# Enhancing Performance, Security, and Management in Network Function Virtualization

Yang Zhang[1], Zhi-Li Zhang[2]
Email: {yazhang, zhzhang}@cs.umn.edu
[1]PhD Student, [2]PhD Supervisor, University of Minnesota, Twin Cities

***Abstract*—In an era of ubiquitous connectivity, various new applications, network protocols, and online services (e.g., cloud services, distributed machine learning, cryptocurrency) have been constantly creating, underpinning many of our daily activities. Emerging demands for networks have led to growing traffic volume and complexity of modern networks, which heavily rely on a wide spectrum of specialized network functions (e.g., Firewall, Load Balancer) for diverse purposes. Although these (virtual) network functions (VNFs) are widely deployed, they are instantiated in an *uncoordinated* manner failing to meet growing demands of evolving networks. In this dissertation, we argue that networks equipped with VNFs can be designed in a fashion similar to how computer software is programmed today. By following the blueprint of modularization, networks can be made more efficient, secure, and manageable.**

## I. Introduction

The classical description of the Internet architecture is based on the hourglass model [1]. In this architecture, the only function in the neck of the hourglass are IP routers which determine routes and forward packets. Today, networks have been growing to meet new and sophisticated demands. For example, demands for securing networks – whether backbone networks of Internet service providers, campus/enterprise networks, data center networks, or even satellite networks – have been growing rapidly; carrier network keeps track of bandwidth consumption to bill users for usage; real-time streaming service designs new protocols for pursuing extremely low latency; IP depletion problem has been discussed for decades. There are many other requirements – load balancing, data compressing and caching, proxing, to name a few, and thus today's networks support far beyond merely forwarding packets.

To satisfy these demands for networks, virtual network functions (VNFs) are inserted into networks to perform specialized functions. VNFs run in one or more virtual machines on top of hardware networking infrastructure, and provide functions such as transforming, inspecting, filtering, or otherwise manipulating traffic for purposes other than packet forwarding. While VNFs are widely deployed to bring various benefits such as regulated control and performance enhancement, they are typically developed and deployed in an isolated way, without interacting with each other. For example, MPTCP proxy implements its own logic to detect and associate MPTCP subflows while this logic may also be required by other VNFs. Parallelism among VNFs can be served as a general framework to support better service function chain performance. However, VNFs are programmed as gigantic black-boxes now and instantiated in an uncoordinated manner, failing to meet the growing demands of evolving networks.

In this thesis, we advocate for modularization in how VNFs are implemented and operated. Instead of designing and deploying VNFs as a gigantic black-boxes – where each VNF has no interaction with others or hosts – we argue that VNF development should break barriers among isolated VNFs. Rather, VNFs should provide necessary interfaces to expose certain information and to be integrated into networks, allowing VNF developers or network administrators to fuse them together. As we will show, VNF modularization is feasible for VNF development and deployment in networks, and brings better network performance, security, and manageability.

## II. Thesis Rationale: Background, Key Challenges, and Related Work

The challenges in realizing the potential of NFV has attracted a plethora of research studies in recent years. In the following we will use several examples to illustrate the key challenges in NFV that motivate our thesis.

### *A. Challenge 1. Parallelizing Network Functions For Accelerating Service Function Chains.*

A Service Function Chain (SFC) defines a sequence of VNFs, and stitches them together [2]. SFC has become a key enabler for network operators to offer diverse services and an important application for Software Defined Networking (SDN) [3], [4], [5]. Recently, operators can create, update, remove, or scale out/in network functions (NFs) *on demand* [6], [7], [8], construct a sequence of NFs to form a SFC [2], and steer traffic through it to meet service requirements [9], [10], [11]. However, virtualization and "softwarization" of NFs pose many new challenges [12]. In particular, traffic traversing virtualized NFs suffers from reduced throughput and increased latency, compared to physical NFs [13], [9], [10], [11]. The flexibility offered by SDN will enable more complex network services to be deployed, which will likely lead to longer SFC. As the length of an SFC (*i.e.,* number of NFs) increases, so does its overhead.

Exploiting parallelism to reduce packet processing latency and increase the overall system throughput is a classical approach that is widely used in computer software design. For example, most of today's web-based cloud computing applications take advantage of the stateless HTTP protocols for parallel HTTP transaction processing. Data analytics frameworks such as Map-Reduce and Spark utilize task level parallelism to speed up massive compute jobs using multiple servers. In terms of NFV, NF-level parallelism is first explored in ParaBox [14] and later in NFP [15] by exploiting order independence of certain NFs for parallel packet processing within an SFC. Both efforts focus on parallelizing packet processing for SFCs on a *single* (multi-core) server. Real-world NFs, on the other hand, will likely be operating in edge clouds or data centers with clusters of servers. How to effectively utilize multiple servers to reduce per-packet processing latency and increase the overall system throughput is the main problem we explore.

## B. Challenge 2. Making Network Functions Aware of MPTCP.

MPTCP is designed to boost data transmission throughput by taking advantage of multiple available paths in network. It is a major extension to TCP and has been standardized by the Internet Engineering Task Force [16]. MPTCP can not only increase data throughput, but also seamlessly perform vertical handover between multiple paths, which makes the data transmission more robust against link failures [17]. Moreover, these features are obtained without requiring any modification at the application level. Thus, MPTCP can be deployed in today's Internet without much impact on the proper functioning of the existing network devices [18].

Although MPTCP is designed to be compatible with most network devices, MPTCP can not be necessarily understood by these network devices. To the best of our knowledge, few network devices are designed with explicit consideration of MPTCP, and little work has been done to investigate how to better support this new protocol in the network. For example, MPTCP is designed to be no more aggressive than a regular TCP on a shared bottleneck link [19]. This implies that multiple subflows of a MPTCP session can only bring throughput improvement if the subflows do not share the same bottleneck link. However, neither end host nor network has this mechanism to avoid common links being traversed by MPTCP subflows in the same MPTCP session. If routers could spread the MPTCP subflows onto disjoint paths, the overall data goodput could be greatly improved [20].

Furthermore, making network devices MPTCP-aware may improve the functionality of certain network services. Take application identification service or intrusion detection service as an example. If an application/malware signature spans across multiple MPTCP subflows, the accuracy of the identification outcome may be improved by assembling subflows together. Likewise, when a subflow that carries the signature is identified/blocked, all other subflows belonging to the same MPTCP session can be identified/blocked too.

Thus, a MPTCP analysis system is developed to associate MPTCP subflows in the same MPTCP session. The main difference between our work and mptcptrace [21] is that mptcptrace works as an offline tool and requires full flow records; it uses token-based approach to associate MPTCP flows. On the other hand, our work is designed to be an online tool that can work with either full or partial flow records. Sandri *et al.* [20] have designed a method to improve MPTCP performance by distributing subflows of the same MPTCP connection across different paths. An OpenFlow controller is used to associate MPTCP subflows and hence it relies on reactive flow processing, which may bring scalability concerns. In addition, their subflow association algorithm is based on token only, ignoring MPTCP meta socket. Our work does not assume a centralized point where all flows would pass through and hence can be used in a more flexible setting.

## C. Challenge 3. Taking Consensus as a Network Service.

SDN simplifies network devices by moving control plane functions to a logically centralized control plane; therefore data plane devices become simple programmable forwarding elements. For scalability and reliability, the logically centralized control plane ("network OS") is often realized via multiple SDN controllers, forming a distributed system. Open Network Operating System (ONOS) [22] and OpenDayLight (ODL) [23] are two such Network OS examples supporting multiple SDN controllers for high availability.

In the distributed network OS such as ONOS and OpenDayLight, the replicated controllers rely on conventional distributed system mechanisms such as consensus protocols for state replication and consistency. Paxos [24] is a widely used distributed consensus protocol in production software [25], [26], [27], [28] to ensure liveness and safety. Unfortunately, Paxos is very difficult to understand and implement in practical systems [29]. Raft [29] attempts to address these complexities by decomposing the consensus problem into relatively independent sub-problems: leader election, log replication, and safety. It implements a more "easy-to-understand" consensus protocol that manages a replicated log to provide a building block for building practical distributed systems. Both ONOS and ODL use certain implementations of Raft to ensure consistency among replicated network states. For example, ONOS maintains a global network view to SDN control programs that is logically centralized, but physically distributed among multiple controllers. It employs Raft to manage the switch-to-controller mastership and to provide distributed primitives to control programs such as ConsistentMap, which guarantees strong consistency for a key-value store.

The reliance of distributed network OS on consensus protocols to maintain *consistent* network state introduces an intricate *inter-dependency* between the network OS (as a distributed system) and the network it attempts to control. This inter-dependency may create new kinds of fault scenarios or instabilities that have neither been addressed in distributed systems nor in networking. In particular, it may severely affect the correct or efficient operations of consensus protocols. The key issue lies in the fact that the design of fault-tolerant distributed system mechanisms such as consensus algorithms typically focuses on server failures alone, while assuming the underlying network will handle connectivity issues on its own. For example, the design of Paxos or Raft assumes that the network may arbitrarily delay or drop messages; however, *as long as the network is not partitioned*, messages from one end point will *eventually* be delivered to another end point. Such assumptions about the network hold true in classical IP networks, where distributed routing algorithms running on routers cooperate with each other to establish new paths after failures. SDN now creates *cyclic dependencies* among *control network connectivity*, *consensus protocols*, and *control logic managing the network*, where the control logic managing the network is built on top of a distributed system (*e.g.,* ONOS) which relies on consensus protocols for consistency and control network connectivity for communication, whereas the network data plane (and control network) hinges on this distributed system to set up rules to control and enforce "who can talk to whom" among networking elements. Consequently, new failure scenarios can arise in SDN.

### D. Challenge 4. Fusing LAN Virtualization with WAN Virtualization.

Many modern enterprises are geographically dispersed across multiple sites over a wide area network (WAN). Typically, branch office site networks are connected to a central office core network or a core data center (private cloud) via "dedicated" WAN links provisioned by one or more service providers. For security and privacy, WAN gateways at each site route enterprise traffic over VPN tunnels connecting edge networks with core/cloud networks. However, WAN link failures happen more frequently than expected [30]. Even with a dedicated WAN, dealing with WAN failures is a key consideration in Google's B4&after systems [31], [?]. With increasing complexity in WAN (e.g., WAN managed by different ISPs; emerging 5G links adopted in WAN), such failures are likely to occur more frequently than before. More importantly, WAN failures have a significant impact on enterprises, and thus dealing with them is a major practical challenge.

Compared to local area network (LAN), WAN connectivity has become prohibitively expensive to meet the growing demands required by applications. This has led to a shift towards novel WAN solutions using SDN to better manage bandwidth intensive traffic traversing private WANs and increase the utilization of expensive WAN links. SD-WAN solutions [32] allow multiple WAN links to be logically combined for higher capacity. When WAN link failures are detected, traffic is re-distributed from failed links to other available links for resilience. We argue that a joint/modular design between LAN and WAN is required for a more resilient SD-WAN solution.

## III. Research Overview: Intellectual Merits

We begin by re-considering network equipped with VNFs from a network administrator's perspective. A global view of deployed VNFs brings new opportunities for performance optimization over the network, and thus we explore parallelism in service function chains composing a sequence of VNFs that are typically traversed in-order by data flows. We then study MPTCP for the purpose of boosting throughput and enhancing security. Instead of implementing a customized solution in every VNF to conquer this common challenge – making VNFs aware of MPTCP, we implement an online service named SAMPO to be readily integrated into VNFs. Following the same principle, we make an attempt to take consensus as a service in software-defined networks. We illustrate new network failure scenarios that are not explicitly handled by existing consensus algorithms such as Raft, thereby severely affecting their correct or efficient operations. Finally, we present Durga, a system fusing wide area network (WAN) virtualization on gateway with local area network (LAN) virtualization technology. It seamlessly aggregates multiple WAN links into a (virtual) big pipe for better utilizing WAN links and also provides fast fail-over thus minimizing application performance degradation under WAN link failures. Without the support from LAN virtualization technology, existing solutions fail to provide high reliability and performance required by today's enterprise applications.

### A. HybridSFC: Accelerating Service Function Chains with Parallelism

We present HybridSFC, a parallelism mechanism to accelerate SFCs spanning multiple servers. Instead of parallelizing NFs as much as possible (*e.g.,* [14], [15]), HybridSFC employs a controller that converts a sequential chain into a hybrid chain and parallelizes packet processing only if it is *beneficial.* Additionally, the controller adopts traffic level parallelism to distribute traffic in an optimized way to satisfy service level objectives of target traffic. The controller programs both software and hardware switches to activate parallelism across NFs spanning multiple physical servers. HybridSFC employs a customized data plane to support hybrid chains without modifying the implementation of existing NFs. Based on the instructions from the controller, HybridSFC data plane *mirrors* packets to parallelized NFs and then *merges* their outputs to ensure correctness – *i.e.,* traffic and NF states changed by a hybrid chain must be identical to what would have been produced by the original sequential SFC. We evaluate the performance of HydridSFC and demonstrate that it can reduce the latency by up to 37.7%.

### B. SAMPO: Online Subflow Association for Multipath TCP with Partial Flow Record

As facilitating VNFs aware of MPTCP is beneficial to both the performance of MPTCP sessions and the quality of network services, we take a first step towards making the network devices MPTCP-aware by investigating how to associate subflows that belong to the same MPTCP session. This is relatively easy to achieve at a place where all flow records are available, e.g., at the end hosts. In this case one can use MPTCP token in TCP option field carried in the MP_JOIN message of each subflow to identify a MPTCP session. However, the problem of associating MPTCP subflows becomes more challenging in network. For example, it is common that network monitoring devices perform sampling on the data streams before processing them in order to reduce processing load. Moreover, flow paths can also change due to network dynamics and hence the monitoring device may only see a portion of the flow. All such complications may cause the

MPTCP packets containing the token to be missing from flow records, and hence a more comprehensive and robust solution is needed for subflow association in network.

We propose SAMPO, an online subflow association mechanism for MPTCP with partial flow record. Our main contribution is a data sequence number (DSN) based algorithm that can associate subflows based on analysis of DSN values of each subflow, their range and overlapping pattern. Through extensive theoretical analysis and experimentation, we find that the DSN based association is very effective even when a very small fraction of packets from each subflow are available. For instance, the algorithm reaches close to 100% accuracy when only 1% of packets are sampled in.

### C. When Raft Meets SDN: How to Elect a Leader and Reach Consensus in an Unruly Network

We illustrate a few network failure scenarios that may arise when applying Raft to a distributed SDN control cluster. We demonstrate how these failure scenarios can severely affect the *correct* or *efficient* operations of Raft: in the best case they significantly reduce the available "normal" operation time of Raft; and in the worst case, they render Raft unable to reach consensus by failing to elect a consistent leader. It is worth noting that the problems highlighted here are different from those addressed by, *e.g.,* the celebrated CAP Theorem in distributed systems [33], [34], which establishes impossibility results regarding simultaneously ensuring availability and (strong) consistency under network partitions. This result has been recently generalized in [35] to SDN networks in terms of impossibility results regarding *ensuring network policy consistency under network partitions*. In contrast, we argue that due to the inter-dependency between the network OS as a distributed system and the network it attempts to control, SDN introduces new network failure scenarios that are not explicitly handled by existing consensus algorithms such as Raft, thereby severely affecting their correct or efficient operations.

We then discuss possible "fixes" to circumvent these problems. In particular, we argue that in order to fundamentally break this inter-dependency, it is crucial to equip the SDN control network with a resilient routing mechanism such as *PrOG* [36] that guarantees connectivity among (non-partitioned) SDN controllers under arbitrary failures. We then propose a network-assisted Raft consensus algorithm that takes advantage of programmable network and offloads certain Raft [29] functionality to P4 [37] switches. Our goal is to improve the performance of Raft without sacrificing scalability. Using a vanilla Raft implementation [38], *PrOG*, and a P4 switch simulator, we provide preliminary evaluation results.

### D. Improving SD-WAN Resilience: From Vertical Handoff to WAN-Aware MPTCP

We develop a novel *WAN-awareness* mechanism that enables end systems with MPTCP support (even with only one network interface) to generate multiple MPTCP subflows using *virtual subnet* addresses. This mechanism is enhanced at SD-WAN gateways to load-balance subflows across WAN links and dynamically reroute them away from failed WAN links in a scalable manner.

Building on top of *WAN-aware MPTCP* (WaMPTCP), we present a novel scalable SD-WAN virtualization framework which not only can aggregate multiple (heterogeneous) WAN links into a (virtual) "big pipe", but is also capable of providing fast failover with minimal application performance degradation. The system is designed to handle diverse enterprise traffic. In addition to WaMPTCP for support of performance critical applications running on hosts with MPTCP kernel modules, it also incorporates MPTCP proxies for legacy TCP connections running on hosts with no MPTCP support as well as the default tunnel handoff mechanism for non-TCP traffic. Through extensive evaluation in both emulated testbeds and real-world deployment, we show the superior performance over existing SD-WAN solutions.

## IV. Conclusion, Lessons Learned & Thoughts for the Future

In this dissertation, we have argued that by following the blueprint of modularization, networks equipped with VNFs can be made more efficient, secure, and manageable. We now discuss a few broad lessons learned over the course of this dissertation, and what they suggest about future VNF development and deployment.

### A. Modularization involves interactions among VNFs and hence requires unified programming interfaces and platform.

In 2012, the European Telecommunications Standards Institute issued a proposal named as Network Functions Virtualization (NFV) [39]. The incentives of proposing NFV is that modern telecoms networks contain an ever increasing variety of proprietary hardware, and thus the launch of new services often demands network reconfiguration and on-site installation of new equipment which in turn requires additional floor space, power, and trained maintenance staff. NFV accelerates and requires greater flexibility and dynamism than hardware-based appliances allow. Hard-wired network with single functions boxes are tedious to maintain, slow to evolve, and prevent service providers from offering dynamic services – similar as the motivation for joint design blueprint proposed in this dissertation.

Along these lines, several projects were designed to conquer the open challenges in the joint design blueprint. For example, VNF orchestraters [40], [41] were proposed for automatically instantiating or closing VNF instances as traffic load changes. The SFC working group in IETF [2] is actively investigating how to best implement routing through multi-middlebox topologies and enforce policies about which traffic receives processing by which VNFs. Many research studies have been carried out to address statefulness of VNFs, e.g. [42], [43], [44], [45], [46], [47]. In terms of NFV behavior modeling, synthesis, testing as well as policy analysis and traffic steering, various novel techniques have been proposed, see, e.g., [48], [49], [50], [51], [52], [53], [3], [5], [4], [54]. The NFV placement problem has also attracted a plethora of research studies, mostly employing mathematical optimization techniques [55]. While we can see the benefits brought by joint design, how to design unified programming interfaces for supporting interactions among VNFs and implement a general platform for the integration of VNFs is required.

### *B. Modularization can be extended to offload certain functions down to hardware programmable switches.*

Several recent projects investigate offloading consensus algorithms to either switches [56] or FPGA devices [57]. NetPaxos [56] proposes to implement the Paxos consensus algorithm in network by leveraging programmable switches. Besides the Paxos roles implemented on servers, NetPaxos requires one switch serving as a Paxos coordinator and several others as Paxos acceptors. NetPaxos can be implemented using P4 [58], a domain specific language that allows the programming of packet forwarding planes. However, Paxos consensus algorithm is very difficult to understand and implement due to its notoriously opaque explanation and lack of details for building practical systems [29]. Thus, offloading such a complex consensus algorithm to network is error-prone. István *et al.* [57] takes the efforts of implementing the entire ZAB consensus algorithm [59] on FPGA devices using a low-level language which is difficult to program. Moreover, this hardware-based solution may not be scalable as it requires the storage of potentially large amounts of consensus states, logic, and even the application data. Even though offloading VNFs to networks is a promising area to explore, it would be demanded to formally validate the correctness of such a decoupled architecture. Moreover, it is also interesting to compare the solution implemented in real P4 switches with other existing FPGA-based or RDMA-based solutions.

### *C. Decomposing VNFs opens the door to pursue further performance enhancement.*

Recent proposals virtualize and decompose NF at different granularity. For example, EdgePlex [60] assigns a single VM for the control plane and a dedicated VM for the data plane of each customer provisioned on a provider edge router. OpenBox [61] performs fine-grained decomposition at the software module level of NFs. In this case, the components of a service chain will be NF modules, instead of VNFs. It would be more challenging to parallelize packet processing at a finer grained level in a chain.

### *D. Enhancing networks equipped with VNFs in the context of 5G technologies.*

In the emerging 5G technologies – besides innovations in radio technologies such as 5G new radio [62], [63], NFV will be a key enabling technology [64], [65], [66], [67], [68], [69] underpinning the envisioned 5G “Cloud RANs”, MECs and packet core networks for support of *network slicing* and diverse services ranging from enhanced mobile broadband to massive machine type communications and ultra-reliable low latency communications. For example, upon a request for a service (e.g., from a mobile user or a machine, say, an autonomous vehicle or an industrial controller), a SFC will be dynamically constructed using a series of VNFs such as firewalls, mobility managers, network address translators, traffic shapers and so forth that are deployed on demand at appropriate locations within a (dynamic) network slice to meet the desired service requirements. It would be challenging to jointly design VNFs in the context of 5G technology, and leverage various 5G VNFs and SFCs to support the development of 5G end-to-end facilities, network slicing, 5G services and vertical trials. Through end-to-end evaluations and 5G service trials, NFV platforms can be further refined and expanded.

## V. Acknowledgment

This research was supported by various sources of funds: DoD ARO MURI Award W911NF-12-1-0385, DTRA grant HDTRA1-09-1-0050, NSF grants CNS-1411636, CNS-1618339, CNS-1617729, CNS 1814322 and CNS183677.

## References

[1] N. R. Council, *Realizing the Information Future: The Internet and Beyond.* Washington, DC: The National Academies Press, 1994.
[2] J. M. Halpern and C. Pignataro, “Service Function Chaining (SFC) Architecture,” 2015.
[3] Z. A. Qazi *et al.*, “SIMPLE-fying Middlebox Policy Enforcement Using SDN,” in *Proc. SIGCOMM*, 2013.
[4] S. K. Fayazbakhsh *et al.*, “Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags,” in *Proc. NSDI*, 2014.
[5] Y. Zhang *et al.*, “StEERING: A software-defined networking for inline service chaining,” in *Proc. ICNP*, 2013.
[6] S. G. Kulkarni, W. Zhang, J. Hwang, S. Rajagopalan, K. Ramakrishnan, T. Wood, M. Arumaithurai, and X. Fu, “NFVnice: Dynamic Backpressure and Scheduling for NFV Service Chains,” in *Proc. SIGCOMM*, 2017.
[7] W. Zhang, J. Hwang, S. Rajagopalan, K. Ramakrishnan, and T. Wood, “Flurries: Countless Fine-Grained NFs for Flexible Per-Flow Customization,” in *Proc. CoNEXT*, 2016.
[8] P. Zave, R. A. Ferreira, X. K. Zou, M. Morimoto, and J. Rexford, “Dynamic Service Chaining with Dysco,” in *Proc. SIGCOMM*, 2017.
[9] S. Kumar, M. Tufail, S. Majee, C. Captari, and S. Homma, “Service Function Chaining Use Cases In Data Centers,” IETF, Internet-Draft draft-ietf-sfc-dc-use-cases-06, 2017.
[10] T. Nadeau and P. Quinn, “Problem Statement for Service Function Chaining,” RFC 7498, 2015.
[11] J. Napper *et al.*, “Service Function Chaining Use Cases in Mobile Networks,” IETF, Tech. Rep., 2016.
[12] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, “Network function virtualization: Challenges and opportunities for innovations,” *IEEE Communications Magazine*, 2015.
[13] J. Hwang, K. K. Ramakrishnan, and T. Wood, “NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms,” in *Proc. NSDI*, 2014.
[14] Y. Zhang, B. Anwer, V. Gopalakrishnan, B. Han, J. Reich, A. Shaikh, and Z.-L. Zhang, “ParaBox: Exploiting Parallelism for Virtual Network Functions in Service Chaining,” in *Proc. SOSR*, 2017.
[15] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, “NFP: Enabling Network Function Parallelism in NFV,” in *Proc. SIGCOMM*, 2017.
[16] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, “Tcp extensions for multipath operation with multiple addresses,” Internet Requests for Comments, RFC, 2013.
[17] C. Paasch, G. Detal, F. Duchene, C. Raiciu, and O. Bonaventure, “Exploring mobile/wifi handover with multipath tcp,” in *Proc. CellNet*, 2012.
[18] C. Raiciu, C. Paasch, S. Barré, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, “How hard can it be? designing and implementing a deployable multipath tcp,” in *Proc. NSDI*, 2012.

[19] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath tcp," in *Proc. NSDI*, 2011.

[20] M. Sandri, A. Silva, L. Rocha, and F. Verdi, "On the benefits of using multipath tcp and openflow in shared bottlenecks," in *Proc. AINA*, 2015.

[21] B. Hesmans and O. Bonaventure, "Tracing multipath tcp connections," in *Proc. SIGCOMM*, 2014.

[22] P. Berde *et al.*, "ONOS: Towards an Open, Distributed SDN OS," in *Proc. HotSDN*, 2014.

[23] "OpenDaylight: Open Source SDN Platform," https://www.opendaylight.org/, 2017.

[24] L. Lamport, "The Part-time Parliament," *ACM Transactions on Computer Systems*, 1998.

[25] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally deployed software defined wan," in *Proc. SIGCOMM*, 2013.

[26] T. D. Chandra, R. Griesemer, and J. Redstone, "Paxos Made Live: an Engineering Perspective," in *Proc. PODC*, 2007.

[27] A. Lakshman and P. Malik, "Cassandra: a Decentralized Structured Storage System," *SIGOPS Operating Systems Review*, 2010.

[28] M. Burrows, "The Chubby Lock Service for Loosely-coupled Distributed Systems," in *Proc. OSDI*, 2006.

[29] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm," in *Proc. USENIX ATC*, 2014.

[30] D. Turner, K. Levchenko, A. C. Snoeren, and S. Savage, "California fault lines: Understanding the causes and impact of network failures," in *Proc. SIGCOMM.*

[31] C.-Y. Hong *et al.*, "B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined wan," in *Proc. SIGCOMM*, 2018.

[32] "Comparison of the SD-WAN vendor solutions," https://www.netmanias.com/en/post/oneshot/12481/sd-wan-sdn-nfv/comparison-of-the-sd-wan-vendor-solutions, 2017.

[33] E. A. Brewer, "Towards robust distributed systems," 2000.

[34] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *SIGACT News*, 2002.

[35] A. Panda, C. Scott, A. Ghodsi, T. Koponen, and S. Shenker, "CAP for Networks," in *Proc. HotSDN*, 2013.

[36] E. Ramadan, H. Mekky, B. Dumba, and Z.-L. Zhang, "Adaptive resilient routing via preorders in sdn," in *Proc. DCC*, 2016.

[37] P. Bosshart *et al.*, "P4: Programming Protocol-Independent Packet Processors," *SIGCOMM CCR*, 2014.

[38] D. Ongaro, "Logcabin: A distributed storage using raft," https://github.com/logcabin, 2016.

[39] "ETSI NFV Standard," http://www.etsi.org/deliver/etsi_gs/nfv/001_099/001/01.01.01_60/gs_nfv001v010101p.pdf, 2017.

[40] OPNFV, "OPNFV: Open Platform for NFV," https://www.opnfv.org, 2018.

[41] S. Palkar *et al.*, "E2: A Framework for NFV Applications," in *Proc. SOSP*, 2015.

[42] S. Rajagopalan *et al.*, "Split/Merge: System Support for Elastic Execution in Virtual Middleboxes," in *Proc. NSDI*, 2013.

[43] A. Gember-Jacobson *et al.*, "OpenNF: Enabling Innovation in Network Function Control," in *Proc. SIGCOMM*, 2014.

[44] J. Khalid, A. Gember-jacobson, R. Michael, A. Abhashkumar, and I. Nsdi, "Paving the Way for NFV: Simplifying Middlebox Modifications Using StateAlyzr This paper is included in the Proceedings of the," in *Proc. NSDI*, 2016.

[45] M. Kablan, A. Alsudais, E. Keller, and F. Le, "Stateless network functions: Breaking the tight coupling of state and processing," in *Proc. NSDI*, 2017.

[49] D. Joseph and I. Stoica, "Modeling middleboxes," *IEEE Network: The Magazine of Global Internetworking*, 2008.

[46] 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," in *Proc. of SIGCOMM*, 2015.

[47] S. Woo, J. Sherry, S. Han, S. Moon, S. Ratnasamy, and S. Shenker, "Elastic scaling of stateful network functions," in *Proc. NSDI*, 2018.

[48] W. Wu, Y. Zhang, and S. Banerjee, "Automatic synthesis of nf models by program analysis," in *Proc. HotNets*, 2016.

[50] R. Stoenescu, M. Popovici, L. Negreanu, and C. Raiciu, "Symnet: Scalable symbolic execution for modern networks," in *Proc. of SIGCOMM*, 2016.

[51] G. P. Katsikas, M. Enguehard, M. Kuźniar, G. Q. Maguire Jr, and D. Kostić, "Snf: synthesizing high performance nfv service chains," *PeerJ Computer Science*, 2016.

[52] S. K. Fayaz *et al.*, "Buzz: Testing context-dependent policies in stateful networks," in *Proc. NSDI*, 2016.

[53] R. Hartert *et al.*, "PGA: Using Graphs to Express and Automatically Reconcile Network Policies," in *Proc. SIGCOMM*, 2015.

[54] B. Anwer, T. Benson, N. Feamster, and D. Levin, "Programming Slick Network Functions," in *Proc. SOSR*, 2015.

[55] X. Li and C. Qian, "A survey of network function placement," in *Proc. of CCNC*, 2016.

[56] H. T. Dang *et al.*, "Netpaxos: Consensus at network speed," in *Proc. SOSR*, 2015.

[57] Z. István *et al.*, "Consensus in a box: Inexpensive coordination in hardware," in *Proc. NSDI*, 2016.

[58] P. Bosshart *et al.*, "P4: Programming Protocol-Independent Packet Processors," in *CCR*, 2014.

[59] P. Hunt *et al.*, "Zookeeper: Wait-free coordination for internet-scale systems," in *Proc. ATC*, 2010.

[60] A. Chiu, V. Gopalakrishnan, B. Han, M. Kablan, O. Spatscheck, C. Wang, and Y. Xu, "EdgePlex: decomposing the provider edge for flexibilty and reliability," in *Proc. SOSR*, 2015.

[61] A. Bremler-Barr, Y. Harchol, and D. Hay, "OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions," in *Proc. SIGCOMM*, 2016.

[62] "Qualcomm-5G-NR," https://www.qualcomm.com/invention/technologies/5g-nr, 2018.

[63] S. Yost, "Decoding 5G New Radio: The Latest on 3GPP and ITU Standards," https://spectrum.ieee.org/telecom/wireless/decoding-5g-new-radio, 2018.

[64] C. J. Bernardos, A. de la Oliva, P. Serrano, A. Banchs, L. M. Contreras, H. Jin, and J. C. Zuniga, "An Architecture for Software Defined Wireless Networking," in *IEEE Wireless Communications Magazine*, 2014.

[65] D. Sabella, P. Rost, A. Banchs, V. Savin, M. Consonni, M. Di Girolamo, M. Lalam, A. Maeder, and I. Berberana, "Benefits and challenges of cloud technologies for 5G architecture," in *Proc. of Vehicular Technology*, 2015.

[66] P. Rost, A. Banchs, I. Berberana, M. Breitbach, M. Doll, H. Droste, C. Mannweiler, M. A. Puente, K. Samdanis, and B. Sayadi, "Mobile network architecture evolution toward 5G," in *IEEE Communications Magazine*, 2016.

[67] M. Moradi, Y. Lin, Z. M. Mao, S. Sen, and O. Spatscheck, "Softbox: A customizable, low-latency, and scalable 5g core network architecture," *IEEE Journal on Selected Areas in Communications*, 2018.

[68] Z. A. Qazi, P. K. Penumarthi, V. Sekar, V. Gopalakrishnan, K. Joshi, and S. R. Das, "KLEIN: A Minimally Disruptive Design for an Elastic Cellular Core ," in *Proc. SOSR*, 2016.

[69] C. Rotsos, D. King, A. Farshad, J. Bird, L. Fawcett, N. Georgalas, M. Gunkel, K. Shiomoto, A. Wang, A. Mauthe, N. Race, and D. Hutchison, "Network service orchestration standardization: A technology survey," in *Computer Standards and Interfaces*, 2017.