Service Chaining for Heterogeneous Middleboxes

Xuzhi Zhang and Russell Tessier
Department of Electrical and Computer Engineering
University of Massachusetts Amherst
xuzhizhang@umass.edu, tessier@umass.edu

***Abstract*—In this paper, we describe a new session-level approach for network function virtualization that steers packets through both FPGA- and processor-based middleboxes. We support inter-domain service chaining, dynamic modification of service chains for ongoing sessions, and application of session-level approaches for UDP-based protocols. To demonstrate our approach, we establish and dynamically update inter-domain service chains for QUIC protocol sessions across a scalable set of FPGA- and processor-based middleboxes. The middleboxes implemented with distributed agents show 96% better latency and 10% better throughput than accelerated software for a firewall application.**

I. Introduction

FPGAs have been used to implement a wide variety of high-performance network functions [1]–[4], such as packet filters and intrusion detection circuits. This approach to network function virtualization (NFV) provides for low-latency, line-rate processing that is orders of magnitude faster than microprocessor-based software implementations. FPGAs allow users to create custom hardware to process packets without the overhead of network stack processing supported by an operating system (OS).

Most NFV systems, including systems with FPGAs, use chains of functions. For large-scale NFV systems, traffic must be steered through functions in a sequence. Although most systems use a centralized controller for sequence orchestration [5, 6], the approach is burdened by the risk of central point failure, difficulties in synchronizing routing table rules, and traffic steering across network domain boundaries. In contrast, the use of distributed agents in *session-based* traffic coordination can effectively overcome these shortcomings [7]. A session includes a series of interactions between two communication endpoints. To date, NFV based on session-level approaches has only been applied to microprocessor-based systems due to difficulties in managing traffic steering across heterogeneous middleboxes and managing partial FPGA reconfiguration when functions updates are needed.

In this paper, we describe a new distributed-agent NFV system that supports the service function chaining of FPGAs and microprocessors. Our new approach deploys distributed agents in end-hosts and FPGA- and processor-based middleboxes. An agent cooperates with a partial reconfiguration core to manage the dynamic reconfiguration of middlebox functions on FPGAs. Agents update packet headers at the transport layer to steer packets belonging to different sessions through corresponding service chains. We verify our new approach with QUIC protocol sessions [8] and show the benefits of implementing our agents with FPGA circuits versus software-based implementations. Our prototype distributed-agent NFV system is assessed using Intel DE5-Net FPGA boards, microprocessor-based middleboxes, and a network router for up to nine middleboxes. The use of FPGA-based service chaining results in a 96% reduction in function packet latency versus a software-based approach.

II. Related Work

Service function chaining (SFC) [9] steers traffic through a set of functions provided by middleboxes to compose complex network services. Traditionally SFC deployments are static and rely on network configurations to stitch middleboxes together. As NFV and software-defined networking (SDN) technologies have matured, dynamically-composed service chains and fine-grained packet forwarding have become possible. Several previous service function chain solutions steer network traffic by controlling network elements (e.g., switches). Stratos [5] and E2 [6] use fine-grained forwarding rules to build static service chains within clouds. OpenNF [10] and Split-Merge [11] provide dynamic service chaining by updating packet forwarding rules in SDN switches. These solutions rely on logically centralized controllers to install and update forwarding rules.

Session-level control of NFV chains has recently been introduced [12] to lessen the impact of centralized NFV control. Existing protocols for session-level service chaining include Network Service Header (NSH) [12], Segment Routing Header (SRH) [13], and Dysco [7]. NSH supports service chaining without the use of forwarding rules. SRH encodes a list of IPv6 addresses of virtual network functions (VNFs) in packet headers to perform SFC. Dysco is a session-level protocol that steers the traffic of a session through an SFC. Like our approach, it can dynamically update the SFC without requiring changes to IP routing, end-host applications, or middlebox applications. However, Dysco does not support connectionless protocols. None of these protocols have been implemented using FPGAs.

Previous research projects [2, 3] have integrated FPGAs into NFV systems and developed resource management algorithms to allocate functions across heterogeneous resources. These systems rely on centralized SDN controllers to update the forwarding rules in SDN switches that direct packets through heterogeneous middleboxes. Tarafdar et al. [14] developed an FPGA-based switch to steer packets through FPGA-based middleboxes based on the addresses of network function kernels in the Ethernet packet header. This approach avoids

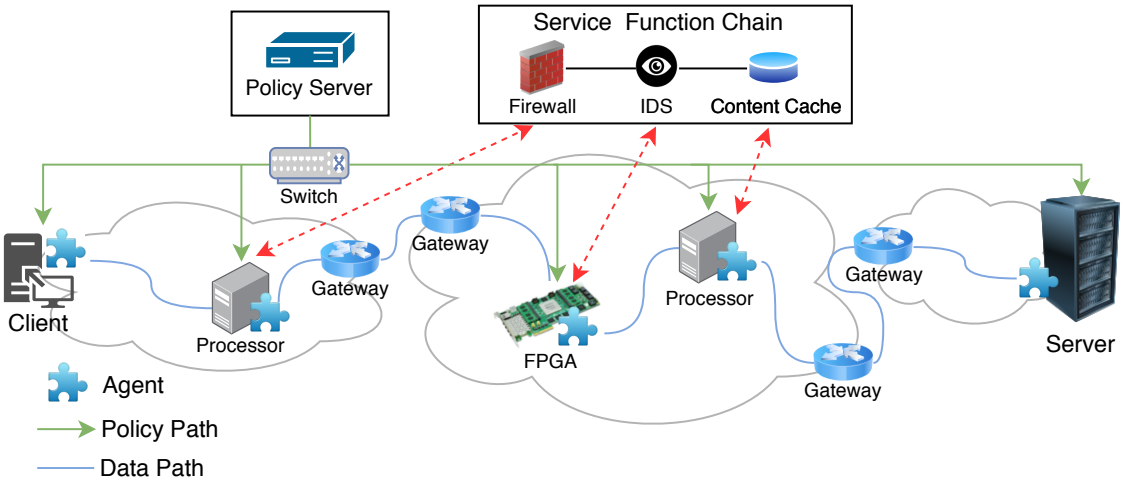

Fig. 1: An example of an inter-domain service chain established with agents and a policy server.

the use of a central controller, but it cannot steer packets at the session level. As a result, it does not support the establishment of service chains across network domain boundaries.

III. Architecture

Network functions in a service chain, such as a firewall, intrusion detection system (IDS), or content cache, can be implemented on custom hardware, virtual machines (VM), or reconfigurable hardware deployed across multiple subnetworks. In Figure 1, the client and server at either end of the service chain are located in different subnetworks. FPGA and processor resources in the local area network (LAN) and cloud are used to deploy network functions to process in-transit data packets.

A session is a series of interactions between two communication endpoints. In Figure 1, end-hosts (i.e., server and client) communicate with each other by establishing sessions. Our approach runs agents on end-hosts and FPGA- and processor-based middleboxes, builds a service chain while creating a session, and directs session packets through network functions in the service chain. Agents rely on basic IP routing and high-level policies, which can be obtained from a policy server, to steer packets between end-hosts and middleboxes located in different subnetworks.

A. Components and interfaces

A service chain for a session includes a series of *subsessions*. Each subsession connects an end-host and a neighboring middlebox, or two neighboring middleboxes. Agents set up individual subsessions in the service chain and an agent can maintain multiple subsessions at the same time. Each subsession is identified by a five-tuple (i.e., source/destination IP address, source/destination port number, and protocol) that is used for a specific service chain. The agent on an end-host or a middlebox in a service chain forwards packets with the original header of the session to the end-host application or the middlebox application; as such, our approach works

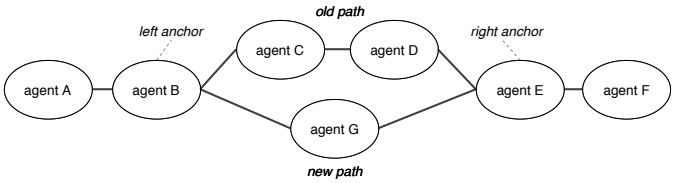

Fig. 2: Agents reconfigure a segment of a service chain, replacing an old path with two middleboxes by a new path with one

with existing application-layer protocols. Before transmitting session packets to the next middlebox or end-host, the agent rewrites packet headers using the subsession five-tuple. In this way, agents steer packets through the service chain.

The first agent in a service chain (e.g., the agent on the client end-host in Figure 1) starts service chain creation according to a chaining policy received from a policy server. The chaining policy specifies an ordered list of middleboxes and end-hosts located in the service chain. Each middlebox or end-host in the policy is identified by its IP address. For example, the policy for establishing the service chain in Figure 1 includes an ordered list of IP addresses for the client end-host, the first processor-based middlebox, the FPGA-based middlebox, the second processor-based middlebox, and the server end-host. During service chain creation, the chaining policy is passed forward by the agents along the service chain from the first agent in the chain to the last one. Each agent sets up a subsession to connect to the next neighboring middlebox or end-host indicated by the policy.

IV. Dynamic Service Chain Modification

A service chain can benefit from dynamic path updates to improve network resource utilization, reduce resource consumption, and adjust network resources to adapt to changes in the network environment. The agents at the two ends of a segment of a service chain are the left anchor and the right

anchor (Figure 2). Here, we define the left anchor as the agent close to the client, and the right anchor as the agent close to the server. Path updates are initiated by the agent acting as the left anchor. The policy server sends a new address list to the left anchor. The new address list includes the IP addresses of the left anchor, middleboxes, and right anchor of the new path that will replace the old path. For example, the left anchor B in Figure 2 receives the address list [B, G, E] in which B is the left anchor and E is the right anchor. The address list [B, G, E] indicates the new path. Service chain modification is achieved by transmitting a series of control packets between the left and right anchors to create a new path and then switching the old path [B, C, D, E] to the new path through two anchors. Control packets are used to resolve contention if multiple portions of a service chain try to change at the same time, set up a new path for the service chain, and cancel modification if a new path cannot be created.

Our approach allows an agent on an FPGA-based middlebox to partially reconfigure a virtual network function (VNF) implementation during service chain modification without interrupting the operation of other VNFs on the same hardware. Agents running on processor-based middleboxes can adjust (reconfigure) middlebox functionality by starting/terminating VNF processes. The need for a specific VNF can be included in the control packet during service chain modification. The agent triggers the partial FPGA reconfiguration of middlebox functionality before forwarding the control packet to the next middlebox. The session on the service chain continues to operate during reconfiguration, as session data is still transmitted on the old path during the FPGA partial reconfiguration.

V. Implementation

In our system, agents can be implemented on FPGA- and processor-based middleboxes, and end-hosts. The agents establish and dynamically update service chains across heterogeneous middleboxes. FPGA-based middleboxes were implemented using two Terasic DE5-Net boards that include Intel Stratix V FPGAs. Processor-based middleboxes were implemented as Docker[1] containers on three twelve-core Intel Xeon workstations (2.4 GHz, 32 GB SDRAM, two 10 Gbps NICs, and four 1 Gbps NICs). QUIC clients and servers run on a 28-core Intel Xeon workstation (2.6 GHz, 128 GB SDRAM, two 10 Gbps NICs, and two 1 Gbps NICs). The policy server was implemented using an Intel Duo server (2.66 GHz, 4 GB).

Agents forward network packets to FPGA- and processor-based VNFs. An agent on general-purpose commodity hardware (i.e. end-hosts, processor-based middleboxes) is implemented as software running in user space. It utilizes the host network stack to communicate with the applications running on the host (e.g. client and server) and receive/send session packets from/to NICs. The agent on the FPGA platform (top of Figure 3), developed as part of this work, is implemented as a dedicated hardware circuit that interacts with packet processors and network interfaces via the Avalon streaming

[1]www.docker.com

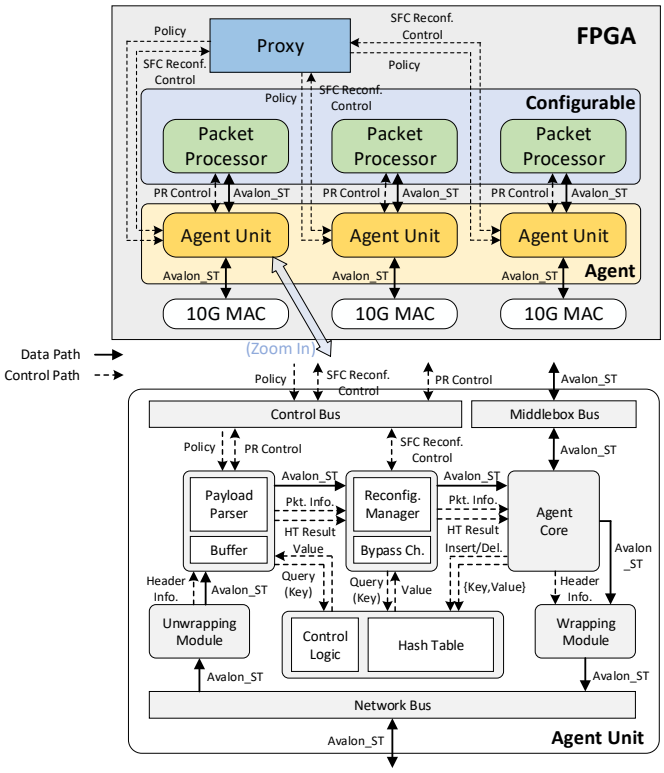

Fig. 3: Implementation of an agent on an FPGA. The agent unit in the top subfigure is expanded in the bottom subfigure.

bus. One FPGA platform can support up to three packet processors that operate as separate network functions. The agent connects each packet processor to a physical network port. FPGA packet processors are dynamically reconfigurable using partial reconfiguration. The policy for reconfiguring local packet processors is obtained from the policy server. A *proxy* module in each middlebox and end-host communicates with the policy server via a TCP connection. For processor-based middleboxes, the proxy module was implemented as a software program. On the FPGA platform it was implemented using a NIOS II processor.

For experimentation, we implemented our custom agent design on a Stratix V 5SGXEA7N FPGA. The FPGA platform has three packet processors deployed (top half of Figure 3). The agent unit module (expanded in the bottom part of Figure 3) is pipelined and divided into five submodules: the unwrapping module, the payload parser module, the reconfiguration manager module, the agent core, and the wrapping module. The agent implemented on an FPGA contains three agent units (one for each packet processor), each of which connects a single packet processor with an individual 10G MAC core through the Avalon streaming bus. The data packets from the network are fed to the unwrapping module. This module extracts the five-tuple information from the packet header and sends it and the payload to the payload parser module. This module uses the five-tuple as a key to search the hash table to obtain the address of the next middlebox and

TABLE I: Resource usage for SFC implementation cores targeted to a Stratix V 5SGXEA7N

	LUTs	FFs	Block Mem bits
Agent unit	20,976	21,948	891,770
Proxy	14,598	19,957	2,934,968
PR region	11,440	5,720	839,680
Firewall	2,822	5,283	504,960
Network interface	12,807	18,082	239,657
Available in FPGA	469,440	938,880	52,428,800

the original five-tuple of the session. The module also extracts the data packet payload and the control information used for service chain establishment and reconfiguration. When needed, the payload parser module updates the control information according to the policy received from the proxy module and adds an address list for constructing a service chain to the control frame. The payload parser module stores the received data packet in a small buffer.

The reconfiguration manager module responds to the SFC reconfiguration control command to generate reconfiguration control packets and start the reconfiguration of a service chain segment. This module uses a state machine to manage the various stages of the service chain reconfiguration process. It obtains the address of the next middlebox on the service chain segment to be reconfigured from the hash table. The agent core module performs hash table operations, rewrites the packet headers for data packets processed by the packet processor, forwards packets emerging from the packet processor, and provides header information about the next subsession to the wrapping module. The packet payload and header information are integrated into a complete data packet in the wrapping module, which is sent to the 10G MAC through the network bus. For experimentation, a firewall packet processor was created and tested. The firewall module was implemented in a partially reconfigurable packet processor region (PR region) on the FPGA device.

VI. Evaluation

Lab experiments were performed using our PC and FPGA-board virtualization system. An open-source tool *ngtcp2*[2] was used to generate QUIC protocol sessions between clients and servers. We measured the latencies for session initiation to quantify the overhead introduced by our agents in establishing a service chain. Then, we measured the throughput of QUIC sessions in a service chain to verify the scalability of chains with agents. Finally, we assessed the ability of the system to modify a service chain path across multiple subnetworks. The FPGA resource counts of the agent, the proxy, the PR region for a single packet processor, the firewall module, and the network interface are shown in Table I. The size of the partial bitstream for the PR region is 5.8 MB, while the entire bitstream for the Stratix V FPGA is 31.4 MB.

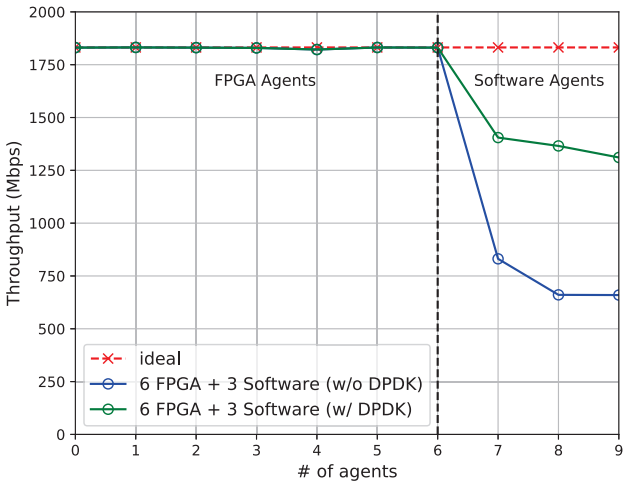

Fig. 4: Throughput scalability for chains of up to 2 FPGAs (3 agents each) and 3 workstations (3 non-DPDK agents or 3 DPDK-based agents). Six sessions were used for this experiment.

A. Throughput test

The service chain throughput of VNFs with software agents implemented with and without the Data Plane Development Kit (DPDK) and with FPGA agents was tested. DPDK allows for packet processing in user space[3]. Six QUIC sessions were run simultaneously on the same service chain. Overall throughput on the service chain was measured using an increasing set of hardware and software middleboxes. Software agents implemented without DPDK were run in containers installed in three workstations (one container per workstation). DPDK-based software agents were run directly on the host OS for three workstations. Two DE5-Net boards were used to implement FPGA versions (three agent units per FPGA). Figure 4 shows the scalability of our agent-based approach working on heterogeneous middleboxes with up to nine agents deployed in the service chain. The first six middleboxes used in the chain are FPGA-based, which show a consistent chain throughput on the left side of the graph. The blue and green plots in Figure 4 show that software agents introduce throughput degradation to the service chain. The serial nature of processor execution causes system performance slowdown. Although the FPGA throughput can reach close to 10 Gbps, the throughput shown at the left of Figure 4 for the chain is limited by the packet generation rate of the processor-based server.

B. Dynamic path modification

Our agent-based approach implements inter-domain service chaining by deploying agents in multiple subnetworks and connecting the middleboxes and end-hosts by establishing subsessions across network boundaries. In a second experiment, QUIC clients and servers, located in different subnetworks, and FPGA and processor-based middleboxes were connected through a level-3 router. Each subnetwork has its own IP

[2]https://github.com/ngtcp2/ngtcp2/tree/draft-23
[3]www.dpdk.org

TABLE II: Throughput and latency comparison of software and FPGA firewall implementations for three QUIC sessions

	Software (w/o DPDK)	Software (w/ DPDK)	FPGA
Throughput (Mbps)	796	1,157	1,270
Latency (μs)	115.6	85.8	3.7

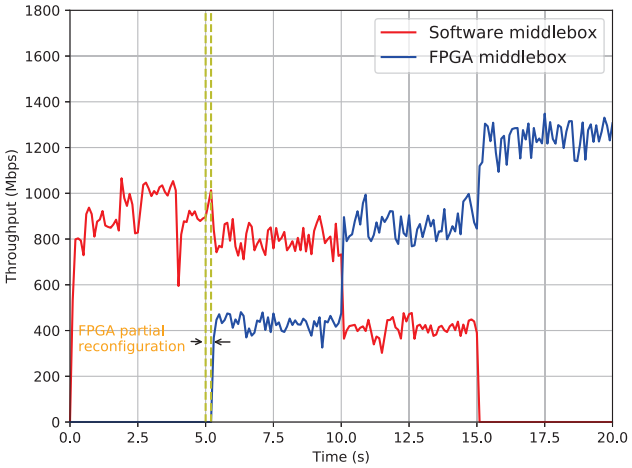

Fig. 5: Throughput of three QUIC sessions on processor and FPGA middleboxes. Initially, all three sessions are implemented on processors (left). Service chain modification is performed every 5 seconds to migrate a QUIC session from a processor (container) middlebox to the FPGA middlebox.

address domain. As shown in Table II, the FPGA-based firewall has significantly better throughput and latency than software versions for three QUIC sessions. FPGA throughput was again limited by the processor-based generation of QUIC packets. During service chain operation, agents are used to modify the service chain three times, each time migrating one QUIC session from a processor-based middlebox to the FPGA-based middlebox.

Figure 5 shows the QUIC throughput of the software-based and FPGA-based middleboxes over time, including three session migrations to the FPGA. The time series represents throughput measures at one-hundred-millisecond intervals. Three QUIC sessions initially pass through the software middlebox with a total traffic rate less than 1,000 Mbps (the software firewall throughput limitation shown in Table II). Service chain modification occurs at 5, 10, and 15 seconds after the start of the experiment. When the first QUIC session is redirected, the control message sent by the left anchor triggers the agent on the FPGA middlebox to configure an FPGA firewall circuit as a packet processor in a PR region. Partial FPGA reconfiguration requires about 196 ms and requires the loading of a firewall partial bitstream from flash. The agent stays active during the partial reconfiguration of the firewall bitstream. The time interval between the two yellow vertical dashed lines in Figure 5 indicates the partial reconfiguration of the firewall on the FPGA middlebox. After all three QUIC sessions have been redirected to pass through the FPGA middlebox, the overall throughput (blue plot on the right) is significantly higher than when all sessions passed through the software-based middleboxes (red plot on the left).

VII. Conclusion

In this paper, we have described the first session-level implementation for the service chaining of heterogeneous middleboxes. Distributed agents implemented on field-programmable gate arrays and microprocessors are used to steer packets through service chains. Our results using QUIC protocol sessions show that the hardware agent implementations have good scalability and better throughput and latency than software implementations during packet processing. Partial reconfiguration is used to migrate functions to FPGAs.

Acknowledgments

This research was supported by National Science Foundation grant CNS-1525836. We thank Intel for the donation of the DE5 boards.

References

[1] S. Byma, J. G. Steffan, H. Bannazadeh, A. L. Garcia, and P. Chow, "FPGAs in the Cloud: Booting Virtualized Hardware Accelerators with OpenStack," in *Proc: FCCM*, 2014, pp. 109–116.

[2] N. Tarafdar, T. Lin, N. Eskandari, D. Lion, A. Leon-Garcia, and P. Chow, "Heterogeneous Virtualized Network Function Framework for the Data Center," in *Proc: FPL*, 2017, pp. 1–8.

[3] X. Zhang, X. Shao, G. Provelengios, N. K. Dumpala, L. Gao, and R. Tessier, "Scalable Network Function Virtualization for Heterogeneous Middleboxes," in *Proc: FCCM*, 2017, pp. 219–226.

[4] ——, "CoNFV: A Heterogeneous Platform for Scalable Network Function Virtualization," *ACM TRETS*, vol. 14, no. 1, pp. 1–29, 2020.

[5] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, V. Sekar, and A. Akella, "Stratos: A Network-Aware Orchestration Layer for Virtual Middleboxes in Clouds," 2013. [Online]. Available: https://arxiv.org/abs/1305.0209

[6] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker, "E2: A Framework for NFV Applications," in *Proc: SOSP*, 2015, pp. 121–136.

[7] P. Zave, R. A. Ferreira, X. K. Zou, M. Morimoto, and J. Rexford, "Dynamic Service Chaining with Dysco," in *Proc: ACM SIGCOMM*, 2017, pp. 57–70.

[8] M. Thomson and S. Turner, "Using Transport Layer Security (TLS) to Secure QUIC," 2020. [Online]. Available: https://tools.ietf.org/html/draft-ietf-quic-tls-27

[9] J. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture," in *RFC 7665*, 2015.

[10] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "OpenNF: Enabling Innovation in Network Function Control," *ACM CCR*, vol. 44, no. 4, pp. 163–174, 2014.

[11] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/Merge: System Support for Elastic Execution in Virtual Middleboxes," in *Proc: NSDI*, 2013, pp. 227–240.

[12] P. Quinn, U. Elzur, and C. Pignataro, "Network Service Header (NSH)." IETF, 2017. [Online]. Available: https://tools.ietf.org/id/draft-ietf-sfc-nsh-17.html

[13] 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: NetSoft*, 2017, pp. 1–5.

[14] N. Tarafdar, T. Lin, E. Fukuda, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "Enabling Flexible Network FPGA Clusters in a Heterogeneous Cloud Data Center," in *Proc: FPGA*, 2017, pp. 237–246.