technologies that are frequently referenced in research papers on VNF implementation, namely the Linux Kernel Forwarding (LKF) and Click modular router, are also used for RTT measurements. For this evaluation, a single VM is launched on the compute node 2 presented in Fig 9 to run the network function. $ii$) Comparison between the centralized approach introduced in [35] and CALVIN : In order to compare the latency performance of proposed CALVIN with the approach used by the state of the art. The centralized approach utilized in [35] is also reimplemented and included in RTT measurements. Since the centralized approach is originally designed to support multiple VNFs, in this comparison, two elementary VNFs are deployed either on a single VM for the centralized approach or on two separate VMs for the CALVIN . To ensure fairness of the comparison, the centralized method uses a VM that is twice the size of the computational resources (virtual CPU and memory) used by the CALVIN . Besides the RTT, the bandwidth of two approaches are also measured by using Iperf with UDP mode [62].

*2) VNF Implementation:* In order to achieve measurement activities, the elementary forwarding and other two basic functions are implemented using four technologies. This section describes these functions and their implementation details.

1) Elementary and basic functions under evaluation:
   a) Elementary Forwarding (FWD): In this function, packets are retrieved from the ingress interface and directly forwarded to the egress interface without any operations. This is the elementary function of each VNF and other functions are built on top of this.
   b) Appending time stamps (ATS): The timestamps of the reception and transmission of the current VNF is appended to the end of the UDP payload of each packet before the packet is sent out. In this function, because of the payload size modification, the checksums of layer 3 and 4 headers must be recalculated. Therefore, it is used to estimate the fundamental latency overhead introduced by the trivial operation on the packet payload.
   c) XORing UDP payload (XOR): This function performs an XOR operation on all bytes of the UDP payload. For each XOR operation, the same and static key is used. Compared to ATS, the XOR function is used to estimate the additional latency introduced by the non-trivial computational operation on the packet payload.
2) Packet IO and processing technologies inside virtual machine:
   a) XDP application: The detailed workflow flow of XDP is described in III-C. Due to following restrictions, only FWD and the XOR with the UDP payload size of 256 bytes are implemented: $i$) The maximal number of instructions per XDP program is currently restricted to 4096 BPF instructions [63]. This limit the complexity of the computational operation as well as the amount of data that can be manipulated. $ii$) The range of operational memory for each packet is limited by the size of the original received packet. Any operation out of this range is not allowed. The ATS function can not be implemented with XDP due to this restriction.
   b) DPDK application: Thanks to the high flexibility and programmability offered by the DPDK, all three

functions can be implemented as DPDK applications. By default, DPDK applications run in polled mode and can consume 100% of CPU resources. In our implementation, a sleeping mechanism is added to reduce the CPU consumption by monitoring the ingress traffic. Using this mechanism, if no traffic reaches the ingress interface for a certain amount of time, the application goes into the sleep mode to decrease the CPU consumption. In the evaluation measurement, the period for checking the state of the ingress traffic is set to 1 $\mu s$. To minimize the processing delay, the number of bursts of the processed packet is set to 1, and the time period to drain the transmission queue is set to 10 $\mu s$.

c) Linux Kernel Forwarding (LKF) is a built-in feature Linux kernel for forwarding packets in network layer [64]. Due to its trivial operations, it's one of the fastest functions running in the kernel space. Since it doest not provide any programmability, this technique cannot be used implement different VNFs in kernel space. It's used as one of the baselines to evaluate the performance of other implementations.

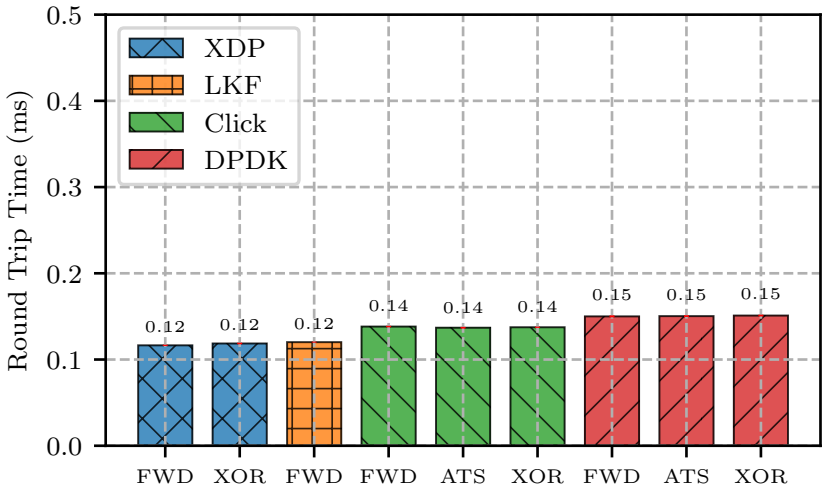
d) Click modular switch is a software framework for building flexible and configurable routers [65]. Compared to the low-level IO operations offered by DPDK, Click provided multiple encapsulated packet processing modules called elements to build the processing pipeline. Elements can be connected with a directed graph described in a router configuration file. To be evaluated VNFs are implemented by combining built-in and customized elements.

The source code of all VNF implementations are open source and can be found in [66].

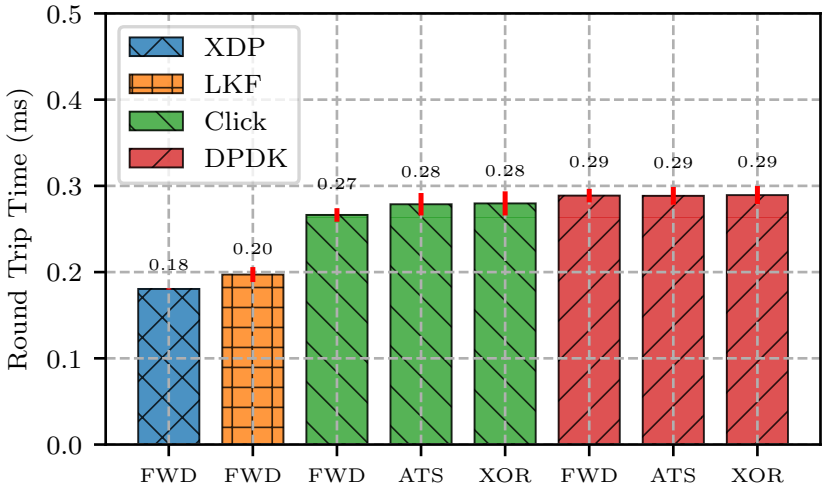
*3) Testbed:* All measurements are performed on our NFV testbed which consists of four COTS computers that are connected by two separate Ethernet networks. Each computer is allocated with 4 CPU cores (Intel 4th Generation Core i5), 16 GB RAM, 128GB SSD and two Gigabit NICs (Intel 9301CT Gigabit CT). The OpenStack (Pike version) [67] is deployed on these computers installed with the Ubuntu 16.04 TLS operation system. As controller and network nodes are deployed on the same physical node, other machines are used as compute nodes. For each node, one NIC is used for management and public traffic, and the other NIC is used to build a separate internal data network for the virtual instance. The purpose of using a separate network is to reduce the impact of management and public traffic on latency measurements. Besides the compute, neworking, identification and storage services of OpenStack, the official SFC and OVS-DPDK plugins are installed. For the management of virtual instances, KVM is used as the hypervisor and the customized Ubuntu Cloud image is used to implement different VNFs.

## *B. Results*

*1) RTT measurement of elementary and basic functions of different technologies:* The average values and 95% confidence intervals of RTT measurement results of elementary forwarding with different technologies are illustrated in Fig 10. For each set of parameters, the measurement are repeated 50 times. In each measurement, 500 UDP segments are sent by the probing client. The results in the figure indicate firstly the latency cost for basic functions is negligible for all technologies. The XOR and ATS consumes fast the same time required by the elementary forwarding. Additionally, the RTT difference between in-kernel and user space technologies is notable in the case of forwarding large packets. For large packets, the forwarding time consumed by the user space technologies is about 50% high than the in-kernel technologies. For in-kernel technologies, although two technologies has the same latency cost for small packets,the XDP is around 10% faster than the kernel forwarding for large packets.




(a) Payload size: 256 Bytes




(b) Payload size: 1400 Bytes

Fig. 10: Round-trip time comparison among technologies in kernel and user space

*2) Performance comparison between centralized approach and CALVIN :* In order to perform a relative comprehensive comparison between two approaches, both RTT and bandwidth performance are measured. $i$) RTT: RTT measurement results of two approaches are illustrated in Fig 11. As presented in Fig 9, the direct forwarding is the baseline of the transmission delay introduced by the underlying virtual networking infrastructure. As shown, the centralized approach exceeds the delay threshold of 0.35 ms in the best case, while the RTT data of the CALVIN has about 70% probability of being within the threshold. According to the measurement results, the average values of the centralized and CALVIN are 2.39 ms and 0.32 ms, respectively. It can be concluded that the CALVIN can reach the

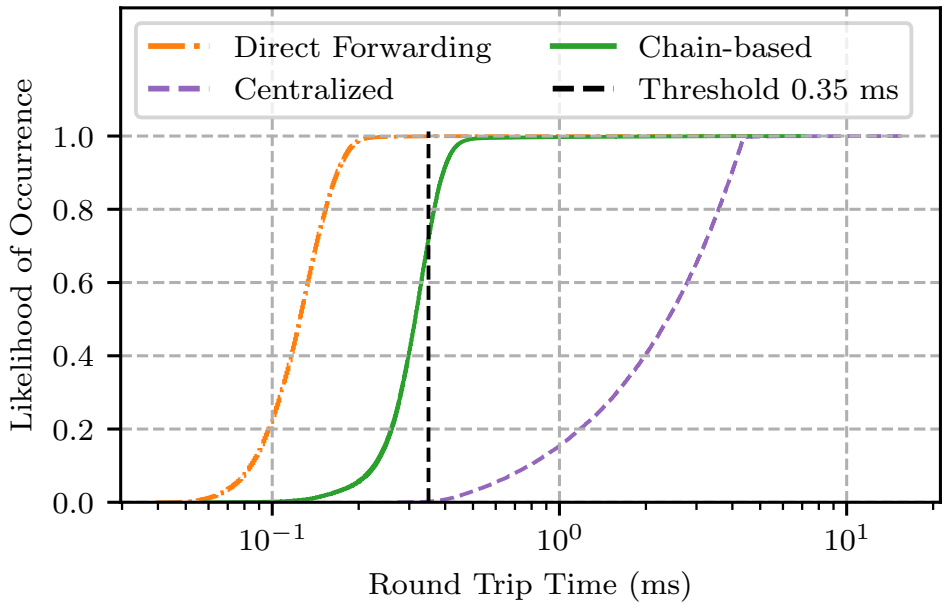



Fig. 11: Round-trip time comparison among different approaches

low latency requirement introduced in I and allow additional data processing inside the virtualized environment. $ii$) Bandwidth: Fig 12 presents the bandwidth measurement results of two approaches with payload size ranging from 256 to 1400 bytes. The bandwidth of centralized approach is always much higher than the CALVIN , especially for large packets. For a payload size of 1400 bytes, the centralized approach has a bandwidth 15 times larger than CALVIN . Compared to centralized approach with a minimal bandwidth of 6 Mbits/s, the maximal bandwidth of CALVIN is smaller than 5 Mbits/s. In addition, the bandwidth of centralized approach shows a significant reduction for smaller payload size, while the performance of CALVIN is more stable for different payload sizes. Since CALVIN uses several mechanisms to reduce the latency, such as reducing the length of the processing queue as much as possible, using single packet processing rather than batch processing, CALVIN significantly reduces bandwidth performance. On the other side, the centralized approach uses the standard Linux socket implementation which contains batch processing mechanisms for the bandwidth enhancement.

## V. Evaluation of Computational Intensive Functions

In the previous section, we evaluated the latency for the elementary function of forwarding the packet. While forwarding is the basic operation required by many network functions, like load balancers and routers, some network functions require more computational resources. In this section, we will evaluate Network Coding and encryption as two operations that have a relatively high computational demand. We first describe the practical applications and relevance of these operations for network functions. Then, we use the previously described test setup to perform an evaluation of the processing delay induced by these operations.

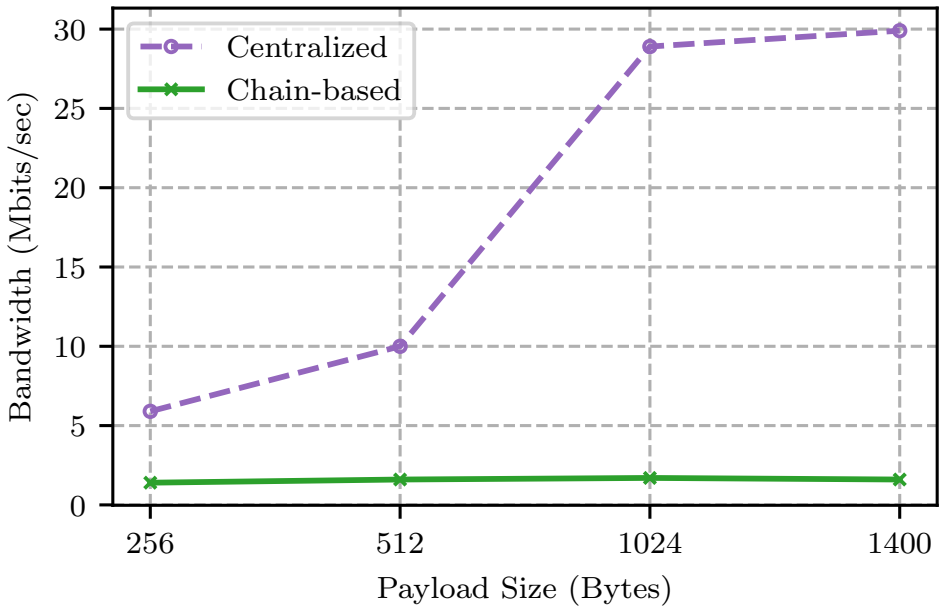



Fig. 12: Bandwidth comparison among different approaches

### A. Network Coding

Network coding combines several original packets with linear equations to form encoded packets that are transferred through the network [68]. In Random Linear Network Coding (RLNC), the values of the coefficients in the linear equations are randomly generated. The key benefits of RLNC include: i.) the ability to recode with partially received data at all nodes in the network without requiring coordination, thus being suitable for distributed environments [69], ii.) versatile coding matrix open for sparsity (adding zeros in an intelligent fashion) to reduce computation complexity [70], [71], iii.) low latency support due to on-the-fly coding capabilities [72], iv.) support of heterogeneous field sizes for communication entities, increasing flexibility in heterogeneous contexts [73] and v.) reduce the overhead between the storage and transmission layers, as the same code can also be used for distributed storage [74].

Research in network coding has suggested many different variants of RLNC. We focus on the two main types of RLNC, which are systematic block codes and convolutional sliding window codes. Block-based RLNC was introduced to reduce the computational requirements for network coding and control for network coding [75]. To further improve the performance, a systematic code does not code every packet, but also sends out the original packets as a “coded” packet [76]. The packets built from linear combinations are then sent in between original packets or at the end of the block [77]. Because of their good computational properties, block-based RLNC is often used in practical evaluations of network coding.

Sliding window network codes have been introduced as a method to reduce the in-order delay of a coded transmission [78]. In the form of a systematic code with a limited coding window, they have been shown to have

superior in-order delay compared to block codes, while requiring comparable computational resources [72].

Although Network Coding has been extensively studied in recent years, the deployment of RLNC in real-world networks is still rare. The main obstacle for the deployment of network coding is the limited availability of programmable computing resources at the nodes within the network, which are currently only used for switching and routing decisions. However, NFV and SDN provide new flexibility for developing and deploying innovative functions within a network. With NFV, network coding can be implemented for abstract virtual machines or containers, which can be instantiated at arbitrary NFV-capable nodes in the network. In addition, SDN can direct the data flows in the network towards the network coding functions and orchestrate them in a service function chain [79], [80].

### *B. Encryption*

Encryption and decryption are critical security components in communication, ensuring the confidentiality and integrity of transferred data. More than 40% of the web traffic is transported encrypted over HTTPS with an increasing trend [81]. As a result, decryption also required to process requests at load balancers, network caches, firewalls and deep packet inspection. As with Network Coding, encryption requires the whole payload to be processed by the network function and thus requires considerable computational effort. We will focus on AES, as it is currently the most commonly used encryption standard for data transfers and storage.

A previous study showed a prohibitive end-to-end latency of at least $30ms$ for encryption as a VNF [82]. However, this setup did not take advantage of fast packet processing mechanisms like DPDK. Other implementations of AES encryption as a VNF use GPUs to increase the throughput and scalability [83], [84]. With this approach, one study showed a latency of over $150\mu s$ [83].

### *C. Measurement Campaign*

Compared to the measurement setup used for elementary and basic functions described in IV-A, additional setups and methodology are required for the evaluation of advanced functions:

1) Metric: Since the RTT of elementary forwarding is evaluation in the last section, this measurement focus only on the processing delay of each packet. This processing refers to the time for a packet to be completely processed by the network functions. For network functions that can generate redundant packets, such as network coding, this delay also includes the time required to create redundant packets.
2) VNF implementation: Due to the limitation of in-kernel technologies, DPDK is used to implement all advanced functions. Both network coding and AES encryption functions are implemented on top of the elementary forwarding application. To implement network coding with different protocols, the network coding library NCKernel introduced in [72] is used. The portable AES Implementation Tiny-AES-C [85] is used to build the encryption application.
3) Methodology: In order to evaluate the impact of system workload on the processing delay, the computational operations should be performed parallel. This requirement is clearly challenging if the probing traffic is generated by a remote VM. Accurate synchronization mechanisms must be deployed on the virtualized networking infrastructure to ensure the probing packet can arrive at each VNF at the same time. Therefore, instead of using an additional client to generate probing UDP traffic, the UDP segments are generated locally by each allocated VM. For the measurement of multiple running VMs, the delay values of warm-up and

tail probing packets are not included in the measurement results. For each number of VNFs, 50000 valid probing packets are generated for processing.

### D. Evaluation

For these complex packet processing functions, the processing time of the function can be prohibitive. Based on the previous evaluation of elementary functions, we know that with DPDK the remaining permissible processing time is only $20\mu s$. Additionally to these strict latency requirements, computationally intensive processes compete for limited CPU resources. A high load on an individual core can result in longer processing times. Therefor, we evaluate the processing time for different numbers of VNFs.

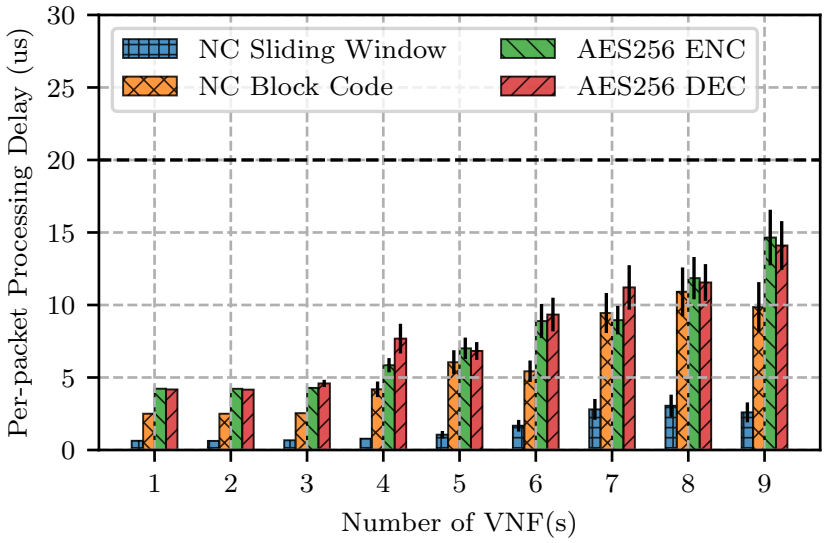
Figure 13 shows the results of our evaluation of the processing time for small packets of 256 bytes and large packets of 1400 bytes. For small packets, the processing time is within the $20\mu s$ requirement for all evaluated functions. With increasing number of VNFs the load on the CPU increases and the processing time increases linearly as soon as the number of VNFs exceeds the number of available cores. For large packets, the processing time increases. While the processing time for network coding remains relatively low and well within the bounds, encryption is not feasible even when the functions have exclusive access to a CPU core. For network coding, the sliding window code clearly outperforms the processing time for the block code. Even for large packets and high contention on the CPU sliding window network coding remains below $7\mu s$.

## VI. Conclustion and Outlook

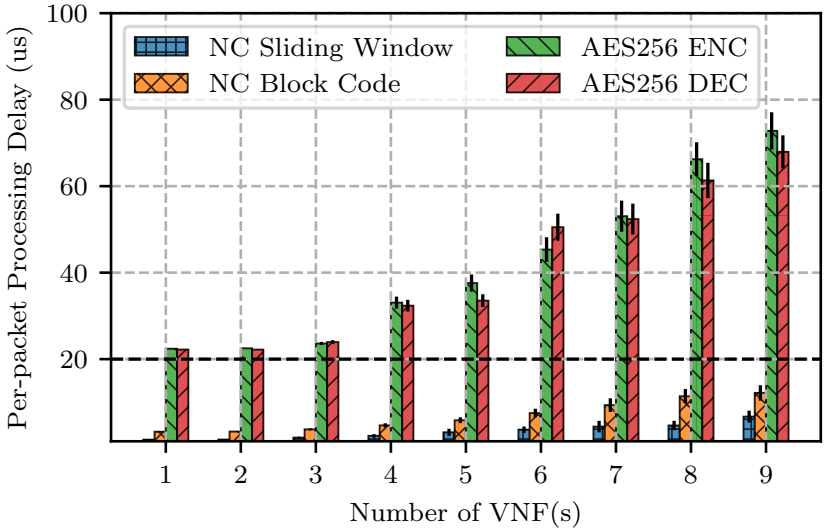
In this paper, we proposed CALVIN, an approach for distributed service function chaining for low-latency requirements. We evaluated multiple VNF implementations with respect to the latency requirements of the Tactile Internet. A critical component in a distributed service function chain is the forwarding of packets from one chain element to the next. We measured the latency of this function with different technologies, namely XDP, native forwarding of the Linux kernel, DPDK and Click. In this comparision, XDP showed the lowest latency of $120\mu s$ for small payloads and $180\mu s$ for large payloads, beating even the native forwarding of the Linux kernel by about $10\%$. The latency of DPDK and Click is up to 50% higher than XDP. Based on these measurements, we conclude that XDP is currently the best choice for implementing virtual network functions with low latency requirements.

However, XDP offers only a limited set of operations and is not suited for arbitrary packet processing. Therefor, we used DPDK to evaluate the latency of computationally intensive functions. We implemented service chains with network coding and encryption functions and evaluated the processing delay. While the latency of network coding remains within the strict requirements of the Tactile Internet, the encryption and decryption of large packets is still a challenge.

The results suggest that the 1ms round trip time targeted by the Tactile Internet can be achieved with the proposed approach. But computationally demanding operations can be problematic. The evaluation of the elementary function of forwarding showed a potential to decrease the latency. By utilizing XDP, the forwarding latency can be dramatically decreased. Future research should investigate if this potential can be utilized even for more complex packet processing functions. Because even the most basic packet processing requires a major chunk of the latency budget, the latency of service chains with increasing length will likely not remain within the requirements. In future research we want to extend the evaluation to longer, heterogeneous service chains.




(a) Payload size: 256 Bytes




(b) Payload size: 1400 Bytes

Fig. 13: Processing time comparison for computationally intensive functions.

## REFERENCES

[1] Y. Yang and A. M. Zador, "Differences in sensitivity to neural timing among cortical areas," *Journal of Neuroscience*, vol. 32, no. 43, pp. 15 142–15 147, 2012.

[2] L. Zhang, H. Gao, and O. Kaynak, "Network-induced constraints in networked control systems – a survey," *IEEE transactions on industrial informatics*, vol. 9, no. 1, pp. 403–416, 2013.

[3] X.-M. Zhang, Q.-L. Han, and X. Yu, "Survey on recent advances in networked control systems," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 5, pp. 1740–1752, 2016.

[4] *Technical Specification Group Services and System Aspects; Study on Communication for Automation in Vertical Domains (Release 16)*, 3GPP, 05 2018, 22.804 TR, V2.0.0.

[5] Q.-Y. Zhang, X.-W. Wang, M. Huang, K.-Q. Li, and S. K. Das, "Software defined networking meets information centric networking: A survey," *IEEE Access*, vol. 6, pp. 39 547–39 563, 2018.

[6] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing & softwarization: A survey on principles, enabling technologies & solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429 – 2453, Third Qu. 2018.

[7] M. Bagaa, T. Taleb, A. Laghrissi, A. Ksentini, and H. Flinck, "Coalitional game for the creation of efficient virtual core network slices in 5G mobile systems," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 469–484, Mar. 2018.

[8] S. Fu, J. Liu, and W. Zhu, "Multimedia content delivery with network function virtualization: The energy perspective," *IEEE MultiMedia*, no. 3, pp. 38–47, Jul.–Sep. 2017.

[9] Y. Harchol, D. Hay, and T. Orenstein, "FTvNF: Fault tolerant virtual network functions," in *Proc. ACM Symp. on Architectures for Netw. and Commun. Sys.*, 2018, pp. 141–147.

[10] B. Yi, X. Wang, K. Li, M. Huang *et al.*, "A comprehensive survey of network function virtualization," *Computer Networks*, vol. 133, pp. 212–262, Mar. 2018.

[11] A. AbdelSalam, F. Clad, C. Filsfils, S. Salsano, G. Siracusano, and L. Veltri, "Implementation of virtual network function chaining through segment routing in a Linux-based NFV infrastructure," in *Proc. IEEE Conf. on Network Softwarization (NetSoft)*, 2017, pp. 1–5.

[12] L. Askari, A. Hmaity, F. Musumeci, and M. Tornatore, "Virtual-network-function placement for dynamic service chaining in metro-area networks," in *Proc. IEEE Int. Conf. on Optical Network Design and Modeling (ONDM)*, 2018, pp. 136–141.

[13] N. F. S. de Sousa, D. A. L. Perez, R. V. Rosa, M. A. Santos, and C. E. Rothenberg, "Network service orchestration: A survey," *arXiv preprint arXiv:1803.06596*, 2018.

[14] A. Gupta, M. F. Habib, U. Mandal, P. Chowdhury, M. Tornatore, and B. Mukherjee, "On service-chaining strategies using virtual network functions in operator networks," *Computer Networks*, vol. 133, pp. 1–16, Mar. 2018.

[15] H. Hantouti, N. Benamar, T. Taleb, and A. Laghrissi, "Traffic steering for service function chaining," *IEEE Communications Surveys & Tutorials, in print*, 2018.

[16] W. Hahn, B. Gajic, F. Wohlfart, D. Raumer, P. Emmerich, S. Gallenmueller, and G. Carle, "Feasibility of compound chained network functions for flexible packet processing," in *Proc. European Wireless Conf.*, 2017, pp. 1–6.

[17] K. Han, S. Li, S. Tang, H. Huang, S. Zhao, G. Fu, and Z. Zhu, "Application-driven end-to-end slicing: When wireless network virtualization orchestrates with NFV-based mobile edge computing," *IEEE Access*, vol. 6, pp. 26 567 – 26 577, 2018.

[18] A. Hmaity, M. Savi, F. Musumeci, M. Tornatore, and A. Pattavina, "Protection strategies for virtual network functions placement and service chains provisioning," *Networks*, vol. 70, no. 4, pp. 373–387, Dec. 2017.

[19] T.-W. Kuo, B.-H. Liou, K. C.-J. Lin, and M.-J. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1562–1576, Aug. 2018.

[20] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz, "Service function chaining in next generation networks: State of the art and research challenges," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 216–223, Feb. 2017.

[21] F. Rath, J. Krude, J. Rüth, D. Schemmel, O. Hohlfeld, J. Á. Bitsch, and K. Wehrle, "SymPerf: Predicting network function performance," in *Proc. of the SIGCOMM Posters and Demos*, 2017, pp. 34–36.

[22] D. Raumer, S. Bauer, P. Emmerich, and G. Carle, "Performance implications for intra-node placement of network function chains," in *Proc. IEEE Int. Conf. on Cloud Networking (CloudNet)*, 2017, pp. 1–6.

[23] A. Morton, "Considerations for benchmarking virtual network functions and their infrastructure," RFC 8172, Jul. 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc8172.txt

[24] B. Yang, Z. Xu, W. K. Chai, W. Liang, D. Tuncer, A. Galis, and G. Pavlou, "Algorithms for fault-tolerant placement of stateful virtualized network functions," in *IEEE Int. Conf. on Commun. (ICC)*, 2018, pp. 20–24.

[25] B. Yi, X. Wang, and M. Huang, "A dynamic heuristic for the recomposition of service function chain," *IET Communications*, vol. 12, no. 16, pp. 1984–1990, Oct. 2018.

[26] E. Ahmed and M. H. Rehmani, "Mobile edge computing: Opportunities, solutions, and challenges," *Future Generation Computer Systems*, vol. 70, pp. 59–63, May 2017.

[27] M. Chen, Y. Zhang, L. Hu, T. Taleb, and Z. Sheng, "Cloud-based wireless network: Virtualized, reconfigurable, smart wireless network to enable 5G technologies," *Mobile Networks and Applications*, vol. 20, no. 6, pp. 704–712, Dec. 2015.

[28] T. Taleb and A. Ksentini, "Follow me cloud: Interworking federated clouds and distributed mobile networks," *IEEE Network*, vol. 27, no. 5, pp. 12–19, Sep.-Oct. 2013.

[29] T. Taleb, M. Corici, C. Parada, A. Jamakovic, S. Ruffino, G. Karagiannis, and T. Magedanz, "EASE: EPC as a service to ease mobile core network deployment over cloud," *IEEE Network*, vol. 29, no. 2, pp. 78–88, Mar.-Apr. 2015.

[30] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657–1681, Third Qu. 2017.

[31] H. Zhang, N. Liu, X. Chu, K. Long, A.-H. Aghvami, and V. C. Leung, "Network slicing based 5G and future mobile networks: Mobility, resource management, and challenges," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 138–145, Aug. 2017.

[32] P. Emmerich, D. Raumer, S. Gallenmüller, F. Wohlfart, and G. Carle, "Throughput and latency of virtual switching with OpenvSwitch: A quantitative analysis," *Journal of Network and Systems Management*, vol. 26, no. 2, pp. 314–338, Apr. 2018.

[33] C.-L. Hsieh and N. Weng, "NF-switch: VNFs-enabled SDN switches for high performance service function chaining," in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. IEEE, 2017, pp. 1–6.

[34] G. Lettieri, V. Maffione, and L. Rizzo, "A survey of fast packet I/O technologies for network function virtualization," in *Int. Conf. on High Performance Computing*. Springer, Cham, Switzerland, 2017, pp. 579–590.

[35] L. Zhang, S. Lai, C. Wu, Z. Li, and C. Guo, "Virtualized network coding functions on the internet," in *Proc. Int. Conf. on Distr. Computing Systems*, 2017, pp. 129–139.

[36] Open vSwitch with DPDK. [Accessed 2018-09-15]. [Online]. Available: http://docs.openvswitch.org/en/latest/intro/install/dpdk/

[37] T. Herbert and A. Starovoitov, "eXpress Data Path (XDP): Programmable and high performance networking data path," Mar. 2016, available from https://github.com/iovisor/bpf-docs/blob/master/Express_Data_Path.pdf, Last acccessed Oct. 2, 2018.

[38] J. D. Brouer, "Linux Networking Subsystem, XDP – eXpress Data Path," 2016, available from https://prototype-kernel.readthedocs.io/en/latest/networking/index.html, Last acccessed Oct. 2, 2018.

[39] "BPF and XDP Reference Guide," 2018, available from https://cilium.readthedocs.io/en/v1.2/bpf, Last acccessed Oct. 2, 2018.

[40] S. Miano, M. Bertrone, F. Risso, and M. Tumolo, "Creating complex network services with eBPF: Experience and lessons learned," in *Proc. IEEE High Performance Switching and Routing (HPSR)*, 2018, pp. 1–8.

[41] Z. Ahmed, M. H. Alizai, and A. A. Syed, "Inkev: In-kernel distributed network virtualization for dcn," *ACM SIGCOMM Computer Commun. Review*, vol. 46, no. 3, p. 4, Jul. 2018.

[42] "Neutron documentation," Sep. 2018, available from https://docs.openstack.org/neutron, Last acccessed Oct. 3, 2018.

[43] M. Bertrone, S. Miano, F. Risso, and M. Tumolo, "Accelerating Linux security with eBPF iptables," in *Proc. ACM SIGCOMM 2018 Conference on Posters and Demos*, 2018, pp. 108–110.

[44] X. Li, Y. Wu, J. Ge, H. Zheng, E. Yuepeng, C. Han, and H. Lv, "A kernel-space POF virtual switch," *Computers & Electrical Engineering*, vol. 61, pp. 339–350, Jul. 2017.

[45] D. Scholz, D. Raumer, P. Emmerich, A. Kurtz, K. Lesiak, and G. Carle, "Performance implications of packet filtering with Linux eBPF," in *Proc. Int. Teletraffic Congress (ITC)*, Sep. 2018, pp. 1–9.

[46] Y. Zhang, B. Anwer, V. Gopalakrishnan, B. Han, J. Reich, A. Shaikh, and Z.-L. Zhang, "Parabox: Exploiting parallelism for virtual network functions in service chaining," in *Proc. ACM Symp. on SDN Research*, 2017, pp. 143–149.

[47] J. L. García-Dorado, F. Mata, J. Ramos, P. M. S. del Río, V. Moreno, and J. Aracil, "High-performance network traffic processing systems using commodity hardware," in *Data traffic monitoring and analysis, LNCS 7754*. Springer, Berlin, Heidelberg, 2013, pp. 3–27.

[48] S. Gallenmüller, D. Scholz, F. Wohlfart, Q. Scheitle, P. Emmerich, and G. Carle, "High-performance packet processing and measurements," in *Proc. IEEE Int. Conf. on Commun. Systems & Networks (COMSNETS)*, 2018, pp. 1–8.

[49] S. Gallenmüller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, "Comparison of frameworks for high-performance packet IO," in *Proc. ACM/IEEE Symp. on Architectures for Netw. and Commun. Systems*, 2015, pp. 29–38.

[50] N. Van Tu, K. Ko, and J. W.-K. Hong, "Architecture for building hybrid kernel-user space virtual network functions," in *Proc. Int. Conf. on Network and Service Management (CNSM)*, 2017, pp. 1–6.

[51] "Kernel NIC Interface," 2018, available from https://doc.dpdk.org/guides/prog_guide/kernel_nic_interface.html, Last acccessed Oct. 3, 2018.

[52] X. Wang, C. Xu, G. Zhao, and S. Yu, "Tuna: an efficient and practical scheme for wireless access point in 5G networks virtualization," *IEEE Commun. Letters*, vol. 22, no. 4, pp. 748–751, Apr. 2018.

[53] C. Li, C. Ding, and K. Shen, "Quantifying the cost of context switch," in *Proceedings of the 2007 workshop on Experimental computer science*. ACM, 2007, p. 2.

[54] M. V. Pedersen, J. Heide, and F. H. Fitzek, "Kodo: An open and research oriented network coding library," in *International Conference on Research in Networking*. Springer, 2011, pp. 145–152.

[55] Z. Xiang, F. Gabriel, G. T. Nguyen, and F. H. P. Fitzek, "Latency measurement of service function chaining on openstack platform," in *2018 IEEE 43rd Conference on Local Computer Networks (LCN) (LCN 2018)*, Chicago, USA, 2018.

[56] OpenStack Nova computing documentation. [Accessed 2018-10-3]. [Online]. Available: https://www.spinics.net/lists/netdev/msg405175.html

[57] Linux netdev mailing list: Patch 4/5. [Accessed 2018-10-3]. [Online]. Available: https://www.spinics.net/lists/netdev/msg405175.html

[58] BCC project homepage. [Accessed 2018-09-20]. [Online]. Available: https://github.com/iovisor/bcc

[59] P. Veitch and T. Long, "A Low-Latency NFV Infrastructure for Performance-Critical Applications," Tech. Rep., [Accessed 2018-9-30]. [Online]. Available: https://software.intel.com/en-us/articles/low-latency-nfv-infrastructure-for-performance-critical-applications

[60] J. Chauhan, D. Makaroff, and A. Arkles, "Is Doing Clock Synchronization in a VM a Good Idea?" Tech. Rep. [Online]. Available: http://www.coker.com.au/bonnie++/

[61] Service Function Chain Extension for OpenStack Networking. [Accessed 2018-10-1]. [Online]. Available: https://docs.openstack.org/networking-sfc/latest/

[62] Iperf homepage. [Accessed 2018-9-10]. [Online]. Available: https://iperf.fr/

[63] BPF and XDP reference guide. [Accessed 2018-10-4]. [Online]. Available: https://cilium.readthedocs.io/en/v1.2/bpf/

[64] Linux Kernel networking documentation. [Accessed 2018-10-4]. [Online]. Available: https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt

[65] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, 2000.

[66] Build-VNF project repository. [Accessed 2018-10-5]. [Online]. Available: https://github.com/stevelorenz/build-vnf

[67] OpenStack Pike documentation. [Accessed 2018-10-4]. [Online]. Available: https://docs.openstack.org/pike/

[68] R. Ahlswede, N. Cai, S.-Y. Li, and R. W. Yeung, "Network information flow," *IEEE Transactions on information theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[69] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4413–4430, 2006.

[70] S. Feizi, D. E. Lucani, and M. Médard, "Tunable sparse network coding," in *22th International Zurich Seminar on Communications (IZS)*. Eidgenössische Technische Hochschule Zürich, 2012.

[71] V. Nguyen, G. T. Nguyen, F. Gabriel, D. E. Lucani, and F. H. Fitzek, "Integrating sparsity into fulcrum codes: Investigating throughput, complexity and overhead," in *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2018, pp. 1–6.

[72] S. Wunderlich, F. Gabriel, S. Pandi, F. H. Fitzek, and M. Reisslein, "Caterpillar rlnc (crlnc): A practical finite sliding window rlnc approach," *IEEE Access*, vol. 5, pp. 20 183–20 197, 2017.

[73] D. E. Lucani, M. V. Pedersen, D. Ruano, C. W. Sørensen, F. H. Fitzek, J. Heide, and O. Geil, "Fulcrum network codes: A code for fluid allocation of complexity," *arXiv preprint arXiv:1404.6620*, 2014.

[74] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE transactions on information theory*, vol. 56, no. 9, pp. 4539–4551, 2010.

[75] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proceedings of the annual Allerton conference on communication control and computing*, vol. 41, no. 1. The University; 1998, 2003, pp. 40–49.

[76] J. Barros, R. A. Costa, D. Munaretto, and J. Widmer, "Effective delay control in online network coding," in *INFOCOM 2009, IEEE*. IEEE, 2009, pp. 208–216.

[77] S. Pandi, F. Gabriel, J. A. Cabrera, S. Wunderlich, M. Reisslein, and F. H. Fitzek, "Pace: Redundancy engineering in rlnc for low-latency communication," *IEEE Access*, vol. 5, pp. 20 477–20 493, 2017.

[78] M. Karzand and D. J. Leith, "Low delay random linear coding over a stream," in *Communication, Control, and Computing (Allerton), 2014 52nd Annual Allerton Conference on*. IEEE, 2014, pp. 521–528.

[79] D. Szabo, A. Gulyas, F. H. Fitzek, and D. E. Lucani, "Towards the tactile internet: Decreasing communication latency with network coding and software defined networking," in *European Wireless 2015; 21th European Wireless Conference; Proceedings of*. VDE, 2015, pp. 1–6.

[80] F. Gabriel, G. T. Nguyen, R.-S. Schmoll, J. A. Cabrera, M. Muehleisen, and F. H. Fitzek, "Practical deployment of network coding for real-time applications in 5g networks," in *Consumer Communications & Networking Conference (CCNC), 2018 15th IEEE Annual*. IEEE, 2018, pp. 1–2.

[81] A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel, and P. Tabriz, "Measuring https adoption on the web," in *26th USENIX Security Symposium*, 2017, pp. 1323–1338.

[82] V.-C. Nguyen, A.-V. Vu, K. Sun, and Y. Kim, "An experimental study of security for service function chaining," in *Ubiquitous and Future Networks (ICUFN), 2017 Ninth International Conference on*. IEEE, 2017, pp. 797–799.

[83] Z. Zheng, J. Bi, C. Sun, H. Yu, H. Hu, Z. Meng, S. Wang, K. Gao, and J. Wu, "Gen: A gpu-accelerated elastic framework for nfv," in *Proceedings of the 2nd Asia-Pacific Workshop on Networking*. ACM, 2018, pp. 57–64.

[84] M. M. Rovnyagin and A. A. Kuznetsov, "Application of hybrid computing technologies for high-performance distributed nfv systems," in *Young Researchers in Electrical and Electronic Engineering (EIConRus), 2017 IEEE Conference of Russian*. IEEE, 2017, pp. 540–543.

[85] Small portable AES implementation in C. [Accessed 2018-9-10]. [Online]. Available: https://github.com/kokke/tiny-AES-c

**Zuo Xiang** (zuo.xiang@tu-dresden.de) is a Ph.D. student at the Deutsche Telekom Chair of Communication Networks at Tu Dresden, Dresden, Germany. He received his diploma (Dipl.-Ing.) degree in electrical engineering from the Technical University Dresden, Germany, in 2018. His research interests lie in the area of network softwarization. His research focuses on Network Function Virtualization (NFV) and Software Defined Network (SDN) for low latency communication in cloud and edge computing environment.

**Frank Gabriel** (frank.gabriel@tu-dresden.de) is a Ph.D. student at the Deutsche Telekom Chair of Communication Networks at Tu Dresden, Dresden, Germany. He received the Dipl.-Inf. degree in computer science from the Technical University Chemnitz, Germany, in 2011. His research focuses on network coding for reliable and low latency communication in multipath and mesh networks.

**Elena Urbano** (elena.urbano_perez@tu-dresden.de) is a junior researcher at the Deutsche Telekom Chair of Communication Networks at TU Dresden, Dresden, Germany. She received her M.Sc. degree in Automation and Industrial Electronics from the University of Málaga, Málaga, Spain. Her research interests are mainly in the areas of control theory and robotics, and her work focuses on the joint design of communication and control in cyber physical systems exploiting adaptive networking concepts to enable global (and not local) cyber physical systems.

**Giang T. Nguyen** (giang.nguyen@tu-dresden.de) is a senior researcher at Deutsche Telekom Chair of Communication Networks, Technical University of Dresden. He received his Master degree (M.Eng.) in Telecommunications from Asian Institute of Technology (AIT), Thailand in 2007 and his Ph.D. (Dr.-Ing.) in computer science from the Technical University of Dresden, Germany in 2016. His research interests lie in the area of 5G and highly adaptive and energy-efficient computing (HAEC). His research focuses on the resilience aspects of Peer-to-Peer video streaming, network coding, cooperative networking, and energy-efficient protocol design.

**Martin Reisslein** (reisslein@asu.edu) (S'96-M'98-SM'03-F'14) is a Professor in the School of Electrical, Computer, and Energy Engineering at Arizona State University (ASU), Tempe. He received the Ph.D. in systems engineering from the University of Pennsylvania, Philadelphia, in 1998. He currently serves as Associate Editor for the *IEEE Transactions on Mobile Computing*, the *IEEE Transactions on Education*, and *IEEE Access*, as well as *Computer Networks*. He is Associate Editor-in-Chief of the *IEEE Communications Surveys & Tutorials* and Co-Editor-in-Chief of *Optical Switching and Networking*. He chairs the steering committee of the *IEEE Transactions on Multimedia*.

**Frank H. P. Fitzek** (frank.fitzek@tu-dresden.de) is the coordinator of the 5G Lab Germany and a professor at Technische Universitat Dresden. He received his diploma (Dipl.-Ing.) degree in electrical engineering from RWTH-Aachen, Germany, in 1997, and his Ph.D. (Dr.-Ing.) in electrical engineering from the Technical University Berlin, Germany in 2002. He has received numerous awards, including the NOKIA Champion Award five times, the NOKIA Achievement Award (2008), the Danish SAPERE AUDE research grant (2010), and the Vodafone Innovation prize (2012). His research focuses on wireless and mobile networks, mobile phone programming, network coding, cross-layer and energy-efficient protocol design, and cooperative networking.