Reducing Latency in Virtual Machines: Enabling Tactile Internet for Human-Machine Co-Working

Zuo Xiang, Frank Gabriel, Elena Urbano, Giang T. Nguyen, Martin Reisslein, *Fellow, IEEE*, and Frank H. P. Fitzek

***Abstract*— Software-defined networking (SDN) and network function virtualization (NFV) processed in multi-access edge computing (MEC) cloud systems have been proposed as critical paradigms for achieving the low latency requirements of the tactile Internet. While virtual network functions (VNFs) allow greater flexibility compared to hardware-based solutions, the VNF abstraction also introduces additional packet processing delays. In this paper, we investigate the practical feasibility of NFV with respect to the tactile Internet latency requirements. We develop, implement, and evaluate Chain-based Low latency VNF ImplemeNtation (CALVIN), a low-latency management framework for distributed Service Function Chains (SFCs). CALVIN classifies VNFs into elementary, basic, and advanced VNFs; moreover, CALVIN implements elementary and basic VNFs in the kernel space, while the advanced VNFs are implemented in the user space. Throughout, CALVIN employs a distributed mapping with one VNF per Virtual Machine (VM) in a MEC system. Furthermore, CALVIN avoids the metadata structure processing and batch processing of packets in the conventional Linux networking stack so as to achieve short per-packet latencies. Our rigorous measurements on off-the-shelf conventional networking and computing hardware demonstrate that CALVIN achieves round-trip times from a MEC ingress point via two elementary forwarding VNFs (one in kernel space and one in user space) and a MEC server to a MEC egress point on the order of 0.32 ms. Our measurements also indicate that MEC network coding and encryption are feasible for small 256 byte packets with an MEC latency budget of 0.35 ms; whereas, large 1400 byte packets can complete the network coding, but not the encryption within the 0.35 ms.**

***Index Terms*— Low-latency network softwarization, kernel space, service function chain (SFC), tactile Internet, user space, virtualized network function (VNF).**

I. Introduction and Motivation

Manuscript received October 10, 2018; revised March 07, 2019; accepted March 13, 2019. Date of publication March 21, 2019; date of current version April 16, 2019. Funded by Deutsche Telekom and by the German Research Foundation (DFG, Deutsche Forschungsgemeinschaft) as part of Germany's Excellence Strategy – EXC 2050/1 – Project ID 390696704 – Cluster of Excellence "Centre for Tactile Internet with Human-in-the-Loop" (CeTI) of Technische Universität Dresden. *(Corresponding author: Martin Reisslein.)*

Z. Xiang and F. H. P. Fitzek are with the Centre for Tactile Internet with Human-in-the-Loop (CeTI), Technische Universität Dresden, 01062 Dresden, Germany, and also with the Deutsche Telekom Chair, Technische Universität Dresden, 01062 Dresden, Germany (e-mail: zuo.xiang@tu-dresden.de; frank.fitzek@tu-dresden.de).

F. Gabriel, E. Urbano, and G. T. Nguyen are with the Deutsche Telekom Chair, Technische Universität Dresden, 01062 Dresden, Germany (e-mail: frank.gabriel@tu-dresden.de; elena.urbano@tu-dresden.de; giang.nguyen@tu-dresden.de).

M. Reisslein is with the School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, AZ 85287-5706 USA, and also with the Centre for Tactile Internet with Human-in-the-Loop (CeTI), Technische Universität Dresden, 01062 Dresden, Germany (e-mail: reisslein@asu.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2019.2906788

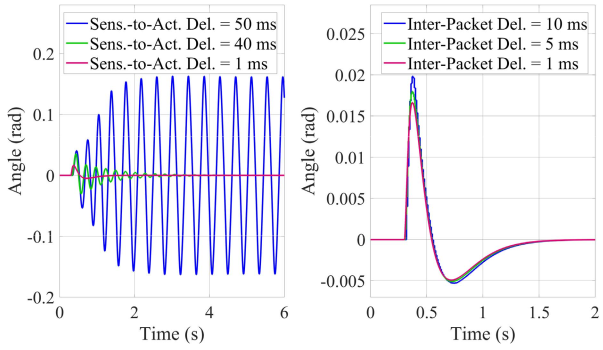


Fig. 1. Angle of an inverted pendulum trying to reach stability for different sensor-to-actuator delays (50 ms, 40 ms, and 1 ms) for 1 ms inter-packet delay as well as for 1 ms sensor-to-actuator delay for different inter-packet delays (10 ms, 5 ms, and 1 ms).

Low latency communication is the central requirement for enabling the tactile Internet for human-machine co-working [1]–[6]. Both, humans and machines require latencies below one millisecond for a wide range of co-working scenarios. For instance, for humans operating in a virtual world and for interactions with robots and other machines, the latencies for visual, audio, or tactile multi-sensoric feedback should be below 15 ms, 3 ms, or 1 ms, respectively [7]. Every machine based on control loops also requires low latencies in order to work efficiently or to operate in a stable manner [8], [9]. 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. Fig. 1 shows the influence of the delay between the angle sensor and pendulum actuator (motor) on the pendulum stability. For long delays (50 ms in Fig. 1), the system becomes unstable, and the pendulum will never reach stability in the inverted position. For shorter delays (40 ms), the system takes some time to achieve stability. This time delay could imply 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 [10] defines the allowed latency requirement of one millisecond for an end-to-end communication. Based on this standard, we consider a typical allocation of the individual 5G communication network delay budget components, as illustrated in Figure 2. Figure 2 assumes that a total of 0.4 ms is consumed in the embedded systems and the wireless links on both ends. For instance, 0.1 ms may be allocated for the embedded sensor computing platform

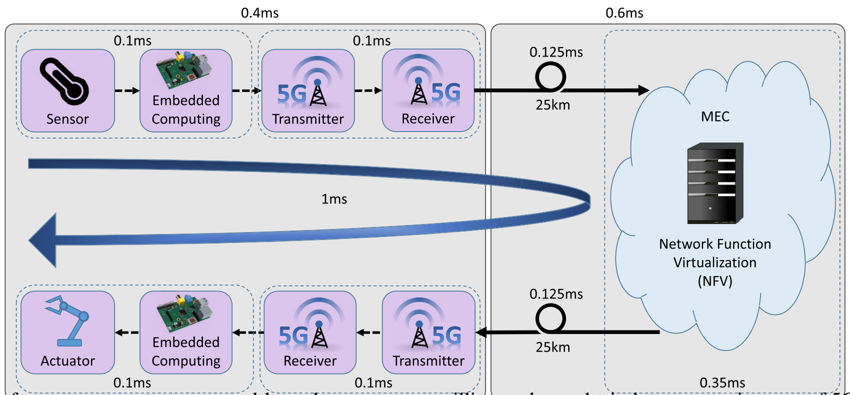


Fig. 2. Typical latency budget for sensor-to-actuator control loop that meets one millisecond round-trip latency requirement of 5G [10]: 0.4 ms are allocated for embedded sensor and actuator processing and wireless communication, leaving 0.6 ms for the wired link propagation as well as the virtualized communication environment ("compute and forward") processing in the MEC.

to evaluate the sensed information, 0.1 ms for (one-way) communication latency in uplink and downlink, and 0.1 ms for embedded computing at the actuator.

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 μs per kilometer) and the physical fiber characteristics. 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 Networking (SDN) [11], Network Function Virtualization (NFV) [12]–[19], and Service Function Chaining (SFC) [20]–[34] standardized by the IETF/IRTF. These novel technologies enable the concept of Multi-access Edge Computing (MEC) [26], [35]–[40], which allows for local processing of data, which in turn will reduce the latencies on the pure communication path. Assuming a maximum distance of 25 km between the sensor/actuator and the MEC, the (round-trip) communication will require 0.25 ms and will leave 0.35 ms for NFV processing in the MEC system.

However, achieving low latencies is a notoriously difficult problem in communication networks [41], [42]. While delays that are proportional to the available transmission bitrate and data amounts can be addressed through scaling up the transmission capacities and compression, processing delays with their various constant delay contributions pose significant challenges [42]–[44]. Moreover, recent studies [45], [46] have demonstrated that NFV, which is highly desirable for flexibility [47], imposes heavy data transfer and computation demands, incurring relatively long latencies. Nowadays, virtual switches are quite fast [48]–[50]; however, virtual machines (VMs), specifically the packet IO and processing operations inside the VM, are slow. The centralized approach proposed in [51] gives more than 2 ms latency inside the virtual communication environment for a single VM running the elementary forwarding function, which is clearly above the 0.35 ms delay budget.

In this empirical measurement study, we examine low-latency NFV in real general-purpose MEC systems built with off-the-shelf hardware and software. We design, implement, and evaluate a low-latency service function chain (SFC) management framework named Chain bAsed Low latency VNF ImplemeNtation (CALVIN). CALVIN implements VNFs either in the kernel space or in the user space and distributes each VNF to its own VM. CALVIN employs fast packet input/output (IO) mechanisms that avoid the metadata structures and the batch processing of data packets in the conventional Linux networking stack. Our extensive measurements demonstrate that CALVIN achieves MEC latencies on the order of 0.32 ms, which allows for additional processing of data, e.g., for network coding and encryption. The proposed CALVIN approach makes it for the first time possible to process advanced network functions, such as network coding and encryption, in a general-purpose virtualized MEC setting while meeting the one millisecond delay target of the tactile Internet. Our CALVIN SFC management and VNF implementation codes for OpenStack, which is the defacto industry standard for general-purpose cloud computing platforms, are openly available at [52], [53].

II. Background and Related Work

A. Background

The typical NFV service loop inside the MEC 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 SFC, transmitted to the requested server, processed by the server, and leaves 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, an SFC

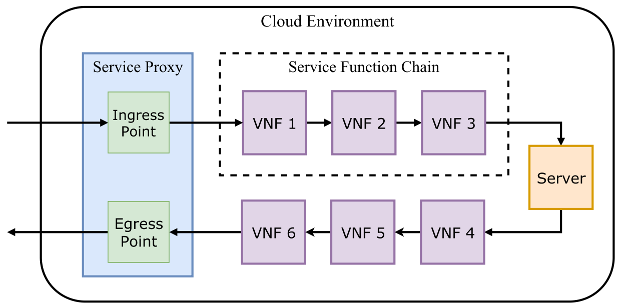


Fig. 3. Illustration of service loop in a cloud (MEC/virtualization) environment: Packets traverse an ingress service function chain (SFC) consisting of an ordered sequence of virtual network functions (VNFs) to reach the server for cloud processing and leave the cloud via the egress point.

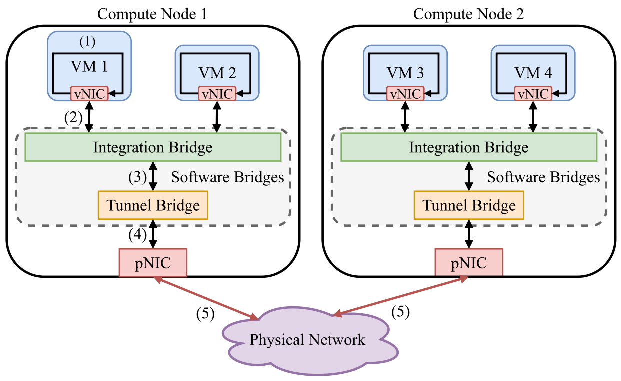


Fig. 4. Illustration of typical virtual network connection between two compute nodes in a cloud environment, whereby each compute node hosts multiple virtual machines (VMs). Each VM connects via a virtual network interface controller (vNIC) to the bridges and onwards via a physical NIC to the physical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

consists of an ordered set of 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 VMs that are orchestrated by the cloud computing platform.

Fig. 4 illustrates a typical cloud computing infrastructure scenario where multiple VMs are connected with a 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. 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 (vNICs) with the physical network interface (pNIC). In particular, 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., Generic Routing Encapsulation (GRE) or Virtual extensible LAN (VXLAN).

The networking components in Fig. 4 introduce different types of latency:

- Between VM and integration bridge (1, 2): This latency component is the focus of this study and includes two main parts:
  - Transfer packets between the integration bridge and VM through vNIC (2): This latency depends on the vNIC technology and has two subparts:
    - Data transfer between virtual bridge data buffer and vNIC ring buffer: With the performance improvements of virtual bridges, e.g., the DPDK fast path in the Open vSwitch (OVS-DPDK) [54], the latency of this subpart has been reduced to the order of microseconds [48].
    - Frame copying between the vNIC ring buffer and the VM memory: This subpart now becomes a bottleneck for low-latency data transfer in the virtual network overlay.
  - Process packets inside VM (1): This latency depends on the implementation of the VNF processing workflow. Optimizations from the implementation perspective should process the frame as fast as possible for the low-latency tactile Internet.
- Between integration bridge and pNIC (3, 4): This latency component depends on the employed virtual bridges as well as the physical resource management and orchestration (MANO) platform [55]. Cloud platforms commonly 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 the best of our knowledge, the reduction of latencies (1) and (2) is an open research question. Our proposed CALVIN approach significantly reduces these latencies so that the overall end-to-end latency of the service loop is within the 5G one millisecond latency requirement.

B. Related Work

Several recent studies based on mathematical analysis and simulations have considered SFC latencies, see e.g., [56]–[68]. These studies have mainly considered moderate to high workloads that result in substantial queueing delays for VNF processing. In contrast, we conduct an experimental study with empirical latency measurements on lightly loaded real networking and computing systems. Our goal is to rigorously empirically investigate the baseline latencies of real SFC implementations. Our measurement study complements the existing mathematical analysis and simulation studies and provides baseline latency values, which can serve as reference points for future analysis and simulation studies on low-latency VNF processing. We also develop and evaluate the CALVIN low-latency SFC framework which achieves significant latency reductions compared to existing frameworks. We note that several recent studies have examined VNF placement strategies, e.g., [69]–[77], mainly from the perspective of mathematical modeling and optimization of the placement. A few recent implementation oriented studies have examined scheduling and flexibility aspects of VNF placement [25], [78]. Complementarily, our implementation oriented study examines the low-latency aspects of VNF placement.

Some recent studies have sought to reduce communications delays and increase communications reliability through specific communications strategies [6], such as short packet transmissions [79], [80]. Complementarily to these communications mechanism focused studies, we address the NFV latency in MEC systems for arbitrary packet sizes.

The remainder of this section surveys existing related SFC 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 exploit the complementary strengths of both kernel and user space technologies to reduce the latency (while efficiently utilizing the CPU resources).

1) Kernel Space: Recent kernel space approaches have typically been based on the eXpress Datapath (XDP) framework (a new system in the Linux kernel) [81]–[84] which is built using custom extended Berkeley Packet Filter (eBPF) programs. 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 [84].

Miano et al. [85] have conducted quantitative characterizations of a range of eBPF aspects. Miano et al. documented several limitations when building complex network services with eBPF. Due to these limitations we do not implement all functions in the kernel space in CALVIN. Our own preliminary XDP evaluations (with Linux kernel 4.17.0-041700-generic), which are not included here due to space constraints, indicated that XDP achieves low latencies for elementary packet processing. However, due to the 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. 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 (OS)), 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 eBPF benefits. Moreover, XDP avoids the need to define a new security model by utilizing the Linux kernel security model.

In-kernel processing with XDP based on the eBPF with an NFV focus has recently been examined in the InKeV study [86]. 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 an SFC consisting of physically distributed VMs. In contrast, we consider physically distributed VMs in our evaluations. InKeV has been compared with the OpenStack neutron [87] networking infrastructure, which we also employ in our evaluations. Thus, InKeV could be a good competitor for the underlying virtual switches (OVSs) used by OpenStack (latency components (3)–(5) in Fig. 4). In contrast, we focus on the latency introduced by the VM, i.e., on the latency components (1) and (2) in Fig. 4.

For completeness, we note that relatively simple networking functions relating to security and virtual switching in the kernel space have recently been studied in [88]–[91].

2) User Space: User space frameworks, which are also referred to as kernel-bypassing frameworks, have been studied more frequently than kernel space frameworks [92]–[94]. However, most user space framework evaluations have focused on throughput and did not consider latency performance in detail. The performance of three widely known kernel-bypassing high-speed packet IO frameworks, including netmap, PF_RING ZC, and Intel DPDK, has been measured in [94], revealing a general trade-off between throughput and latency. Recent studies have examined several aspects of user space frameworks, including CPU scheduling [95], flexible programmability [96], [97], and resilience [98], [99]. These recent studies are complementary to our CALVIN study and did not specifically focus on low latency. For instance, the NetStar study [96] has examined flexible asynchronous network function programming, which is mainly useful for network management traffic, e.g., flow table updates; whereas we focus on low-latency processing of data traffic. The HyperVDP study [97] has developed a hypervisor that flexibly offloads some CPU compute-intensive tasks to programmable network interface cards to reduce the resource usage (at the expense of slightly reduced throughput and increased latency).

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 VNFs have been explored in [100]. 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 OS 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. [51] have recently implemented network coding on typical VMs by employing DPDK [101] with the kernel network interface (KNI) [102]. 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 the kernel space and user space at least four times (red lines in Fig. 5 indicate slow path or bottleneck). Additional copies of packets between the kernel space and user space introduce non-negligible latencies, especially in a virtual guest OS. Moreover, the resources, especially virtual CPU (vCPU) resources, need to be shared between multiple

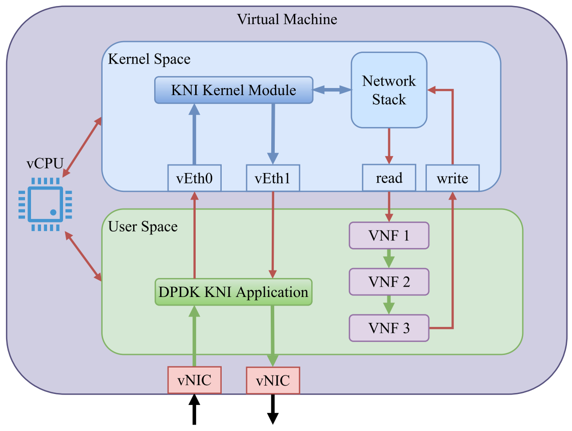


Fig. 5. Graphical illustration of operational principles of centralized combined kernel and user space approach described in [51]. The combined approach requires four or more packet transmissions between kernel space and user space, illustrated by the vertical red arrows crossing over between kernel space and user space.

processes both in kernel and user space; these context switches also incur latencies. Furthermore, the cache behavior of the CPU cannot be optimized because of this context switching; specifically, some processing related instructions cannot be pre-fetched and consistently stored in the CPU cache. The resulting relatively high latency is one main drawback of the centralized approach. The latency cannot be readily scaled down; for instance, utilizing two vCPU cores (one core to handle packet receptions and another core to handle packet transmissions) does not reduce the latency (as we have verified with our own measurements of the centralized approach). To overcome the drawbacks of the centralized approach, our main CALVIN strategy is to distribute VNFs over a chain of VMs that run the network functions *either completely in the kernel space or completely in the user space*.

We also note for completeness that a specific combined kernel and user space approach, referred to as "Tuna", for a 5G wireless access point has been developed in [103]. The Tuna approach places 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 both the high-performance in-kernel network data path and user space (kernel bypass) techniques. At the same time, CALVIN avoids the overhead of context switching of the centralized approach [51]. This section presents the overview, architectural design, and workflow of CALVIN.

A. Overview

The underlying idea of CALVIN is to assess the nature of a VNF in terms of its complexity when processing packets. Accordingly, CALVIN decides to implement each VNF in either the kernel space or the user space. Each implemented VNF is then encapsulated inside a separate VM. These two CALVIN design choices completely eliminate the context switching overhead of the vCPU of each VM to process packets in different spaces and to schedule different VNF programs, thus reducing the overall end-to-end service latency.

The main advantages of this CALVIN strategy are:

- Reduce cost of context switching inside VM: As quantified in [104], context switching can produce direct and indirect costs. Direct costs 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 a VNF in a single space mitigates the negative effects of both costs on latency.
- Avoid copying packet data structures 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, data exchanges between spaces are critical bottlenecks for the VNF processing. Conducting all packet operations in a single space reduces data copying.
- Increase scalability and flexibility: In a centralized approach, such as [51], 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 contrast, the CALVIN performance can be scaled due to the flexible structure of the dynamic SFC; each new VNF can be assigned a specific VM with appropriately allocated resources for the current functional requirements.

B. VNF Classification

Based on these design imperatives, we take a first step towards the development of the CALVIN SFC management framework by classifying VNFs. According to the introduced VNF classification, a given VNF is either implemented in the kernel space or in the user space. The network functions are divided into three main categories in CALVIN:

1) Elementary (Skeleton) Functions: This type covers 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 virtual egress interfaces. These functions can be implemented in both kernel and user spaces.

2) Basic Functions: The main characteristics of basic functions are: i) Operations are mainly performed on the header (or metadata); not on the packet payload. Headers have typically small sizes; thus, header operations can be performed without iterative execution, which is not fully supported in the current eBPF and XDP versions. ii) The computational intensity and complexity are low so that the processing delay is limited to an acceptable range, even without acceleration mechanisms, such as Single Instruction, Multiple Data (SIMD), hugepages, or CPU cache prefetching, which are not fully available in most in-kernel frameworks. $iii)$ The implementation has no strict dependencies on specific running environments or frameworks.Basic functions with these characteristics, such as router, load balancer, network address translation (NAT), and packet filter, are suitable for kernel space implementation.

3) Advanced Functions: Compared to basic functions, advanced functions involve complex and compute-intensive operations with the following main characteristics: $i)$ Both packet header and payload need to be processed. $ii)$ Acceleration mechanisms in the user space are required to keep processing delays within reasonable ranges. $iii)$ The implementation of advanced functions typically requires specific runtime or execution environments. For example, the widely used open source network coding library Kodo [105] requires the C++ runtime environment, which is not available in the kernel space. According to these characteristics it is beneficial to implement advanced functions, such as data encryption, compression, and network coding, in the user space.

C. Selecting VNF Implementations for VNF Classes

Compared to the numerous kernel-bypassing approaches [94], [106], [107], such as `Netmap`, `PF_RING`, `DPDK`, relative few in-kernel fast packet IO approaches have been developed. To the best of our knowledge, XDP is currently the fastest in-kernel programmable network data path which provides bare metal packet processing at the lowest point in the software stack [84], [108]. Therefore, we select XDP for the in-kernel VNF implementation in CALVIN.

Based on a thorough literature review that covered [94], [106], [107] and related studies, we selected DPDK for the user space VNF implementation, mainly for the following reasons: i) High performance: According to the evaluation in [94], DPDK has demonstrated favorable bandwidth and latency performance. ii) Open source, low level with high configurability and very good documentation: This allows our VNF implementation to have full control of all processing functions invoked in the packet IO and to adopt design optimizations to reduce latency. For example, we have integrated the network coding library Kodo [105] with the native DPDK data structure to avoid data transfer overheads, thus achieving short latencies for advanced network coding VNFs. iii) Wide support: DPDK supports a wide range of physical, paravirtualized, and software NICs. The Open vSwitch also implements the DPDK fast path [54], simplifying the deployment of CALVIN on the OpenStack cloud platform, which uses Open vSwitch as default software bridge.

However, we acknowledge that bypassing the kernel with DPDK (Version 18.02) has several disadvantages [108]: i) The driver can only run in the polling mode. ii) Network and upper layer protocols require third-party implementations (whereas these protocols are already implemented in the mainline Linux kernel). iii) The Linux kernel has its own security model for managing the networking hardware. Bypassing the kernel requires a new security model in the user space, which is an important direction for future research.

D. Architecture Design

The architectural design of CALVIN is illustrated in Fig. 6. CALVIN is built on top of the research-oriented SFC framework SFC-Ostack [46], [52]. We extend the SFC-Ostack control and data plane components to enable latency optimization strategies:

- Control plane: We add the VNF classifier module which translates between the VNF description and the SFC description, including the VNF classification 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 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 the kernel space or in the user space. Packets are received from the ingress interface (vNIC), processed by the function program, and sent out through the egress interface (vNIC). VMs in the service chain are connected in a prescribed order by Open vSwitch with DPDK (OVS-DPDK).

Notice in Fig. 6 that CALVIN uses the distributed mapping with one VM for one VNF (and not the compact mapping of multiple VNFs in the same VM). This CALVIN design choice is mainly based on the reasoning in Section II-B.3 about the flexibility of VNF resource allocation and on the results in [25], which indicate that for non-trivial computational VNF tasks, i.e., in particular for our advanced functions, the distributed mapping achieves lower latencies.

E. Setup and Workflow

The CALVIN operation with the distinct processing of basic and advanced functions requires various settings for both hardware and software used by the OpenStack cloud platform.

1) Configurations for Accelerating Virtual Networking Infrastructure:

- The physical NIC of each node must support DPDK [101], [109] to deploy the official OVS-DPDK plugin of OpenStack.
- Each compute node should have dedicated CPUs assigned only for OVS-DPDK: The current version of OVS-DPDK works in polling mode and any interruptions from other processes can cause performance degradations and even lead to packet losses. Consequently, dedicated CPU cores must be configured for OVS-DPDK by using the Linux CPU isolation tool.
- 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 needs to allocate hugepages for the guest OS of each VM instance.

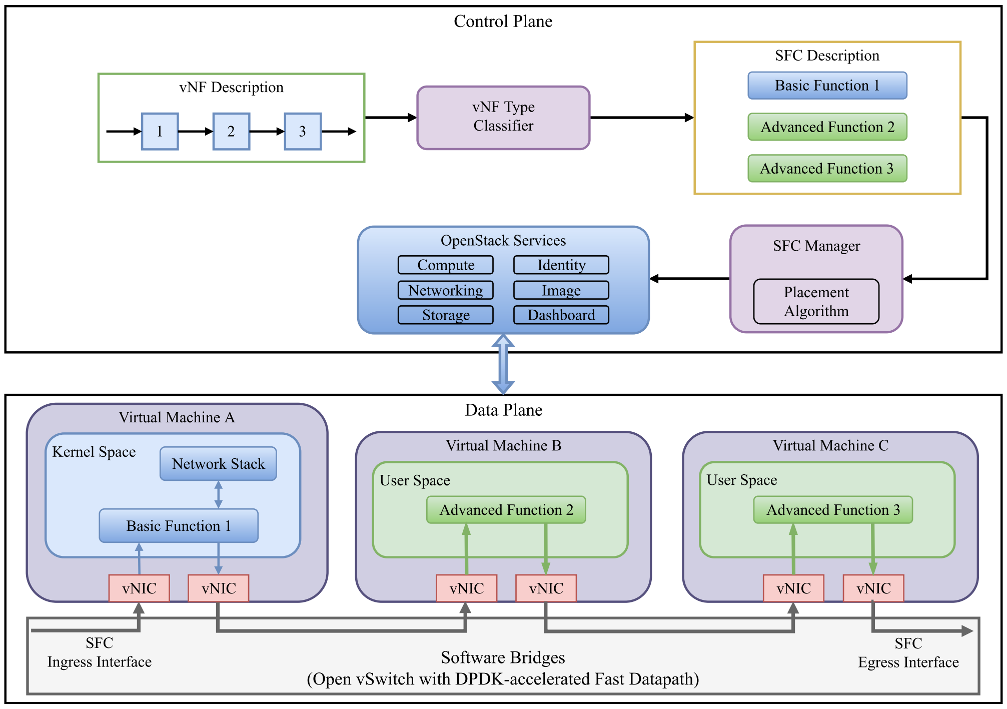


Fig. 6. Illustration of CALVIN architectural design, including fundamental components in control and data plane: The SFC manager utilizes OpenStack services to deploy VNFs (which are classified into basic and advanced functions) into a chain of VMs. The functions are implemented either in the kernel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

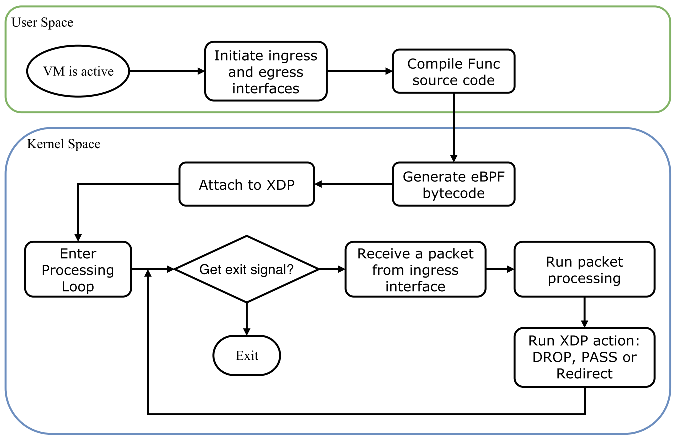


Fig. 7. CALVIN workflow for basic functions running in the kernel space.

2) VNF Processing Configurations: Figs. 7 and 8 present the workflow of running basic functions in the kernel space and running advanced functions in the user space; the full source code is available from [53]. Since the Kernel-based Virtual Machine (KVM) is the default hypervisor of the OpenStack computing service (until latest version: rocky) [110], the following configurations are deployed for VMs launched by the KVM hypervisor:

a) Kernel space:

- Due to the XDP requirements [111], the virtual interface of each VM should support the allocation of a dedicated transmit queue (TX queue), e.g., through the virtio_net patch [111] to OpenStack or through extending the OpenStack Nova project.
- The Linux kernel of the guest OS running inside the VM should be updated to at least version 4.8 to run the XDP program [112].
- The eBPF compiler framework BCC [112] should be installed to compile the XDP program and to attach the compiled program to the virtual interfaces.

b) User space:

- Virtual interfaces should be assigned with IOMMU for minimal latency overhead.
- Sufficient 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 transmission.
- The DPDK kernel module igb_uio needs to be loaded to enable the kernel Poll Mode Drivers (PMD) driver. This driver should be bound to the VM ingress and egress interfaces.

We note that exchanging data between VNFs purely in the user space in a given VM is not straightforward. Normally, Inter Process Communication (IPC) mechanisms e.g., the Unix Domain Socket, are used to exchange data between processes running on the same VM. However, the IPC mechanisms are provided by the OS, which inject packets back into the kernel space. In order to avoid being forced back into the kernel space for inter-VNF data exchanges, we adopt the distributed mapping of one VNF per VM for CALVIN. Both the centralized approach and CALVIN use fast DPDK frame IO (which is substantially faster than the conventional raw socket

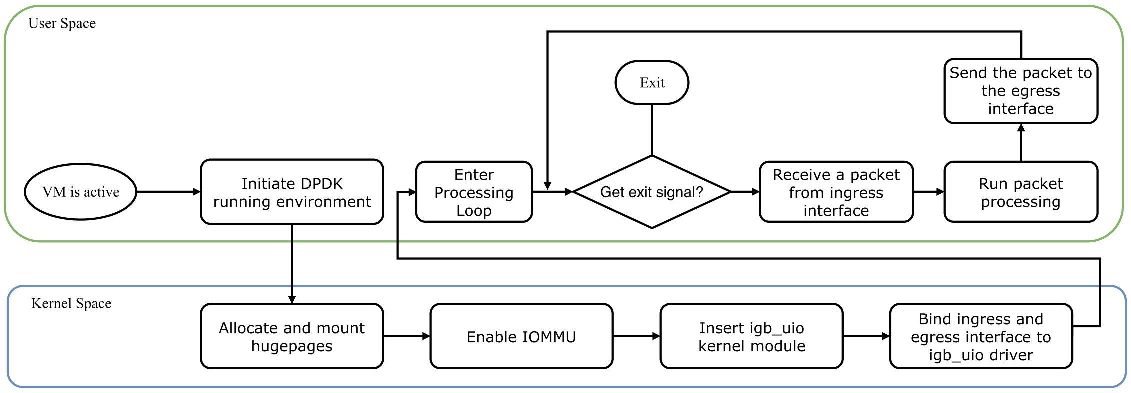

User Space
VM is active
Initiate DPDK running environment
Enter Processing Loop
Get exit signal?
Exit
Receive a packet from ingress interface
Run packet processing
Send the packet to the egress interface
Kernel Space
Allocate and mount hugepages
Enable IOMMU
Insert igb_uio kernel module
Bind ingress and egress interface to igb_uio driver

Fig. 8. CALVIN workflow for advanced functions running in the user space.

frame handling by the OS) for the ingress and egress vNICs. The critical difference between CALVIN and the centralized approach is that CALVIN avoids injecting packets back into the kernel space of a given VM.

We also note that legacy advanced VNF, such as network coding function programs, have been written for normal layer 3 or raw sockets, which are typically not directly compatible with DPDK. For CALVIN, we implemented the examined advanced VNFs that run in user space based on the DPDK architecture and data structures. Building advanced VNFs on the native DPDK data structures avoids data transfers, aiding in achieving short latencies.

3) Future Research Directions for CALVIN Workflow: The current CALVIN version developed and evaluated in this study employs the distributed mapping of one VM per one VNF. The data is exchanged between VMs with low-latency OVS-DPDK software bridging, see Fig. 6, and is processed with the respective workflows illustrated in Figs. 7 and 8. The software bridging avoids the latencies due to the conventional IPC mechanisms provided by the OS. Ongoing research and development of kernel space and user space processing may enable low-latency data exchanges between distinct VNFs purely in the kernel space or purely in the user space. For instance, eBPF supports tail calls and maps to chain and to exchange data between multiple eBPF programs purely in the kernel space. Future research could design and evaluate low-latency XDP-based in-kernel SFC mechanisms that could process the elementary and basic VNFs. In the user space, the principle of shared memory regions [113], [114] could enable low-latency data exchanges between VNFs and form the foundation for low-latency DPDK-based user space SFC mechanisms for processing advanced VNFs.

Another (albeit less important) motivation for the distributed mapping in the current CALVIN version is the flexibility of dynamic resource scaling of distributed VMs. Dynamic VM resource scaling without service downtime can be achieved through live resizing, which has recently become feasible in the commercial VMware integrated OpenStack [115], but is not yet fully supported in OpenStack [116]. Alternatively, a new VM with more vCPUs can be provisioned when VNFs become bottlenecks. The compact mapping with all VNFs operating on a single VM requires that the old VM and the new VM operate simultaneously during the transition phase, which can be demanding for the typically limited memory resources of cloud compute systems. The distributed mapping allows the transition to be performed sequentially, with only the old VM and new VM supporting one VNF operating simultaneously. Nevertheless, the ongoing advances may make dynamic resource scaling (live resizing) of VM instances common in the future. It would then be interesting to examine live resizing in the context of low-latency VNF processing.

As low-latency inter-VNF data exchanges purely in the kernel or user space in a given VM and live resizing of VMs mature, it would be interesting to develop a compact mapping version of CALVIN that processes all basic VNFs in a single kernel-space-focused VM and all advanced VNFs in a single user-space-focused VM. It would then be insightful to compare this compact mapping CALVIN version with the distributed mapping CALVIN that is examined in this paper.

IV. Performance Evaluation of Elementary and Basic Functions

In order to evaluate the feasibility of CALVIN for the 1 ms control loop of the tactile Internet, we first conduct measurements for elementary and basic functions with a minuscule computational workload in this section. The measurement results for elementary and basic functions indicate the baseline latencies. The evaluation methodology and testbed are extended from the testbed introduced in our conference paper [46].

A. Measurement Set-Up

Fig. 9 illustrates the measurement set-up. VMs are launched on two compute nodes to measure the latency of the elementary service loop introduced in Section II-A. We use an active measurement strategy to measure the latency for UDP traffic. We consider the UDP protocol for the probing traffic since UDP is typically used for low-latency applications. Latency measurements for TCP are difficult due to the complex TCP flow control and congestion control. UDP traffic provides us with full control of both the sending and receiving processes.

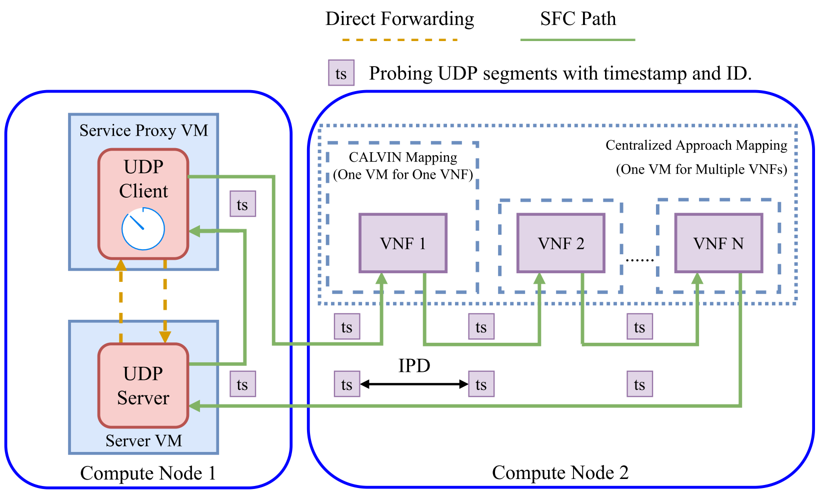

Direct Forwarding
SFC Path
ts Probing UDP segments with timestamp and ID.
Service Proxy VM
UDP Client
ts
UDP Server
Server VM
Compute Node 1
CALVIN Mapping (One VM for One VNF)
Centralized Approach Mapping (One VM for Multiple VNFs)
VNF 1
VNF 2
VNF N
IPD
Compute Node 2

Fig. 9. Illustration of RTT measurement set-up: The service proxy and the server are allocated on compute node 1. Multiple VNFs are launched on compute node 2 to implement the SFC for the probing UDP traffic generated by the service proxy. The RTT is measured at the UDP client.

1) Architecture: The measurement setup is based on the service loop in Fig. 3. The service proxy runs the UDP client, which sends probing packets to the server located on the same compute node 1. The server simply bounces all received packets back to the client as fast as possible. Alternatively, probing packets can be forwarded directly to the server to measure the direct forwarding latency (without any VNFs) of the underlying network infrastructure.

In order to reflect authentic practical networking scenarios, the service proxy VM and the server VM do not use fast IO technologies, such as DPDK or XDP. In particular, the service proxy and the server are normally working in the network layer (in contrast to VNFs running in the data link layer). Therefore, we employ the conventional Linux networking stack in the service proxy VM and server VM, so as to capture the latencies introduced by the networking layer.

Compared to the centralized measurement setup in [86], the components of our measurement architecture are distributed over two different physical nodes, which mimics practical cloud environments.

A time stamp (ts) and an identification number (ID) are added before the payload part of each probing packet. The time stamps are used for the latency calculation. The IDs are used to identify out-of-order packets which would indicate the accumulation of packets in queues [117]. We set the packet traffic rates sufficiently low, see Section IV-A.5 to avoid out-of-order packets.

2) Testbed: All measurements were performed on our NFV testbed consisting of off-the-shelf computers that are connected by two separate Gigabit Ethernet networks. Each computer had 4 CPU cores (Intel 4th Generation Core i5), 16 GB RAM, 128GB SSD, and two Gigabit NICs (Intel 9301CT Gigabit CT). OpenStack (Pike version) [118] was deployed on these computers, installed with the Ubuntu Server 16.04 TLS operation system. 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, networking, identification, and storage services of OpenStack, the official SFC and OVS-DPDK plugins were installed. In particular, we used Virtio [119] for the vNIC and OVS-DPDK as the virtual bridge. For the management of the virtual instances, KVM was used as the hypervisor and the customized Ubuntu cloud image was used to implement different VNFs.

3) Elementary and Basic VNFs: We implemented and measured elementary forwarding and two basic VNFs.

FWD Elementary Forwarding (FWD): Packets are retrieved from the VNF ingress interface and directly forwarded to the egress interface without any operations. FWD is an elementary function of each VNF and other VNF functions are built on top of the FWD function.

ATS Appending Time Stamps (ATS): The timestamps of the reception and transmission of a given packet by the current VNF are appended to the end of the UDP payload of the packet just before the packet is sent out. The ATS function modifies the payload size; therefore, the layer 3 and layer 4 header checksums must be recalculated. In order to ensure fair latency comparisons, we recalculate the checksum with standard methods in software (and do not employ checksum offloading). Thus, the ATS function can be used to estimate the latency introduced by a trivial operation on the packet payload.

XOR XORing UDP payload (XOR): This function performs an XOR operation with the same static key on all bytes of the UDP payload. Compared to ATS, the XOR function can be used to estimate the additional latency introduced by a non-trivial computational operation on the packet payload.

We make the source code of all VNF implementations openly available at [53].

4) Metrics:

a) Latency: We adopt the Round-Trip-Time (RTT) (i.e., the per-packet delay) of each UDP packet sent by the

UDP client program running on the service proxy VM as the latency metric for the following reasons: i) The RTT includes the delay introduced by the forward and backward paths. ii) The RTT measurement does not require time synchronization between the VMs. (According to the experiments in [120], VM-level time synchronization is error prone due to the interference between the VMs.) Note that the measured RTT includes all latency components (1)–(5) illustrated in Fig. 4 across the entire SFC.

b) Bandwidth: We measure the maximum achieved bandwidth (packet throughput in bit/second) with iPerf (version 2) in the UDP mode [121] (We did not use iPerf for the RTT measurements because iPerf currently does not support per-packet delay measurements; we wrote our own Python tools [53] for the per-packet RTT measurements.) We run an iPerf UDP bounce server on the server VM and an iPerf UDP client on the service proxy VM (all on compute node 1 in Fig. 9). Through trials that manually adjusted the iPerf client target bandwidth in steps of 10 kbit/s, each lasting for five minutes (with ten independent repetitions), we determined the maximum bandwidth such that there are no lost or out-of-order packets detected at the iPerf client side.

5) Active Measurement Parameters: Active measurements require the setting of two UDP traffic parameters, namely the Inter-Packet-Delay (IPD) and the payload size. Based on our preliminary measurements (which are not included in detail due to space constraints), relative small IPDs on the order of a few milliseconds give consistent RTT values. Much shorter IPDs and correspondingly high traffic rates would lead to queueing for the VNF processing, as analyzed in [57], [58], [61], [63], [64], [66]–[68]; in contrast, our measurements focus on the latencies of lightly loaded systems without significant queueing. The RTT values slightly increase with increasing IPD, which is mainly due to OS batching mechanisms and the queuing mechanisms of the underlying virtual bridges that are activated for low network traffic loads. To avoid the additional latency introduced by batching (queuing) and to ensure consistent measurements, we set the IPD to 5 ms in all measurements.

We select 256 and 1400 bytes as the lower and upper limits of the payload size. According to our preliminary evaluations, UDP segments with less than 256 bytes of payload cannot be properly processed by XDP. At the same time, the official SFC plugin of OpenStack [122] currently does not support jumbo frames. The maximum UDP payload size depends on the Maximum Transmission Unit (MTU) of the underlying physical network. For the default MTU of Ethernet of 1500 bytes, the maximum UDP payload size is limited to 1472 bytes ($1500 - 20$ (minimum 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.

For each scenario, the measurements were repeated 50 times; for each measurement, 500 UDP segments were sent by the probing client.

6) Measurement Scenarios:

a) Comparison of different VNF technologies: We selected XDP [84] to implement VNFs in kernel space and DPDK [101] to implement VNF in user space for CALVIN (see Section III-C). To benchmark the latencies of these two selected technologies, we also consider two other frequently considered VNF technologies, namely Linux Kernel Forwarding (LKF) as a kernel space technology and the Click modular router [123] as a user space technology.

XDP: The XDP workflow flow is outlined in Section III-E. Due to following XDP restrictions, only the FWD and XOR functions with the 256 bytes UDP payload size are implemented: i) The maximum number of instructions per XDP program is currently 4096 BPF instructions [124], limiting the complexity of the computational operations as well as the data amounts that can be manipulated. ii) The range of the XDP operational memory for each packet is limited by the size of the original received packet. Operations outside this range are prohibited. Thus, the ATS function cannot be implemented with XDP.

DPDK: The high flexibility and programmability of DPDK allow all three functions to be implemented as DPDK applications. By default, DPDK applications run in polled mode and can consume 100% of the CPU resources. The CPU and corresponding power consumption could be reduced with a sleeping mechanism. To minimize the processing delay, we set the number of packets in a processed batch (burst) to one.

LKF: Linux Kernel Forwarding (LKF) is a built-in Linux kernel feature for network-layer packet forwarding [125]. Due to its trivial operations, LKF is one of the fastest functions running in the kernel space. Since LKF does not provide any programmability, LKF cannot be used to implement different VNFs in the kernel space.

Click: The Click modular switch is a software framework for building flexible and configurable routers [123]. Compared to the low-level IO operations offered by DPDK, Click provides multiple encapsulated packet processing modules called elements to build processing pipelines. Elements can be connected with a directed graph described in a router configuration file. The evaluated VNFs are implemented by combining built-in and customized elements.

For the VNF technologies comparison, we launch a single VM on compute node 2 in Fig. 9 to run the network function.

b) Comparison of centralized approach [51] vs. CALVIN: We benchmark the proposed CALVIN approach against the state-of-the-art centralized approach [51], which was one of the first well-studied approaches for implementing advanced functions, such as network coding, as VNFs. (We re-implemented the centralized approach as its source code was not publicly available.) In order to examine the distributed VNF aspect of CALVIN, we consider two FWD VNFs on two separate VMs for CALVIN, while for clarity of comparison we implement only one FWD VNF in the centralized approach. We give the centralized approach a VM with twice the computational resources (vCPU and memory) compared to each individual VM used by CALVIN.

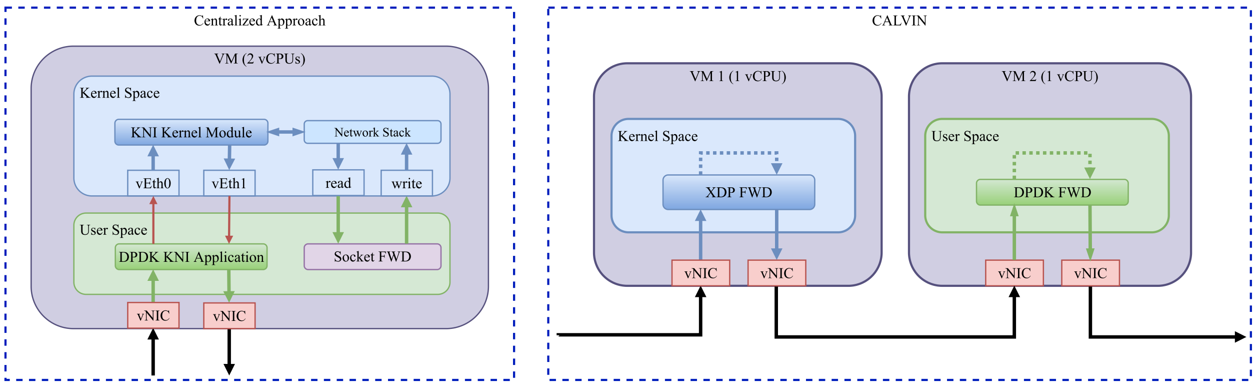


Fig. 10. Detailed illustration of measurement set-up for the RTT comparison between centralized approach and CALVIN on compute node 2 of Fig. 9.

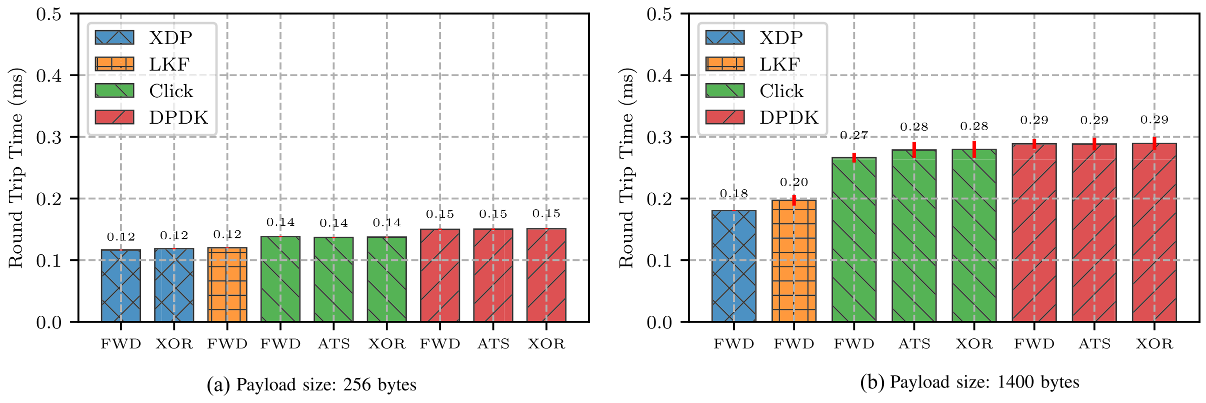


Fig. 11. Means and 95% confidence intervals for round-trip time (RTT) of different VNF technologies in kernel space (XDP and LKF) and user space (Click and DPDK). The 95% confidence intervals for the 256 byte payload size are very tight and barely visible in the plot.

Thus, the comparison is effectively between two FWD VNFs in CALVIN and one FWD VNF in the centralized approach, whereby both approaches have the same computational resources. More specifically, for the centralized approach, which is illustrated in the left part of Fig. 10, packets enter the VM through the left vNIC, traverse the FWD VNF and exit through right vNIC. For CALVIN, which is illustrated in the right part of Fig. 10, packets enter the left vNIC of VM1, traverse the XDP FWD, exit through the right vNIC of VM1, enter the left vNIC of VM2, traverse the DPDK FWD, and exit through the right vNIC of VM2.

B. Results

1) RTT Measurements of Elementary and Basic VNFs for Different VNF Technologies: The average values and 95% confidence intervals of the measured RTTs of the elementary and basic VNFs (one FWD VNF, one ATS VNF, or one XOR VNF) implemented with different VNF technologies on compute node 2 are plotted in Fig. 11. We observe from Fig. 11 that the RTTs for the two basic VNFs, namely ATS and XOR, are equivalent to the respective RTTs for the elementary FWD VNF for all VNF technologies. This result implies that the additional latencies for the payload processing in the basic ATS and XOR VNFs are negligible compared to the elementary FWD latencies.

Examining closely the in-kernel VNF technologies, we observe from Fig. 11 that although XDP and LKF have the same FWD RTTs for small packets, XDP FWD is around 10% faster than LKF FWD for large packets. Moreover, we observe from Fig. 11 that the RTTs for in-kernel processing (XDP and LKF)) and user space processing (Click and DPDK) are very similar for the small 256 bytes packet size. On the other hand, for the large 1400 bytes packet size, user space processing incurs substantially longer (about 50% longer) RTTs than in-kernel processing.

These observed latency differences appear to be primarily due to the two main types of overhead introduced by basic functions: (i) The overhead of copying frames [126] from the virtual NIC ring buffer to the VM memory, and (ii) the overhead of additional metadata pre-processing of the frames. In the Linux networking stack (LKF), a data structure `sk_buff` containing some metadata is allocated for each received frame. The latency overheads for extracting the metadata and allocating `sk_buff` are non-trivial [24], [84], [127]. In contrast, XDP operates at the lowest point in the Linux software networking stack, without allocating a metadata data structure, nor any parsing and pre-processing of packets. XDP supports the packet forwarding to another interface (XDP_REDIRECT Action) without the metadata processing and without checking the routing table of the kernel (MAC addresses are provided and written by the eBPF program). For the conventional Linux kernel forwarding (LKF), the metadata structures must be created and the kernel requires checking the routing table to write the proper MAC addresses. Thus, LKF is slightly slower than XDP for large packets (which require more time to extract the metadata). In summary, both XDP and LKF copy the frames into kernel space memory; XDP does not allocate metadata structures, whereas LKF does allocate metadata structures.

Turning to the user space technologies, Click and DPDK copy the frame data into user space memory and allocate different metadata structures. In particular, in DPDK (kernel bypassing), the Poll Mode Driver (PMD) [128] copies frames from the NIC ring buffer to the DPDK user space memory pool (a pre-allocated fixed-length memory using hugepages which is resident in the physical memory [101], [129]). During this copying, an additional `mbuf` data structure [130] containing metadata is created for each frame. These metadata are required by DPDK for further processing and many advanced features. For large packets, this copying into the user space memory pool and metadata structure creation can incur significantly longer latencies compared to the in-kernel approaches. Click involves similar latency-increasing copy and metadata operations.

2) Comparison Between Centralized Approach and CALVIN:

a) RTT: Fig. 12(a) presents the RTT measurement results for the elementary FWD VNF. As presented in Fig. 9, the direct forwarding is the baseline of the transmission delay introduced by the underlying virtual networking infrastructure. We observe from Fig. 12(a) that the centralized approach exceeds the 0.35 ms delay threshold in the best case, while the CALVIN RTT is below the 0.35 ms threshold with about 70% probability for 1400 bytes packets, and with nearly 100% for 256 bytes packets. The measured average (mean) RTTs (accounting for all latency components (1)–(5) in Fig. 4) for the 256 bytes and 1400 bytes packets are as follows: CALVIN: 0.19 ms and 0.32 ms, respectively; centralized approach: 2.30 ms and 2.39 ms, respectively. Thus, we conclude that CALVIN can meet the low latency requirement introduced in Section I and allow additional data processing inside the virtualized environment.

We remark that the measurements without fast packet IO in [25] indicated that for VNFs with very low computation demands, the compact mapping gave lower latencies than the distributed mapping, while for computationally intensive VNFs, the distributed mapping gave lower latencies. Our latency measurements with our fast packet IO based CALVIN demonstrate that the distributed mapping achieves low latencies within the latency constraints of the tactile Internet for elementary and basic VNFs with low computational demands. Intuitively, with the accelerated packet IO and the correspondingly high supported packet rates and high CPU resource consumption for fast packet IO (compared to the conventional slow networking stack), even VNFs with low computational processing demands do no longer present a trivially low computation load on the CPU. Thus, we believe that the distributed mapping is a good design choice for implementing all classes of low-latency VNFs in CALVIN. The distributed mapping allows each VNF to focus on conducting its processing tasks quickly in its own space and leaves the communication to the underlying software integration bridge, which has mature low-latency performance. The allocation of one VM per VNF with the distributed mapping may be considered wasteful; however, each VM could be optimized for its usage and the distributed mapping is consistent with the emerging unikernel concept [131], [132]. For example, if the packet processing occurs mainly in the userspace, then the kernel components could be slimmed down to make the kernel space as small as possible.

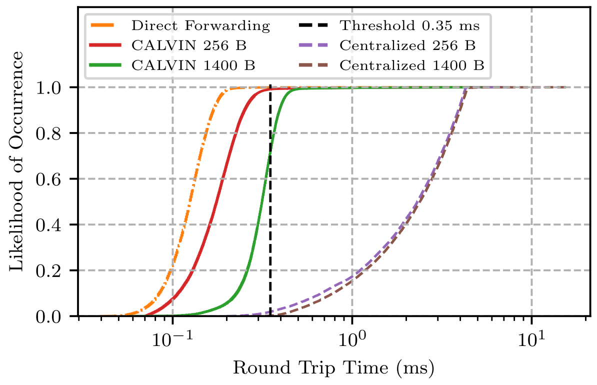
b) Bandwidth: Fig. 12(b) presents the bandwidth measurement results for payload sizes ranging from 256 to 1400 bytes. We observe from Fig. 12(b) that the bandwidth supported by the centralized approach is substantially higher than the CALVIN bandwidth, especially for large packets. For a payload size of 1400 bytes, the centralized approach bandwidth is over 15 times higher than the CALVIN bandwidth. Compared to the centralized approach with a minimal supported bandwidth around 6 Mbits/s, the maximal CALVIN bandwidth is in the range from 1.4–1.7 Mbits/s.

c) Latency-bandwidth trade-off: Taken together, the RTT results in Fig. 12(a) and the bandwidth results in Fig. 12(b) demonstrate the tradeoff between per-packet latency and throughput in real system implementations. The centralized approach uses the standard Linux socket implementation which was designed primarily for high-throughput best-effort service applications, e.g., file transfers, which have typically bursty traffic. Accordingly, the kernel networking stack includes a range of throughput enhancing mechanisms, such as batch processing. Batch processing collects multiple packets and then processes the batch of packets with acceleration methods, e.g., with single instruction multiple data (SIMD) instructions and enhanced CPU caching. Batch processing increases the per-packet delay as the first packet in a batch must wait for subsequent packets to fill up a batch before processing commences.

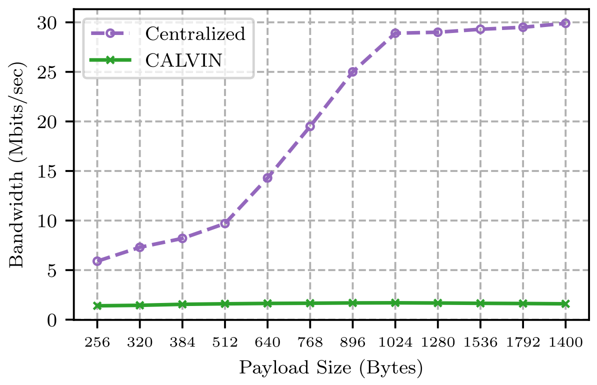
The batch processing in the Linux networking stack slows down the centralized approach, even if no batch processing is employed in the DPDK user space application. This is because the KNI injects packets back into the normal Linux kernel networking stack, where batching is employed. The batching in the kernel stack cannot be avoided without modifying the kernel source code.

CALVIN avoids this batch processing in the kernel space by employing fast path technologies, e.g., XDP, for in-kernel VNF implementation. Since our goal is to reduce the per-packet delay, CALVIN avoids batch processing in all VNF implementations. The VNF implementations in CALVIN operate in a run-to-completion mode, i.e., they retrieve one packet, process it (batch size of one packet), and send it out as quickly as possible. CALVIN thus reduces the per-packet latency, as observed from Fig. 12(a), at the expense of supporting a lower packet throughput, as observed in Fig. 12(b).

For the tactile Internet for human-machine co-working, low per-packet delay is typically much more important than support for high bandwidth. The human-machine co-working packet traffic, e.g., the control messages of a robot arm are typically small so that support for low bandwidths is sufficient. Also, the 5 ms IPD is sufficient as a sample period of an angle sensor for a typical pendulum application,



(a) RTT for packets with 256 bytes and 1400 bytes.



(b) Supported bandwidth

Fig. 12. FWD VNF performance comparison: Proposed CALVIN (two FWD VNFs, namely XDP FWD and DPDK FWD, see right side of Fig. 10) vs. the state-of-the-art centralized approach [51] (one FWD VNF, see left side of Fig. 10), whereby both CALVIN and centralized approach utilize same computational resources (two equivalent vCPUs).

TABLE I
CPU USAGE OF THE PHYSICAL COMPUTE NODE

Approach	User (%)	Sys (%)	Guest (%)	IDLE (%)
Centralized	25.2	47.2	2.9	24.7
CALVIN	25.2	23.0	2.2	49.6

as illustrated by the right side of Fig. 1. On the other hand, control messages that are delayed by batch processing can profoundly disrupt human-machine co-working. Therefore, CALVIN trades support for only low bandwidth for reduced per-packet latency. The bandwidth supported by CALVIN can be improved in future work with bandwidth management mechanisms, e.g., a load balancer that distributes a packet flow over a set of duplicated VNFs to enable parallel processing. Parallel processing requires a fast per-flow load balancer and a traffic merger to handle out-of-order packets arising from the parallelism.

d) CPU resource usage: We measured the usage of the four cores of the physical CPU on compute node 2 in Fig. 9 with `mpstat` [133]. Since the scheduling of CPU resources for all running VMs is managed by the OpenStack compute service (Nova), the global average usage of all cores has been measured. We measured with a sample period of 1 second for a duration of 10 minutes. Table I shows the CPU usage levels of the centralized approach and CALVIN at the user level (User), kernel level (Sys), and for a niced guest (Guest).

We observe from Table I that compared to CALVIN, the centralized approach doubles the CPU resources consumed at the kernel (Sys) level. KVM uses the Linux kernel of the host OS as the hypervisor and uses POSIX threads to implement vCPUs of the guest OS [134]; thus, the Sys CPU usage reflects the usage of the vCPU of the VM running the VNF. By avoiding the overhead of context switching and metadata processing of each vCPU, CALVIN significantly reduces the kernel (Sys) CPU usage of the host OS. Thus, CALVIN leaves a significantly higher proportion of time of the physical CPU idle. This higher CPU idle time achieved by CALVIN can be utilized to run cloud management services, software bridges, and other essential background processes to stabilize the latency performance.

V. EVALUATION OF COMPUTATION-INTENSIVE ADVANCED VNFS

The evaluations in Section IV-B.2 indicated that CALVIN can complete elementary VNFs with an RTT (within the MEC) on the order of 0.32 ms. Thus, considering the 0.35 ms MEC latency budget from Fig. 2 there is still a remaining latency budget of about 0.02 ms for some advanced VNFs. This section evaluates network coding and encryption as two examples of advanced VNFs that have relatively high computational demands. We first describe the practical applications and relevance of these two VNFs. Then, we use the test setup from Section IV-A to evaluate the processing delay incurred by these advanced VNFs. The purpose of this evaluation is to assess whether the RTT reduction for elementary VNFs achieved by CALVIN is sufficient to permit practical advanced VNFs within the latency requirements of the tactile Internet.

A. Network Coding

Network coding linearly combines several original packets with coding coefficients to form encoded packets that are transferred through the network [135]. In Random Linear Network Coding (RLNC), the coding coefficients 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 [136], ii.) versatile coding matrix, permitting sparsity (judiciously added zeros) to reduce computation complexity [137], [138], iii.) low latency support due to on-the-fly coding capabilities [139], [140], iv.) support of heterogeneous field sizes for communication entities, increasing flexibility in heterogeneous contexts [141], and v.) reduced overhead between the storage and transmission layers, as the same code can also be used for distributed storage [142].

Network coding research has proposed many different RLNC variants. We focus on the two main RLNC types, namely systematic block codes and convolutional sliding window codes. Block-based RLNC was introduced to reduce the computational requirements and control for network coding [143]. To further improve the performance, a systematic code does not code every packet, but sends original packets as "coded" packets [144]. The packets built from linear combinations are then sent in between original packets or at the end of the block [145].

Sliding window network coding has been introduced to reduce the in-order delay of coded transmissions [146]. In the form of a systematic code with a limited coding window, sliding window network coding has shorter in-order delay compared to block codes, while generally requiring comparable computational resources [140].

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 network nodes, which are currently only used for switching and routing decisions. However, NFV and SDN provide new flexibilities for deploying innovative functions within a network [16]. With NFV, network coding can be implemented for abstract VMs or containers, which can be instantiated at arbitrary NFV-capable network nodes. In addition, SDN can direct the data flows towards the network coding VNFs and orchestrate them in an SFC [147], [148]. However, the latency of network coding as a VNF in a general-purpose MEC system has to the best of our knowledge not been previously examined in detail.

Our per-packet processing delay measurements consider the encoding (which is computationally equivalent to recoding in a network node) with a Galois field size of $GF(2^8)$ and 25% redundancy. We consider block coding with a block size of 32 packets and sliding window coding with a window size of 8 packets.

B. Encryption

Encryption and decryption are critical security components in communication, ensuring the confidentiality and integrity of the transferred data. More than 40% of the web traffic is transported in encrypted form over HTTPS, with an increasing trend [149]. As a result, decryption is required for a multitude of network functions, e.g., caching and deep packet inspection. As with network coding, encryption requires the entire payload to be processed which is a considerable computational effort. We focus on the Advanced Encryption Standard (AES), which is a commonly used encryption standard for data transfers and storage.

A previous study showed a prohibitive end-to-end latency of at least 30 ms for encryption as a VNF [150]. However, this previous study did not take advantage of fast packet processing mechanisms, such as DPDK. Other VNF implementations of AES encryption have used Graphics Processing Units (GPUs) to increase the throughput and scalability [151], [152], while incurring latencies of over 150 μs [151]. In contrast, our CALVIN approach enables encryption of small packets within a 20 μs delay budget on a general-purpose MEC system.

C. Measurement Set-Up

Compared to the measurement set-up for elementary and basic functions described in Section IV-A, the following additional considerations are required for evaluating advanced functions:

1) VNF Implementation: Due to the limitations of in-kernel technologies, CALVIN uses DPDK to implement all advanced VNFs. Both network coding and AES encryption functions are implemented on top of the elementary DPDK FWD application. We implemented network coding with the Network Coding Kernel Library (NCKernel), which is built on top of the Kodo library [105], to support the different variants of network coded communications. We used the portable AES Implementation Tiny-AES-C [153] to build the encryption application.

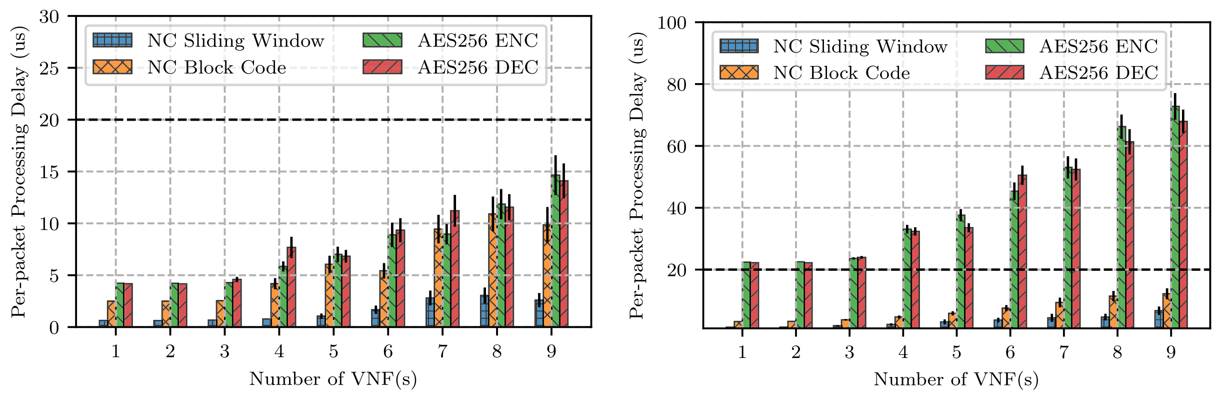
We implement multiple VNFs in parallel (each VNF on its own VM according to the CALVIN architecture principles, see Section III-D) so as to evaluate the scalability of our VNF implementation. Scalability is a key performance indicator for virtualization systems since a key aspect of virtualization is to run multiple virtual instances on limited hardware resources.

2) Metric: Since the RTT of elementary forwarding has been evaluated in Section IV, the measurements for the advanced VNFs focus on the packet processing delay. We define processing delay as the time duration required for the complete processing of a packet by a VNF, i.e., as the latency component (1) in Fig. 4. For VNFs that can generate redundant packets, such as network coding, this processing delay also includes the time duration required to create redundant packets.

3) Methodology: In order to evaluate the impact of VNF processing demands on the processing delay, the computational operations should be performed in parallel. This requirement is very challenging if the probing traffic is generated by a remote VM. Accurate synchronization mechanisms would need to be deployed on the virtualized networking infrastructure to ensure that probing packets arrive at each VNF at the same time. Therefore, instead of using an additional client to generate probing UDP traffic, for the evaluation of the advanced VNFs, the UDP segments are generated locally by each allocated VM. The locally generated traffic ensures that the VMs are continuously backlogged so that we obtain the worst-case processing delay: Every VM is always busy working and the OpenStack scheduler needs to handle the resource allocation among them. 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

The evaluation of elementary functions in Section IV-B.2, indicated a mean delay of 0.32 ms for the elementary FWD VNF of 1400 bytes packets in CALVIN. Considering the MEC latency budget of 0.35 ms from Fig. 2 and a safety margin



(a) Payload size: 256 bytes

(b) Payload size: 1400 bytes

Fig. 13. Means and 95% confidence intervals for processing times in microseconds for computationally intensive advanced VNFs: Per-packet processing latency in given VNF (latency component (1) in Fig. 4) on a given VM as a function of the number of VNFs running in parallel (with one VNF per VM) on one compute node with four CPU cores.

around 0.01 ms, we consider a latency budget of 20 μs for the advanced function processing (for smaller packets this latency budget could be larger as 256 bytes packets had only 0.19 ms mean RTT and 0.25 ms 90%ile RTT in Section IV-B.2).

Figure 13 shows the measured processing times for small 256 bytes packets and large 1400 bytes packets. For small packets, the processing time is within the 20 μs 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 CPU cores that are available exclusively for VM processing (one of the available four CPU cores is heavily utilized by the OVS-DPDK software bridge, which runs in polling mode with default DPDK functionalities). The latency increase is due to the contention for CPU resources. For a prescribed maximum latency requirement, e.g., 5 μs, we can read off the number of permitted parallel running VNFs, e.g., three VNFs. With a load balancer redirecting a given flow to multiple VNFs (and VMs), the larger numbers of supported VNFs would correspond to increased supported throughput.

For the large 1400 byte packets, we observe from Figure 13 increased processing times compared to the small 256 bytes packets. While the processing time for network coding remains relatively low and well within the 20 μs budget, encryption is not feasible even when the VNFs have exclusive access to a CPU core, i.e., for three or less VNFs. For network coding, the sliding window code has substantially shorter processing times than the block code. Even for large packets and high contention for the CPU resources, e.g., for nine parallel VNFs, the sliding window network coding delays remain below 7 μs.

VI. Conclusion

We have designed, implemented, and evaluated Chain bAsed Low latency VNF ImplemeNtation (CALVIN), an approach for managing distributed service function chains (SFCs) for low-latency tactile Internet applications. CALVIN implements virtual network functions (VNFs) either in the kernel space (if VNFs require only simple processing) or in the user space (if VNFs require advanced processing) so as to avoid transmissions between kernel space and user space for processing a given VNF. CALVIN further implements VNFs in a distributed manner with one VNF per VM and employs fast packet input/output (IO) to avoid the metadata and batch processing of the conventional Linux network stack.

We initially measured the elementary forwarding latencies of various current VNF implementations. We found that the eXpress Data Path (XDP) achieved latencies of 120 μs for small payloads and 180 μs for large payloads, while the native Linux kernel incurred about 10% higher forwarding latencies. The Data Plane Development Kit (DPDK) approach and the Click router approach have up to 50% higher latencies than XDP. Based on these measurements, we adopted XDP for implementing computationally simple VNF in CALVIN, while we adopted DPDK for implementing computationally complex VNFs in CALVIN.

We extensively benchmarked CALVIN against the state-of-the-art centralized SFC management approach [51], which processes a given VNF with both the kernel space and the user space. Our measurements demonstrated that CALVIN achieves significantly shorter latencies (0.32 ms mean latency for 1400 byte packets) for an SFC consisting of two distributed elementary forwarding VNFs (XDP forwarding and DPDK forwarding) compared to a single elementary forwarding VNF in the centralized approach (2.39 ms for 1400 byte packets). On the downside, CALVIN supports only a lower packet throughput (bandwidth) of around 1.5 Mbit/s than the centralized approach (between 6 and close to 30 Mbit/s depending on the packet size). CALVIN thus trades in reduced packet throughput in order to achieve shorter per-packet latency, which is required for typical tactile Internet applications with a 1 ms round-trip delay budget.

There are many important future research directions for SFC management in the tactile Internet. The implementation and measurements reported in this article have focused on the network function virtualization (NFV) in the MEC, i.e., the rightmost dashed box (the MEC cloud) in Fig. 2. Future research could integrate the MEC into a holistic 5G testbed that encompasses the entire end-to-end sensor-to-actuator loop in Fig. 2. Another direction is to examine novel "compute and forward" functions, such as video frame preprocessing for object detection [154] or transcoding [155], that bring more intelligence to the network edge [156]. The video preprocessing with limited edge cloud computing could extract key information to reduce the amount of data that needs to be transmitted through the network to the remote computationally powerful cloud.

REFERENCES

[1] K. Antonakoglou, X. Xu, E. Steinbach, T. Mahmoodi, and M. Dohler, "Toward haptic communications over the 5G tactile Internet," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3034–3059, 4th Quart., 2018.

[2] H. Chen *et al.*, "Ultra-reliable low latency cellular networks: Use cases, challenges and approaches," *IEEE Commun. Mag.*, vol. 56, no. 12, pp. 119–125, Dec. 2018.

[3] K.-C. Chen, T. Zhang, R. D. Gitlin, and G. Fettweis, "Ultra-low latency mobile networking," *IEEE Netw.*, to be published.

[4] O. Holland *et al.*, "The IEEE 1918.1 'tactile Internet' standards working group and its standards," *Proc. IEEE*, to be published.

[5] Z. Hou, C. She, Y. Li, T. Q. Quek, and B. Vucetic, "Burstiness-aware bandwidth reservation for ultra-reliable and low-latency communications in tactile Internet," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 11, pp. 2401–2410, Nov. 2018.

[6] A. Nasrallah *et al.*, "Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 88–145, 1st Quart., 2019.

[7] Y. Yang and A. M. Zador, "Differences in sensitivity to neural timing among cortical areas," *J. Neurosci.*, vol. 32, no. 43, pp. 15142–15147, Oct. 2012.

[8] L. Zhang, H. Gao, and O. Kaynak, "Network-induced constraints in networked control systems—A survey," *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 403–416, Feb. 2013.

[9] X.-M. Zhang, Q.-L. Han, and X. Yu, "Survey on recent advances in networked control systems," *IEEE Trans. Ind. Informat.*, vol. 12, no. 5, pp. 1740–1752, Oct. 2016.

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

[11] 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. 39547–39563, 2018.

[12] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2429–2453, 3rd Quart., 2018.

[13] 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 J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 469–484, Mar. 2018.

[14] S. Fu, J. Liu, and W. Zhu, "Multimedia content delivery with network function virtualization: The energy perspective," *IEEE Multimedia*, vol. 24, no. 3, pp. 38–47, Jul./Sep. 2017.

[15] Y. Harchol, D. Hay, and T. Orenstein, "FTvNF: Fault tolerant virtual network functions," in *Proc. ACM Symp. Archit. Netw. Commun. Syst.*, 2018, pp. 141–147.

[16] W. Kellerer, P. Kalmbach, A. Blenk, A. Basta, M. Reisslein, and S. Schmid, "Adaptable and data-driven softwarized networks: Review, opportunities, and challenges," *Proc. IEEE*, to be published.

[17] L. Linguaglossa, D. Rossi, S. Pontarelli, D. Barach, D. Marjon, and P. Pfister, "High-speed data plane and network functions virtualization by vectorizing packet processing," *Comput. Netw.*, vol. 149, pp. 187–199, Feb. 2019.

[18] I. Trajkovska *et al.*, "SDN-based service function chaining mechanism and service prototype implementation in NFV scenario," *Comput. Standards Interfaces*, vol. 54, pp. 247–265, Nov. 2017.

[19] B. Yi, X. Wang, S. K. Das, K. Li, and M. Huang, "A comprehensive survey of network function virtualization," *Comput. Netw.*, vol. 133, pp. 212–262, Mar. 2018.

[20] 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. Netw. Softw. (NetSoft)*, Jul. 2017, pp. 1–5.

[21] 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. Opt. Netw. Design Modeling (ONDM)*, May 2018, pp. 136–141.

[22] N. F. S. de Sousa, D. A. L. Perez, R. V. Rosa, M. A. Santos, and E. C. Rothenberg. (2018). "Network service orchestration: A survey." [Online]. Available: https://arxiv.org/abs/1803.06596

[23] 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," *Comput. Netw.*, vol. 133, pp. 1–16, Mar. 2018.

[24] H. Hantouti, N. Benamar, T. Taleb, and A. Laghrissi, "Traffic steering for service function chaining," *IEEE Commun. Surveys Tuts.*, to be published.

[25] W. Hahn *et al.*, "Feasibility of compound chained network functions for flexible packet processing," in *Proc. Eur. Wireless Conf.*, May 2017, pp. 1–6.

[26] K. Han *et al.*, "Application-driven end-to-end slicing: When wireless network virtualization orchestrates with NFV-based mobile edge computing," *IEEE Access*, vol. 6, pp. 26567–26577, 2018.

[27] 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.

[28] 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 Trans. Netw.*, vol. 26, no. 4, pp. 1562–1576, Aug. 2018.

[29] 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 Commun. Mag.*, vol. 55, no. 2, pp. 216–223, Feb. 2017.

[30] F. Rath *et al.*, "SymPerf: Predicting network function performance," in *Proc. SIGCOMM Posters Demos*, 2017, pp. 34–36.

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

[32] A. Morton, *Considerations for Benchmarking Virtual Network Functions and Their Infrastructure*, document RFC 8172, Jul. 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc8172.txt

[33] B. Yang *et al.*, "Algorithms for fault-tolerant placement of stateful virtualized network functions," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 20–24.

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

[35] E. Ahmed and M. H. Rehmani, "Mobile edge computing: Opportunities, solutions, and challenges," *Future Gener. Comput. Syst.*, vol. 70, pp. 59–63, May 2017.

[36] 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 Netw. Appl.*, vol. 20, no. 6, pp. 704–712, Dec. 2015.

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

[38] T. Taleb *et al.*, "EASE: EPC as a service to ease mobile core network deployment over cloud," *IEEE Netw.*, vol. 29, no. 2, pp. 78–88, Mar./Apr. 2015.

[39] 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 Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1657–1681, 3rd Quart., 2017.

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

[41] S. Cheshire. (1996). *It is the Latency, Stupid*. Accessed: Jan. 16, 2019. [Online]. Available: http://www.stuartcheshire.org/rants/latency.html

[42] S. Cheshire, "Latency and the quest for interactivity," Volpe Welty, San Francisco, CA, USA, White Paper, 1996, pp. 1–8. Accessed: Jan. 16, 2019. [Online]. Available: http://www.stuartcheshire.org/papers/LatencyQuest.pdf

[43] O. S. Sella, A. W. Moore, and N. Zilberman, "FEC killed the cut-through switch," in *Proc. ACM Workshop Netw. Emerging Appl. Techn.*, 2018, pp. 15–20.

[44] N. Zilberman *et al.*, "Where has my time gone?" in *Proc. Int. Conf. Passive Active Netw. Meas.*, 2017, pp. 201–214.

[45] N. Hanford, V. Ahuja, M. K. Farrens, B. Tierney, and D. Ghosal, "A survey of end-system optimizations for high-speed networks," *ACM Comput. Surv.*, vol. 51, no. 3, pp. 54:1–54:36, Jul. 2018.

[46] Z. Xiang, F. Gabriel, G. T. Nguyen, and F. H. P. Fitzek, "Latency measurement of service function chaining on OpenStack platform," in *Proc. IEEE Conf. Local Comput. Netw. (LCN)*, Chicago, IL, USA, Oct. 2018, pp. 473–476.

[47] M. He, A. M. Alba, A. Basta, A. Blenk, and W. Kellerer, "Flexibility in softwarized networks: Classifications and research challenges," *IEEE Commun. Surveys Tuts.*, to be published.

[48] P. Emmerich, D. Raumer, S. Gallenmüller, F. Wohlfart, and G. Carle, "Throughput and latency of virtual switching with Open vSwitch: A quantitative analysis," *J. Netw. Syst. Manage.*, vol. 26, no. 2, pp. 314–338, Apr. 2018.

[49] C.-L. Hsieh and N. Weng, "NF-switch: VNFs-enabled SDN switches for high performance service function chaining," in *Proc. IEEE Int. Conf. Netw. Protocols (ICNP)*, Oct. 2017, pp. 1–6.

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

[51] L. Zhang, S. Lai, C. Wu, Z. Li, and C. Guo, "Virtualized network coding functions on the Internet," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, Jun. 2017, pp. 129–139.

[52] *SFC-Ostack: A Simple Research Framework for SFC on OpenStack*. Accessed: Jan. 15, 2019. [Online]. Available: https://github.com/stevelorenz/sfc-ostack

[53] *Build-VNF Project Repository*. Accessed: Oct. 5, 2018. [Online]. Available: https://github.com/stevelorenz/build-vnf/tree/master/CALVIN

[54] *Open vSwitch With DPDK*. Accessed: Sep. 15, 2018. [Online]. Available: http://docs.openvswitch.org/en/latest/intro/install/dpdk/

[55] M.-A. Kourtis *et al.*, "T-NOVA: An open-source MANO stack for NFV infrastructures," *IEEE Trans. Netw. Service Manage.*, vol. 14, no. 3, pp. 586–602, Sep. 2017.

[56] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, "Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 668–682, Mar. 2019.

[57] D. Cho, J. Taheri, A. Y. Zomaya, and L. Wang, "Virtual network function placement: Towards minimizing network latency and lead time," in *Proc. IEEE Int. Conf. Cloud Comput. Techn. Sci. (CloudCom)*, Dec. 2017, pp. 90–97.

[58] R. Gouareb, V. Friderikos, and A. H. Aghvami, "Delay sensitive virtual network function placement and routing," in *Proc. IEEE Int. Conf. Telecommun. (ICT)*, Jun. 2018, pp. 394–398.

[59] H. Halabian, "Distributed resource allocation optimization in 5G virtualized networks," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 627–642, Mar. 2019.

[60] H. Hawilo, M. Jammal, and A. Shami, "Network function virtualization-aware orchestrator for service function chaining placement in the cloud," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 643–655, Mar. 2019.

[61] D. Liao *et al.*, "AI-based software-defined virtual network function scheduling with delay optimization," *Cluster Comput.*, to be published.

[62] W. Miao *et al.*, "Stochastic performance analysis of network function virtualisation in future Internet," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 613–626, Mar. 2019.

[63] L. Qu, C. Assi, and K. Shaban, "Delay-aware scheduling and resource optimization with network function virtualization," *IEEE Trans. Commun.*, vol. 64, no. 9, pp. 3746–3758, Sep. 2016.

[64] L. Qu, C. Assi, K. Shaban, and M. J. Khabbaz, "A reliability-aware network service chain provisioning with delay guarantees in NFV-enabled enterprise datacenter networks," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 3, pp. 554–568, Sep. 2017.

[65] M. Savi, M. Tornatore, and G. Verticale. (2017). "Impact of processing-resource sharing on the placement of chained virtual network functions." [Online]. Available: https://arxiv.org/abs/1710.08262

[66] L. Tang, H. Yang, R. Ma, L. Hu, W. Wang, and Q. Chen, "Queue-aware dynamic placement of virtual network functions in 5G access network," *IEEE Access*, vol. 6, pp. 44291–44305, 2018.

[67] Q. Xu, J. Wang, and K. Wu, "Resource capacity analysis in network slicing with ensured end-to-end performance bound," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.

[68] Q. Ye, W. Zhuang, X. Li, and J. Rao, "End-to-end delay modeling for embedded VNF chains in 5G core networks," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 692–704, Feb. 2019.

[69] S. Agarwal, F. Malandrino, C. F. Chiasserini, and S. De, "VNF placement and resource allocation for the support of vertical services in 5G networks," *IEEE/ACM Trans. Netw.*, to be published.

[70] I. Benkacem, T. Taleb, M. Bagaa, and H. Flinck, "Optimal VNFs placement in CDN slicing over multi-cloud environment," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 616–627, Mar. 2018.

[71] J. Cao, Y. Zhang, W. An, X. Chen, Y. Han, and J. Sun, "VNF placement in hybrid NFV environment: Modeling and genetic algorithms," in *Proc. IEEE Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2016, pp. 769–777.

[72] X. Chen, W. Ni, I. B. Collings, X. Wang, and S. Xu, "Distributed placement and online optimization of virtual machines for network service chains," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.

[73] C. Galdamez, R. Pamula, and Z. Ye, "On efficient virtual network function chaining in NFV-based telecommunications networks," *Cluster Comput.*, to be published.

[74] A. Laghrissi, T. Taleb, M. Bagaa, and H. Flinck, "Towards edge slicing: VNF placement algorithms for a dynamic & realistic edge cloud environment," in *Proc. IEEE GLOBECOM*, Dec. 2017, pp. 1–6.

[75] A. Laghrissi, T. Taleb, and M. Bagaa, "Conformal mapping for optimal network slice planning based on canonical domains," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 519–528, Mar. 2018.

[76] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Commun. Surveys Tuts.*, to be published.

[77] M. A. T. Nejad, S. Parsaeefard, M. A. Maddah-Ali, T. Mahmoodi, and B. H. Khalaj, "vSPACE: VNF simultaneous placement, admission control and embedding," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 542–557, Mar. 2018.

[78] Y. Hu and T. Li, "Towards efficient server architecture for virtualized network function deployment: Implications and implementations," in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2016, pp. 1–8.

[79] G. Durisi, T. Koch, J. Östman, Y. Polyanskiy, and W. Yang, "Short-packet communications over multiple-antenna Rayleigh-fading channels," *IEEE Trans. Commun.*, vol. 64, no. 2, pp. 618–629, Feb. 2016.

[80] G. Durisi, T. Koch, and P. Popovski, "Toward massive, ultrareliable, and low-latency wireless communication with short packets," *Proc. IEEE*, vol. 104, no. 9, pp. 1711–1726, Aug. 2016.

[81] T. Herbert and A. Starovoitov. (Mar. 2016). *The Express Data Path (XDP): Programmable and High Performance Networking Data Path*. Accessed: Oct. 2, 2018. [Online]. Available: https://github.com/iovisor/bpf-docs/blob/master/Express_Data_Path.pdf

[82] J. D. Brouer. (2016). *Linux Networking Subsystem, XDP–Express Data Path*. Accessed: Oct. 2, 2018. [Online]. Available: https://prototype-kernel.readthedocs.io/en/latest/networking/index.html

[83] (2018). *BPF and XDP Reference Guide*. Accessed: Oct. 2, 2018. [Online]. Available: https://cilium.readthedocs.io/en/v1.2/bpf

[84] T. Høiland-Jørgensen *et al.*, "The eXpress Data Path: Fast programmable packet processing in the operating system kernel," in *Proc. ACM Int. Conf. Emerg. Netw. Exp. Techn. (CoNEXT)*, 2018, pp. 54–66.

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

[86] Z. Ahmed, M. H. Alizai, and A. A. Syed, "InKeV: In-kernel distributed network virtualization for DCN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 46, no. 3, p. 4, Jul. 2018.

[87] (Sep. 2018). *Neutron Documentation*. Accessed: Oct. 3, 2018. [Online]. Available: https://docs.openstack.org/neutron

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

[89] X. Li *et al.*, "A kernel-space POF virtual switch," *Comput. Elect. Eng.*, vol. 61, pp. 339–350, Jul. 2017.

[90] 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 Congr. (ITC)*, Sep. 2018, pp. 1–9.

[91] Y. Zhang *et al.*, "Parabox: Exploiting parallelism for virtual network functions in service chaining," in *Proc. ACM Symp. SDN Res.*, 2017, pp. 143–149.

[92] 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* (Lecture Notes in Computer Science), vol. 7754. Berlin, Germany: Springer, 2013, pp. 3–27.

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

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

[95] A. Anthony, S. R. Chowdhury, T. Bai, R. Boutaba, and J. François, "UNiS: A user-space non-intrusive workflow-aware virtual network function scheduler," in *Proc. IEEE Int. Conf. Netw. Service Manage. (CNSM)*, Nov. 2018, pp. 152–160.

[96] J. Duan, X. Yi, J. Wang, C. Wu, and F. Le, "*NetStar*: A future/promise framework for asynchronous network functions," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 600–612, Mar. 2019.

[97] C. Zhang, J. Bi, Y. Zhou, and J. Wu, "HyperVDP: High-performance virtualization of the programmable data plane," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 556–569, Mar. 2019.

[98] J. Duan, X. Yi, S. Zhao, C. Wu, H. Cui, and F. Le, "*NFVactor*: A resilient NFV system using the distributed actor model," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 586–599, Mar. 2019.

[99] M. Zhang *et al.*, "Tripod: Towards a scalable, efficient and resilient cloud gateway," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 570–585, Mar. 2019.

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

[101] P. Emmerich, M. Pudelko, S. Bauer, and G. Carle, "User space network drivers," in *Proc. ACM Appl. Netw. Res. Workshop*, 2018, pp. 91–93.

[102] (2018). *Kernel NIC Interface*. Accessed: Oct. 3, 2018. [Online]. Available: https://doc.dpdk.org/guides/prog_guide/kernel_nic_interface.html

[103] 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. Lett.*, vol. 22, no. 4, pp. 748–751, Apr. 2018.

[104] C. Li, C. Ding, and K. Shen, "Quantifying the cost of context switch," in *Proc. USENIX Workshop Experim. Comput. Sci.*, 2007, pp. 1–4.

[105] M. V. Pedersen, J. Heide, and F. H. Fitzek, "Kodo: An open and research oriented network coding library," in *Proc. Int. Conf. Res. Netw.*, in (Lecture Notes in Computer Science), vol. 6827. Berlin, Germany: Springer, 2011, pp. 145–152.

[106] D. Cerovic, V. Del Piccolo, A. Amamou, K. Haddadou, and G. Pujolle, "Fast packet processing: A survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3645–3676, 4th Quart., 2018.

[107] D. Barach, L. Linguaglossa, D. Marion, P. Pfister, S. Pontarelli, and D. Rossi, "High-speed software data plane via vectorized packet processing," *IEEE Commun. Mag.*, vol. 56, no. 12, pp. 97–103, Dec. 2018.

[108] *IOVisor Homepage*. Accessed: Jan. 24, 2019. [Online]. Available: https://www.iovisor.org/

[109] M.-A. Kourtis *et al.*, "Enhancing VNF performance by exploiting SR-IOV and DPDK packet processing acceleration," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2015, pp. 74–78.

[110] *OpenStack Nova Computing Documentation*. Accessed: Oct. 3, 2018. [Online]. Available: https://www.spinics.net/lists/netdev/msg405175.html

[111] *Linux Netdev Mailing List: Patch 4/5*. Accessed: Oct. 3, 2018. [Online]. Available: https://www.spinics.net/lists/netdev/msg405175.html

[112] *BCC Project Homepage*. Accessed: Sep. 20, 2018. [Online]. Available: https://github.com/iovisor/bcc

[113] J. Hwang, K. K. Ramakrishnan, and T. Wood, "NetVM: High performance and flexible networking using virtualization on commodity platforms," *IEEE Trans. Netw. Service Manage.*, vol. 12, no. 1, pp. 34–47, Mar. 2015.

[114] W. Zhang *et al.*, "OpenNetVM: A platform for high performance network service chains," in *Proc. ACM Workshop Hot Topics Middleboxes Netw. Function Virtualization*, 2016, pp. 26–31.

[115] (Nov. 2018). *VMware Docs, Enable Live Resize*. Accessed: Mar. 4, 2019. [Online]. Available: https://docs.vmware.com/en/VMware-Integrated-OpenStack/4.1/com.vmware.openstack.admin.doc/GUID-FEB48287-04AD-4BAB-9B42-99543DCC9733.html

[116] (2018). *OpenStack Compute (Nova), Instance Live Resize*. Accessed: Jan. 5, 2019. [Online]. Available: https://blueprints.launchpad.net/nova/+spec/instance-live-resize

[117] P. Veitch and T. Long, "A low-latency NFV infrastructure for performance-critical applications," Intel, Santa Clara, CA, USA, Tech. Rep., 2017, Accessed: Sep. 30, 2018. [Online]. Available: https://software.intel.com/en-us/articles/low-latency-nfv-infrastructure-for-performance-critical-applications

[118] *OpenStack Pike Documentation*. Accessed: Oct. 4, 2018. [Online]. Available: https://docs.openstack.org/pike/

[119] *Virtio Documentation*. Accessed: Jan. 20, 2019. [Online]. Available: https://www.linux-kvm.org/page/Virtio

[120] J. Chauhan, D. Makaroff, and A. Arkles, "Is doing clock synchronization in a VM a good idea?" in *Proc. IEEE Int. Perform. Comput. Commun. Conf.*, Dec. 2010, pp. 1–2.

[121] *Iperf Homepage*. Accessed: Sep. 10, 2018. [Online]. Available: https://iperf.fr/

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

[123] 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.

[124] *BPF and XDP Reference Guide*. Accessed: Oct. 4, 2018. [Online]. Available: https://cilium.readthedocs.io/en/v1.2/bpf/

[125] *Linux Kernel Networking Documentation*. Accessed: Oct. 4, 2018. [Online]. Available: https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt

[126] C. Sieber, R. Durner, M. Ehm, W. Kellerer, and P. Sharma, "Towards optimal adaptation of NFV packet processing to modern CPU memory architectures," in *Proc. ACM Workshop Cloud-Assist. Netw.*, 2017, pp. 7–12.

[127] Y. Hu, M. Song, and T. Li, "Towards full containerization in containerized network function virtualization," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 467–481, Mar. 2017.

[128] *DPDK Poll Mode Driver*. Accessed: Jan. 20, 2019. [Online]. Available: https://doc.dpdk.org/guides/prog_guide/poll_mode_drv.html

[129] D. Vladislavić, D. Huljenić, and J. Ožegović, "Enhancing VNF's performance using DPDK driven OVS user-space forwarding," in *Proc. IEEE Int. Conf. Softw., Telecommun. Comput. Netw. (SoftCOM)*, Sep. 2017, pp. 1–5.

[130] *DPDK Mbuf Library*. Accessed: Jan. 20, 2019. [Online]. Available: https://doc.dpdk.org/guides/prog_guide/mbuf_lib.html

[131] L. A. D. Knob, B. G. Xavier, and T. Ferreto, "An unikernels provisioning architecture for OpenStack," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2018, pp. 903–908.

[132] P. L. Ventre *et al.*, "On the fly orchestration of unikernels: Tuning and performance evaluation of virtual infrastructure managers," *IEEE Trans. Cloud Comput.*, to be published.

[133] *Mpstat Manpage*. Accessed: Jan. 24, 2019. [Online]. Available: http://man7.org/linux/man-pages/man1/mpstat.1.html

[134] H. D. Chirammal, P. Mukhedkar, and A. Vettathu, *Mastering KVM Virtualization*. Birmingham, U.K.: Packt, 2016.

[135] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.

[136] T. Ho *et al.*, "A random linear network coding approach to multicast," *IEEE Trans. Inf. Theory*, vol. 52, no. 10, pp. 4413–4430, Oct. 2006.

[137] S. Feizi, D. E. Lucani, and M. Médard, "Tunable sparse network coding," in *Proc. Int. Zurich Seminar Commun. (IZS)*, 2012, pp. 1–5.

[138] 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 *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, May 2018, pp. 1–6.

[139] F. Gabriel, S. Wunderlich, S. Pandi, F. H. Fitzek, and M. Reisslein, "Caterpillar RLNC with feedback (CRLNC-FB): Reducing delay in selective repeat ARQ through coding," *IEEE Access*, vol. 6, pp. 44787–44802, 2018.

[140] 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. 20183–20197, 2017.

[141] D. E. Lucani *et al.*, "Fulcrum: Flexible network coding for heterogeneous devices," *IEEE Access*, vol. 6, pp. 77890–77910, 2018.

[142] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 56, no. 9, pp. 4539–4551, Sep. 2010.

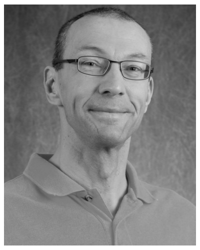
[143] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. Allerton Conf. Commun., Control, Comput.*, vol. 41, 2003, pp. 40–49.
[144] J. Barros, R. A. Costa, D. Munaretto, and J. Widmer, "Effective delay control in online network coding," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 208–216.
[145] 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. 20477–20493, 2017.
[146] M. Karzand and D. J. Leith, "Low delay random linear coding over a stream," in *Proc. Allerton Conf. Commun., Control, Comput.*, 2014, pp. 521–528.
[147] 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 *Proc. Eur. Wireless*, 2015, pp. 1–6.
[148] 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 *Proc. IEEE Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2018, pp. 1–2.
[149] A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel, and P. Tabriz, "Measuring HTTPS adoption on the Web," in *Proc. USENIX Secur. Symp.*, 2017, pp. 1323–1338
[150] V.-C. Nguyen, A.-V. Vu, K. Sun, and Y. Kim, "An experimental study of security for service function chaining," in *Proc. IEEE Int. Conf. Ubiquitous Future Netw. (ICUFN)*, Jul. 2017, pp. 797–799.
[151] Z. Zheng *et al.*, "GEN: A GPU-accelerated elastic framework for NFV," in *Proc. ACM Asia–Pacific Workshop Netw.*, 2018, pp. 57–64.
[152] M. M. Rovnyagin and A. A. Kuznetsov, "Application of hybrid computing technologies for high-performance distributed NFV systems," in *Proc. IEEE Conf. Russian Young Researchers Electr. Electron. Eng. (EIConRus)*, Feb. 2017, pp. 540–543.
[153] *Small portable AES Implementation in C*. Accessed: Sep. 10, 2018. [Online]. Available: https://github.com/kokke/tiny-AES-c
[154] J. Ren, Y. Guo, D. Zhang, Q. Liu, and Y. Zhang, "Distributed and efficient object detection in edge computing: Challenges and solutions," *IEEE Netw.*, vol. 32, no. 6, pp. 137–143, Nov./Dec. 2018.
[155] I. Benkacem, T. Taleb, M. Bagaa, and H. Flinck, "Performance benchmark of transcoding as a virtual network function in CDN as a service slicing," in *Proc. Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2018, pp. 1–6.
[156] D. Sanvito, G. Siracusano, and R. Bifulco, "Can the network be the AI accelerator?" in *Proc. ACM Workshop Netw. Comput. (NetCompute)*, Aug. 2018, pp. 20–25.

Zuo Xiang received the Diploma (Dipl.Ing.) degree in electrical engineering from Technical University Dresden (TU Dresden), Dresden, Germany, in 2018, where he is currently pursuing the Ph.D. degree with the Deutsche Telekom Chair of Communication Networks. His research interests lie in the area of high-performance network softwarization. His research focuses on network function virtualization (NFV) and software-defined network (SDN) for low latency communication in cloud and edge computing environments.

Frank Gabriel received the Dipl.-Inf. degree in computer science from Technical University Chemnitz, Germany, in 2011. He is currently pursuing the Ph.D. degree with the Deutsche Telekom Chair of Communication Networks, Technical University Dresden, Dresden, Germany. His research focuses on network coding for reliable and low latency communication in multipath and mesh networks.

Elena Urbano received the M.Sc. degree in automation and industrial electronics from the University of Málaga, Spain. She is currently a Junior Researcher with the Deutsche Telekom Chair of Communication Networks, TU Dresden, Dresden, Germany. 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 received the master's degree (M.Eng.) in telecommunications from the Asian Institute of Technology (AIT), Thailand, in 2007, and the Ph.D. (Dr. Ing.) degree in computer science from the Technical University of Dresden, Germany, in 2016. He is currently a Senior Researcher with the Deutsche Telekom Chair of Communication Networks, Technical University of Dresden. 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 (S'96–M'98–SM'03–F'14) received the Ph.D. degree in systems engineering from the University of Pennsylvania, Philadelphia, PA, USA, in 1998. He is currently a Professor with the School of Electrical, Computer, and Energy Engineering, Arizona State University (ASU), Tempe, AZ, USA. He chairs the Steering Committee of the IEEE TRANSACTIONS ON MULTIMEDIA. He currently serves as an Associate Editor for the IEEE TRANSACTIONS ON MOBILE COMPUTING, the IEEE TRANSACTIONS ON EDUCATION, IEEE ACCESS, as well as *Computer Networks*. He is an Associate Editor-in-Chief of the IEEE COMMUNICATIONS SURVEYS & TUTORIALS and a Co-Editor-in-Chief of *Optical Switching and Networking*.

Frank H. P. Fitzek received the Diploma (Dipl.Ing.) degree in electrical engineering from the University of Technology–Rheinisch-Westfälische Technische Hochschule (RWTH) Aachen University, Aachen, Germany, in 1997, and the Ph.D. (Dr.Ing.) degree in electrical engineering from Technical University Berlin, Germany, in 2002. He became an Adjunct Professor with the University of Ferrara, Italy, in 2002. He is currently a Professor and the Head of the Deutsche Telekom Chair of Communication Networks, TU Dresden, Germany, where he is also coordinating the 5G Lab. He is the Spokesman of the DFG Cluster of Excellence CeTI. He joined Aalborg University in 2003 as an Associate Professor and later became a Professor. His current research interests are in the areas of wireless and 5G communication networks, network coding, cloud computing, compressed sensing, cross layer and energy-efficient protocol design, and cooperative networking. He was selected to receive the NOKIA Champion Award several times from 2007 to 2011. He has received the YRP Award for the work on MIMO MDC in 2005, the Nokia Achievement Award for his work on cooperative networks in 2008, the SAPERE AUDE Research Grant from the Danish Government in 2011, the Vodafone Innovation Prize in 2012, and the Young Elite Researcher Award of Denmark. He has also received the honorary degree *Doctor Honoris Causa* from the Budapest University of Technology and Economics (BUTE) in 2015.