INVITED SURVEY PAPER

Research Challenges for Network Function Virtualization – Re-Architecting Middlebox for High Performance and Efficient, Elastic and Resilient Platform to Create New Services –

Kohei SHIOMOTO[†a)], ***Fellow***

SUMMARY Today's enterprise, data-center, and internet-service-provider networks deploy different types of network devices, including switches, routers, and middleboxes such as network address translation and firewalls. These devices are vertically integrated monolithic systems. Software-defined networking (SDN) and network function virtualization (NFV) are promising technologies for dis-aggregating vertically integrated systems into components by using "softwarization". Software-defined networking separates the control plane from the data plane of switch and router, while NFV decouples high-layer service functions (SFs) or Network Functions (NFs) implemented in the data plane of a middlebox and enables the innovation of policy implementation by using SF chaining. Even though there have been several survey studies in this area, this area is continuing to grow rapidly. In this paper, we present a recent survey of this area. In particular, we survey research activities in the areas of re-architecting middleboxes, state management, high-performance platforms, service chaining, resource management, and trouble shooting. Efforts in these research areas will enable the development of future virtual-network-function platforms and innovation in service management while maintaining acceptable capital and operational expenditure.

key words: *network function virtualization, software-defined networking, service chain, policy management, resource management*

1. Introduction

Many different types of network devices, including switches, routers, and middleboxes [1], such as network address translation (NAT) and firewalls, are deployed in today's enterprise, data-center, and internet-service-provider (ISP) networks. Figure 1 illustrates a general model of network devices including switch, router, and middlebox. Packet processing modules (PPMs) that terminate network interface and handle layer 2 and 3 packet processing are connected to ports of switch module (SM). Control modules (CMs) that handles control and management planes, e.g., routing and signaling protocols, are connected to ports of SM. PMMs that handle higher than layer 3 are connected to ports of SM. These network devices are vertically integrated monolithic systems; individual vendors exercise proprietary design for system architecture of their products. These devices are becoming more complicated because network-equipment vendors frequently add new features to their products, resulting in thousands of embedded features and protocols with tens of millions of lines of source code [2]. Therefore, these network devices are *black boxes*, that is, it is sometimes difficult to diagnose network-service problems caused by failure of such devices and difficult to improve such devices by introducing new features. Discontinuing the production of a certain component could risk the failure of the entire network-device system (so-called "end of life (EOL)"). Softwarization is ushering in a new paradigm for disaggregation of traditionally vertically integrated network devices. Software-defined networking (SDN) and network function virtualization (NFV) are the first steps in disaggregation: they disaggregate vertically integrated systems into software and hardware components.

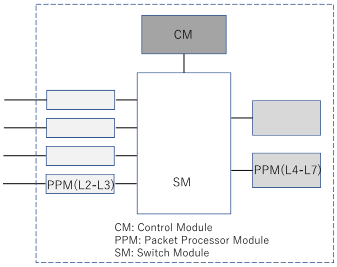


Fig. 1 General model of network devices including switch, router, and middlebox.

Software-defined networking [3], [4] decouples the control plane from network devices such as routers and switches. The logically centralized controller determines network control including reachability, load balancing, access control, security, and interface configuration. The management of Internet Protocol (IP) networks is becoming difficult as they serve as infrastructures for a wide range of services and applications; therefore, providing reliability and high performance. Control over the paths to deliver traffic with target performance objectives (also known as *traffic-engineering*) is difficult to execute with traditional routing mechanisms implemented in switches or routers (e.g., OSPF, BGP, SPT). The routing control platform [5] and path computation element [6] are the results of early efforts in separating the control plane from switches or routers. The 4D architecture [7] was proposed to refactor network management to decouple the control plane from the hardware data plane (forwarding plane), benefiting simple decision logic in logically centralized control, which enables independent development and advances in the control and data planes. Separation of an

Manuscript received April 7, 2017.
Manuscript revised June 30, 2017.
Manuscript publicized July 21, 2017.

[†] The author is with Faculty of Knowledge Engineering, Tokyo City University, Tokyo, 158-8557 Japan.

a) E-mail: shiomoto@tcu.ac.jp

DOI: 10.1587/transcom.2017EBI0001

intelligent control plane and simple data plane allows us to visualize network flexibility required in enterprise, data-center, and service-provider networks. Advanced features, such as traffic-engineering and virtual private networks (VPNs), can be implemented as part of the decision logic in the logically centralized controller. SANE [8] and Ethane [9], [10], which use a centralized controller for admission and routing control of flows to improve reliability and security of enterprise networks, are the result of early SDN proof-of-concepts and provide the foundation for openflow architecture [11]. A fair amount of research has been conducted, as reported in several survey studies [3], [4], [12]–[15].

NFV [16], [17] decouples NFs or SFs implemented in the data plane from middleboxes as software running on commodity off-the-shelf (COTS) X86 servers due to advances in IT virtualization technologies (e.g., virtual machine (VM) and container). SF is realized by an associated NF and therefore we use NF and SF interchangeably hereafter. Service functions implemented in the data plane can be complex; they modify traffic across multiple protocol layers (e.g., packet-header rewriting, payload rewriting, and session-level behavior modification); therefore, they are normally implemented as software. Thus, a middlebox is implemented as an appliance that runs service function software on a platform consisting of general purpose processors (GPUs) with possible optional hardware accelerators. By decoupling functions from the hardware components of middleboxes, NFV brings about benefits of elastic capacity, resource efficiency, performance, resiliency, and fault tolerance. A group of middleboxes will be used to implement a network policy, which typically requires a packet to go through a series of middleboxes, which is called *SF chaining*. For example, a network policy in an enterprise network requires that HTTP traffic incoming from the Internet should go through a series of an IP firewall (IP-FW), intrusion-detection system (IDS), and proxy, and non-HTTP traffic should go through an IP-FW and IDS. Both SDN and NFV will have to evolve to maximize the benefits of coordinated software-based control and data planes. Several studies [16], [18], [19] surveyed research on NFV, gave an overview of NFV projects and standardization efforts, and identified research directions.

SDN and NFV are promising driving technologies to disaggregate vertically integrated system into components by using “softwarization”. Software-defined networking separates the control plane from the data plane of a switch and routers, while NFV decouples high-layer SFs implemented in the data plane of a middlebox and enables innovation in policy implementation by using SF chaining. Both SDN and NFV will have to evolve to maximize the benefits of coordinated software-based control and data planes. Even though there have been several survey studies in this area [3], [4], [12]–[16], [18], [19], this area is continuing to grow rapidly; therefore, it is useful to survey state-of-art research activities. NFV are rapidly growing in topics such as re-architecting middleboxes, state management, high-performance platforms, service chaining, and resource management. In this paper, we present a recent survey of these areas with a special focus on the above topics, with which we have been witnessing remarkable advances in the past few years. The rest of the paper is organized as follows. In Sect. 2 we give an overview of the technical aspects of today’s middleboxes to understand the problems to be solved by NFV. We survey reseach and discuss challenge in the areas of re-architecting middlebox (Sect. 3), high-performance platform (Sect. 4), state management (Sect. 5), service chaining (Sect. 6), resource management (Sect. 7), and trouble shooting (Sect. 8). In Sect. 9 we conclude the paper.

2. Middlebox

The end-to-end argument of the initial Internet principle [20] is rarely seen in today’s enterprise and internet-service-provider (ISP) networks. A middlebox [1], which executes more complex packet processing functions in the data plane than normal IP routers, is placed like “bumps-in-the-wire” between a source host and destination host to provide value-added services to customers, including improved performance, improved security, and reduced bandwidth costs. Network deployments handle changing applications, workload, and policy requirements via the deployment of middleboxes. Today, middleboxes are expensive and closed systems and are acquired from independent vendors and deployed as standalone devices with little cohesiveness in how the ensemble of middleboxes is managed. A recent survey of 57 enterprise networks [21]–[23] revealed that today’s middlebox infrastructure requires high capital and operational expenditure (CAPEX and OPEX) and induces new failure modes. The number of middleboxes is comparable to the number of routers and switches in enterprise networks, which could translate into high capital expenditure of millions of dollars in five years for very large enterprise networks (with more than 100k hosts). Very large enterprise networks could even deploy several categories of middleboxes, which requires broad expertise and consequently large management teams. Such management teams perform daily tasks of monitoring and diagnostics, policy configuration tasks on-demand, and appliance configuration tasks in installment and replacement time, while performing long-term tasks including manufacturer interaction and training for upgrades occurring in roughly four-year cycles. Misconfiguration, overload, and physical and electrical failures are most common causes of failures in the deployment of middleboxes. Middlebox deployment induces high CAPEX, complex management inducing high OPEX, and failures from overload and physical and electrical failures.

These challenges can be solved by outsourcing middleboxes to a cloud-based middlebox infrastructure [2], [22], [23]. The middlebox-placement problem can be solved by using SDN. “Plumbing” and “services” are separated and under different organizations; an enterprise network only forwards data, and any additional processing is provided as cloud services. Network function virtualization is well in line with this direction. Cloud-service providers construct physical infrastructures using a farm of COTS X86 servers and have diversified types of middlebox applications that run on top of these physical hardware infrastructures.

Enforcement of network-wide policies involves network-management tasks ranging across stateful policy routing, access-control and rate-limiting traffic, and diagnostics for performance debugging, forensics for detecting malicious activity, etc. Among them, stateful policy routing for packet traverses a given sequence of middleboxes is challenging for its enforcement and verification. The root cause of this problem is that as packets traverse the network, their headers and contents may be dynamically modified by middleboxes. These modifications make it difficult to ensure that the desired set of policies are consistently applied throughout the network. This is particularly challenging because middleboxes often rely on proprietary internal logic for affecting such dynamic traffic transformations.

Details of middlebox functionality and deployment configuration is written in vendor manuals or not clearly described. There have been few studies describing diversified middlebox functionality and deployment configuration, except [1], [24], [25]. Some middleboxes rewrite the packet header, such as NAT, and others do payload such as the redundancy eliminator (RE). Even some middleboxes modify session-level behavior such as a wide area network (WAN)-optimizer. It is crucial to understand diversified types of middleboxes before we discuss the NFV research challenges such as service chaining, state management, high performance platforms, re-architecting middlebox, and trouble-shooting.

2.1 Diversified Types of Middleboxes

Diversified middleboxes are deployed in enterprise [22], ISP [26], and data-center networks [27]. The middleboxes such as IP Firewall (IP-FW), application firewall (AFW), NAT, load balancer (LB), proxy, deep packet inspection (DPI), intrusion detection system (IDS), intrusion-prevention system (IPS), virtual router (VR), redundancy eliminator (RE), WAN optimizer, etc. are deployed. Serving gateway (S-GW) and packet data network gateway (P-GW) are deployed in mobile carrier networks while Home Gateway (HGW) and broadband network gateway (BNG) are deployed for broadband residential access networks. Even though the above-mentioned middleboxes are not exhaustive, we review these middleboxes to see how much their functionalities differ before we discuss the NFV research challenges such as service chaining, state management, high performance platforms, re-architecting middlebox, and trouble-shooting.

IP-FW and AFW: The IP-FW monitors and drops IP packets based on the IP header and transport-layer header. It looks up the access control list (ACL) when it receives a packet. It drops the packet if the ACL dictates that it should not pass through. No packet modification is done. The stateless IP firewall typically allows all IP fragments to pass through because it does not have enough upper-layer header information to make a filtering decision. This middlebox reassembles IP fragments to avoid leaking IP fragments. The AFW acts as a protocol end point.

NAT: NAT is normally placed at the boundary between *public* and *private* domains for security reasons or convenient and flexible address management. It dynamically allocates a source port at its public IP address when a host in the *private* domain initiates a new Transmission Control Protocol (TCP) connection or sends a new UDP packet. It rewrites the IP address and transport layer port number fields to multiplex outbound IP packets over the same *public* IP address and to demultiplex inbound IP packets to a host in the *private* domain. Even though NAT seems to be a simple middlebox, different types of implementations exist and their behaviors differ: Restricted Cone NAT, and Symmetric NAT [28]. It should be noted that vendors may differ in their detailed implementation options for even the same class of middlebox.

LB: LB is placed in front of a farm of servers (e.g., Web, Domain Name System (DNS), etc.). When it receives the fist packet of a new flow to a Web site, it dynamically selects a Web-server instance for the flow and records it in the state table. It rewrites the destination IP address to that of the web server. The subsequent packets of the flow follow the same address translation. When it receives packets from a Web-server instance, it looks up the state table and sends the packet out after rewriting the source IP address to that of the LB. The LB can perform *persistent* load-balancing; it redirects a session from a certain customer to the same Web-server instance persistently using Cookie for the instance. Algorithms for Web-server-instance selection differ from vendor to vendor (e.g., leased-loaded, round-robin, etc.). Some vendors can use leased-loaded while others can use round-robin or a mixture of both. The layer-7 LB executes URI-based load-balancing. It terminates the TCP sessions and parses the HTTP GET request messages. It can also offload Secure Socekts Layer (SSL) processing for encrypted traffic.

Proxy: Proxy acts as both a server and client for the purpose of making requests on behalf of other clients. A “transparent proxy” is a proxy that does not modify the request or response beyond what is required for proxy authentication and identification. A “non-transparent proxy” is a proxy that modifies the request or response to the user agent, such as group-annotation services, media-type transformation, protocol reduction, or anonymity filtering. Web proxy terminates an IP flow and recreates another one in between the user agent and web server. A proxy is used for different purposes such as monitoring and filtering, performance improvement, anonymous service access, and security. Squid [29] is a caching and forwarding web proxy.

DPI: DPI [30] is a function that deeply inspects packets at an inspecting point in a network for various purposes such as bandwidth management, user profiling, network security, surveillance, network forensics, and censorship and content regulation [31], [32]. It deeply inspects a packet to recognize the characteristics contained in the packet. If it recognizes that the packet needs some treatment for its purpose, appropriate actions will be triggered. Characteristics include protocols, applications, URLs, media content, text string, special-format data, viruses, malware, and other cyber-security concerns. For example, for use of bandwidth management, DPI recognizes a user's application type and executes appropriate bandwidth allocation for each application (e.g., high priority on real-time applications and low for bandwidth-hungry but non-real-time applications). For use of network security, the DPI recognizes malicious traffic by inspecting certain bit patterns (signature) in packets as part of the IDS, as described below.

IDS and IPS: The IDS monitors a network to detect malicious activities or policy violations. For malicious detection activities, the IDS is categorized into two types: signature and anomaly based. The signature-based IDS searches for specific patterns in traffic, such as Byte-sequence pattern and known malicious instruction sequences used by malware. The anomaly-based IDS detects traffic deviated from a well behaved traffic model that is often constructed using machine learning algorithms and marks it as malicious traffic. The IPS can be considered as an extension of the IPS in that it also detects malicious traffic or policy violations, and it takes such actions as sending an alarm, dropping detected malicious packets, resetting a connection, or blocking traffic from the offending IP address. Bro [33], [34] is an open-source UNIX-based traffic-monitoring tool, which is often used as an IDS. Snort [35], [36] is an open-source UNIX-based IPS. Other IDS/IPS products include Palo Alto Networks [37], OpenVPN [38], Passive Real-time Asset Detection System (PRADS) [39].

VR: The VR instance is separated from the physical router platform to support live migration from one physical platform to another [40], [41]. The complexity of network management is reduced by relieving a task for maintaining consistency between the physical and logical configuration by using VR. Major router vendors released router virtualization technology to allow multiple VR instances running over the same physical router platform. Currently, VR runs on a COTS server platform (e.g., X86 machine) to deliver 10-Gb/s throughput per CPU core [42].

RE: The RE is widely used by data centers and enterprise networks to improve network efficiency by eliminating redundant data between end points [43]–[51]. It executes application-agnostics (protocol-agnostic) redundant elimination. An upstream RE device stores packets in memory over a certain period. Packet contents are indexed using *fingerprints*, which work as hooks pointing to content in random locations within the packet. For each incoming packet, the upstream RE device checks if the packet's fingerprints have appeared in earlier in-memory packets. Each matching fingerprint indicates a certain region of partial overlap between the incoming packet and an earlier packet. The matching packets are compared to identify the maximal region of overlap. Such overlapping regions are removed from the incoming packet and a shim, which provides the fingerprint that caused the match and byte range for the matching in-cache packet, is inserted to notify the downstream RE device of how to decode the packet. A packet can carry multiple shims, each potentially matching a different in-memory packet. The downstream RE device uses the shim in the encoded packet to retrieve the matching packet(s) and fills in the corresponding missing byte range(s). It is assumed that the cache on the downstream RE device is consistent with the upstream one. Redundancy eliminator can be implemented as software running on a commodity server [52].

WAN-Optimizer: A WAN-optimizer is used to improve data-transmission efficiency between two end points (e.g., between branch office and headquarters, between a data center and another data center) across a WAN. Even though most earlier WAN-optimizers have been used a TCP proxy to tackle high bandwidth-delay product issues in the TCP, they now use a variety of techniques, e.g., latency improvement, data compression, caching, redundancy elimination, traffic shaping, and forward-error correction. Several vendors provide WAN-optimizer products [51], [53]–[56].

S-GW and P-GW: Long-Term Evolution (LTE) networks consist of a radio access network (RAN) and evolved packet core (EPC) [57], [58]. A RAN includes eNodeBs that provide wireless access to user terminals. An EPC consists of network entities that both manage devices and route the data traffic. A S-GW and P-GW are routers that provide connectivity to devices. The GPRS Tunneling Protocol (GTP) is used to establish a session between eNodeB and P-GW. The control plane elements consist of the mobility management entity (MME), home subscriber server (HSS), and policy and charging rules function (PCRF). The HSS and PCRF are database servers for user-subscription information and quality of service/billing policies, respectively. An MME is the key control node that manages both device connectivity and mobility in the EPC network. In addition to being the entry point for control plane messages from devices, it manages other control plane entities using 3GPP standard specific well-defined interfaces.

HGW and BNG: An HGW and BNG are deployed for broadband residential-access networks [59], [60]. The Point-to-Point Protocol over Ethernet (PPPoE) or IP over Ethernet (IPoE) is used to establish a session for the customer admitted via authentication, authorization, and accounting procedures.

2.2 Traffic Modification by Middlebox

Different types of middleboxes are characterized by a few key attributes. For example, Qazi et al. [61] classifies different type of middleboxes in terms of key attributes, e.g., the inputs they operate on, their actions, time-scale at which dynamic traffic modifications occur, and information they require for operation. Regarding the inputs they operate on, they use either packet header, payload, flow, session, or a combination of them. Regarding their actions, they can take different actions, e.g., no change, drop packet, rewrite header, rewrite payload, redirect traffic, or map sessions. Regarding the time-scale at which dynamic traffic modifications occur, they can modify traffic at time-scale of either per-packet, per-flow, or per-session. We observe that different types of middleboxes operate at different granularity (per-packet, per-

Table 1 Middle box.

Middlebox	Modification	Actions
IP-FW and AFW	No	Pass or Drop
NAT	IP address, Port num.	Header conversion
LB	IP address	Address conversion and Switch
Proxy	Session	Session termination and re-creation
DPI	No	Inspect packet data
IDS and IPS	No	Inspect packet data and Drop or Report if malicious
VR	No	Forward packets
RE	Payload	Overlapping bytes are removed and fingerprinted at sending side, re-created at receiving side
WAN optimizer	YES	Latency improvement, data compression, caching, RE, shaping, FEC
S-GW and P-GW	YES	S-GW and P-GW for handling GTP tunneling in data plane, MME for mobility management in control plane
HGW and BNG	YES	HGW and BNG for handling PPoE or IPoE tunneling in data plane, Radius server for AAA in control plane

flow, per-session), modify traffic differently (rewrite header, rewrite payload, map sessions), and operate at different timescales (per-packet, per-flow, per-session). Operators have to pay attention to the impact of those behaviors of middleboxes when they place middleboxes in their networks and configure routing and policy. Table 1 lists diversified middleboxes and highlights how they modify packets and behave. Bearing those middlebox behavior in mind, in the rest of paper, we discuss the NFV research challenges such as re-architecting middlebox, high-performance platform, state management, service chaining, resource management, and trouble-shooting.

3. Re-Architecting Middleboxes

Network deployments handle changing applications, workload, and policy requirements via the deployment of middleboxes. Middleboxes are expensive and closed systems, acquired from independent vendors, and deployed as standalone devices with little cohesiveness in how the ensemble of middleboxes is managed. A middlebox is developed and managed as a standalone device. That is, a new type of middlebox is developed and introduced into the infrastructure as a one-shot solution to a specific requirement. In addition, each middlebox has its own configuration command line interface; therefore, it is difficult to manage different types of middleboxes in a unified way. Infrastructure hardware resource is under-utilized because each middlebox is provisioned for its own peak load hour, which may be different from one to another. To address these issues, several research projects have been conducted to re-architect middleboxes [62]–[65].

3.1 Software-Based Programmable Router

Click [66] provides a framework that allows us to build routers by composing graphs of *elements*, each having a single simple atomic function. A Click router [66] is built from a set of packet processing modules called *elements*, which implement basic router functions. Click has been used as an NFV platform because it allows developers to easily compose middlebox functionality by reassembling the elements provided by Click and customized elements developed by the developers. Different from the computer architecture assumed when Click was originally developed, modern commodity hardware exhibits characteristics such as hardware multi-queue, multi-core processors and non-uniform memory access (NUMA). Barbette et al. [67] carried out an extensive study of the integration of packet-processing mechanisms and user-space packet I/O framework into the Click modular router and proposed *FastClick* by incorporating a high-speed packet I/O framework to enhance the performance of Click. They demonstrated that FastClick with both Netmap and DPDK exhibits up to about 2.3 × increase in speed compared to conventional software implementations. Click does not provide native TCP support for applications; Therefore, its applicability to middlebox applications is limited. Click does not provide blocking I/O; therefore, it leads to waste of CPU resources while being busy waiting on I/O. To work around these two limitations of Click, Laufer et al. [68] proposed a Click-based middlebox architecture called CliMB, which introduces the concept of blocking tasks in Click to allow network applications to efficiently wait on I/O without consuming CPU resources. It also provides a full-fledged modular TCP implementation, supporting blocking and non-blocking I/O as well as socket and zero-copy APIs for application portability and high performance.

Sekar et al. [62], [63] proposed *CoMb*, a middlebox infrastructure that converts expensive and specialized standalone middlebox devices into software-centric implementations of consolidated to run a shared COTS hardware platform by decoupling middlebox application software from a standalone device. The CoMb targets consolidation at two levels: individual middleboxes and managing an ensemble of middleboxes. At the level of an individual middlebox, by having middlebox applications run on a consolidated hardware platform, they reuse low-level modules such as packet capture, parsing headers, reconstructing session state, and parsing application-layer protocol. At the level of managing an ensemble of middleboxes, each CoMb box runs multiple software-based middlebox applications. The network controller of CoMb assigns processing responsibilities across the network. Sekar et al. presented a CoMB single-box design assuming a number of parallel computation cores and multiple queues at the NIC. By having all middlebox applications within a service chain (what they call *hyperapp* in their paper) run on the same processor core and associating each queue of the NIC with a *policy shim (pshim)* layer representing a *hyperapp*, it avoids inter-core synchronization overheads and shared data structures across cores. They compared it with having identical middlebox applications run on the same core and demonstrated its advantage compensates its disadvantage of it could incurring overhead due to the context switches and contention over shared resources on a single core. They demonstrated proof-of-concept by implementing a Clik router-based platform.

The CoMb is targeted at per-packet processing; therefore, it is less applicable to stream or flow-based processing, which is common to the class of load-balance switch (LBS) middleboxes. Such middleboxes require operating on a byte-stream rather than individual packets and require communication among multiple network elements to execute dynamic forwarding and rewriting. The TCP connection collapsing and HTTP request rewriting for persistent connections are also required for LBS middleboxes. To meet these goals, Anderson et al. [64] proposed an extensible open middlebox (xOMB) software architecture for building flexible, programmable, and incrementally scalable middleboxes based on commodity server components. The xOMB consists of commodity hardware switches, front-end software middleboxes, back-end application servers, and a controller for coordination. A front-end software middlebox parses, processes, and forwards streams of requests and responses between clients and the back-end servers. The xOMB supports arbitrary protocol and application logic through user-defined processing modules (PMs). Deployments can scale processing capacity by stacking xOMB servers in either way that every server runs the same modules or that servers run different modules and form a processing chain. To handle arbitrary byte streams, they terminate client TCP connections at the middlebox, execute the appropriate modular processing pipeline containing user-defined processing logic on an incoming byte stream, then transmit the resulting byte stream over a new TCP connection to the appropriate back-end server. The xOMB [64] demonstrates the feasibility of constructing extensible middleboxes with commodity servers and operating systems (OSes). Through the use of general programmable network-processing pipelines, the xOMB creates a service chain in one single OS. The xOMB uses a general programmable pipeline for network processing, composed of xOMB-provided and user-defined C++ modules responsible for arbitrary parsing, transforming, and forwarding messages and streams. It provides a single, unified platform for implementing the various functions of static load balancing/filtering, dynamic request routing, and protocol acceleration.

Greenhalgh et al. [65] discussed a class of system architectures for building in-network processing platforms called “flowstream architectures” to strike a balance between performance, scalability and flexibility. They proposed the implementation of network functionalities in virtualized machines/servers/routers running on top of commodity PCs. The traffic flow among these virtual network entities is controlled by a programmable network switch implementing Openflow. A flowstream architecture consists of an Ethernet switch and commodity servers attached to the Ethernet switch via ports for additional processing. The switch and servers are managed as a single platform from the operators’ point of view by a controller.

Many types of middleboxes execute a variety of advanced network functionalities. A closer look at these middleboxes reveals that they have many shared primitive functions: most middleboxes execute packet header analysis, and many middleboxes examine the application payload and/or reconstruct the TCP session. Bremler-Barr et al. [25] proposed *OpenBox*, a logically-centralized framework that decouples the control plane of middleboxes from their data plane. They investigated different types of existing middleboxes and modeled data plane processing of middleboxes as an ordered list of stages of the unified processing defined by a set of rules consisting of header match, payload match, and instructions. They showed how multiple middlebox applications can be implemented by using a subset of processing stages to provide the requested functionality. As mentioned above, OpenBox decouples the control plane of middlebox from their data plane. A high-level control plane defines monitoring and performance goals. A low-level data plane defines the processing stages of multiple middleboxes as instructed by the control. The data plane is composed of either separate or consolidated OpenBox service instances (OBIs) that provide the necessary functionality of the different processing stages and low-level processing entities that perform one or more stages of the unified processing stages. Each OBI receives a *processing graph* (PG) and a set of processing rules from the *OpenBox controller*, which is a logically centralized server that communicates with OBIs and is in charge of making service chains and enforcing them by communicating with the OBIs. Bremler-Barr et al. defined the OpenBox protocol that allows adding and removing rules, specifying PGs, sending alerts and log messages, and inquire OBIs for statistics. They defined the box abstraction layer as an abstract API for writing OpenBox applications to specify a processing path and a set of rules. OpenBOX is expected to replace legacy middlebox architecture, make advanced packet processing easier and faster, provide more flexible and scalable deployment, and allow innovative applications to be easily developed.

The move from hardware middleboxes to software Network Functions (NFs) as advocated by NFV has proven challenging. Developing new network functions remains tedious, while providing isolation between NFs relies on VM or container technologies; therefore, incurs high performance overhead. Panda et al. [69] proposed Net Bricks to address this challenge in both *building* and *running* middleboxes. Regarding building middleboxes, they argue that NetBricks supports only a limited set of abstractions whose actions can be optimized through user code rather than supporting a totally general framework where the developer must take on the task of optimizing the resulting code, as in Click [66]. NetBricks is focused on a core set with well-known semantics and highly optimized implementations and provides the necessary generality by allowing customization through the user of user-defined functions. Regarding running middleboxes, they proposed a technique to compile-time and runtime checks to enforce memory isolation in software (they call this technique *zero-copy software isolation*) rather than relying on isolation provided by using VM and container technologies which incur significant performance penalties. NetBricks builds on Rust, a safe language and runtime environments, and uses Low Level Virtual Machine (LLVM) [70] to provide the same memory isolation as containers and VMs, without incurring the same performance penalties by relying on compile-time and runtime checks to enforce memory isolation in software. Recent advances in language and runtime design with the widespread adoption of LLVM as an optimization backend for compilers has improved performance of safe languages and runtime environments which provide memory isolation mechanisms in software.

Palkar et al. [71] proposed *E2*, a framework for NFV applications with motivation for scale-out central offices where BNGs and EPC gateways are located supporting a wide range of higher-level traffic-processing functions - content caching, DPI, parental controls, WAN and application acceleration, traffic scrubbing for Distributed Denial of Service (DDOS) attach prevention, and encryption - in addition to traditional NFs. The E2 framework meets system requirements such as NF placement, elastic scaling, service composition, fault tolerance, and monitoring. It automates the common tasks of placement, service interconnection, and dynamic scaling. It simplifies the building, deploying, and managing of NFs. The policy statement called *pipelets* defines a traffic class and directed acyclic graph that captures how this traffic class should be processed by NFs. The E2 framework is responsible for implementing the policy statements that are translated from network-wide policies by the SDN controller. It uses three components: (1) scaling, (2) placement, and (3) interconnection. The scaling component dynamically computes the required number of NF instances to support dynamically varying traffic demand and generates an instance graph called *iGraph*. The placement component translates the iGraph into an assignment of NF instances to specific servers. The interconnection component configures the network to steer traffic across appropriate NF instances. Palkar et al. implemented a data plane using SoftNIC [72], a software switch that allows packet PMs to be configured as a data-flow graph in a similar manner to Click [66].

Anwer et al. [73] proposed *Slick*, a programming framework that allows a programmer to write a single high-level program based on Python to implement policy. They developed a programming abstraction where a specific function is implemented as an *element*, in a similar manner to Click [66], to compose custom network functions and specify SF chains. Slick runtime implements the programming abstraction by decomposing it into constituent functions and placing those functions at appropriate locations. Slick runtime executes both placement of modular packet-processing elements and steering traffic through those elements.

Bezahaf et al. [74] proposed *FlowOS*, a flow-based programmable platform for commodity hardware middleboxes. It provides a development environment for flow PMs that can be regarded as a middlebox that works on a stream of a flow. It provides a socket-like interface for writing middlebox software that processes application byte streams instead of IP packets. It exposes the byte streams in a clean way and hides the intricacies of the IP packet structure or TCP segments.

Table 2 lists summary of related work on software-based programmable router to re-architect middlebox.

3.2 Hardware-Based Programmable Router

Bosshart et al. [77], [78] proposed the reconfigurable match tables (RMT) model for high-speed programmable data switch. In the RMT model, a programmer is allowed to (1) modify the definitions of a field, (2) modify specification of match tables, and (3) define new actions. Specifically, it allows the definition of arbitrary headers and header sequences, arbitrary matching of fields by an arbitrary number of tables, arbitrary writing of packet-header fields (but not the packet payload), and state update per packet. An RMT switch consists of a parser followed by a number of match stages. The parser allows field definitions to be modified or added. The parser output is a packet-header vector, which is a set of header fields with meta-data fields. The vector flows through a sequence of match stages, each of which abstracts a logical unit of packet processing. Each logical match stage allows the match-table size to be configured. An input selector selects the fields to be matched. Packet modifications are done using a wide instruction, very long instruction word that can operate on all fields in the header vector concurrently. There is an action unit for each field F in the header vector, which can rewrite F by taking input arguments of the header vector and the action data results of the match. Control flow is made possible by an additional output. The next-table-address from each table match provides the index of the next table to execute. In summary, the RMT model allows new fields to be added by modifying the parser, new fields to be matched by modifying match memories, new actions by modifying stage instructions, and new queuing by modifying the queue discipline for each queue. The forwarding plane can be changed in the field without modifying hardware. Bosshart et al. demonstrated a 640-Gb/s switch chip (10-Gb/s per port) using an industry standard 28 nm process.

Bosshart et al. [79] proposed “P4”, a high-level language for programming protocol-independent packet pro-

Table 2 Summary of research projects on "re-architecting middlebox".

Research project	Type	Distinctive Features
"Click", Morris et al. [66], [75], [76] (1999)	Building block for software-based router	Click provides a set of packet processing modules called *elements* that allows developers to compose a middlebox functionality by reassembling the elements.
"Flowstream Architectures", Greenhalgh et al. [65] (2009)	Ethernet switch and commodity servers	The switch and servers are managed as a single platform by a controller. Virtualized machines/servers/routers run on top of commodity PCs.
"CoMb", Sekar et al. [62], [63] (2012)	Consolidated middlebox	Reusing low-level modules such as packet capture, parsing headers, reconstructing session state, parsing application-layer protocol. All middlebox applications within a service chain run on the same core of processor. Click router-based implementation.
"xOMB", Anderson et al. [64] (2012)	eXtensible open MiddleBox software architecture	xOMB consists of commodity hardware switches, front-end software middleboxes, back-end application servers, and a controller for coordination. It realizes a service chain in one single OS. A front-end software middlebox terminates client TCP connections, executes user-defined modular processing pipeline, and transmits the resulting byte stream over a new TCP connection to a back-end server.
"FlowOS", Bezahaf et al. [74] (2013)	a flow-based platform	It provides socket like interface for writing middlebox software that process application byte streams instead of IP packets.
"OpenBox", Bremler-Barr et al. [25] (2015)	a logically-centralized framework	Middleboxes modeled as a ordered list of stages of the unified processing defined by a set of rules consisting of header match, payload match, and instructions.
"E2", Palkar et al. [71] (2015)	a framework for NFV applications for BNGs and EPC	Policy statement called *pipelets* defines a traffic class and directed acyclic graph (DAG) that captures how this traffic class should be processed by network functions. It employs three components: (1) scaling component, (2) placement component, and (3) interconnection component.
"Slick", Anwer et al. [73] (2015)	Python programming framework to implement policy.	A function is implemented as *element* in a similar manner of Click [66]. Slick runtime performs placement of modular packet processing elements and steering traffic.
"FastClick", Barbette et al. [67] (2015)		Integration of high-speed packet I/O and userspace packet I/O framework (e.g., netmap and DPDK) into the Click modular router.
"NetBricks", Panda et al. [69] (2016)	A limited core set of abstractions with well-known semantics	Compile-time and runtime checks to enforce memory isolation in software called *Zero-Copy Software Isolation (ZSCI)*. NetBricks builds on a safe language (Rust) and uses LLVM [70] for software isolation.
"CliMB", Laufer et al. [68] (2016)	Click improvement	It provides a modular TCP implementation, supporting blocking and non-blocking I/O as well as socket and zero-copy APIs.

cessors. It works in conjunction with SDN control protocols such as OpenFlow. Its goals include (1) reconfigurability in the field, (2) protocol independence, and (3) target independence. Regarding reconfigurability, the controller should be able to redefine the packet parsing and processing in the field. Regarding protocol independence, the controller should be able to specify (i) a packet parser for extracting header fields with particular names and types and (ii) a collection of typed match+action tables that process these headers. Switches forward packets via a programmable parser followed by multiple stages of match+action, arranged in series, parallel, or a combination of both. An abstract model generalizes how packets are processed in different forwarding devices and by different technologies. This allows us to devise a common language, P4, to represent how packets are processed in terms of a common abstract model. The forwarding model is controlled by two types of operations: configure and populate. Configure operations program the parse, set the order of match+action stages, and specify the header fields processed by each stage. Populate operations add and remove entries to the match+action tables that were specified during configuration. Arriving packets are first handled by the parser. The extracted header fields are then passed to the match+action tables. A packet-processing language allows the programmer to express serial dependencies between header fields. Dependencies can be identified by analyzing table-dependency graphs (TDGs). A two-step compilation process is used. A compiler translates the P4 representation to TDGs to facilitate dependency analysis then maps a TDG to a specific switch target. The P4 language is designed to make it easy to translate a P4 program into a TDG.

Song et al. [80] proposed protocol-oblivious forwarding (POF) to remove the dependency on protocol-specific configurations on forwarding elements (FEs) and to enhance the data path with new stateful instructions. An FE extracts and assembles the search keys from the packet header, conducts the table lookups, and executes the associated instructions. The FEs do not have any information on any forwarding protocols. The packet parsing is directed by the controller through a sequence of generic key-assembly and table-lookup instructions. To achieve this, they expand the packet meta-data as a generic scratch pad associated for each packet in the processing pipeline. Song et al. argue that a concise set of protocol-independent instructions is sufficient to design a simpler and more generic forwarding plane model. They proposed the flow instruction set. The search key is simply defined by one or more {*offset, length*} tuples. Forwarding instructions are made protocol agnostic. For the instructions that manipulate the packet or meta-data (e.g. insert, delete, and modify), {*offset, length*} is used to locate the target data. A new instruction *AddField* is added to cope with the situation in which we need to push a new header. Simple mathematical instructions (addition, subtraction, and shift) and logic (AND, OR, NOT, and XOR) are also added. Instructions that allow the data path to actively manipulate the flow tables are added. They allow each flow entry to invoke the same set of instructions with different parameters efficiently by storing the unique parameters with flow entries and letting the flow entries that share the same set of instructions point to a common instruction block. Statistic counters are treated as a shared resource that is allocated on an on-demand basis. Possible lookup tables are categorized into different types such as direct table, exact match, longest prefix match, and masked match.

In terms of programmability, recent programmable switch chips [77], [78], [81]–[83] allow only packet parsing and forwarding but do not state modification in the data plane while many data plane algorithms create and modify the algorithmic state such as congestion control, scheduling, measurement, and active queue management. Sivaraman et al. [84] proposed an abstraction to program and implement data plane algorithms called a *packet transaction*. It is a sequential code block that is atomic and isolated from other such code blocks, with the semantics that any visible state is equivalent to a serial execution of packet transactions across packets in the order of packet arrival. Packet transactions let the programmer focus on the operations needed for each packet without worrying about other concurrent packets. To enable packet transactions, they make three contributions. First, *Banzai*, a machine model for programmable line-rate switches that models two important constraints for stateful line-rate operations: the inability to share a state between different packet-processing units, and the requirement that any switch-state modification be visible to the next packet entering the switch. Banzai abstracts pipelines consisting of match-action tables arranged in stages at the ingress and egress of recent programmable switch architectures and extends them with stateful processing units to implement a data plane algorithm. Second, *Domino*, a new domain-specific language for data plane algorithms with packet transactions at its core, is an imperative language with C-like syntax. Third, a compiler from Domino-packet transactions to a Banzai target, which extracts *codelets* from transactions: code fragments, which if executed atomically, guarantee a packet transaction's semantics. It then uses program synthesis to map codelets to atoms, rejecting the transaction if the atom cannot execute the codelet. Sivaraman et al. demonstrate that these targets are feasible in a 32-nm standard cell library in area relative to a 200 mm^2 baseline switching chip.

Li et al. [85] proposed "ClickNP", a field-programmable gate array (FPGA)-accelerated platform for flexible and high-speed NFs achieving 40-Gb/s line rate with commodity servers. It exhibits a modular architecture similar to the Click router. Elements are programmable and written in high-level C-like languages. ClickNP elements are compiled into binaries on the CPU and low-level hardware description language for FPGAs. High-level synthesis (HLS) tools are used for this. ClickNP uses a set of optimization techniques to use massive parallelism in FPGAs. ClickNP organizes each element into a logic block in an FPGA and connects them with first in, first out buffers so that all these elements blocks can run in full parallel. The processing function is written to minimize the dependency among operations, which allows the HLS tools to generate maximum parallel logics. Delayed write and memory scattering is used to address the read-write dependency and pseudo-memory dependency. Operations in different stages are balanced to match their processing speed to maximize the overall throughput of pipelines. The ClickNP host process has one manager and zero or multiple worker threads. The manager thread loads the FPGA image into the hardware, starts worker threads, initializes ClickNP elements in both the FPGA and CPU, and controls their behaviors by sending signals to elements at runtime. Each worker thread may process one or more modules if they are assigned to the CPU.

Table 3 lists summary of related work on hardware-based programmable router to re-architect middlebox.

4. High-Performance Platform

4.1 Software Router

There are a lot of works on high-performance packet processing. Veal et al. [86] investigated the performance scalability of a multi-core Web server. They found that flow-level parallelism is well exploited; thus, the performance is scaled. An address bus is a bottleneck of scaling performance, and the performance is scaled until the capacity of the address bus is saturated. Bolla et al. [87] investigated the architectural bottleneck of software routers running on PCs with multi-core processors. Santos et al. [88] proposed changing the architecture to Xen to overcome the overhead issues of the driver-domain model. They moved the copy operation to the guest CPU to increase the cache locality, used hardware support of NICs to place data directly into the guest domain to avoid data copy between domains, and relaxed the memory isolation property to minimize the cost of granting the driver domain access to the guest-domain pages.

Dobrescu et al. [89] proposed "RouteBricks", a software router architecture that exploits parallelism across multiple servers as well as multiple cores within a server. RouteBricks is a cluster router architecture, which is aimed to scale up to ones with hundreds of 10-G/bs ports. All clusters of commodity servers are interconnected in a full-mesh topology, reducing the interconnecting link speedup ratio. Direct valiant load balancing (Direct VLB) [90] is used to achieve robust load-balancing against a variable and unpredictable traffic matrix. To minimize packet re-ordering in TCP or User Datagram Protocol (UDP) flow, a set of same-flow packets arriving at the cluster within δ ms from one another are sent through the same intermediate node like the Flare load-balancing method [91]. Dobrescu et al. note that due to limited per-node fan-out, a low-degree multi-hop topology

Table 3 Summary of research projects on "re-architecting middlebox (forwarding)".

Research project	Type	Distinctive Features
Bosshart et al. [77], [78]	reconfigurable match tables (RMT) model	Programmer is allowed to (1) modify the definitions of field, (2) modify specification of match tables, (3) define new actions. 640 Gb/s switch chip (10Gb/s per port) using an industry standard 28nm process.
"Protocol-oblivious Forwarding (POF)", Song et al. [80]	Protocol-free configuration on forwarding elements	The search key is simply defined by one or more {*offset, length*} tuples. Instructions for mathematics, logic, and header and flow manipulation are defined.
Sivaraman et al. [84]	abstraction to program and implement data plane algorithms	"Banzai", a machine model, abstracts pipelines consisting of match-action tables arranged in stages. "Domino", a new domain-specific language (DSL) for data plane algorithms, is an imperative language with C-like syntax.
"ClickNP", Li et al. [85]	FPGA-accelerated platform	It exhibits a modular architecture like Click router. Elements are programmable and written in high-level C-like languages. ClickNP elements are compiled into binaries on CPU and low-level hardware description language (HDL) for FPGAs. ClickNP host process has one manager and zero or multiple worker threads. The manager thread loads the FPGA image into the hardware, starts worker threads, initializes ClickNP elements in both FPGA and CPU, and controls their behaviors. Each worker thread may process one or more modules if they are assigned to CPU.

Table 4 Summary of research projects on "high performance platform" (Part 1).

Research project	Type	Distinctive Features
Veal et al. [86] (2007)	Performance scalability of a multi-core Web server.	Flow-level parallelism is well exploited. Address bus is a bottleneck.
Santos et al. [88] (2008)	Xen architecture change	It overcomes the overhead issues of driver domain model.
Ram et al. [92] (2009)	Overhead of network I/O of driver domain I/O in Xen.	Multi-queue NIC is supported in driver domain model in Xen.
"RouteBricks", Dobrescu et al. [89] (2009)	Software router architecture based on a cluster router architecture.	All clusters of commodity servers are interconnected in a full-mesh topology, where Direct Valiant load balancing (Direct VLB) [90] is employed. Each packet should be handled by a single core to avoid overhead of synchronizing cores to transfer the packet and potential additional L3 cache misses.

based on a k-ary n-fly (a generalized butterfly topology that interconnects N nodes with $n = \log_k N$ stages of k-degree nodes) will be, in practice, used for large-scale nodes. To exploit parallelism at the CPU level, each packet should be handled by a single core to avoid overhead of synchronizing cores to transfer the packet and potential additional Level 3 (L3) cache misses. Batch packet processing is used to reduce overhead per-packet book-keeping overhead-reading and -updating socket-buffer descriptors and the data structures that point to them. They demonstrated that there is no difference in throughput for different data placements contrary to the preliminary concern on throughput drops in the NUMA architecture. They constructed a prototype called RB4, which implements Click-based software routers on four Nehalem servers interconnected through a full-mesh topology with direct-VLB routing with 10 Gb/s of external links.

Table 4 lists the summary of related work in the area of software router architecture.

4.2 Recent Technologies for High-Performance General Purpose Processors

Virtual Network Function (VNF) is a network function that was embedded into a dedicated hardware appliance, implemented as software that is supposed to run on VM or container. A frequently raised issue about VNFs is performance. By having network SFs implemented as software and running on COTS X86 server, NFVs could lead to large variations in latency and erratic throughput even when the underlying physical resource is only lightly used. For example, Wang and Ng measured the end-to-end networking performance of the Amazon EC2 cloud and found that very unstable TCP/UDP throughput, fluctuating between zero and 1 Gb/s at the tens of ms time granularity, and the delay variations among Amazon EC2 instances can be 100 times larger than most propagation delays, which are smaller than 0.2 ms, even when the network is not heavily loaded [93]. The unstable networking characteristics caused by virtualization can obviously affect the performance and deployment of virtual appliances.

Three challenges need to be addressed to accomplish software-based packet processing at line speed, i.e., interruption, data copy, and network I/O. First, packets arrive at unpredictable times, causing interruptions in notifying an OS that received packet data are ready for processing. Handling millions of interruptions per second quickly exceeds the time spent packet processing at 10 Gb/s. Second, the

OS reads incoming packets into kernel space then copies the data into user space. Those packet copies will exacerbate performance, ending up with a longer latency and lower throughput, especially when we conduct a chain of NFs across multiple VMs. Finally, network I/O in virtualization can have even greater overheads with additional copies between the hypervisor and guest OSes. Again those additional copies will exacerbate performance, especially when we conduct a chain of NFs across multiple VMs.

The new API (NAPI) has been supported by Linux 2.6 and later. The NAPI is a modification of the packet-processing framework in Linux device drivers to improve the performance of high-speed networking. It disables some interruptions when the network-traffic load is high and switches to polling the devices instead of avoiding frequent interruptions sharing the same message that there are many packets to process. Another advantage of this polling-based approach is that when the kernel is overwhelmed, the packets that cannot be handled in time are simply dropped in the device queues (i.e., overwritten in the incoming buffer).

Intel Data Plane Development Kit (DPDK) [94] is another software-based acceleration for high-speed networking applications that also uses polling to avoid the overhead of interrupt processing. Intel DPDK reduces the overheads caused by interruption and data copies by allowing user-space applications to directly poll the NIC for data. It uses huge memory pages and allows applications to write and read data directly into these pages using direct memory access (DMA). It allows applications to access the NIC card directly without involving kernel processing. While DPDK enables high throughput user-space applications, its pass-through mode that provides DMA to and from a VM can have lower performance than native network I/O. While DPDK supports single-rooted I/O virtualization (SR-IOV) to allows multiple VMs to access a single NIC, and a MAC address is used to switch packets in a virtual switch. Extra data copies are required if packets are forwarded between VMs.

Ram et al. [92] developed two mechanisms to address the issues on the overhead of network I/O of the driver-domain I/O virtualization model in Xen. First, a multi-queue NIC is supported in the driver-domain model in Xen to eliminate the overheads of packet de-multiplexing and copying. Second, to reduce the grant overhead for memory-access protection in the driver-domain model, a grant-reuse mechanism is developed based on a software I/O address-translation table.

Egi et al. [95] investigated the performance of the software router Click [66], [75], [76] running on multi-core CPUs with multi-queue NICs. They revealed that multi-queuing does not completely eliminate the need for task synchronization in software routers and that replication of full forwarding paths on CPU cores offers the best allocation policy because it is better at using spare CPU cycles through higher parallelism.

Rizzo et al. [96] proposed "netmap", a framework that enables OSes to handle millions of packets per seconds. To remove data copying, meta-data management, and system call overheads, they implemented a number of techniques in FreeBSD and Linux: a lightweight meta-data representation, pre-allocated linear fixed size packet buffer, granting applications direct-protected access to the packet buffers, and supporting multi-queue NIC.

Kim et al. [97] proposed "The Power of Batching in the Click Modular Router." They investigated a method to run the Click in recent high-performance commodity servers consisting of multi-core CPUs in a NUMA architecture with multi-queue NIC. They demonstrated that batching in terms of both packet I/O and computation is crucial, and we should take care of NUMA architecture and multi-queue NIC for multi-core CPU.

Recent work by Hwang et al. [98] extends the DPDK libraries to provide low-latency and high-throughput networking in virtualized environments. The NetVM platform extends DPDK to achieve high-speed inter-VM communications [99], [100]. It allows zero-copy delivery of packet data to VMs through a small shared ring-buffer shared between the hypervisor and each individual VM that is used to transmit packet descriptors. NetVM also uses a huge page region shared with a group of trusted VMs that allows chained NF applications to directly read or write packet data. It introduces a lockless memory sharing design, which uses multiple queues and receive-side scaling (RSS). Therefore, it does not require synchronizations to allow one application to have control of the descriptor containing a packet's address, preventing overheads caused by inter-core communications and context switching, which could cause the system to fall behind; thus, may result in tens of packets being dropped. It also introduces NUMA-aware design to ensure that as a packet is processed by either the host or guest OS, it always stays in a local memory bank, and cache lines will never need to be passed between sockets. NetVM implementation includes the NetVM manager and NetVM core engine, drivers for an emulated PCI device, modifications to the KVM's CPU-allocation policies, and NetLib (library for building in-network functionality in a VM's user space). The NetVM manager runs in the hypervisor and provides a communication channel to the NetVM core engine. The NetVM core engine, which is a DPDK user-space application running in the hypervisor, receives packets and delivers them to VMs using zero copy. It also communicates with the NetVM manager to synchronize information about new VMs. It polls the NIC to read packets directly into the huge page area using DMA, and NetVM inserts a descriptor of the packet in the ring buffer that is a small shared memory region shared between the hypervisor and each individual VM that is used to transmit packet descriptors. Emulated (PCI) is used to get around the limitation that QEMU and KVM do not allow memory to be shared between the hypervisor and VMs. NetVM can compose complex NF chains from multiple pipelined VMs and achieve throughput up to 10 Gb/s, which is an improvement of more than 250% compared to existing the DPDK framework that uses SR-IOV.

Network operators need to instantiate and migrate virtual appliances dynamically and efficiently. The native solution of running VNFs in Linux or other commodity OS VMs has a slow instantiation time (several seconds) and a relatively large memory footprint. Martins et al. [98] recently proposed ClickOS, a tiny Xen-based VM to facilitate NFV. ClickOS can be instantiated within around 30 ms and requires about 5 MB of memory when running. ClickOS relies on hypervisor virtualization (in particular, para-virtualization) to achieve flexibility, isolation, and multi-tenancy. It adopts Click [66] as the main programming abstraction for middleboxes and creates a tailor-made middlebox VM to run Click configurations. It provides tools to build and manage ClickOS VMs including inserting, deleting, and inspecting middlebox states. A ClickOS VM consists of the Click router running on top of miniOS that implements the basic functionalities needed to run as a Xen VM. The miniOS has a single address space that does not require separation of the address space between the kernel and user and the cooperative scheduler-reducing context switch overhead. To achieve high-performance, it overhauls the Xen I/O subsystem by changing the back-end switch from Open vSwitch to VALE [101], optimizing virtual net devices (e.g., reducing the number of hyper calls, use of batching, and removing unnecessary software layers and data paths), and redesigning back-end and front-end drivers. A ClickOS VM is small (5 MB), boots quickly (around 30 ms), and adds a small delay (45 µs). Over one hundred ClickVMs can be concurrently run over a 10-Gb/s pipe on a commodity server.

Even though high-performance network I/O libraries, such as netmap [96] and DPDK [94], enables packet-processing rates of 10 Gb/s and higher by avoiding the kernel's networking stack and allowing direct access to packet data from a user-space application, they dedicate NIC ports to a single process; therefore, it is impossible to run multiple processes on the same server. Thus, ClickOS [98] and E2 [71], which use netmap and DPDK, respectively, are designed for static service-function chaining because the SF chaining is hard coded and cannot be modified dynamically by an NFV manager. Zhang et al. [102] proposed OpenNetVM, a packet-processing framework that allows dynamic SF chaining. It runs NFs as standard user-space processes inside a lightweight docket container while providing them high-performance network I/O via DPDK using shared memory accessible to each docket container within a common security domain. Unlike a VM whose disk image is gigabytes in size, the docker container, defined by a configuration list of packages and files to be installed, greatly simplifies the deployment and sharing of containers, allowing NFs to start in less than a second.

Hirschman et al. [103] investigated the applicability, in terms of performance, of general-purpose processors (e.g., Intel Xeon processor) for the EPC, which is a key component of LTE systems. They argued that general-purpose processors can execute EPC functions for representative market call models and that workloads can scale across bearer and control planes at a line rate without acceleration technologies.

Bronstein et al. [104] addressed abstraction of hardware accelerators (HWAs) to improve performance of VNFs in the NFV infrastructure (NFVI). An HWA combines a general-purpose CPU with a specialized HW such as application-specific integrated circuit (ASIC), FPGA, multicore network-processor unit (MC-NPU), and graphics processing unit (GPU). The VNF provides the expected performance improvements in case HWA is available; otherwise, it keeps running even with lower performance. Bronstein et al. proposed extensions to the virtual infrastructure manager (VIM) and VNF manager (VNFM) to conduct HWA life-cycle management allowing HWAs to be shared and dynamically allocated to VNFs.

Table 5 lists the summary of related work in the area of software router architecture based on recent high performance processor technologies.

4.3 Graphical Processing Units (GPUs) Based Approaches

Multi-core Central Processing Units (CPUs) and many-core Graphic Processing Units (GPUs) are used to accelerate packet processing [111]–[123]. The key software technique to exploit hardware performance include batching, pipelining, and parallelization.

Han et al. [113], [124] proposed "PacketShader", a high-speed software-based packet-processing framework. It uses GPUs as co-processors of CPUs to offload memory-intensive jobs such as IP table lookup, or computation-intensive ones, such as IPsec encryption, to scale packet processing with massive parallelism of GPUs. It implements optimized packet I/O to eliminate per-packet memory management overhead and process packets in batch.

Sun et al. [114] proposed "SNAP", a high-speed software-based packet-processing engine that exploits parallelism using GPUs. Snap is built on top of the Clik modular router. Click is a modular software router that provides an efficient pipeline-like abstraction for packet processing on PC hardware. A packet processor is constructed by connecting small software modules called *elements*, the building blocks of packet processing, into a graph. It enables individual elements to be implemented as GPU code to offload heavy packet-processing jobs on the GPU. It adds a set of new features including batched packet processing, memory structure optimized for offloading to the GPU, and asynchronous scheduling with in-order completion, while maintaining flexibility in Click, which allows users to build complex packet-processing pipelines from simple elements. Therefore, it amortizes overhead occurring every time a host CPU instructs GPUs to execute a program code called "kernel" in GPU programming terminology.

Kim et al. [115] proposed "network balancing act (NBA)", a software-based packet-processing framework that exploits the latest hardware technologies including multicore CPUs, many-core GPUs, and multi-queue 10-GbE NICs. It uses batch processing with small overheads by using memory management and branch prediction and adaptive CPU/GPU load balancing to automatically obtain the maximum throughput. It extends the Click modular-router

Table 5 Summary of research projects on "high performance platform" (Part 2).

Research project	Type	Distinctive Features
Egi et al. [95] (2010)	Performance of Click running on multi-core CPUs with multi-queue NIC.	Multi-queuing does not completely eliminate the need for the task synchronization in software routers. Replication of full forwarding paths on CPU cores offers the best allocation policy.
"netmap", Rizzo et al. [96] (2012)	Framework that enables operating systems to handle the millions of packets per seconds.	Implemented a number of techniques in FreeBSD and Linux: a lightweight meta-data representation, pre-allocated linear fixed size packet buffer, granting applications direct protected access to the packet buffers, supporting multi-queue.
Kim et al. [97] (2012)	Click on multi-core CPU.	Batching in terms of both packet I/O and computation is crucial in a NUMA architecture with multi-queue NIC.
"IX", Belay et al. [105] (2014)	High I/O performance with strong protection property.	Uses hardware virtualization to separate management and scheduling functions of the kernel (control plane) from network processing (data plane).
"Arrakis", Peter et al. [106] (2014)	Remove the kernel from the I/O data path.	It uses device hardware to deliver I/O directly to a customized user-level library.
Touitou et al. [107] (2014)	Fast Path Offloading.	NFV handles a fraction of flow requiring intensive higher layer protocol processing, which is small portion of the entire flow, by offloading the rest of flow to the fast path of data plane.
"mTCP", Jeong et al. [108] (2014)	User-level TCP stack for multi-core systems.	High-performance packet I/O libraries allow applications to directly access the packets in order to address performance issues caused by the dominating number of short TCP connections, each of which handling in the kernel could need 70% to 80% of CPU cycles. Context-switch overhead is amortized by batching packet-level and socket-level events.
Marinos et al. [109], [110] (2014)	Renouncing generality of network stacks in commodity operating system.	Userspace stacks built on top of netmap framework [96]: Standstrom, a specialized userspace stack for serving static web content, and Namestorn, a specialized userspace stack implementing a high performance DNS server.
"NetVM", Hwang et al. [98] (2014)	Extensions of DPDK libraries for virtualized environments.	Overheads of inter-core communications and context switching is prevented: (1) Zero-copy delivery of packet data to VMs through a small shared ring-buffer shared between the hypervisor, (2) Huge page region shared with a group of trusted VMs that allows chained network function applications to directly read or write packet data, and (3) Lockless memory sharing design, multiple queue and receive side scaling (RSS).
"ClickOS", Martins et al. [98] (2014)	Tiny Xen-based VM to facilitate NFV.	ClickOS VM consists of the Click router running on top of miniOS. The miniOS has a single address space that does not require separation of address space between kernel and user, and a cooperative scheduler reducing context switch overhead. Xen I/O subsystem is overhauled by changing from Open vSwitch to VALE switch [101], optimizing virtual net devices, and redesigning back-end and front-end drivers.
"SoftNIC", Han et al. [72] (2015)	Hybrid software-hardware architecture.	In order to incorporate diverse network functionality into NIC, it provides a hardware abstraction layer (HAL) to developer to implement features.

architecture to hide details of hardware and software architecture specifics and provides a declarative abstraction for GPU offloading to reduce implementation efforts.

Vasiliadis et al. [120] proposed "MIDeA", a software-based stateful network-traffic analysis tool that runs application software based on Snort [36] on off-the-shelf general purpose hardware that combines multiple CPUs, multiple GPUs, and multi-queue NICs. It takes advantage of the parallelism offered by these types of hardware to offload a computation-intensive code implementing the Aho-Corasick string search algorithm [125] on GPUs to achieve high performance in a scalable way. To mitigate the overhead caused by extra data transfer between a CPU and GPU over the PCIe bus, it uses a pipelining that allows CPU and GPU execution to overlap.

Jamshed et al. [118] proposed "Kargus", a software-based IDS that exploits massive parallelism by balancing the pattern-matching workloads with multi-core CPUs and many-core GPUs. It offloads a computation-intensive code implementing the Aho-Corasick string search algorithm [125] on GPUs. It also uses batch processing and parallel execution with load balancing to achieve high performance. By fetching multiple packets from the NIC simultaneously, it has each pattern-matching function handle a batch of packets simultaneously. It uses a load-balancing algorithm that selectively offloads Pattern-matching tasks to a GPU only if the CPU is under heavy load condition, which dynamically adjusts the offloading threshold.

Jang et al. [117] proposed "SSLShader", an SSL acceleration that exploits GPUs to offload SSL cryptographic operations. It selectively offloads cryptographic operations to a GPU depending on the load level.

Vasiliadis et al. [119] proposed "GASPP", a programmable network-traffic processing framework tailored to modern GPUs. It integrates into a purely GPU-powered implementation for many of the most common operations used by different types of network-traffic-processing applications, including network-flow tracking and TCP-stream reassembly. It also implements novel mechanisms for sharing memory context between network interfaces and the GPUs to avoid redundant data movement. It allows developers to focus on core application logic, alleviating the low-level technical challenges of data transfer to and from the GPU, packet batching, asynchronous execution, synchronization issues, connection-state management, and so on.

Even though the studies mentioned so far rely on massive parallelism of GPUs to obtain high-throughput packet processing, Kalia et al. [116] explored a hypothesis that much of the advantage of using GPUs for packet processing comes from the way GPUs are programmed and that less comes from the hardware advantage of GPUs over CPUs that is computational efficiency by having many processing cores and huge memory bandwidth. They argue that the key advantage of a GPU is that it can transparently hide the latency to retrieve data from main memory by using massive parallelism and fast hardware thread switching. They applied group pre-fetching and software pipelining to CPU packet handling code to boost its performance and presented a preliminary language and compiler-based framework that incorporates these code-optimization techniques.

Belay et al. [105] proposed “IX”, a data plane OS with high I/O performance while providing strong protection that existing kernels offer. With commodity OS design, it was not assumed that recent hardware technologies, such as multi-core, high-speed NICs, kernel schedulers, networking APIs, and network stacks, have been designed under the assumption of multiple applications sharing a single processing core and packet inter-arrival times being many times higher than the latency of interrupts and system calls. To address this mismatch between OS and recent hardware technology, IX uses hardware virtualization to separate management and scheduling functions of the kernel (control plane) from network processing (data plane). It separates the control function of the kernel, responsible for resource configuration, provisioning, scheduling, and monitoring, from the data plane. The control plane multiplexes and schedules resources among data planes, but in a course-grained manner in space and time. Entire cores are dedicated to data planes, memory is allocated at large page granularity, and NIC queues are assigned to data plane cores.

Peter et al. [106] proposed “Arrakis”, an OS designed to remove the kernel from the I/O data path without compromising process isolation. It uses device hardware to deliver I/O directly to a customized user-level library.

Touitou et al. [107] proposed fast path offloading to accelerate NFV performance. They claimed that NFV should be used to handle a fraction of flow requiring intensive higher layer protocol processing, which is a small portion of the entire flow in most cases, by offloading the rest of the flow to the fast path of the data plane. They implemented their proposal on HA-proxy, an open source layer-7 LB to demonstrate performance improvement.

Jeong et al. [108] proposed “mTCP”, a user-level TCP stack for multi-core systems to address the performance issues caused by the dominating number of short TCP connections, each of which handling in the kernel may require 70 to 80% of CPU cycles. They leveraged high-performance packet I/O libraries that allow applications to directly access the packets. They amortized the context-switch overhead by batching packet-level and socket-level events. They also took advantage of load-balancing of concurrent flows by using multi-core CPUs with RSS.

Marinos et al. [109], [110] proposed specialized network stacks, renouncing generality of network stacks in commodity OSes. The implemented specialized user-space stacks built on top of netmap framework [96], i.e., Standstrom, a specialized user-space stack for serving static web content, and Namestorn, a specialized user-space stack implementing a high-performance DNS server.

Han et al. [72] proposed “SoftNIC”, a hybrid software-hardware architecture to incorporate diverse network functionality into NICs. It provides a hardware abstraction layer to the developer to implement features in software while incurring minimal performance overhead.

There have been several studies using a GPU for acceleration. Vasiliadis et al. [122] proposed “Gnort”, an intrusion-detection system based on Snort that exploits a GPU to offload pattern-matching operations. They ran the Aho-Corasick algorithm [125], a multi-pattern matching algorithm, on the GPU exploiting the Single Instruction Multiple Data (SIMD) instructions to accelerate performance. Sun et al. [121] proposed “GPUstore”, a framework for integrating GPU computing into storage systems within the Linux kernel. They took advantage of the parallelism of a GPU to accelerate expensive computations in storage systems such as encryption, checksums, error correcting codes. Wang et al. [123] proposed a name lookup engine that exploits GPUs for massive parallel processing power.

Table 6 lists the summary of related work in the area of GPU-based high performance approaches.

5. State Management

The management functionality should be able to support sharing spare resources and elastic provisioning of network services effectively. The NFV infrastructure should be able to instantiate VNFs in the right locations at the right time. Elastic resource allocation to VNFs in response to variable workload and quick resource provisioning to instantiate new active VNFs in response to hardware failure is crucial to provide quality service level. This process can be fully automated, and the number of virtual instances of a particular VNF can be changed in response to workload and failure. Since VNFs can be dynamically created/migrated, it provides an additional dimension of complexity in terms of keeping track of where a given VNF is running. One of the main problems is the fact that many NFs are stateful. A typical structure of the NF application state can be divided into two classes [126], [127]. The first class contains a global state accessed independently of the processed traffic. The second class contains a partitioned state that consists of chunks of states directly related to network flows or sessions

Table 6 Summary of research projects on "high performance platform (GPU)".

Research project	Type	Distinctive Features
"PacketShader", Han et al. [113], [124] (2010)	High-speed software-based packet processing framework.	It employs graphical processing units (GPUs) as co-processor of CPU to offload memory-intensive jobs or computation-intensive ones. It implements optimized packet I/O to eliminate per-packet memory management overhead and to process packets in batch.
"SNAP", Sun et al. [114] (2013)	High-speed software-based packet processing engine.	Built on top of the Clik modular router, Click elements are implemented as GPU code. Batched packet processing, memory structure optimized for offloading to GPU, and asynchronous scheduling with in-order completion are added.
"NBA (Network Balancing Act)", Kim et al. [115] (2015)	Software-based packet processing framework for multi-core CPUs, many-core GPUs, and multi-queue 10 GbE NICs.	It employs a batch processing by use of memory management and branch prediction and an adaptive CPU/GPU load balancing. Click-based implementation with declarative abstraction.
"MIDeA", Vasiliadis et al. [120] (2011)	Software-based stateful network traffic analysis tool running Snort [36] on multiple CPUs, multiple GPUs, and multi-queue NICs.	Computation-intensive codes for Aho-Corasick string search algorithm [125] is offloaded on GPUs. Extra data transfer between CPU and GPU over the PCIe bus is mitigated, by pipelining CPU and GPU execution.
"Kargus", Jamshed et al. [118] (2012)	Software-based IDS balancing the pattern matching workloads with CPUs and GPUs.	It offloads a computation-intensive codes implementing the Aho-Corasick string search algorithm [125] on GPUs. It employs batch processing and parallel execution with load balancing. A load balancing algorithm selectively offloads pattern matching tasks to GPU only if the CPU is under heavy load condition.
"SSLShader", Jang et al. [117] (2011)	SSL acceleration offloading SSL cryptographic operations to GPU.	It selectively offloads cryptographic operations to GPU depending on the load level.
"Gnort", Vasiliadis et al. [122] (2008)	Intrusion detection system based on the Snort.	Aho-Corasick algorithm [125] for multi-pattern matching runs on the GPU exploiting the SIMD instructions.
"GPUstore", Sun et al. [121] (2012)	Framework for integrating GPU computing into storage systems.	Parallelism of GPU is exploited to accelerate expensive computations in storage system such as encryption, checksums, error correcting codes within the Linux kernel.

processed by the NF. These state chunks can be identified with the same information used to identify single flows or sessions. For IP connections, this is done using 5-tuples consisting of source IP, target IP, source port, target port, and protocol. In typical NFs, most parts of the application state are represented by the second class, which can easily be distributed across multiple VNF instances [126], [127].

Gember et al. [128] focused on a network-aware orchestration layer composed of a forwarding controller, which forwards traffic flows according to the service-chaining specification, and a middlebox controller, which monitors application performance and receives resource statistics from the middlebox.

Gember et al. [127] presented OpenNF, a unified control plane architecture to manage both the network forwarding state and internal NF state, which enables middleboxes to migrate and recover. OpenNF provides efficient, coordinated control of both the internal NF state and network-forwarding state to allow quick, safe, and fine-grained reallocation of flows across NF instances. Consider a scenario in which a VNF is overloaded and must be scaled out to satisfy SLAs on throughput. We can easily launch a new VNF instance and reroute some in-progress flows to the new instance. The VNF accuracy may be impacted due to some VNF-internal states not being copied or shared. When an internal VNF state is being moved, packets may arrive at the source instance after the move starts, or at the destination instance before the state transfer finishes. Unless care is taken, updates to the VNF state due to such packets may either be lost or occur out of order. Similarly, when a state is copied across NF instances, updates occurring simultaneously may cause the state to become inconsistent. To account for race conditions, the authors introduced two constructs: (1) an event abstraction to externally observe and prevent local state changes inside NFs, and (2) a two-phase scheme for updating network-forwarding state. They ensure that state updates are not lost or reordered during state moves and the shared state remains consistent.

Rajagopalan et al. [126] proposed a framework called *Split/Merge* to achieve elastic flow-processing capacity by replicating middleboxes and accommodating the flows into each replica middlebox. The Split/Merge framework provides a hypervisor-level abstraction for virtual middleboxes. It allows middlebox applications to be written and configured regardless of the number of replicas that may be instantiated. Thus, it achieves transparent and balanced elasticity by creating and destroying VM replicas while maintaining the load between them. A Split/Merge-aware VM is abstractly defined as a set of identical state machine replicas. Consistency is achieved by ensuring that each replicated state machine can access the state required to produce the appropriate output. Through a specialized shared library instead of using system-provided functions, the Split/Merge framework requires middleboxes to allocate and access all per-flow states required to process packets within the flow and cross-flow states required to process packets across flows. This allows the framework to transfer and replicate the middlebox state without serializing or updating middlebox internal structures.

Rajagopalan et al. [129] proposed Pico Replication (PR) to achieve a high-availability (HA) middlebox framework. It takes advantage of the Split/Merge framework to perform a flow-level replication for high availability rather than at the VM level. It fragments a set of flows on a replica into disjoint subsets called *replication groups*. Each replication group has its own output buffer, checkpoint frequency, and replication middlebox target. By executing a flow-level replication, it operates at much higher frequency than VM-based replication and controls the replication frequencies and targets of different flows independently. Instead of suspending and checkpointing an entire middlebox at the VM level, PR allows middlebox applications to enable fine-grained flow-level replications. It consists of three modules: *state-management module* (SMM), *packet-management module* (PMM), and *replication module* (PM). The SMM contains a set of flow states that consists of backups of other replica's states called *standby states* and provides an interface to middlebox applications to identify the flow-related state by leveraging the concept of the Split/Merge framework [126]. The PMM acts as an intermediary between a middlebox application and the network to ensure that per-flow state replication maintains consistency through input and output buffers. Using output buffering, the system ensures that the network output from a middlebox replica is not seen by the external world until a checkpoint of the flow states is committed at the backup node. A checkpoint begins by suspending any execution that may affect the state. Halting input of a particular flow is effectively a suspend operation and is sufficient for a checkpoint of the flow-state commence. Once a flow is suspended and its state is copied, the appropriate output buffer is released. The RM interacts with the SMM and PMM to implement a policy for replication. The RM instructs the PMM to halt a flow. Then, the RM obtains the flow state from the SMM. The RM copies the flow state to the SMM elsewhere in the network. Finally, after receiving confirmation that the flow state is backed up, the RM instructs the PMM to release the output buffer. The RM also informs the SDN controller of the flow backup targets so that it can quickly recover from failure.

It is a challenge to develop a fail-over mechanism that correctly restores states for middleboxes that are stateful. Active-active replication, where master and slave are executed on all inputs but only the master's output is released to users, will not work because of the *non-deterministic* nature of packet processing in middleboxes. Sherry et al. [130], [131] proposed a *fault-tolerant middlebox*, a new design for fault-tolerant middleboxes that achieves correctness, fast recovery with only a slight increase in latency. They took a *replay-based* approach that maintains a log of inputs to the system and recreates the lost state by replaying the inputs from the log in the event of a failure. To accommodate non-determinism, they use the approach of intercepting and/or recording the outcome of all potentially non-deterministic operations. No output can be released to the external world until all the information necessary to recreate the internal state consistent with that output has been committed to stable storage. It uses *ordered logging*, which is lightweight logging of the information needed after recovery, and the *parallel release* algorithm that ensures that recovery is always correct. They identified determinants information that must be recorded to correctly replay operations that are vulnerable to non-determinism. A packet access log is recorded to represent non-determinism caused from races between threads accessing shared variables. It ensures that, before the middlebox transmits a packet, it has successfully logged to stable storage all the information needed to recreate the internal state consistent with an execution that would have generated the packet. They implemented a prototype using Click and demonstrated 30 µs of latency to median packet latencies and rapid recovery in 40-275 ms for practical system configurations.

Kablan et al. [132] proposed *stateless NFs*. They argued that an NF should be stateless similar to agile stateless cloud-scale applications that store their states into a dedicated cache and back-end stores. They argued that the state of an NF should be separated and remain in a back-end store or cache without loss of performance. They demonstrated that a stateless NF can overcome the performance limitation by leveraging low-latency data-store-access methods such as RAMCloud [133], [134] and FaRM [135]. They implemented a prototype of stateless NAT using Click and RAMCloud cluster connected via InfiniBand. where the NAT translation table is maintained in a RAMCloud cluster. They implemented three modes of operation: blocking, asynchronous (non-blocking), and cache. They argued that a stateless NF allows for easier scalability and higher availability by decomposing applications into micro services that rely on back-end data stores and middle-tier caching layers to provide the needed state on-demand.

To achieve elastic resource allocation to VNFs in response to variable workload and quick resource provisioning to instantiate new active VNFs in response to hardware failure, stateful middlebox applications need to be modified to handle the middlebox internal state. Khalid et al. [136] proposed *StateAlyzr*, a system that reduces the effort involved in making modifications to middlebox software to perform custom state allocation, track updates to state, and (de)serialize state objects by automating identification of state objects that require explicit handling. Numerous data structures and procedures, large call graphs, heavy use of multi-level pointers, and indirect calls to packet processing routines are middlebox attributes that make the code modification tasks difficult. Khalid et al. used program analysis techniques, such as slicing and pointer Analysis, to automatically identify (1) variables corresponding to state objects that pertain to individual or groups of flows, (2) variables

Table 7 Summary of research projects on "state management".

Research project	Type	Distinctive Features
"OpenNF", Gember et al. [128] (2013)	Network-aware orchestration layer	OpenNF composed of a forwarding controller that forwards traffic according to the service-chaining specification, and an middlebox controller that monitors application performance and resource statistics.
"Split/Merge", Rajagopalan et al. [126] (2013)	Hypervisor-level abstraction for virtual middleboxes.	A set of identical state machine replicas is created. Each replicated state machine access the state required to produce the appropriate output through a shared library.
"Pico Replication (PR)", Rajagopalan et al. [129] (2013)	High available (HA) middlebox framework.	Based on Split/Merge framework it performs a flow-level replication and operates at higher frequency than VM-based replication. Using output buffering, the system ensures that the network output from a middlebox replica is not seen by the external world until a checkpoint of the flow states is committed a the backup node.
"OpenNF", Gember et al. [127] (2014)	Unified control plane architecture.	Prevention of local state changes inside NFs and a two-phase scheme for updating state are used to ensure that state updates are not lost or reordered during state moves and shared state remains consistent.
"Fault-Tolerant MiddleBox (FTMB)", Sherry et al. [130], [131] (2015)	Fault-tolerant middlebox.	Replay-based approach maintains a log of inputs to the system and recreate lost state by replaying the inputs form the log. No output can be released to the external world until all the information necessary to recreate consistent internal state. A prototype using Click is implemented.
"Stateless Network Functions," Kablan et al. [132] (2015)	Stateless cloud-scale applications platform.	States of network function are separated and stored in a back-end store A prototype of stateless NAT using Click and RAMCloud cluster connected via InfiniBand is implemented.
"StateAlyzr", Khalid et al. [136] (2016)	Reduction of efforts in modifications to middlebox software.	Program analysis techniques, e.g., slicing and pointer analysis automatically identify (1) variables for state objects of flows, (2) variables for state objects updated by an incoming packet, and (3) middlebox I/O actions.
Khalid et al. [140] (2016)	Standard southbound API for VNF management.	Identifies four core operations: State Management, Service Chaining, Bottleneck Detection, Configuration. Presents a framework-agnostic API for managing the network function's state, chaining the network functions, and quantifying the network function's performance.

corresponding to state objects that can be updated by an incoming packet at runtime, (3) the flow space corresponding to a state object, (3) middlebox I/O actions that are impacted by each state object, and (4) objects updated at runtime by an incoming packet. They leveraged a typical middlebox code structure that consists of three basic parts, i.e., initialization, packet receive loop, and packet processing, to design algorithms that use static program-analysis techniques in a way that significantly improves precision without compromising soundness.

Several frameworks have been proposed for NFV management [61], [126], [128], [132], [137]–[139]. Middlebox application requires modifications to be controlled under these frameworks. Unfortunately, the lack of a standard API for NFV management frameworks and NFVs prevents adoption of NFVs. Khalid et al. [140] recently proposed a standard southbound API for VNF management. They identified four core operations that a generic framework-agnostic API should provide: state management, service chaining, bottleneck detection, and configuration. They present a high-level framework-agnostic API for managing an NF's state, chaining NFs (setting up a sequence of NFs and providing contextual information to another NF in the chain), and quantifying an NF's performance. They argue that it helps facilitate adoption of NFVs and enable innovation in the design of both management frameworks and NFs.

As Split/Merge proposed [126], providing APIs for NFs to create/update state is one approach, but it restricts how internal NF state is structured and may not accommodate the state allocation/access needs of some packet processing logic. Instead, they design a novel southbound API for NFs that allows a controller to request the export or import of NF state without changing how NFs internally manage state. Gember et al. [127] implemented their northbound API using Floodlight, and they constructed several control applications that use this API. They also augmented four NFs – Bro, Squid, iptables, and PRADS – to support their southbound API.

Table 7 lists the summary of related work in the area of state management.

6. Service Chaining

Enforcement of network-wide policies involves network-management tasks ranging across stateful policy routing, access control and rate limiting traffic, and diagnostics for performance debugging, forensics for detecting malicious activity, etc. Stateful policy routing for packet traversing a given sequence of middleboxes is challenging for its enforcement and verification. The root cause of this problem is that as packets traverse the network, their headers and contents may be dynamically modified by middleboxes. For example, a proxy terminates sessions while NAT and LB rewrite headers. These modifications make it difficult to ensure that the desired set of policies are consistently applied throughout the network. This is particularly challenging because middleboxes often rely on proprietary internal logic for affecting such dynamic traffic transformations.

The enforcement of traditional service-chaining policy is highly complex. This complexity originates from the need to carefully plan the network topology and manually set up rules to steer traffic flows through the desired sequence of middleboxes. Traditional solutions are inflexible because service-chaining policy is executed in a manual mode. Thus, the service chain should be more flexible due to the growth of traffic flows. Network function virtualization offers a promising alternative to tackle the challenges based on commodity hardware and virtualization technologies. Through decoupling the control and data planes, SDN maintains a global view of the network and enables the network control to become flexibly programmable. Thus, service-chaining solutions combined with SDN have emerged to provide flexible adjustments. Service chaining should allow network operators to specify a logical middlebox-routing policy and automatically translate this into forwarding rules that take into account the physical topology, switch capacities, and middlebox-resource constraints.

Joseph et al. [27], [141] proposed *PLayer*, a new policy-aware layer-2 architecture for data centers. To introduce PLayer in existing data centers with less effort, they argue that it is important to minimize changes to existing layer-2 switches and keep middleboxes unmodified. The PLayer architecture includes policy-aware switches called *pswitches* and unmodified middleboxes. An unmodified middlebox is placed off-path and connected to one of the Pswtiches that are inter-connected to build data-center network infrastructure. Its design principles include: (i) separating policy from reachability and (ii) taking middleboxes off the physical network path. Off-path middlebox placement simplifies topology modifications and enables efficient use of existing middleboxes. Redirecting traffic through off-path middleboxes is based on the principle of *indirection* [142]–[146].

A Pswitch plays a key role in the PLayer architecture; A Pswitch maintains the middlebox traversal requirements of all applications in the form of policy specifications. It classifies incoming traffic and explicitly redirects them to appropriate middleboxes. Based on policies specified by administrators, Pswitches explicitly forward different types of traffic through different sequences of middleboxes; thus, guaranteeing middlebox traversal in the policy-mandated sequence. They implemented a prototype Pswitch using Click [66] and demonstrated the proof-of-concept.

Fayazbakhsh et al. [139], [147] proposed FlowTags for enforcing network-wide policies in the presence of dynamic middlebox actions. They identified flow tracking as a key capability for policy enforcement in the presence of dynamic traffic transformations caused by middleboxes. The key idea in FlowTags is to tag packets with the necessary middlebox context to tackle the dynamic traffic transformations caused by middleboxes in the service chain. They attempted to integrate *extended middleboxes* into SDN-based networks with minimal impact. With FlowTags, minimal extensions to middleboxes are expected to add the relevant contextual information. They extended the middlebox at the ingress of a service chain in a manner that a middlebox adds a "tag" in the packet header, enabling downstream middlboxes as well as SDN switches to easily track the service-chain flow. The SDN controller configures the actions on switches and middleboxes to use these Tags (added by other middleboxes) as part of their data plane operations to correctly enforce network-wide policies. They presented a controller-middlebox interface, policy abstractions and rule-generation mechanisms, and FlowTag-enabled middlebox design.

Qazi et al [61], [148] proposed SIMPLE-fying Middlebox Policy Enforcement Using SDN. They presented the design and implementation of SIMPLE, an SDN-based policy enforcement layer for middlebox-specific traffic steering [27], [141]. One of the challenges is packet modifications: Middleboxes modify packet headers (e.g, NATs) and even change session-level behaviors (e.g.,WAN optimizers and proxies use persistent connections). Operators have to take into account these effects via careful placement or manually reason the impact of these modifications on routing configurations. Due to the proprietary nature of middleboxes, however, an SDN controller may have limited visibility to set up forwarding rules that take into account such transformations. Due to the diverse and proprietary behaviors of a vast array of middleboxes, it is impractical to model their behavior. Qazi et al. regarded middleboxes as a black-boxes and attempted to learn their input-output behavior from measurement. They proposed a method for automatically inferring mappings between the incoming and outgoing traffic of middleboxes that may modify packet headers and session-level behaviors. They also developed a flow-correlation algorithm that computes payload similarity scores to correlate the incoming and outgoing traffic. They implemented a proof-of-concept of a SIMPLE controller that extends POX.

Ding et al. [149] proposed an open platform for a service chain as a service by using the tangible capabilities of SDN together with NFV. In this platform, the service chain can be considered as a service; SDN is used to improve flexibility, NFV is applied to enhance adaptability, and encapsulating the service chain can help guarantee scalability. They demonstrated a new design and architecture by using the benefits of SDN and NFV for enforcing service-chaining policy without modifying current SDN standards or mandating any implementation constraints on middleboxes. This design encapsulate service-chain identifier based on source MAC to guarantee scalability and achieve auto-provisioning based on NFV to provide adaptability. The design uses VMs to achieve adaptability by automatically creating software-defined middleboxes to process traffic flows and automatically setting up flow rules to steer traffic flows.

Lin et al. [150] proposed an extended SDN architecture for NFV with a case study on intrusion prevention. In conventional SDN, a controller classifies the traffic redirected from a switch to determine the path to NFV modules. The redirection generates a large volume of control plane traffic. They proposed an architecture for intrusion detection where two-layer traffic classification is performed in the

Table 8 Summary of research projects on "service chaining".

Research project	Type	Distinctive Features
"PLayer", Joseph et al. [27], [141] (2008)	A policy-aware layer-2 architecture based on indirection paradigm.	Policy-aware switches called *pswitches* classifies incoming traffic and explicitly redirect them to appropriate middleboxes. Off-path middlebox placement simplifies topology modifications and enables efficient usage of existing middleboxes. Implemented pswitch using Click.
"FlowTags", Fayazbakhsh et al. [147] (2013)	Tag packets with the middlebox context.	Contextual information is associated with a traffic flow as it traverses the network, even if packet headers and contents are modified. The tags are generated by the first middlebox and used by other middleboxes. SDN-capable switches use the tags as part of their forwarding actions.
"SIMPLE", Qazi [61], [148] (2013)	SDN-based policy enforcement layer for middlebox-specific traffic steering.	Automatically infers mappings between the incoming and outgoing traffic of middleboxes that may modify packet headers and session-level behaviors. A flow correlation algorithm that computes payload similarity scores to correlate the incoming and outgoing traffic is proposed.
Ding et al. [149] (2015)	an open platform for service chain as a service using the capabilities of SDN together with NFV.	Enforcing service-chaining policy without modifying current SDN standards or mandating any implementation constraints on middleboxes. It encapsulates Service Chain Identifier (SC-ID) based on Source MAC to guarantee scalability, and realizes auto-provisioning based on NFV to offer adaptability. It uses VM to achieve adaptability by automatically creating software-defined middleboxes to process traffic flows, and automatically setting up flow rules to steer traffic flows.
Lin et al. [150] (2015)	SDN architecture for network function virtualization for intrusion prevention.	It moves traffic classification from the controller to the data plane, and extends OpenFlow messages with a matched field of network events. The classification (CLA) module is located on the switch. Payload analysis is shifted to the DPI function as an NFV module.

data plane by extending OpenFlow compared with deploying NFV modules in an OpenFlow-based SDN. The classification module is located on the switch. Payload analysis is shifted to the DPI function as an NFV module because the analysis is too expensive to be conducted on the switch. This architecture moves traffic classification from the controller to the data plane and extends OpenFlow messages with a matched field of network events. Two cases of intrusion prevention were investigated: (1) detecting SYN flooding from the TCP/IP headers and (2) detecting SQL injection and cross-site scripting attacks for web applications. Their extended SDN architecture reduces the traffic overhead to the controller for providing NFV.

Table 8 lists the summary of related work in the area of service chain.

7. Resource Management

The problem of mapping each virtual network to specific nodes and links in the substrate network is called *virtual network embedding*. Many techniques have been developed for solving the virtual-network embedding problem [151]–[153]. The problem of NF placement and chaining has similarity with that of virtual-network embedding. However, the solution to virtual-network embedding is not appropriate for the NF placement and chaining problem because we need to consider two-level mappings, as described in what follows. Network-function placement and chaining consists of interconnecting a set of NFs through the network to ensure network flows are given the correct treatment. These flows must go through end-to-end paths traversing a specific set of functions. This problem can be decomposed into three phases: (i) placement, (ii) assignment, and (iii) chaining. The placement phase consists of determining how many NF instances are necessary to meet the demand and where to place them in the infrastructure. Virtual NFs are expected to be placed on groups of commodity servers located as computing infrastructure at points of presence (PoPs). In the assignment phase, VNF instances in the PoPs are selected to serve each flow. Instances are assigned to the flow in a manner that prevents processing times from causing intolerable latencies. Finally, the requested functions are chained, which consists of creating paths that interconnect the NFs placed and assigned in the previous phases taking into account path latencies and processing delays added by different VNFs.

Luizelli et al. [154] formalized the NF placement and chaining problem as an integer linear programming (ILP) optimization model and proposed a heuristic procedure to solve the optimization model. They considered three basic types of SFC components: (i) line, (ii) bifurcated path with different endpoints, and (iii) bifurcated path with a single endpoint. They formulated the problem as an ILP optimization problem. Their objective function is aimed at minimizing the number of VNF instances mapped on the infrastructure. Their constraints are placed on a resource of CPU and bandwidth in two-level mappings; SF chaining requests are mapped onto VNF instances, which are mapped onto the physical infrastructure. Since the optimal approach searches an extensive number of symmetrical feasible solutions leading to prohibitive computational complexity, Luizelli et al. developed a heuristic approach based on a binary search in terms of the number of NF instances. Both optimal and heuristic approaches are evaluated considering different use cases and metrics, such as the number of instantiated VNFs, physical- and virtual-resource consumption, and end-to-end latencies.

Cohen et la. [155] proposed "Near optimal placement of virtual network functions." They addressed the issues of placement of VNFs in a physical network and tackled the problem of allocating server resources to NFs at various locations such that all flows requiring a service are satisfied while minimizing operational cost. They used the facility-location problem and generalized assignment problem. Their goal was to locate NFs in a manner that minimizes the overall network costs, i.e., (1) connection cost that accounts for the distance between a client and the facility that provides its service and (2) setup cost that is required to open a facility at a node.

Basta et al. addressed virtualization of S-GW and P-GW in LTE networks [156]. They studied two alternatives: virtualization and decomposition. Virtualization refers to the situation in which S-GWs and P-GWs are separated from transport nodes and located at data centers (note that both data and control planes of S-GWs and P-GWs are located at data centers), while decomposition refers to the situation in which the control plane of S-GWs and P-GWs is separated and located in data centers and the data plane of S-GWs and P-GWs remains in advanced transport network nodes capable of GTP tunneling, charging, etc. They formulated an optimization problem to minimize the transport-network-load overhead against several parameters such as data plane delay, number of potential data centers, and SDN-control overhead.

Dwaraki et al. [157] addressed the problem of routing service-chain requests and placing processing functions in an SDN/NFV-based network. The location of the NF can be anywhere within the SDN/NFV-based network and placed on a pre-computed path from the source and destination. The routing challenge includes: (1) determining a flow path that traverses nodes providing requested NFs in a requested order, and (2) considering network load and other dynamic characteristics. They transformed the network graph into a *layered graph* by adding k layers to the graph, where k represents the number of network functions in the service chain to be implemented. Every $(i-1, i)$ layer pair is connected vertically only by edges between node v^{i-1} and v^{i} if that node provides the i-th network function required in the service chain. These vertical edges are weighted by a cost that is defined by the processing cost for using a NF on node v. Shortest path algorithms, such as Dijkstra's, are used to compute the path. They use delay as a cost metric, consisting of the sum of communication delays on the horizontal links connecting nodes and processing delay on VNF nodes. Their proposed adaptive service routing algorithm is simple and routes traffic adaptively based on instantaneous network latency.

Bari et al. [158] addressed the optimization problem on the number and placement of VNFs that they call the VNF orchestration problem (VNF-OP)". Their goal is to minimize the weighted sum of VNF deployment cost, energy cost, and VNF-traffic-forwarding cost while maintaining service level objectives. They formulated the VNF-OP as a multi-commodity, multi-plant, capacitated facility location problem [159], which is an ILP. Because it is an NP-hard problem, they developed a heuristic algorithm based on the Viterbi algorithm [160]. For a given traffic request, they model it as a multi-stage directed graph, where the i-th stage represents the $(i-1)$-th VNF in the service chain. They compute a set of VNF placements in the service chain by running The Viterbi algorithm on the multi-stage graph.

Clayman et al. [161] addressed the architecture for managing virtualized infrastructures. They proposed a system that consists of a service orchestrator, monitoring manager, and placement engine. The orchestrator manages the creation and removal of the virtual nodes, as well as configuring, monitoring, running and stopping applications in these nodes. Automatic virtual node placement and resource allocation is executed in support with the monitoring system that collects and reports on system behavior. Moens et al. [162] investigated a formal model for resource allocation of NFVs, a problem they refer to as the VNF placement (NFV-P) problem. They assume a scenario in which a base load is carried by physical hardware and burst spillover is carried by using virtual service instances. The problem is formulated as ILP. Mehraghdam et al. [163] proposed a context-free language to describe the SF chaining requests. Given the SF requests, they formulated the SFC mapping problem as a mixed-integer quadratically constrained program. Mohammadkhan et al. [164] formulated service-Function-placement and flow-steering problems jointly in a single mixed integer linear problem and developed several heuristic algorithms. Sahhaf et al. [165] proposed an algorithm to map Network SF chains to the physical-network infrastructure. They decomposed an abstract NF graph into a set of specific component NFs interconnected into a graph. Their algorithm consists of two steps: decomposition selection and mapping. Decomposition is in favor of the same type of component network functions. Mapping the NFs in descending order in terms of requirements. Ghaznavi et al. [166] addressed the elastic VNF-P problem and proposed a method called simple lazy facility location that optimizes the placement of VNF instances in response to on-demand workload. Gil et al. [19] presented a comprehensive survey on the NFV resource-allocation (NFV-RA) problem. They argued that the NFV-RA problem consists of three stages: VNF chain composition, VNF forwarding graph embedding, and VNF scheduling. They classified the related studies as to which stage they are addressed.

8. Trouble Shooting

Switches, routers, and middleboxes are vertically integrated monolithic systems; individual vendors exercise proprietary design for system architecture of their products on their own. Softwarization ushers in a new paradigm for disaggregation of traditionally vertically integrated network devices and disaggregates vertically integrated systems into components. Even though softwarization has the benefits of cost reduction, elastic capacity, functional extensibility, extension of system life, etc., it incurs added complexity; individual components interact with each other, which could cause instability and fragility. The traditional network-management paradigm is

Table 9 Summary of research projects on "trouble shooting".

Research project	Type	Distinctive Features
"vNMF", Miyazawa et al. [169] (2015)	Distributed management function address the increasing complexity in management of NFVs.	It collects statistics such as CPU usage, memory usage, disk read-write I/O, in-out octets/packets, error packets from NFVI and analyzes them using Self-Organizing Map (SOM), a clustering algorithm based on unsupervised artificial neural network. They demonstrated that the proposed method is applicable to two usecases of memory-leak and network congestion in NFV system.
Sanchez et al. [170] (2016)	A multi-layer self-diagnosis framework for networking services in SDN and NFV environments.	They used Bayesian networks approach for root cause calculation. Root-cause analysis module reasons over the service dependency graph, which is modeled as Bayesian network, a probabilistic dependency graph. When the proposed system detects a service failure, it computes service dependency graph related to the service and run root-cause analysis algorithm on the Bayesian network to find the root-cause of the service failure. The fine-grained templates need to be predefined.
Kushnir et al. [171] (2016)	data-driven root-cause analysis method based on clustering alarm correlation and inference of causality between alarms.	The proposed method collects the time series data for all alarm, computes the pairwise alarm type correlation, creates the correlation graph, and run the clustering. It finally creates directed causality graph within each cluster. They do not assume apriori information on network topology.

based on a *mechanism-driven* approach; we need to understand the precise mechanism of the behaviors of individual components of a system and construct a system behavior model. We believe that it is not adequate to rely on such a traditional approach for highly complex systems based on softwarization such as NFV and SDN. Instead, we argue that a *data-driven* approach is suitable, with which we consider the entire system as a *black box* and measure the input and output of the system to understand the system's behavior. We believe that machine learning is a key technology for this data-driven approach [167]. There a few studies in this domain for NFV [168]–[171]. Gardikis et al. [168] addressed NFV monitoring issues and proposed an open-source measurement software framework called T-NOVA, which collects metrics from hardware and hypervisor in the NFVI to report them at the virtualized infrastructure-management level.

Miyazawa et al. [169] proposed a distributed management function called the virtualized network-management function to address the increasing complexity in the management of NFV. It collects statistics such as CPU usage, memory usage, disk read-write I/O, in-out octets/packets, and error packets from NFVI, and analyzes them using self-organizing map, a clustering algorithm based on unsupervised artificial neural networks. They demonstrated that the proposed function is applicable to two use cases of memory-leak and network congestion in an NFV system.

Sanchez et al. [170] proposed a multi-layer self-diagnosis framework for networking services in SDN and NFV environments. Their framework consists of a (1) multi-layered template, (2) self-modeling module, and (3) root-cause analysis (RCA) module. They modeled dependencies among components by defining fine granularity templates. They used the Bayesian network approach for root-cause calculation. The multi-layered template allows operators to fine-grain model physical, logical, virtual, and service layers. The self-modeling module builds a network-dependency graph by assembling a set of fine-grained templates, which describes their internal components and their dependencies among them, and computes a service-dependency graph from the network-dependency graph and a virtual-resource dependency graph. The RCA module reasons over the service-dependency graph, which is modeled as a Bayesian network, a probabilistic dependency graph. When their framework detects a service failure, it computes a service-dependency graph related to the service and runs the RCA algorithm on the Bayesian network to find the root cause of service failure. The fine-grained templates need to be predefined.

Kushnir et al. [171] proposed a data-driven RCA method based on clustering alarm correlation and inference of causality between alarms. The proposed method collects the time series data for all alarms, computes the pairwise alarm-type correlation, creates the correlation graph, and runs the clustering. It finally creates a directed-causality graph within each cluster. They do not assume a priori information on network topology. They evaluated their proposed method for data sets from a network application that was running on a cloud network, where 160 VMs were running to demonstrate that their method extracts a causality graph.

Table 9 lists the summary of related work in the area of trouble shooting.

9. Conclusion

We surveyed research activities in the area of re-architecting middleboxes, high-performance platforms, state management, service chaining, and resource management.

Recent research activities on re-architecting middleboxes have revealed a few new NFV platform data plane architectures using hardware platforms based on general purpose processors, and a control plane architecture. State management of NFs has been the focus of recent intensive research activities to achieve elastic and fault-tolerant VNF deployment without disruption. These research efforts coupled with high-performance platforms will be combined with efforts on re-architecting middleboxes to materialize future VNF platforms.

Recent research activities on service chaining have been contributing to realization of simple management to implement diversified service policies by chaining NFs located in clouds. They will be coupled with research efforts on resource management to innovate service management while maintaining acceptable CAPEX and OPEX. Softwarization has ushered in a new paradigm for disaggregation of traditionally vertically integrated network devices. As research activities on these areas are maturing, innovative services creation is expected to be accelerated in the future.

References

[1] B. Carpenter and S. Brim, "Middleboxes: Taxonomy and issues," RFC 3234, RFC Editor, Feb. 2002. http://www.rfc-editor.org/rfc/rfc3234.txt

[2] G. Gibb, H. Zeng, and N. McKeown, "Outsourcing network functionality," Proc. First Workshop on Hot Topics in Software Defined Networks, HotSDN '12, pp.73–78, ACM, New York, NY, USA, 2012.

[3] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn," Queue, vol.11, no.12, pp.20:20–20:40, Dec. 2013.

[4] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: An intellectual history of programmable networks," SIGCOMM Comput. Commun. Rev., vol.44, no.2, pp.87–98, April 2014.

[5] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and implementation of a routing control platform," Proc. 2Nd Conference on Symposium on Networked Systems Design Implementation - Volume 2, NSDI'05, pp.15–28, USENIX Association, Berkeley, CA, USA, 2005.

[6] A. Farrel, J.P. Vasseur, and J. Ash, "A path computation element (PCE)-based architecture," RFC 4655, RFC Editor, Aug. 2006. http://www.rfc-editor.org/rfc/rfc4655.txt

[7] A. Greenberg, G. Hjalmtysson, D.A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4D approach to network control and management," SIGCOMM Comput. Commun. Rev., vol.35, no.5, pp.41–54, Oct. 2005.

[8] M. Casado, T. Garfinkel, A. Akella, M.J. Freedman, D. Boneh, N. McKeown, and S. Shenker, "SANE: A protection architecture for enterprise networks," Proc. 15th Conference on USENIX Security Symposium - Volume 15, USENIX-SS'06, USENIX Association, Berkeley, CA, USA, 2006.

[9] M. Casado, M.J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," Proc. 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '07, pp.1–12, ACM, New York, NY, USA, 2007.

[10] M. Casado, M.J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," SIGCOMM Comput. Commun. Rev., vol.37, no.4, pp.1–12, Aug. 2007.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," SIGCOMM Comput. Commun. Rev., vol.38, no.2, pp.69–74, March 2008.

[12] K. Suzuki, K. Sonoda, N. Tomizawa, Y. Yakuwa, T. Uchida, Y. Higuchi, T. Tonouchi, and H. Shimonishi, "A Survey on openflow technologies," IEICE Trans. Commun., vol.E97-B, no.2, pp.375–386, Feb. 2014.

[13] I.F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in sdn-openflow networks," Comput. Netw., vol.71, pp.1–30, Oct. 2014.

[14] M. Casado, N. Foster, and A. Guha, "Abstractions for software-defined networks," Commun. ACM, vol.57, no.10, pp.86–95, Sept. 2014.

[15] D. Kreutz, F.M.V. Ramos, P.E. Veríssimo, C.E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," Proc. IEEE, vol.103, no.1, pp.14–76, Jan. 2015.

[16] R. Mijumbi, J. Serrat, J.L. Gorricho, N. Bouten, F.D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," IEEE Commun. Surveys Tuts., vol.18, no.1, pp.236–262, Firstquarter 2016.

[17] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," IEEE Commun. Mag., vol.53, no.2, pp.90–97, Feb. 2015.

[18] Y. Li and M. Chen, "Software-defined network function virtualization: A survey," IEEE Access, vol.3, pp.2542–2553, 2015.

[19] J.G. Herrera and J.F. Botero, "Resource allocation in NFV: A comprehensive survey," IEEE Trans. Netw. Serv. Manage., vol.13, no.3, pp.518–532, Sept. 2016.

[20] J.H. Saltzer, D.P. Reed, and D.D. Clark, "End-to-end arguments in system design," ACM Trans. Comput. Syst., vol.2, no.4, pp.277–288, Nov. 1984.

[21] J. Sherry and S. Ratnasamy, "A survey of enterprise middlebox deployments," Technical Report, UCB/EECS-2012-24, EECS Department, University of California, Berkeley, Feb. 2012.

[22] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," Proc. ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM '12, pp.13–24, ACM, New York, NY, USA, 2012.

[23] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," SIGCOMM Comput. Commun. Rev., vol.42, no.4, pp.13–24, Aug. 2012.

[24] D. Joseph and I. Stoica, "Modeling middleboxes," IEEE Netw., vol.22, no.5, pp.20–25, Sept. 2008.

[25] A. Bremler-Barr, Y. Harchol, and D. Hay, "Openbox: Enabling innovation in middlebox applications," Proc. 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization, HotMiddlebox '15, pp.67–72, ACM, New York, NY, USA, 2015.

[26] T. Benson, A. Akella, and A. Shaikh, "Demystifying configuration challenges and trade-offs in network-based isp services," Proc. ACM SIGCOMM 2011 Conference, SIGCOMM '11, pp.302–313, ACM, New York, NY, USA, 2011.

[27] D.A. Joseph, A. Tavakoli, and I. Stoica, "A policy-aware switching layer for data centers," Proc. ACM SIGCOMM 2008 Conference on Data Communication, SIGCOMM '08, pp.51–62, ACM, New York, NY, USA, 2008.

[28] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, "STUN - simple traversal of user datagram protocol (UDP) through network address translators (NATs)," RFC 3489, RFC Editor, March 2003. http://www.rfc-editor.org/rfc/rfc3489.txt

[29] "Squid," http://squid-cache.org/

[30] C. Xu, S. Chen, J. Su, S.M. Yiu, and L.C.K. Hui, "A survey on regular expression matching for deep packet inspection: Applications, algorithms, and hardware platforms," IEEE Commun. Surveys Tuts., vol.18, no.4, pp.2991–3029, Fourthquarter 2016.

[31] "Procera, application awareness," https://www.proceranetworks.com/customers/vendors/

[32] "Allot communications, deep packet inspection," http://www.allot.com/technology/dart-dpi/

[33] V. Paxson, "Bro: A system for detecting network intruders in real-time," Proc. 7th Conference on USENIX Security Symposium - Volume 7, SSYM'98, p.3, USENIX Association, Berkeley, CA, USA, 1998.

[34] V. Paxson, "Bro: A system for detecting network intruders in real-time," Comput. Netw., vol.31, no.23-24, pp.2435–2463, Dec. 1999.

[35] “Snort,” http://www.snort.org/
[36] M. Roesch, “Snort - lightweight intrusion detection for networks,” Proc. 13th USENIX Conference on System Administration, LISA ’99, pp.229–238, USENIX Association, Berkeley, CA, USA, 1999.
[37] “Palo alto networks,” http://www.paloaltonetworks.com/
[38] “Openvpn,” http://www.openvpn.com/
[39] “Passive real-time asset detectoin system,” http://gamelinux.github.io/prads/
[40] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford, “Virtual routers on the move: Live router migration as a network-management primitive,” Proc. ACM SIGCOMM 2008 Conference on Data Communication, SIGCOMM ’08, pp.231–242, ACM, New York, NY, USA, 2008.
[41] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford, “Virtual routers on the move: Live router migration as a network-management primitive,” SIGCOMM Comput. Commun. Rev., vol.38, no.4, pp.231–242, Aug. 2008.
[42] “Brocade vrouter,” http://www.brocade.com/en/products-services/software-networking/network-functions-virtualization/vrouter.html
[43] N.T. Spring and D. Wetherall, “A protocol-independent technique for eliminating redundant network traffic,” Proc. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM ’00, pp.87–95, ACM, New York, NY, USA, 2000.
[44] N.T. Spring and D. Wetherall, “A protocol-independent technique for eliminating redundant network traffic,” SIGCOMM Comput. Commun. Rev., vol.30, no.4, pp.87–95, Aug. 2000.
[45] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, “Packet caches on routers: The implications of universal redundant traffic elimination,” Proc. ACM SIGCOMM 2008 Conference on Data Communication, SIGCOMM ’08, pp.219–230, ACM, New York, NY, USA, 2008.
[46] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, “Packet caches on routers: The implications of universal redundant traffic elimination,” SIGCOMM Comput. Commun. Rev., vol.38, no.4, pp.219–230, Aug. 2008.
[47] A. Anand, V. Sekar, and A. Akella, “SmartRE: An architecture for coordinated network-wide redundancy elimination,” Proc. ACM SIGCOMM 2009 Conference on Data Communication, SIGCOMM ’09, pp.87–98, ACM, New York, NY, USA, 2009.
[48] A. Anand, V. Sekar, and A. Akella, “SmartRE: An architecture for coordinated network-wide redundancy elimination,” SIGCOMM Comput. Commun. Rev., vol.39, no.4, pp.87–98, Aug. 2009.
[49] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, “Redundancy in network traffic: Findings and implications,” Proc. Eleventh International Joint Conference on Measurement and Modeling of Computer Systems, SIGMETRICS ’09, pp.37–48, ACM, New York, NY, USA, 2009.
[50] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, “Redundancy in network traffic: Findings and implications,” SIGMETRICS Perform. Eval. Rev., vol.37, no.1, pp.37–48, June 2009.
[51] “Bluecoat, wan optimization - mach5,” https://www.bluecoat.com/products-and-solutions/wan-optimization-mach5
[52] S. Song, D. Kim, H. Park, B.Y. Choi, and T. Choi, “CO-REDUCE: Collaborative redundancy reduction service in software-defined networks,” Proc. 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization, HotMiddlebox ’15, pp.61–66, ACM, New York, NY, USA, 2015.
[53] “Riverbed networks,” http://www.riverbed.com/
[54] “Aryaka,” http://www.aryaka.com/
[55] “Cisco, wide area application services,” http://www.cisco.com/c/en/us/products/routers/wide-area-application-services/index.html
[56] “Citrix, wan optimization with netscaler sd-wan,” https://www.citrix.com/content/dam/citrix/en_us/documents/products-solutions/wan-optimization-with-netscaler-sdwan.pdf
[57] “3gpp ts 23.002: Network architecture.”
[58] A. Banerjee, R. Mahindra, K. Sundaresan, S. Kasera, K. Van der Merwe, and S. Rangarajan, “Scaling the lte control-plane for future mobile access,” Proc. 11th ACM Conference on Emerging Networking Experiments and Technologies, CoNEXT ’15, pp.19:1–19:13, ACM, New York, NY, USA, 2015.
[59] “Cisco asr 9000 series aggregation services router broadband network gateway,” http://www.cisco.com/c/en/us/td/docs/routers/asr9000/software/asr9k_r6-0/bng/configuration/guide/b-bng-cg60xasr9k/b-bng-cg60xasr9k_chapter_010.html
[60] “Juniper broadband network gateway solution – juniper networks,” https://www.juniper.net/us/en/solutions/bng/
[61] Z.A. Qazi, C.C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, “SIMPLE-fying middlebox policy enforcement using SDN,” Proc. ACM SIGCOMM 2013 Conference on SIGCOMM, SIGCOMM ’13, pp.27–38, ACM, New York, NY, USA, 2013.
[62] V. Sekar, N. Egi, S. Ratnasamy, M. Reiter, and G. Shi, “Design and implementation of a consolidated middlebox architecture,” Technical Report, UCB/EECS-2011-110, EECS Department, University of California, Berkeley, Oct. 2011.
[63] V. Sekar, N. Egi, S. Ratnasamy, M.K. Reiter, and G. Shi, “Design and implementation of a consolidated middlebox architecture,” Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), pp.323–336, USENIX, San Jose, CA, 2012.
[64] J.W. Anderson, R. Braud, R. Kapoor, G. Porter, and A. Vahdat, “xOMB: Extensible open middleboxes with commodity servers,” Proc. Eighth ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS ’12, pp.49–60, ACM, New York, NY, USA, 2012.
[65] A. Greenhalgh, F. Huici, M. Hoerdt, P. Papadimitriou, M. Handley, and L. Mathy, “Flow processing and the rise of commodity network hardware,” SIGCOMM Comput. Commun. Rev., vol.39, no.2, pp.20–26, March 2009.
[66] 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.
[67] T. Barbette, C. Soldani, and L. Mathy, “Fast userspace packet processing,” 2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), pp.5–16, May 2015.
[68] R. Laufer, M. Gallo, D. Perino, and A. Nandugudi, “CliMB: Enabling network function composition with click middleboxes,” Proc. 2016 Workshop on Hot Topics in Middleboxes and Network Function Virtualization, HotMiddlebox ’16, pp.50–55, ACM, New York, NY, USA, 2016.
[69] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker, “NetBricks: Taking the V out of NFV,” 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), GA, pp.203–216, USENIX Association, Nov. 2016.
[70] C. Lattner and V. Adve, “LLVM: A compilation framework for lifelong program analysis transformation,” International Symposium on Code Generation and Optimization, 2004. CGO 2004., pp.75–86, March 2004.
[71] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker, “E2: A framework for nfv applications,” Proc. 25th Symposium on Operating Systems Principles, SOSP ’15, pp.121–136, ACM, New York, NY, USA, 2015.
[72] S. Han, K. Jang, A. Panda, S. Palkar, D. Han, and S. Ratnasamy, “SoftNIC: A software NIC to augment hardware,” Technical Report, UCB/EECS-2015-155, EECS Department, University of California, Berkeley, May 2015.
[73] B. Anwer, T. Benson, N. Feamster, and D. Levin, “Programming slick network functions,” Proc. 1st ACM SIGCOMM Symposium on Software Defined Networking Research, SOSR ’15, pp.14:1–14:13, ACM, New York, NY, USA, 2015.
[74] M. Bezahaf, A. Alim, and L. Mathy, “FlowOS: A flow-based platform for middleboxes,” Proc. 2013 Workshop on Hot Topics in Middleboxes and Network Function Virtualization, HotMiddlebox

'13, pp.19–24, ACM, New York, NY, USA, 2013.

[75] R. Morris, E. Kohler, J. Jannotti, and M.F. Kaashoek, "The click modular router," SIGOPS Oper. Syst. Rev., vol.33, no.5, pp.217–231, Dec. 1999.

[76] R. Morris, E. Kohler, J. Jannotti, and M.F. Kaashoek, "The click modular router," Proc. Seventeenth ACM Symposium on Operating Systems Principles, SOSP '99, pp.217–231, ACM, New York, NY, USA, 1999.

[77] P. Bosshart, G. Gibb, H.S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," Proc. ACM SIGCOMM 2013 Conference on SIGCOMM, SIGCOMM '13, pp.99–110, ACM, New York, NY, USA, 2013.

[78] P. Bosshart, G. Gibb, H.S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," SIGCOMM Comput. Commun. Rev., vol.43, no.4, pp.99–110, Aug. 2013.

[79] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," SIGCOMM Comput. Commun. Rev., vol.44, no.3, pp.87–95, July 2014.

[80] H. Song, "Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane," Proc. Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, pp.127–132, ACM, New York, NY, USA, 2013.

[81] "Barefoot: The world's fastest and most programmable networks," https://barefootnetworks.com/media/white_papers/Barefoot-World s-Fastest-Most-Programmable-Networks.pdf

[82] "Intel flexpipe," http://www.intel.com/content/dam/www/public/us/en/documents/product-bri efs/ethernet-switchfm6000-series-brief.pdf

[83] "Xpliant ethernet switch product family," http://www.cavium.com/XPliant-Ethernet-Switch-Product-Family.html

[84] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, and S. Licking, "Packet transactions: High-level programming for line-rate switches," Proc. 2016 Conference on ACM SIGCOMM 2016 Conference, SIGCOMM '16, pp.15–28, ACM, New York, NY, USA, 2016.

[85] B. Li, K. Tan, L.L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, and E. Chen, "ClickNP: Highly flexible and high performance network processing with reconfigurable hardware," Proc. 2016 Conference on ACM SIGCOMM 2016 Conference, SIGCOMM '16, pp.1–14, ACM, New York, NY, USA, 2016.

[86] B. Veal and A. Foong, "Performance scalability of a multi-core web server," Proc. 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems, ANCS '07, pp.57–66, ACM, New York, NY, USA, 2007.

[87] R. Bolla and R. Bruschi, "Pc-based software routers: High performance and application service support," Proc. ACM Workshop on Programmable Routers for Extensible Services of Tomorrow, PRESTO '08, pp.27–32, ACM, New York, NY, USA, 2008.

[88] J.R. Santos, Y. Turner, G. Janakiraman, and I. Pratt, "Bridging the gap between software and hardware techniques for I/O virtualization," USENIX 2008 Annual Technical Conference, ATC'08, pp.29–42, USENIX Association, Berkeley, CA, USA, 2008.

[89] M. Dobrescu, N. Egi, K. Argyraki, B.G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "RouteBricks: Exploiting parallelism to scale software routers," Proc. ACM SIGOPS 22Nd Symposium on Operating Systems Principles, SOSP '09, pp.15–28, ACM, New York, NY, USA, 2009.

[90] H. Liu and R. Zhang-Shen, "On direct routing in the valiant load-balancing architecture," GLOBECOM '05. IEEE Global Telecommunications Conference, 2005., pp.6 pp.–726, Dec. 2005.

[91] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," SIGCOMM Comput. Commun. Rev., vol.37, no.2, pp.51–62, March 2007.

[92] K.K. Ram, J.R. Santos, Y. Turner, A.L. Cox, and S. Rixner, "Achieving 10 Gb/s using safe and transparent network interface virtualization," Proc. 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE '09, pp.61–70, ACM, New York, NY, USA, 2009.

[93] G. Wang and T.S.E. Ng, "The impact of virtualization on network performance of amazon EC2 data center," INFOCOM, 2010 Proceedings IEEE, pp.1–9, March 2010.

[94] "Data plane development kit," http://dpdk.org

[95] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, L. Mathy, and P. Papadimitriou, "Forwarding path architectures for multicore software routers," Proc. Workshop on Programmable Routers for Extensible Services of Tomorrow, PRESTO '10, pp.3:1–3:6, ACM, New York, NY, USA, 2010.

[96] L. Rizzo, "Netmap: A novel framework for fast packet I/O," 2012 USENIX Annual Technical Conference (USENIX ATC 12), Boston, MA, pp.101–112, USENIX Association, June 2012.

[97] J. Kim, S. Huh, K. Jang, K. Park, and S. Moon, "The power of batching in the click modular router," Proc. Asia-Pacific Workshop on Systems, APSYS '12, pp.14:1–14:6, ACM, New York, NY, USA, 2012.

[98] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the art of network function virtualization," 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), pp.459–473, USENIX Association, Seattle, WA, April 2014.

[99] J. Hwang, K.K. Ramakrishnan, and T. Wood, "NetVM: High performance and flexible networking using virtualization on commodity platforms," 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), pp.445–458, USENIX Association, Seattle, WA, April 2014.

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

[101] L. Rizzo and G. Lettieri, "VALE, a switched ethernet for virtual machines," Proc. 8th International Conference on Emerging Networking Experiments and Technologies, CoNEXT '12, pp.61–72, ACM, New York, NY, USA, 2012.

[102] W. Zhang, G. Liu, W. Zhang, N. Shah, P. Lopreiato, G. Todeschi, K. Ramakrishnan, and T. Wood, "OpenNetVM: A platform for high performance network service chains," Proc. 2016 Workshop on Hot Topics in Middleboxes and Network Function Virtualization, HotMiddlebox '16, pp.26–31, ACM, New York, NY, USA, 2016.

[103] B. Hirschman, P. Mehta, K.B. Ramia, A.S. Rajan, E. Dylag, A. Singh, and M. Mcdonald, "High-performance evolved packet core signaling and bearer processing on general-purpose processors," IEEE Netw., vol.29, no.3, pp.6–14, May 2015.

[104] Z. Bronstein, E. Roch, J. Xia, and A. Molkho, "Uniform handling and abstraction of NFV hardware accelerators," IEEE Netw., vol.29, no.3, pp.22–29, May 2015.

[105] A. Belay, G. Prekas, A. Klimovic, S. Grossman, C. Kozyrakis, and E. Bugnion, "IX: A protected dataplane operating system for high throughput and low latency," Proc. 11th USENIX Conference on Operating Systems Design and Implementation, OSDI'14, pp.49–65, USENIX Association, Berkeley, CA, USA, 2014.

[106] S. Peter, J. Li, I. Zhang, D.R.K. Ports, D. Woos, A. Krishnamurthy, T. Anderson, and T. Roscoe, "Arrakis: The operating system is the control plane," 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pp.1–16, USENIX Association, CO, 2014.

[107] D. Touitou and E. Roch, "Accelerating NFV with fast path offloading," 2014 IEEE 11th Consumer Communications and Networking Conference (CCNC), pp.893–898, Jan. 2014.

[108] E. Jeong, S. Wood, M. Jamshed, H. Jeong, S. Ihm, D. Han, and

K. Park, “mTCP: A highly scalable user-level tcp stack for multicore systems,” 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), pp.489–502, USENIX Association, Seattle, WA, 2014.

[109] I. Marinos, R.N. Watson, and M. Handley, “Network stack specialization for performance,” Proc. 2014 ACM Conference on SIGCOMM, SIGCOMM ’14, pp.175–186, ACM, New York, NY, USA, 2014.

[110] I. Marinos, R.N. Watson, and M. Handley, “Network stack specialization for performance,” SIGCOMM Comput. Commun. Rev., vol.44, no.4, pp.175–186, Aug. 2014.

[111] “Cavium networks octeon ii processors,” http://www.caviumnetworks.com/OCTEON_II_MIPS64.html

[112] “Cisco quantumflow processors,” http://www.cisco.com/en/US/prod/collateral/routers/ps9343/solution_overview_c22-448936.html

[113] S. Han, K. Jang, K. Park, and S. Moon, “PacketShader: A GPU-accelerated software router,” Proc. ACM SIGCOMM 2010 Conference, SIGCOMM ’10, pp.195–206, ACM, New York, NY, USA, 2010.

[114] W. Sun and R. Ricci, “Fast and flexible: Parallel packet processing with gpus and click,” Architectures for Networking and Communications Systems, pp.25–35, Oct. 2013.

[115] J. Kim, K. Jang, K. Lee, S. Ma, J. Shim, and S. Moon, “NBA (network balancing act): A high-performance packet processing framework for heterogeneous processors,” Proc. Tenth European Conference on Computer Systems, EuroSys ’15, pp.22:1–22:14, ACM, New York, NY, USA, 2015.

[116] A. Kalia, D. Zhou, M. Kaminsky, and D.G. Andersen, “Raising the bar for using gpus in software packet processing,” 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), pp.409–423, USENIX Association, Oakland, CA, May 2015.

[117] K. Jang, S. Han, S. Han, S. Moon, and K. Park, “SSLShader: Cheap SSL acceleration with commodity processors,” Proc. 8th USENIX Conference on Networked Systems Design and Implementation, NSDI’11, pp.1–14, USENIX Association, Berkeley, CA, USA, 2011.

[118] M.A. Jamshed, J. Lee, S. Moon, I. Yun, D. Kim, S. Lee, Y. Yi, and K. Park, “Kargus: A highly-scalable software-based intrusion detection system,” Proc. 2012 ACM Conference on Computer and Communications Security, CCS ’12, pp.317–328, ACM, New York, NY, USA, 2012.

[119] G. Vasiliadis, L. Koromilas, M. Polychronakis, and S. Ioannidis, “GASPP: A GPU-accelerated stateful packet processing framework,” 2014 USENIX Annual Technical Conference (USENIX ATC 14), pp.321–332, USENIX Association, Philadelphia, PA, 2014.

[120] G. Vasiliadis, M. Polychronakis, and S. Ioannidis, “MIDeA: A multi-parallel intrusion detection architecture,” Proc. 18th ACM Conference on Computer and Communications Security, CCS ’11, pp.297–308, ACM, New York, NY, USA, 2011.

[121] W. Sun, R. Ricci, and M.L. Curry, “GPUstore: Harnessing gpu computing for storage systems in the OS kernel,” Proc. 5th Annual International Systems and Storage Conference, SYSTOR ’12, pp.9:1–9:12, ACM, New York, NY, USA, 2012.

[122] G. Vasiliadis, S. Antonatos, M. Polychronakis, E.P. Markatos, and S. Ioannidis, “Gnort: High performance network intrusion detection using graphics processors,” Proc. 11th International Symposium on Recent Advances in Intrusion Detection, RAID ’08, pp.116–134, Springer-Verlag, Berlin, Heidelberg, 2008.

[123] Y. Wang, Y. Zu, T. Zhang, K. Peng, Q. Dong, B. Liu, W. Meng, H. Dai, X. Tian, Z. Xu, H. Wu, and D. Yang, “Wire speed name lookup: A GPU-based approach,” Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), pp.199–212, USENIX, Lombard, IL, 2013.

[124] S. Han, K. Jang, K. Park, and S. Moon, “PacketShader: A GPU-accelerated software router,” SIGCOMM Comput. Commun. Rev., vol.40, no.4, pp.195–206, Aug. 2010.

[125] A.V. Aho and M.J. Corasick, “Efficient string matching: An aid to bibliographic search,” Commun. ACM, vol.18, no.6, pp.333–340, June 1975.

[126] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, “Split/merge: System support for elastic execution in virtual middleboxes,” Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), pp.227–240, USENIX, Lombard, IL, 2013.

[127] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, “OpenNF: Enabling innovation in network function control,” Proc. 2014 ACM Conference on SIGCOMM, SIGCOMM ’14, pp.163–174, ACM, New York, NY, USA, 2014.

[128] A. Gember, A. Krishnamurthy, S.S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar, “Stratos: A network-aware orchestration layer for middleboxes in the cloud,” CoRR, vol.abs/1305.0209, 2013.

[129] S. Rajagopalan, D. Williams, and H. Jamjoom, “Pico replication: A high availability framework for middleboxes,” Proc. 4th Annual Symposium on Cloud Computing, SOCC ’13, pp.1:1–1:15, ACM, New York, NY, USA, 2013.

[130] J. Sherry, P.X. Gao, S. Basu, A. Panda, A. Krishnamurthy, C. Maciocco, M. Manesh, J.a. Martins, S. Ratnasamy, L. Rizzo, and S. Shenker, “Rollback-recovery for middleboxes,” Proc. 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM ’15, pp.227–240, ACM, New York, NY, USA, 2015.

[131] J. Sherry, P.X. Gao, S. Basu, A. Panda, A. Krishnamurthy, C. Maciocco, M. Manesh, J.a. Martins, S. Ratnasamy, L. Rizzo, and S. Shenker, “Rollback-recovery for middleboxes,” SIGCOMM Comput. Commun. Rev., vol.45, no.4, pp.227–240, Aug. 2015.

[132] M. Kablan, B. Caldwell, R. Han, H. Jamjoom, and E. Keller, “Stateless network functions,” Proc. 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization, HotMiddlebox ’15, pp.49–54, ACM, New York, NY, USA, 2015.

[133] D. Ongaro, S.M. Rumble, R. Stutsman, J. Ousterhout, and M. Rosenblum, “Fast crash recovery in RAMCloud,” Proc. Twenty-Third ACM Symposium on Operating Systems Principles, SOSP ’11, pp.29–41, ACM, New York, NY, USA, 2011.

[134] J. Ousterhout, A. Gopalan, A. Gupta, A. Kejriwal, C. Lee, B. Montazeri, D. Ongaro, S.J. Park, H. Qin, M. Rosenblum, S. Rumble, R. Stutsman, and S. Yang, “The RAMCloud storage system,” ACM Trans. Comput. Syst., vol.33, no.3, pp.7:1–7:55, Aug. 2015.

[135] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro, “FaRM: Fast remote memory,” Proc. 11th USENIX Conference on Networked Systems Design and Implementation, NSDI’14, pp.401–414, USENIX Association, Berkeley, CA, USA, 2014.

[136] J. Khalid, A. Gember-Jacobson, R. Michael, A. Abhashkumar, and A. Akella, “Paving the way for NFV: Simplifying middlebox modifications using StateAlyzr,” 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16), pp.239–253, USENIX Association, Santa Clara, CA, March 2016.

[137] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, “OpenNF: Enabling innovation in network function control,” Proc. 2014 ACM Conference on SIGCOMM, SIGCOMM ’14, pp.163–174, ACM, New York, NY, USA, 2014.

[138] A. Gember-Jacobson and A. Akella, “Improving the safety, scalability, and efficiency of network function state transfers,” Proc. 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization, HotMiddlebox ’15, pp.43–48, ACM, New York, NY, USA, 2015.

[139] S.K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J.C. Mogul, “Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags,” Proc. 11th USENIX Conference

on Networked Systems Design and Implementation, NSDI'14, pp.533–546, USENIX Association, Berkeley, CA, USA, 2014.

[140] J. Khalid, M. Coatsworth, A. Gember-Jacobson, and A. Akella, "A standardized southbound API for VNF management," Proc. 2016 Workshop on Hot Topics in Middleboxes and Network Function Virtualization, HotMiddlebox '16, pp.38–43, ACM, New York, NY, USA, 2016.

[141] D.A. Joseph, A. Tavakoli, and I. Stoica, "A policy-aware switching layer for data centers," SIGCOMM Comput. Commun. Rev., vol.38, no.4, pp.51–62, Aug. 2008.

[142] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," Proc. 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '02, pp.73–86, ACM, New York, NY, USA, 2002.

[143] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," SIGCOMM Comput. Commun. Rev., vol.32, no.4, pp.73–86, Aug. 2002.

[144] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," IEEE/ACM Trans. Netw., vol.12, no.2, pp.205–218, April 2004.

[145] R. Gold, P. Gunningberg, and C. Tschudin, "A virtualized link layer with support for indirection," Proc. ACM SIGCOMM Workshop on Future Directions in Network Architecture, FDNA '04, pp.28–34, ACM, New York, NY, USA, 2004.

[146] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker, "Middleboxes no longer considered harmful," Proc. 6th Conference on Symposium on Opearting Systems Design Implementation - Volume 6, OSDI'04, p.15, USENIX Association, Berkeley, CA, USA, 2004.

[147] S.K. Fayazbakhsh, V. Sekar, M. Yu, and J.C. Mogul, "FlowTags: Enforcing network-wide policies in the presence of dynamic middlebox actions," Proc. Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, pp.19–24, ACM, New York, NY, USA, 2013.

[148] Z.A. Qazi, C.C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying middlebox policy enforcement using SDN," SIGCOMM Comput. Commun. Rev., vol.43, no.4, pp.27–38, Aug. 2013.

[149] W. Ding, W. Qi, J. Wang, and B. Chen, "OpenSCaaS: An open service chain as a service platform toward the integration of SDN and NFV," IEEE Netw., vol.29, no.3, pp.30–35, May 2015.

[150] Y.D. Lin, P.C. Lin, C.H. Yeh, Y.C. Wang, and Y.C. Lai, "An extended SDN architecture for network function virtualization with a case study on intrusion prevention," IEEE Netw., vol.29, no.3, pp.48–53, May 2015.

[151] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," SIGCOMM Comput. Commun. Rev., vol.38, no.2, pp.17–29, March 2008.

[152] M. Chowdhury, M.R. Rahman, and R. Boutaba, "ViNEYard: Virtual network embedding algorithms with coordinated node and link mapping," IEEE/ACM Trans. Netw., vol.20, no.1, pp.206–219, Feb. 2012.

[153] A. Fischer, J.F. Botero, M.T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," IEEE Commun. Surveys Tuts., vol.15, no.4, pp.1888–1906, Fourth 2013.

[154] M.C. Luizelli, L.R. Bays, L.S. Buriol, M.P. Barcellos, and L.P. Gaspary, "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions," 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), pp.98–106, May 2015.

[155] R. Cohen, L. Lewin-Eytan, J.S. Naor, and D. Raz, "Near optimal placement of virtual network functions," 2015 IEEE Conference on Computer Communications (INFOCOM), pp.1346–1354, April 2015.

[156] A. Basta, W. Kellerer, M. Hoffmann, H.J. Morper, and K. Hoffmann, "Applying NFV and SDN to LTE mobile core gateways, the functions placement problem," Proc. 4th Workshop on All Things Cellular: Operations, Applications, Challenges, AllThingsCellular '14, pp.33–38, ACM, New York, NY, USA, 2014.

[157] A. Dwaraki and T. Wolf, "Adaptive service-chain routing for virtual network functions in software-defined networks," Proc. 2016 Workshop on Hot Topics in Middleboxes and Network Function Virtualization, HotMiddlebox '16, pp.32–37, ACM, New York, NY, USA, 2016.

[158] M.F. Bari, S.R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," 2015 11th International Conference on Network and Service Management (CNSM), pp.50–56, Nov. 2015.

[159] H. Pirkul and V. Jayaraman, "A multi-commodity, multi-plant, capacitated facility location problem: Formulation and efficient heuristic solution," Computers Operations Research, vol.25, no.10, pp.869–878, 1998.

[160] G.D. Forney, "The viterbi algorithm," Proc. IEEE, vol.61, no.3, pp.268–278, March 1973.

[161] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, "The dynamic placement of virtual network functions," 2014 IEEE Network Operations and Management Symposium (NOMS), pp.1–9, May 2014.

[162] H. Moens and F.D. Turck, "VNF-P: A model for efficient placement of virtualized network functions," 10th International Conference on Network and Service Management (CNSM) and Workshop, pp.418–423, Nov. 2014.

[163] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," 2014 IEEE 3rd International Conference on Cloud Networking (CloudNet), pp.7–13, Oct. 2014.

[164] A. Mohammadkhan, S. Ghapani, G. Liu, W. Zhang, K.K. Ramakrishnan, and T. Wood, "Virtual function placement and traffic steering in flexible and dynamic software defined networks," The 21st IEEE International Workshop on Local and Metropolitan Area Networks, pp.1–6, April 2015.

[165] S. Sahhaf, W. Tavernier, D. Colle, and M. Pickavet, "Network service chaining with efficient network function mapping based on service decompositions," Proc. 2015 1st IEEE Conference on Network Softwarization (NetSoft), pp.1–5, April 2015.

[166] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," 2015 IEEE 4th International Conference on Cloud Networking (CloudNet), pp.255–260, Oct. 2015.

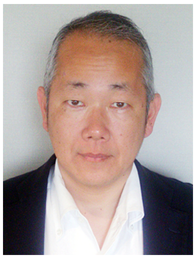
[167] T.S. Buda, H. Assem, L. Xu, D. Raz, U. Margolin, E. Rosensweig, D.R. Lopez, M.I. Corici, M. Smirnov, R. Mullins, O. Uryupina, A. Mozo, B. Ordozgoiti, A. Martin, A. Alloush, P. O'Sullivan, and I.G.B. Yahia, "Can machine learning aid in delivering new use cases and scenarios in 5G?," NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium, pp.1279–1284, April 2016.

[168] G. Gardikis, I. Koutras, G. Mavroudis, S. Costicoglou, G. Xilouris, C. Sakkas, and A. Kourtis, "An integrating framework for efficient nfv monitoring," 2016 IEEE NetSoft Conference and Workshops (NetSoft), pp.1–5, June 2016.

[169] M. Miyazawa, M. Hayashi, and R. Stadler, "vNMF: Distributed fault detection using clustering approach for network function virtualization," 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), pp.640–645, May 2015.

[170] J.M. Sánchez, I.G.B. Yahia, and N. Crespi, "Self-modeling based diagnosis of services over programmable networks," 2016 IEEE NetSoft Conference and Workshops (NetSoft), pp.277–285, June 2016.

[171] D. Kushnir and M. Goldstein, "Causality inference for failures in NFV," 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp.929–934, April 2016.



Kohei Shiomoto is a Professor of Tokyo City University, Tokyo Japan. His current interest research areas include software-defined networking, network function virtualization, machine-learning, and network management. From 1989 to 2017, in NTT Laboratories, he was engaged in research and development of high-speed networks including ATM networks, IP/MPLS networks, GMPLS networks, network virtualization, traffic management, network analytics. From 1996 to 1997 he was engaged in research in high-speed networking as a visiting scholar at Washington University in St. Louis, MO, USA. He received his B.E., M.E., and Ph.D. degrees in information and computer sciences from Osaka University, Osaka in 1987 1989, and 1998. He is a Fellow of IEICE, a Senior Member of IEEE, and a member of ACM.