Lecture 7: Network Function Virtualization

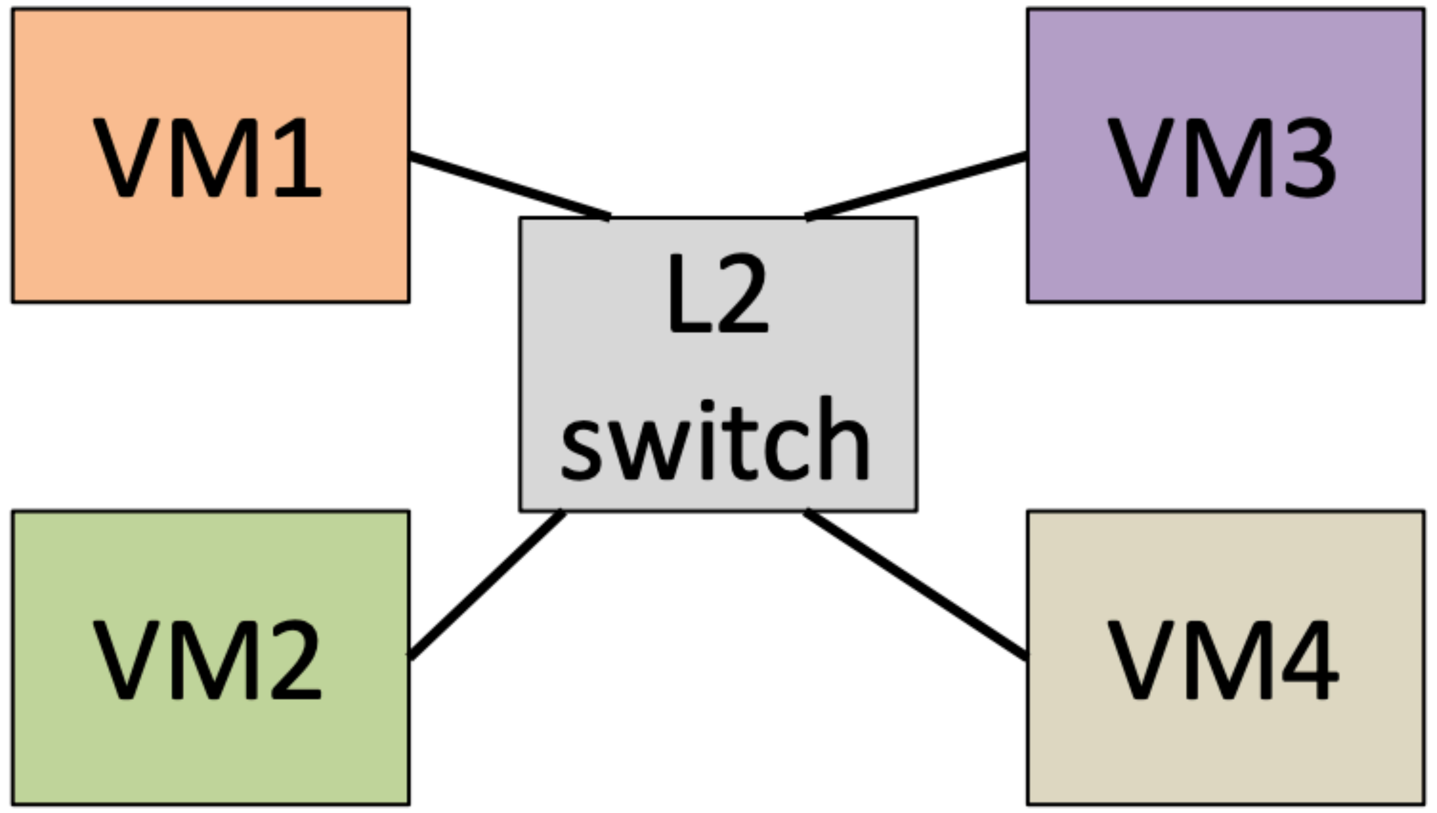
CS 234 / NetSys 210: Advanced Computer Networks

Sangeetha Abdu Jyothi

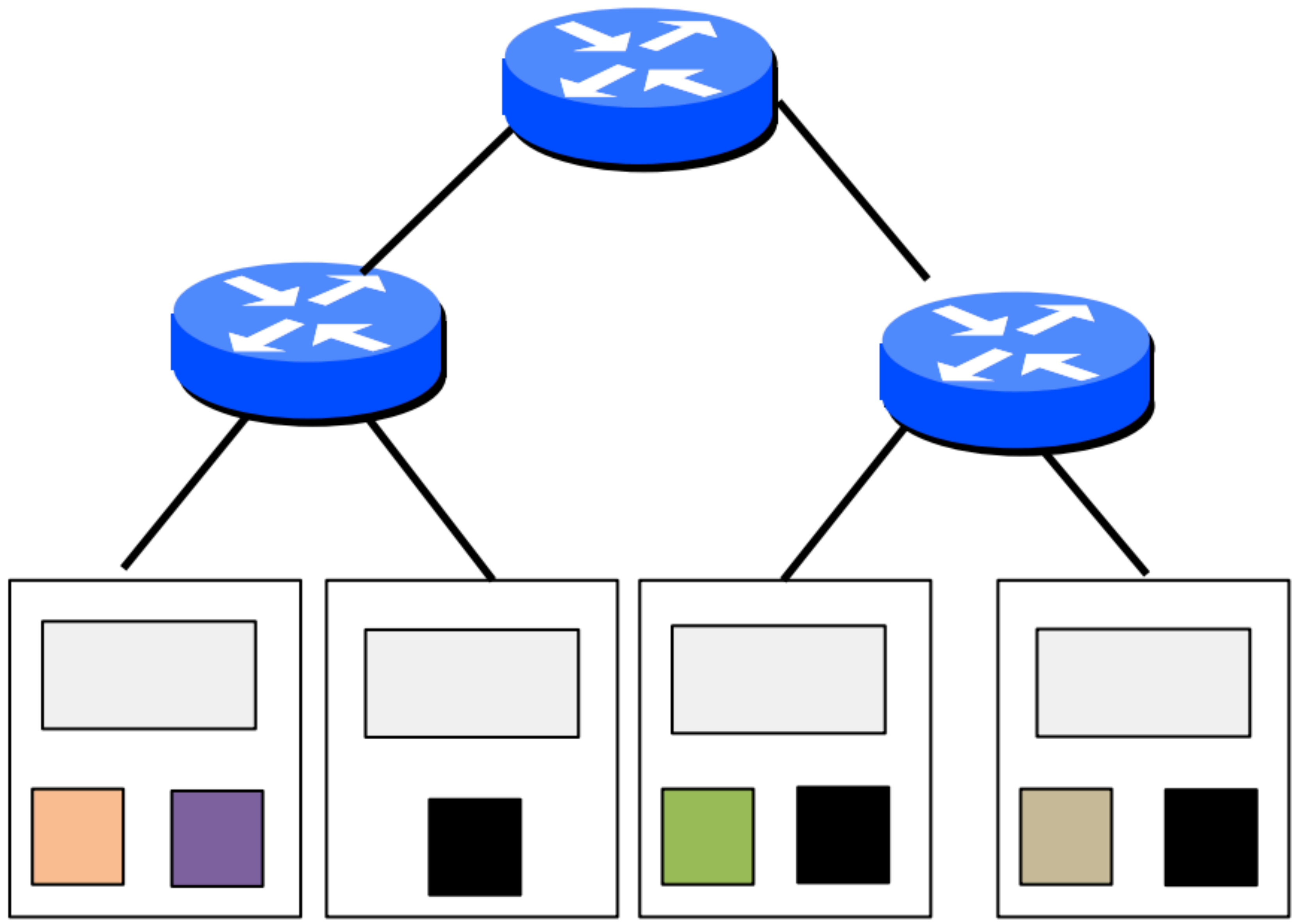
UCIrvine

This lecture uses material from Radhika Mittal (ECE/CS598HPN) and Nick McKeown

Last Class: Network Virtualization

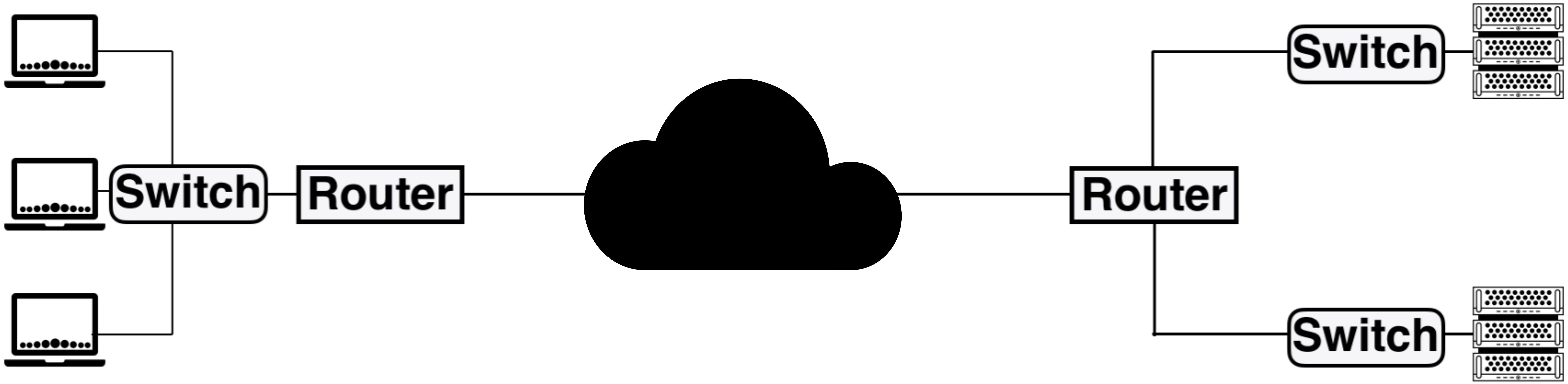


Abstraction



Physical Topology

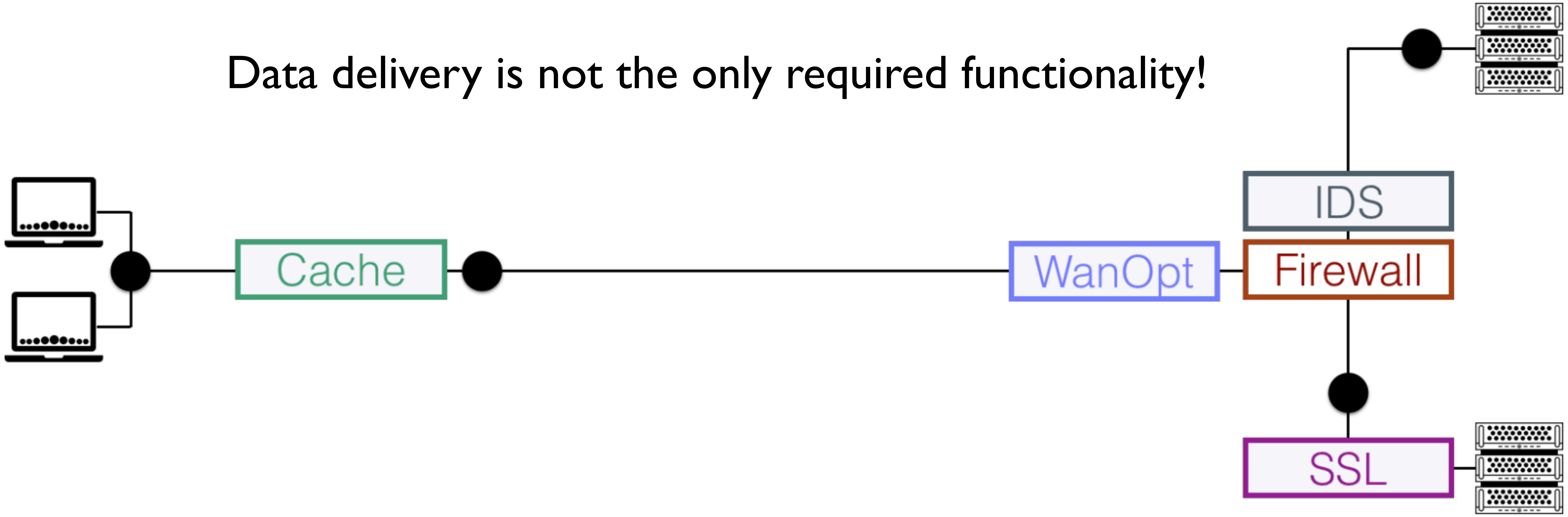
Conventional View of Networks



Data delivery is the only functionality provided by such a network

Rise of Middleboxes

Data delivery is not the only required functionality!



Security (IDS, Firewall): identify and block unwanted traffic

Performance (Cache) : Load content faster

Performance (WanOpt): reduce bandwidth usage

Application support (SSL): protocol for legacy application.

Middlebox Prevalence

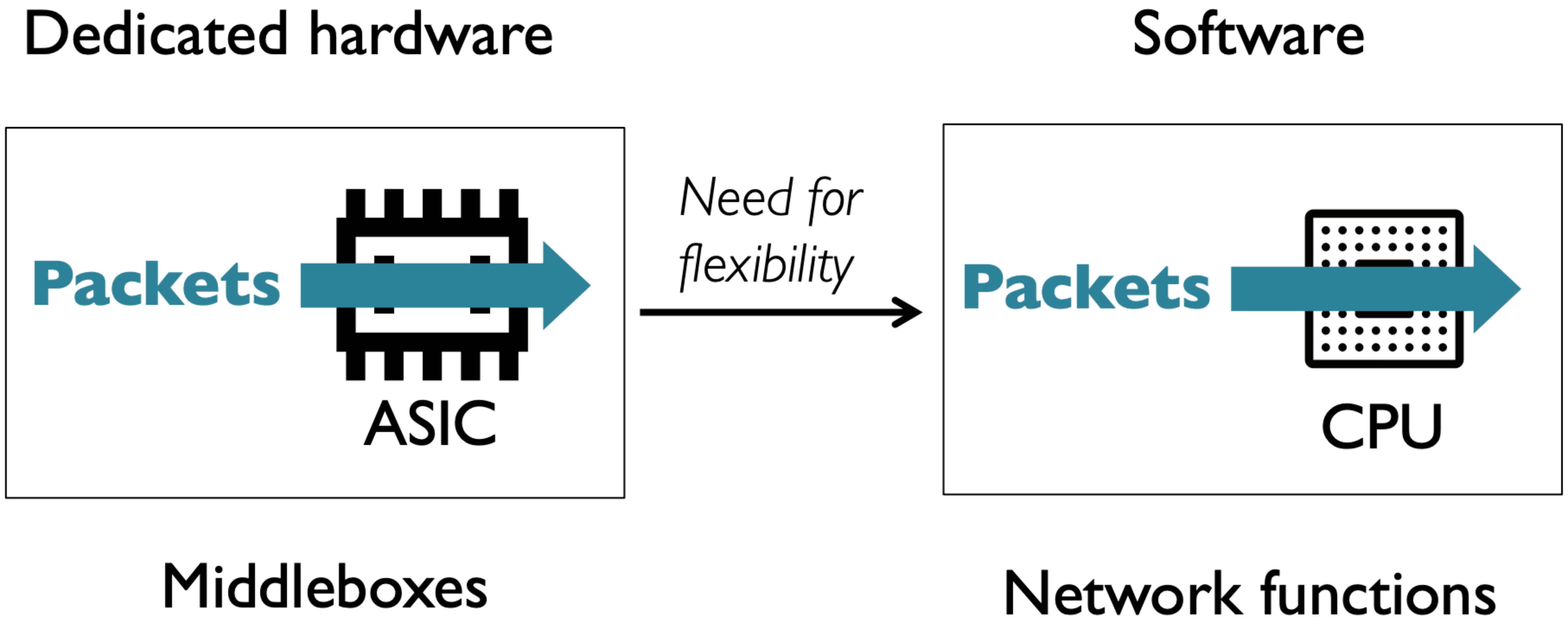
One-third of all network devices in enterprises are middleboxes!

[Making middleboxes someone else's problem, SIGCOMM’12]

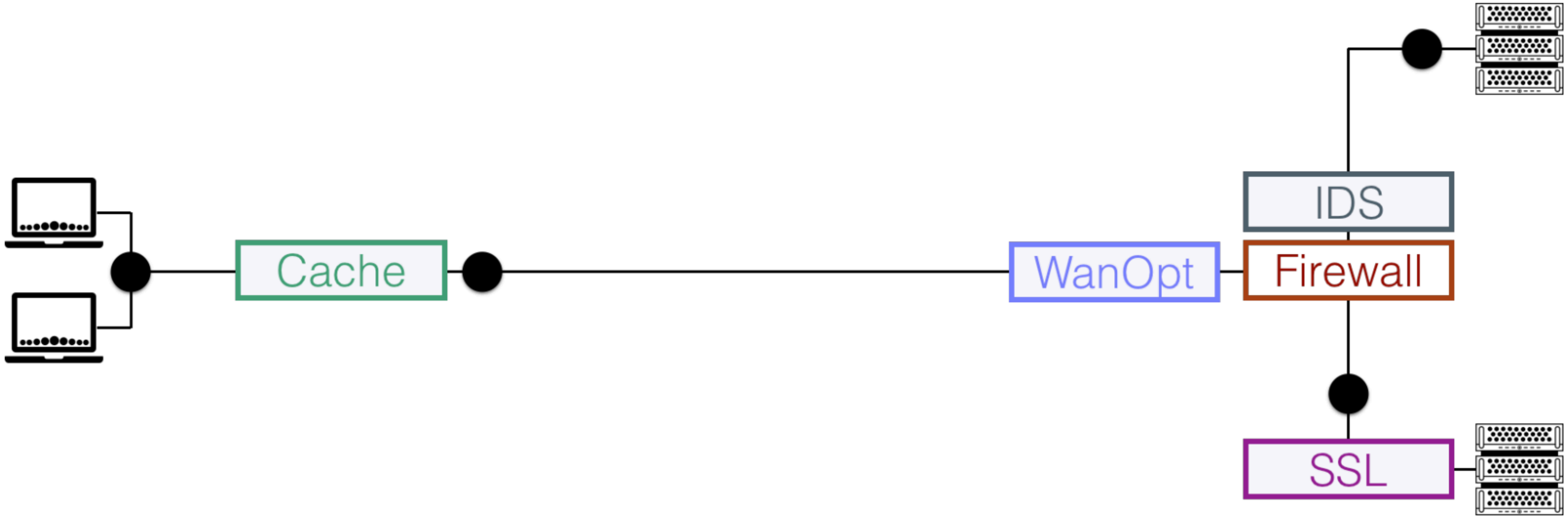
Problems with Hardware Middleboxes

- Dedicated
- Fixed function with little/no programmability
- Specialized hardware/software
- Custom Management APIs

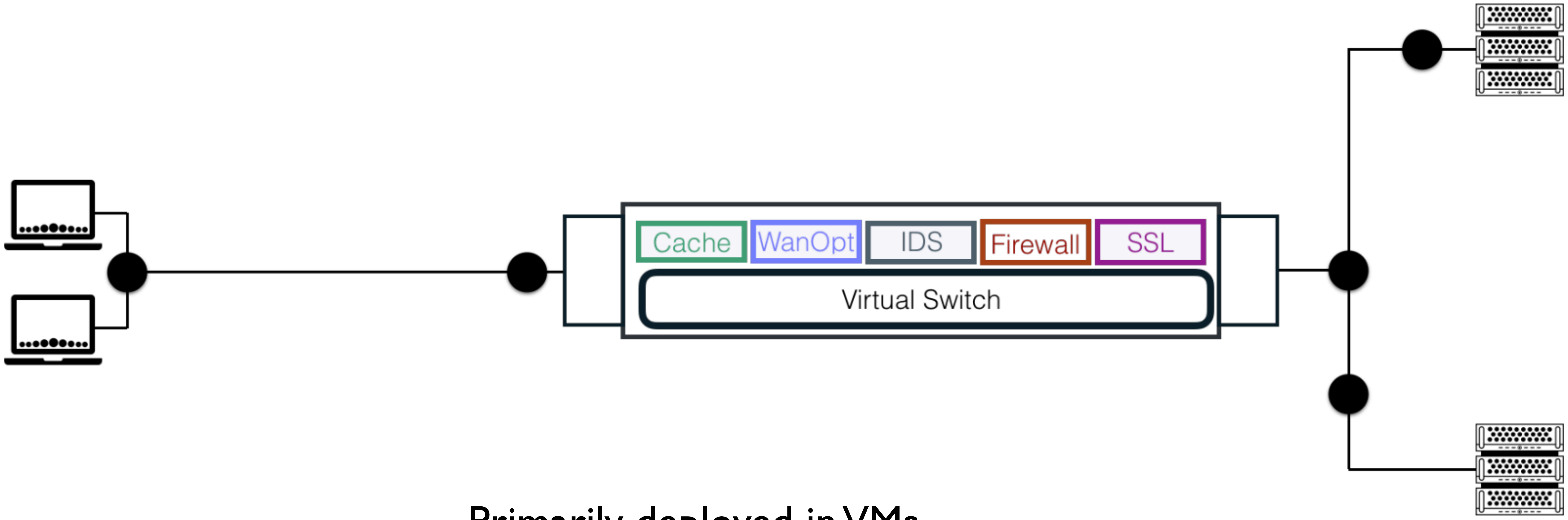
Evolution of Middleboxes: Network Function Virtualization



From Hardware Middleboxes….

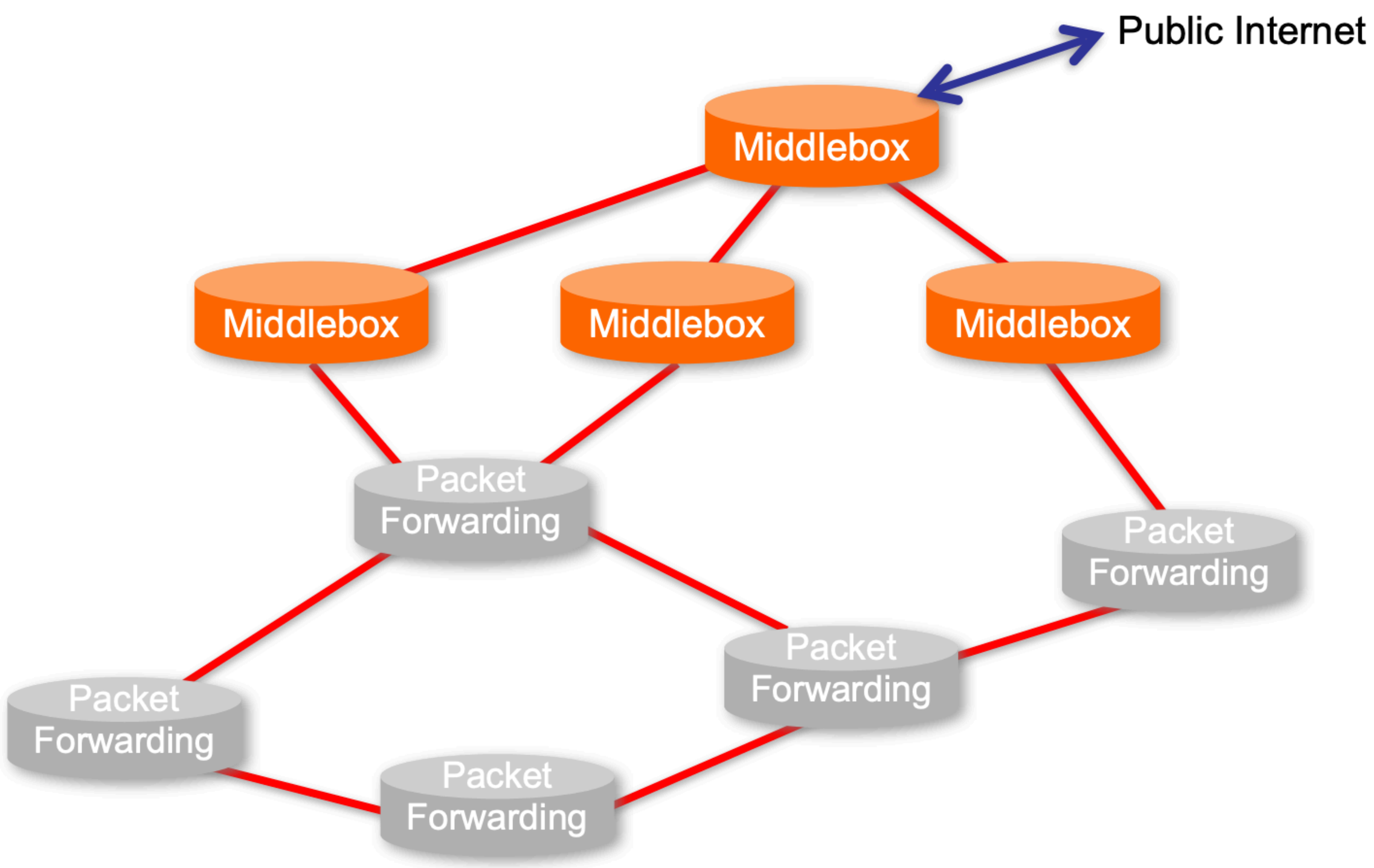


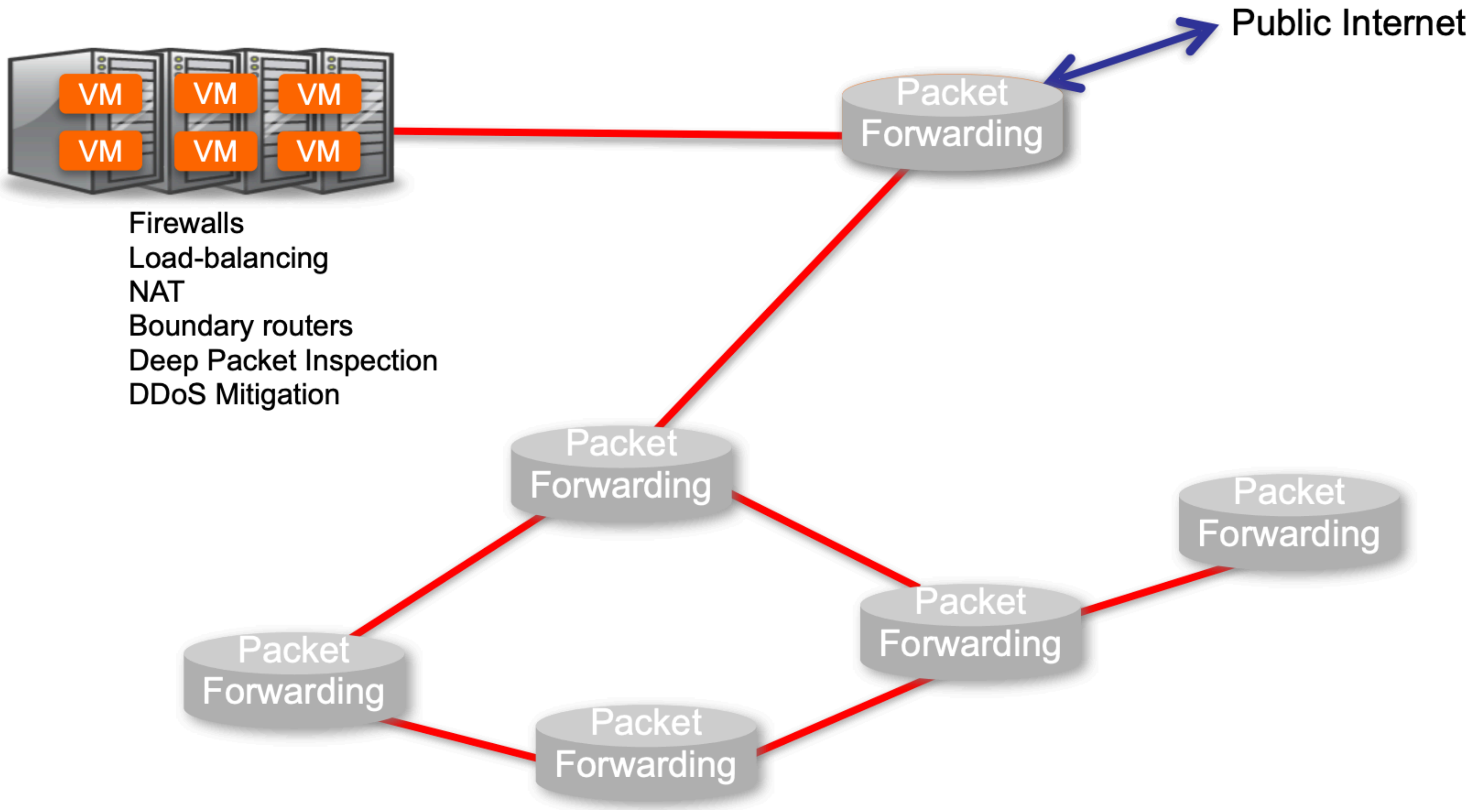
… to software Network Functions (NFs)



Primarily deployed in VMs

Network View

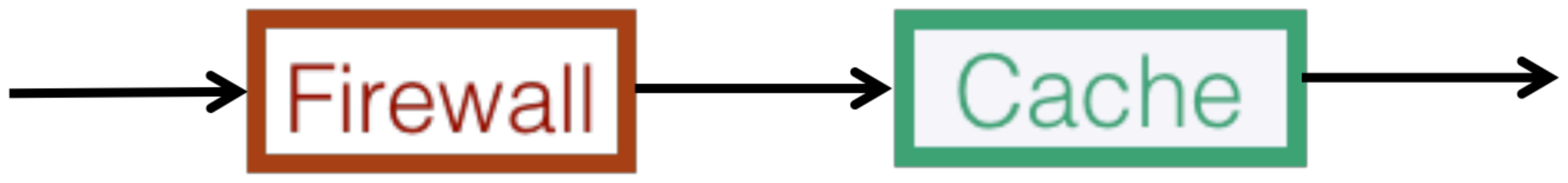




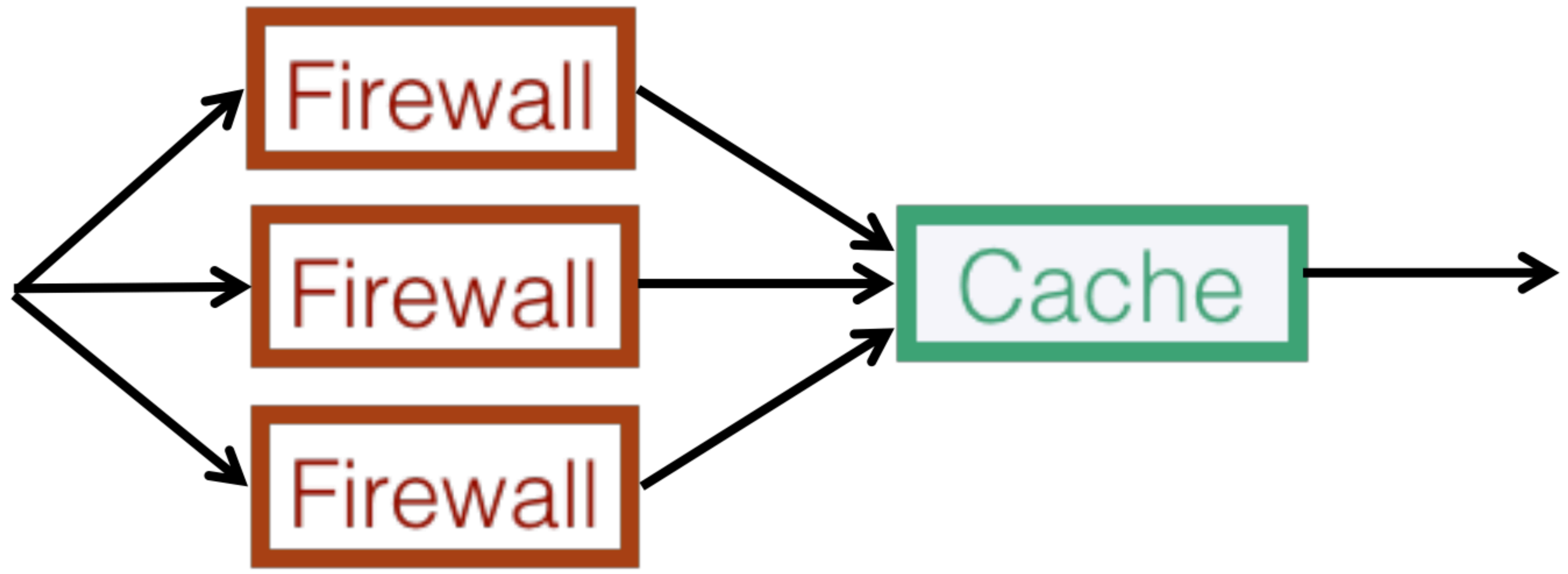
Firewalls
Load-balancing
NAT
Boundary routers
Deep Packet Inspection
DDoS Mitigation

Key Benefits of Software Network Functions

- Programmability
  - ability to update and create new NFs
- Cost benefits of commodity solutions
- Efficiency of statistical multiplexing
- Ease of deployment, configuration, and management



NF Service Chain



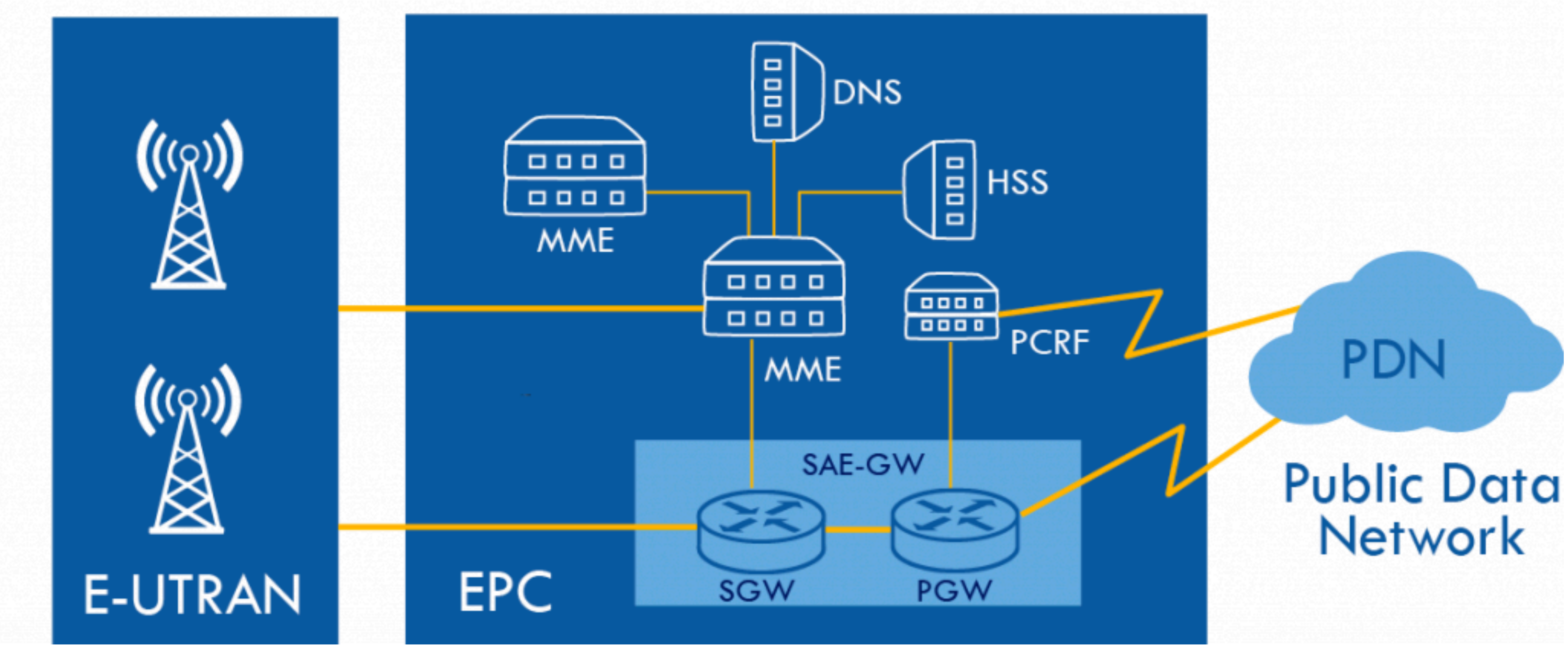
Challenges with Network Function Virtualization

- Complex and costly state management
- Custom per-app management APIs
- Unpredictable performance
- Performance degradation

E2: A Framework for NFV Applications

- End-to-end management of Network Functions
- Provide general solutions for common tasks
- Benefits
    - Frees NF developers to focus on NF-specific logic
    - Automates/consolidates management for operators

Cellular Backend

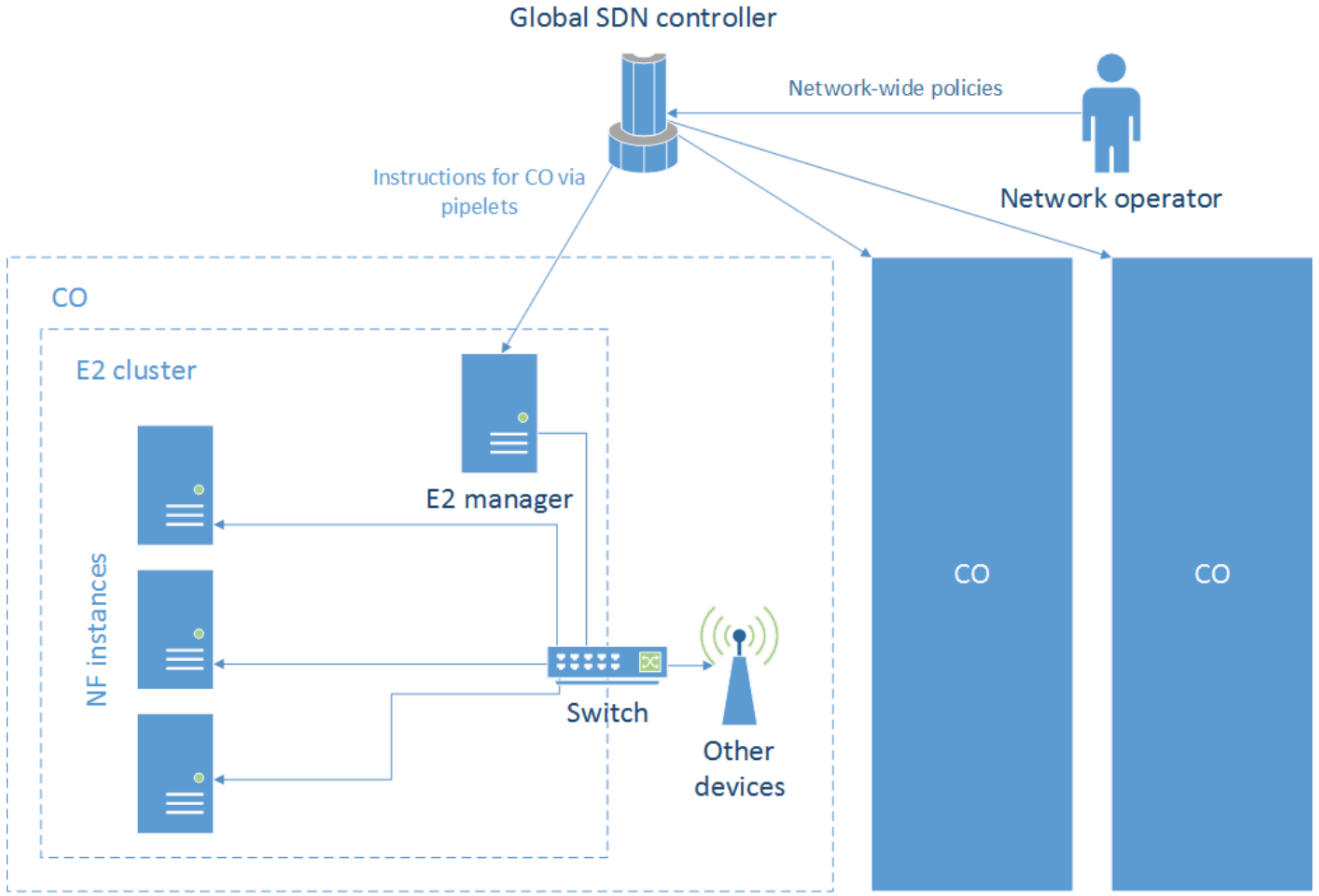


NF Placement Options

- Thread-Based
    - Lightweight
    - No resource isolation
- Virtual Machine-Based
    - Additional overheads
    - Resource Isolation

E2 is VM-based

Design Overview



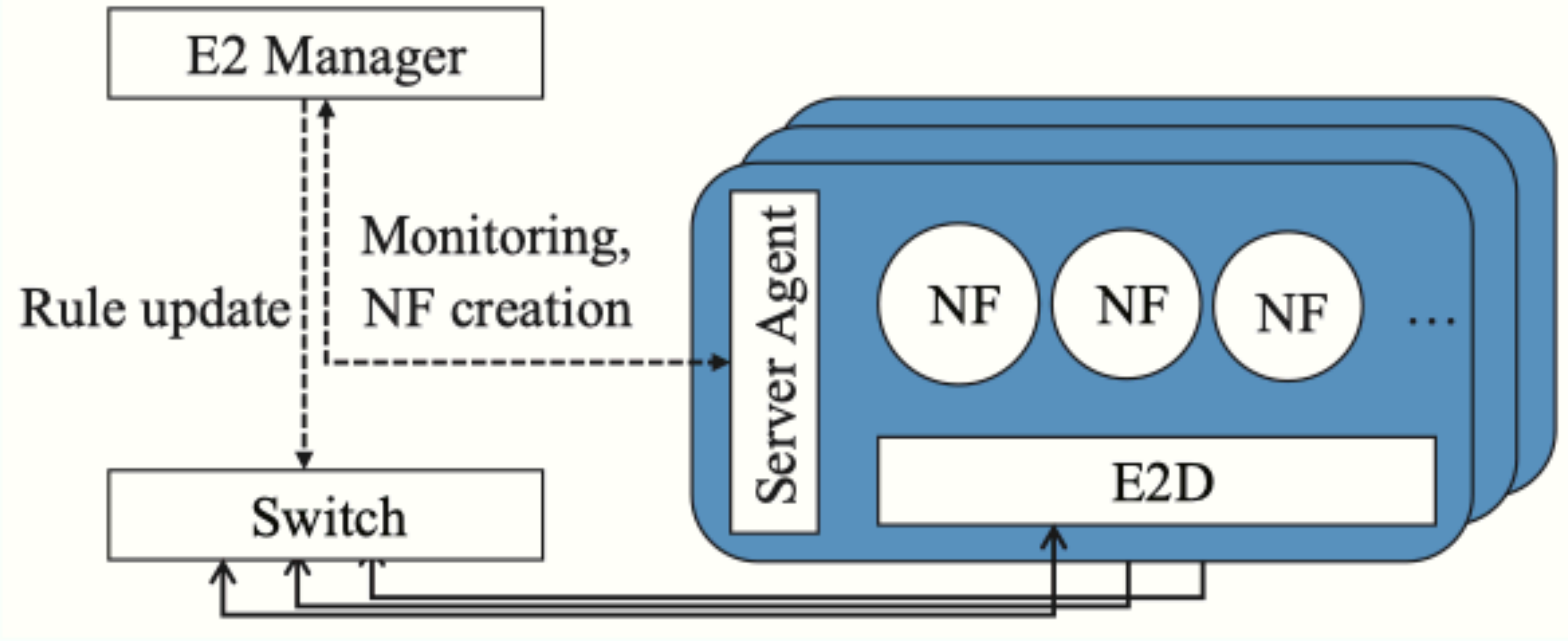
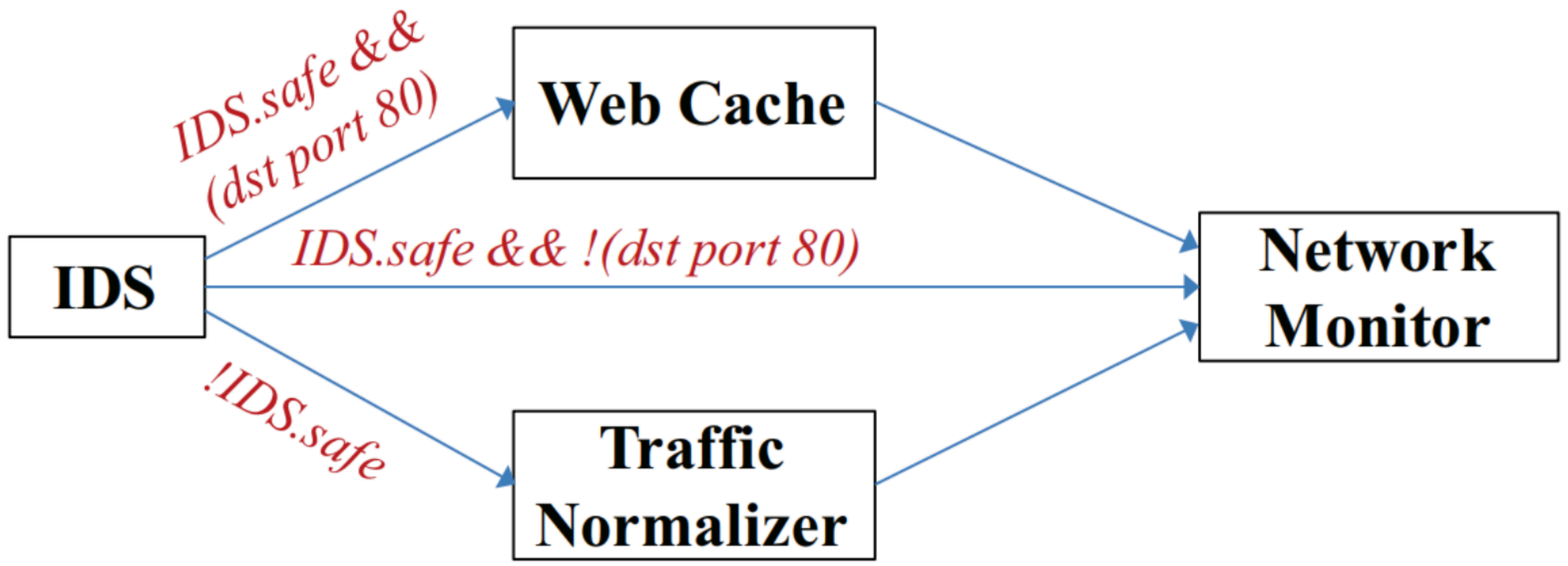


Figure 3: The overall E2 system architecture.

Pipelets

- NFV jobs represented as ‘pipelets’
    - a traffic class and a DAG that captures how this traffic class should be processed by NFs



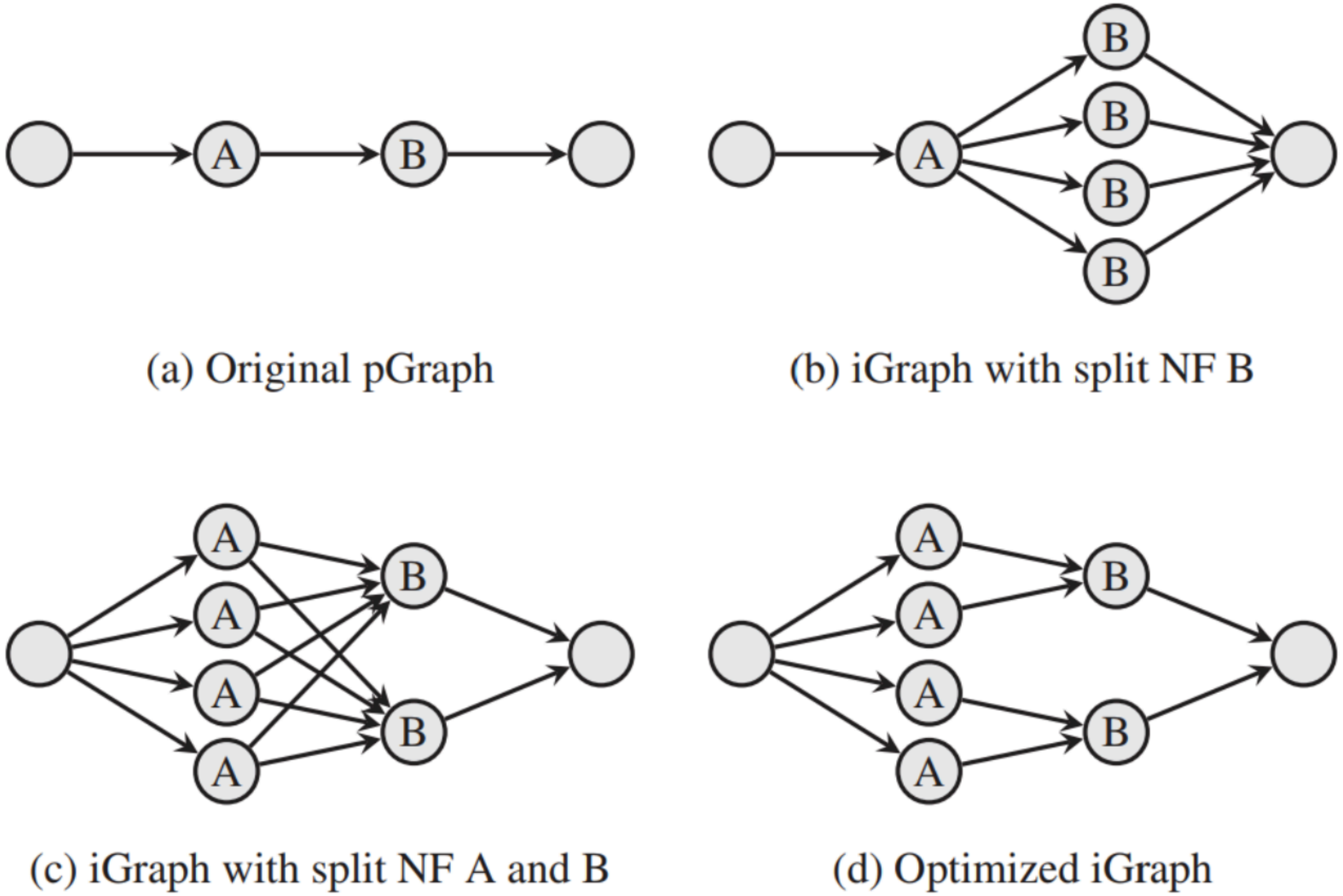
E2 Dataplane

- Modular architecture based on SoftNIC
- Highly efficient (uses Intel DPDK)
- Why OVS is not suitable?
    - expressiveness and functionality are limited by the flow-table semantics
    - performance optimizations that improve the efficiency of NFs more important in this context

E2 Control Plane

- Executing Pipelets
  - Sizing: How many NF instances?
  - Placement: Where to place NF instances?
  - Composition: How to steer traffic between NFs?
  - Dynamic scaling: Adapting to traffic changes
  - Ensuring affinity constraints of NFs

NF placement example



(a) Original pGraph

(b) iGraph with split NF B

(c) iGraph with split NF A and B

(d) Optimized iGraph

Figure 4: Transformations of a pGraph (a) into an iGraph (b, c, d).

Comments from students

- Move E2 to container-based implementation - multiple students
- Single point of failure - multiple students
- "There are certain hardware constraints that E2 takes into account. More work is needed to figure out how to exploit richer resources like CPU cache, GPUs, programmable switches, specialized accelerators, etc." - Rakshit Mehra
- "The paper does not address consistency issues that arise when global or aggregate state is spread across multiple NF instances, which could be a significant challenge for managing cross-NF state in a dynamic scaling scenario." - Sagar Krishna
- "Future work on the E2 framework should focus on exploring fault-tolerance and energy-efficient management and monitoring to enhance its robustness and sustainability in real-world deployments." - Yurun Song

High Performance NF Implementations

Microboxes: High Performance NFV with Customizable, Asynchronous TCP Stacks and Dynamic Subscriptions

Guyue Liu[*], Yuxin Ren[*], Mykola Yurchenko[*],
K.K. Ramakrishnan[†], Timothy Wood[*]
[*]George Washington University, [†]University of California, Riverside

mOS: A Reusable Networking Stack for Flow Monitoring Middleboxes

Muhammad Jamshed, YoungGyoun Moon, Donghwi Kim, Dongsu Han, and KyoungSoo Park
School of Electrical Engineering, KAIST

Abstr

FlowBlaze: Stateful Packet Processing in Hardware

Salvatore Pontarelli[1,2], Roberto Bifulco[3], Marco Bonola[1,2], Carmelo Cascone[4],
Marco Spaziani[2,5], Valerio Bruschi[2,5], Davide Sanvito[6], Giuseppe Siracusano[3],
Antonio Capone[6], Michio Honda[3], Felipe Huici[3] and Giuseppe Bianchi[2,5]

[1]Axbryd, [2]CNIT, [3]NEC Laboratories Europe, [4]Open Networking Foundation,
[5]University of Rome Tor Vergata, [6]Politecnico di Milano

Abstract

While programmat
handle growing net
yet simple abstracti
in hardware remain
problem with Flow
stateful packet pro
straction is based o
troduces the explici
Blaze to leverage f
pressive, supporting
tions, and easy to u
tation issues from t
FlowBlaze on a Ne
tency (in the order
tively little power,
thousands of flows,
for even higher spe
ware and software
licly available.

1 Introductio

Network infrastruct
network functions t
and server load bala
such as access con
examples. Given
the need to contin

NetBricks: Taking the V out of NFV

Aurojit Panda[†] Sangjin Han[†] Keon Jang[‡] Melvin Walls[†] Sylvia Ratnasamy[†] Scott Shenker[†*]
[†] UC Berkeley [‡] Google [*] ICSI

Abstract

The move from hardware middleboxes to software network functions, as advocated by NFV, has proven more challenging than expected. Developing new NFs remains a tedious process, requiring that developers repeatedly rediscover and reapply the same set of optimizations, while current techniques for providing isolation between NFs (using VMs or containers) incur high performance overheads. In this paper we describe NetBricks, a new NFV framework that tackles both these problems. For building NFs we take inspiration from modern data analytics frameworks (*e.g.,* Spark and Dryad) and build a small set of customizable network processing elements. We also embrace type checking and safe runtimes to provide isolation in software, rather than rely on hardware isolation. NetBricks provides the same memory isolation as containers and VMs, without

standard tools for managing VMs; (c) faster development, which now requires writing software that runs on commodity hardware; and (d) reduced costs by consolidating several NFs on a single machine. However, despite these promised advances, there has been little progress towards large-scale NF deployments. Our discussions with three major carriers revealed that they are only just beginning small scale test deployments (with 10-100s of customers) using simple NFs *e.g.,* firewalls and NATs.

The move from hardware middleboxes to software NFs was supposed to speed innovation, so why has progress been so slow? We believe this delay is because traditional approaches for both *building* and *running* NFs are a poor match for carrier networks, which have the following requirements: *performance*, NF deployments should be able to provide per-packet latencies on the order of 10s of μs, and throughput on the order of 10s of Gbps; *efficiency*,

ClickNP: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware

Bojie Li[§†] Kun Tan[†] Layong (Larry) Luo[‡] Yanqing Peng[•†] Renqian Luo[§†]
Ningyi Xu[†] Yongqiang Xiong[†] Peng Cheng[†] Enhong Chen[§]
[†]Microsoft Research [§]USTC [‡]Microsoft [•]SJTU

RACT

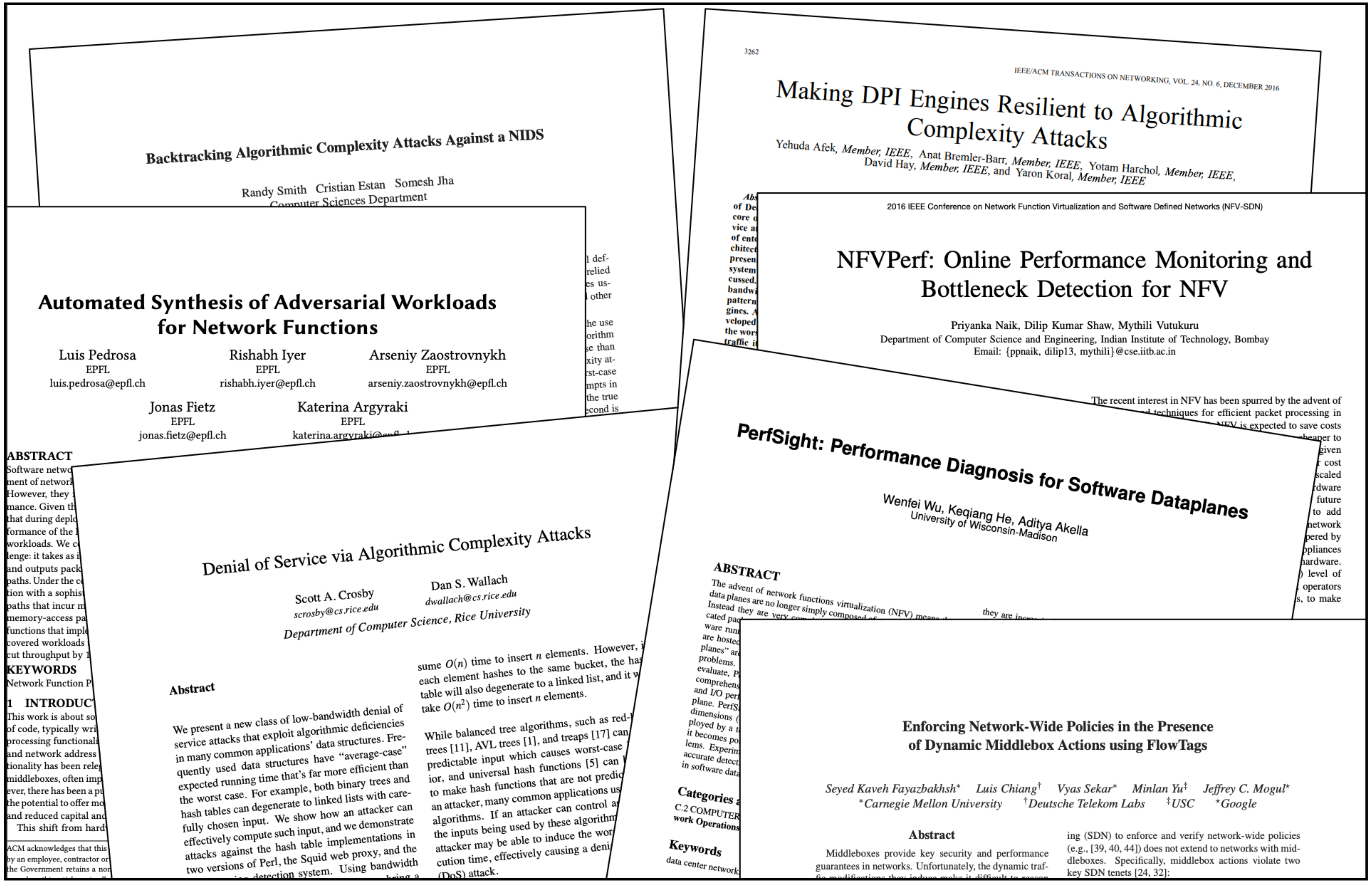
lexible software network functions (NFs) are cru-
ponents to enable multi-tenancy in the clouds. How-
tware packet processing on a commodity server has
capacity and induces high latency. While software
ld scale out using more servers, doing so adds sig-
cost. This paper focuses on accelerating NFs with
mable hardware, *i.e.*, FPGA, which is now a ma-
nology and inexpensive for datacenters. However,
predominately programmed using low-level hard-
cription languages (HDLs), which are hard to code
cult to debug. More importantly, HDLs are almost
ible for most software programmers. This paper presents
a FPGA-accelerated platform for highly flexible
-performance NFs with commodity servers. ClickNP
flexible as it is completely programmable using
l C-like languages, and exposes a modular program-
traction that resembles Click Modular Router. ClickNP
gh performance. Our prototype NFs show that they
ess traffic at up to 200 million packets per second
a-low latency (< 2μs). Compared to existing soft-
nterparts, with FPGA, ClickNP improves through-
0x, while reducing latency by 10x. To the best of
vledge, ClickNP is the first FPGA-accelerated plat-
NFs, written completely in high-level language and
g 40 Gbps line rate at any packet size.

1. INTRODUCTION

Modern multi-tenant datacenters provide shared infrastructure for hosting many different types of services from different customers (*i.e.*, tenants) at a low cost. To ensure security and performance isolation, each tenant is deployed in a *virtualized network* environment. Flexible network functions (NFs) are required for datacenter operators to enforce isolation while simultaneously guaranteeing Service Level Agreements (SLAs).

Conventional hardware-based network appliances are not flexible, and almost all existing cloud providers, *e.g.*, Microsoft, Amazon and VMWare, have been deploying software-based NFs on servers to maximize the flexibility [23, 30]. However, software NFs have two fundamental limitations – both stem from the nature of software packet processing. First, processing packets in software has limited capacity. Existing software NFs usually require multiple cores to achieve 10 Gbps rate [33, 43]. But the latest network links have scaled up to 40∼100 Gbps [11]. Although one could add more cores in a server, doing so adds significant cost, not only in terms of capital expense, but also more operational expense as they are burning significantly more energy. Second, processing packets in software incurs large, and highly variable latency. This latency may range from tens of microsecond to milliseconds [22,33,39]. For many low latency applications (*e.g.*, stock trading), this inflated latency is un-

Understanding NF Performance



Backtracking Algorithmic Complexity Attacks Against a NIDS

Randy Smith Cristian Estan Somesh Jha
Computer Sciences Department

Automated Synthesis of Adversarial Workloads for Network Functions

Luis Pedrosa, EPFL, luis.pedrosa@epfl.ch
Rishabh Iyer, EPFL, rishabh.iyer@epfl.ch
Arseniy Zaostrovnykh, EPFL, arseniy.zaostrovnykh@epfl.ch
Jonas Fietz, EPFL, jonas.fietz@epfl.ch
Katerina Argyraki, EPFL

ABSTRACT

KEYWORDS

1 INTRODUCTION

Denial of Service via Algorithmic Complexity Attacks

Scott A. Crosby, scrosby@cs.rice.edu
Dan S. Wallach, dwallach@cs.rice.edu
Department of Computer Science, Rice University

Abstract

We present a new class of low-bandwidth denial of service attacks that exploit algorithmic deficiencies in many common applications' data structures. Frequently used data structures have "average-case" expected running time that's far more efficient than the worst case. For example, both binary trees and hash tables can degenerate to linked lists with carefully chosen input. We show how an attacker can effectively compute such input, and we demonstrate attacks against the hash table implementations in two versions of Perl, the Squid web proxy, and the

sume $O(n)$ time to insert n elements. However, i each element hashes to the same bucket, the has table will also degenerate to a linked list, and it w take $O(n^2)$ time to insert n elements.

While balanced tree algorithms, such as red- trees [11], AVL trees [1], and treaps [17] can predictable input which causes worst-case ior, and universal hash functions [5] can to make hash functions that are not predic an attacker, many common applications us algorithms. If an attacker can control a the inputs being used by these algorithm attacker may be able to induce the wor cution time, effectively causing a deni (DoS) attack.

3262 IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 24, NO. 6, DECEMBER 2016

Making DPI Engines Resilient to Algorithmic Complexity Attacks

Yehuda Afek, *Member, IEEE*, Anat Bremler-Barr, *Member, IEEE*, Yotam Harchol, *Member, IEEE*, David Hay, *Member, IEEE*, and Yaron Koral, *Member, IEEE*

2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)

NFVPerf: Online Performance Monitoring and Bottleneck Detection for NFV

Priyanka Naik, Dilip Kumar Shaw, Mythili Vutukuru
Department of Computer Science and Engineering, Indian Institute of Technology, Bombay
Email: {ppnaik, dilip13, mythili}@cse.iitb.ac.in

The recent interest in NFV has been spurred by the advent of techniques for efficient packet processing in

PerfSight: Performance Diagnosis for Software Dataplanes

Wenfei Wu, Keqiang He, Aditya Akella
University of Wisconsin-Madison

ABSTRACT

The advent of network functions virtualization (NFV) means data planes are no longer simply composed of

Categories

Keywords

Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags

*Seyed Kaveh Fayazbakhsh** *Luis Chiang*† *Vyas Sekar** *Minlan Yu*‡ *Jeffrey C. Mogul**
**Carnegie Mellon University* †*Deutsche Telekom Labs* ‡*USC* **Google*

Abstract

Middleboxes provide key security and performance guarantees in networks. Unfortunately, the dynamic traf-

ing (SDN) to enforce and verify network-wide policies (e.g., [39, 40, 44]) does not extend to networks with middleboxes. Specifically, middlebox actions violate two key SDN tenets [24, 32]:

Providing Guarantees about NF Behavior

A Formally Verified NAT

Arseniy Zaostrovnykh
EPFL, Switzerland
arseniy.zaostrovnykh@epfl.ch

Solal Pirelli
EPFL, Switzerland
solal.pirelli@epfl.ch

Luis Pedrosa
EPFL, Switzerland
luis.pedrosa@epfl.ch

Katerina Argyraki
EPFL, Switzerland
katerina.argyraki@epfl.ch

George Candea
EPFL, Switzerland
george.candea@epfl.ch

ABSTRACT

There exists a lot of prior work on network verification, but,
knowledge, none that reasons about both the
Most
their
hough
oning
5, 59]).
notion
-level
+) [35].
tateful
, even
ecution
at it is
out the

Software Dataplane Verification

Mihai Dobrescu and Katerina Argyraki
EPFL, Switzerland

Abstract

Software dataplanes are emerging as an alternative to traditional hardware switches and routers, promising programmability and short time to market. These advantages are set against the risk of disrupting the network with bugs, unpredictable performance, or security vulnerabilities. We explore the feasibility of verifying software dataplanes to ensure smooth network operation. For general programs, verifiability and performance are competing goals; we argue that software dataplanes are different—we can write them in a way that enables verification *and* preserves performance. We present a verification tool that takes as input a software dataplane, written in a way that meets a given set of conditions, and (dis)proves that the dataplane satisfies crash-freedom, bounded-execution, and filtering properties. We evaluate our tool on stateless and simple stateful Click pipelines; we perform complete and sound verification of these pipelines within tens of minutes, whereas a state-of-the-art general-purpose tool fails to complete the same task within several hours.

1 Introduction

Software dataplanes are emerging from both research [17,26,27,37] and industry [2,3] backgrounds as a more flexible alternative to traditional hardware switches and routers. They promise to cut network provisioning costs by half, by enabling dynamic allocation of packet-processing tasks to network devices [42]; or to turn the Internet into an evolvable architecture, by enabling continuous functionality update of devices located at strate-

The subject of this work is a verification tool that takes as input the executable binary of a software dataplane and proves that it does (or does not) satisfy a target property; if the target property is not satisfied, the tool should provide counter-examples, i.e., packet sequences that cause the property to be violated. Developers of packet-processing apps could use such a tool to produce software with guarantees, e.g., that never seg-faults or kernel-panics, no matter what traffic it receives. Network operators could use the tool to verify that a new packet-processing app they are considering for deployment will not destabilize their network, e.g., it will not introduce more than some known fixed amount of per-packet latency. One might even envision markets for packet-processing apps—similar to today's smartphone/tablet app markets—where network operators would shop for new code to "drop" into their network devices. The operators of such markets would need a verification tool to certify that their apps will not disrupt their customers' networks.

For general programs, verifiability and performance are competing goals. Proving properties of real programs (unlike searching for bugs) remains an elusive goal for the systems community, at least for programs that consist of more than a few hundred lines of code and are written in a low-level language like C++. A high-level language like Haskell can guarantee certain properties (like the impossibility of buffer overflow) by construction, but typically at the cost of performance.

For software dataplanes, it does not have to be this way: we will argue that we can write them in a way that enables verification and preserves performance. The key question then is: what defines a "software dataplane" and how much more restricted is it than a "general program"?
do we need to restrict our dataplane program-

Verifying Reachability in Networks with Mutable Datapaths

Aurojit Panda* Ori Lahav† Katerina Argyraki‡ Mooly Sagiv◇ Scott Shenker*♠
*UC Berkeley †MPI-SWS ‡EPFL ◇TAU ♠ICSI

Abstract

Recent work has made great progress in verifying the forwarding correctness of networks [26–28, 35]. However, these approaches cannot be used to verify networks containing middleboxes, such as caches and firewalls, whose forwarding behavior depends on previously observed traffic. We explore how to verify reachability properties for networks that include such "mutable datapath" elements, both for the original network and in the presence of fai
The main challenge lies in handling large and
networks. We achieve scaling by d
aging the concept of slices
verification to only re
network. We
verify

boxes without changes in the physical infrastructure [13]. Given their complexity and prevalence, middleboxes are the cause of many network failures; for instance, 43% of a network provider's failure incidents i
ved middleboxes,
and between 4% and 15%
idents were the
result of middlebox
Our goal
exten
gurations by
that contain
ilding our
properties
verifying
pproach
d then
r two
rge
far
or

SymNet: scalable symbolic execution for modern networks

Radu Stoenescu, Matei Popovici, Lorina Negreanu, Costin Raiciu
University Politehnica of Bucharest
Splaiul Independentei 313, Bucharest, Romania
firstname.lastname@cs.pub.ro

Abstract

We present SymNet, a network static analysis tool based on symbolic execution. SymNet injects symbolic packets and tracks their evolution through the network. Our key novelty is SEFL, a language we designed for expressing data plane processing in a symbolic-execution friendly manner.

SymNet statically analyzes an abstract data plane model that consists of the SEFL code for every node and the links between nodes. SymNet can check networks containing routers with hundreds of thousands of prefixes and NATs in seconds, while verifying packet header memory-safety and covering network functionality such as dynamic tunneling, stateful processing and encryption. We used SymNet to debug middlebox interactions from the literature, to check properties of our department's network and the Stanford backbone.

Modeling network functionality is not easy. To aid users we have developed parsers that automatically generate SEFL models from router and switch tables, firewall configurations and arbitrary Click modular router configurations. The parsers rely on prebuilt models that are exact and fast to analyze. Finally, we have built an automated testing tool that combines symbolic execution and testing to check whether the model is an accurate representation of the real code.

•Software and its

or distributed firewall policy compliance is difficult before deploying the network configuration, and deployment can disrupt live traffic. Dynamic testing (packet generation and tracing) can only catch common issues (e.g. lack of connectivity) but does not scale to large networks.

Static analysis of network data planes allows cheap, fast and exhaustive verification of deployed networks for packet reachability, absence of loops, bidirectional forwarding, etc. We do not aim to verify the control plane (e.g. routing protocols, SDN controllers etc.). Control plane verification is a hard problem that includes checking the correctness of the control plane configuration [8, 9, 4], proving convergence after link additions or failures and characterizing the transient behavior until convergence is reached [2, 11]. We assume the control plane configuration is stable and the control plane has converged and analyze the resulting data plane.

All static analysis tools take as input a *model* of the processing performed by each network box, the links between boxes and a snapshot of the forwarding state, and are able to answer queries about the network without resorting to dynamic testing [23, 14, 19, 20, 21]. What is the best modeling language for networks? If possible, we should simply use the implementation of network boxes (e.g. a C program), as this is the most accurate and is easiest to use. If we view packets as variables being passed between different network boxes, static network analysis becomes akin to software testing. This is a problem that has been studied for decades, and the leading approach is to use *symbolic execution* [3].

Symbolic execution is really powerful: it explores all pos-
through the program, providing possible values
at every point. In the context of
symbolic execution lies

State Management

Split/Merge: System Support for Elastic Execution in Virtual Middleboxes

Shriram Rajagopalan[†‡], Dan Williams[†], Hani Jamjoom[†], and Andrew Warfield[‡]
[†]IBM T. J. Watson Research Center, Yorktown Heights, NY
[‡]University of British Columbia, Vancouver, Canada

Elastic Scaling of Stateful Network Functions

Shinae Woo[⋆†], Justine Sherry[‡], Sangjin Han[⋆], Sue Moon[†], Sylvia Ratnasamy[⋆], and Scott Shenker[⋆§]
[⋆]University of California, Berkeley [†]KAIST [‡]CMU [§]ICSI

Abstract

Elastic scaling is a central pro
hard to realize in practice. Th
most Network Functions (NF
need to be *shared* across N
state sharing while meeting
requirements placed on NFs
no solution exists that meets
for the full spectrum of NFs.
S6 is a new framework t
of NFs without compromis
builds on the insight that a
straction is well-suited to th
state as a distributed shar

Stateless Network Functions:
Breaking the Tight Coupling of State and Processing

Murad Kablan, Azzam Alsudais, Eric Keller
University of Colorado, Boulder

Franck Le
IBM Research

ewalls, intrusion detection syster
nslators, and load balancers no long
rietary hardware, but can run in so
ty servers, in a virtualized enviro
roughput [25]. This shift away fro
s should bring several benefits inclu
lastically scale the network functio
ickly recover from failures.
thers have reported, achieving thos
that simple [44, 45, 23, 40].

Rollback-Recovery for Middleboxes

Justine Sherry⋆ Peter Xiang Gao⋆ Soumya Basu⋆ Aurojit Panda⋆
Arvind Krishnamurthy• Christian Maciocco† Maziar Manesh† João Martins◁
Sylvia Ratnasamy⋆ Luigi Rizzo‡ Scott Shenker⋆⋄

⋆ UC Berkeley • University of Washington † Intel Research

ABSTRACT

OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions

Anat Bremler-Barr *
bremler@idc.ac.il

Yotam Harchol †
yotamhc@cs.huji.ac.il

David Hay†
dhay@cs.huji.ac.il

* School of Comput
† School of Computer Scie
The Interdisciplinary Center, Herzliya, Israel
The Hebrew University, Jerusalem, Israel

ABSTRACT

We present OpenBox — a software-
for network-wide development, dep
agement of *network functions* (NF
tively decouples the control plane of
plane, similarly to SDN solutions
network's forwarding plane.
OpenBox consists of three log
OpenBox applicatio

Pico Replication: A High Availability Framework for Middleboxes

Shriram Rajagopalan[†‡] Dan Williams[†] Hani Jamjoom[†]
[†]IBM T. J. Watson Research Center, Yorktown Heigh
[‡]University of British Columb

Abstract

Middleboxes are being rearchitect
ented, composable, extensible, and
level support for high availability (
troduce significant performance ove
we propose *Pico Replication (PR)*, a
work for middleboxes that exploits
structure to achieve low overhead,
HA. Unlike generic (virtual machine
PR operates at the

Paving the Way for NFV:
Simplifying Middlebox Modifications using StateAlyzr

Junaid Khalid, Aaron Gember-Jacobson, Roney Michael,
Anubhavnidhi Abhashkumar, Aditya Akella
University of Wisconsin-Madison

Abstract

central contribution of this paper is a novel, framework-

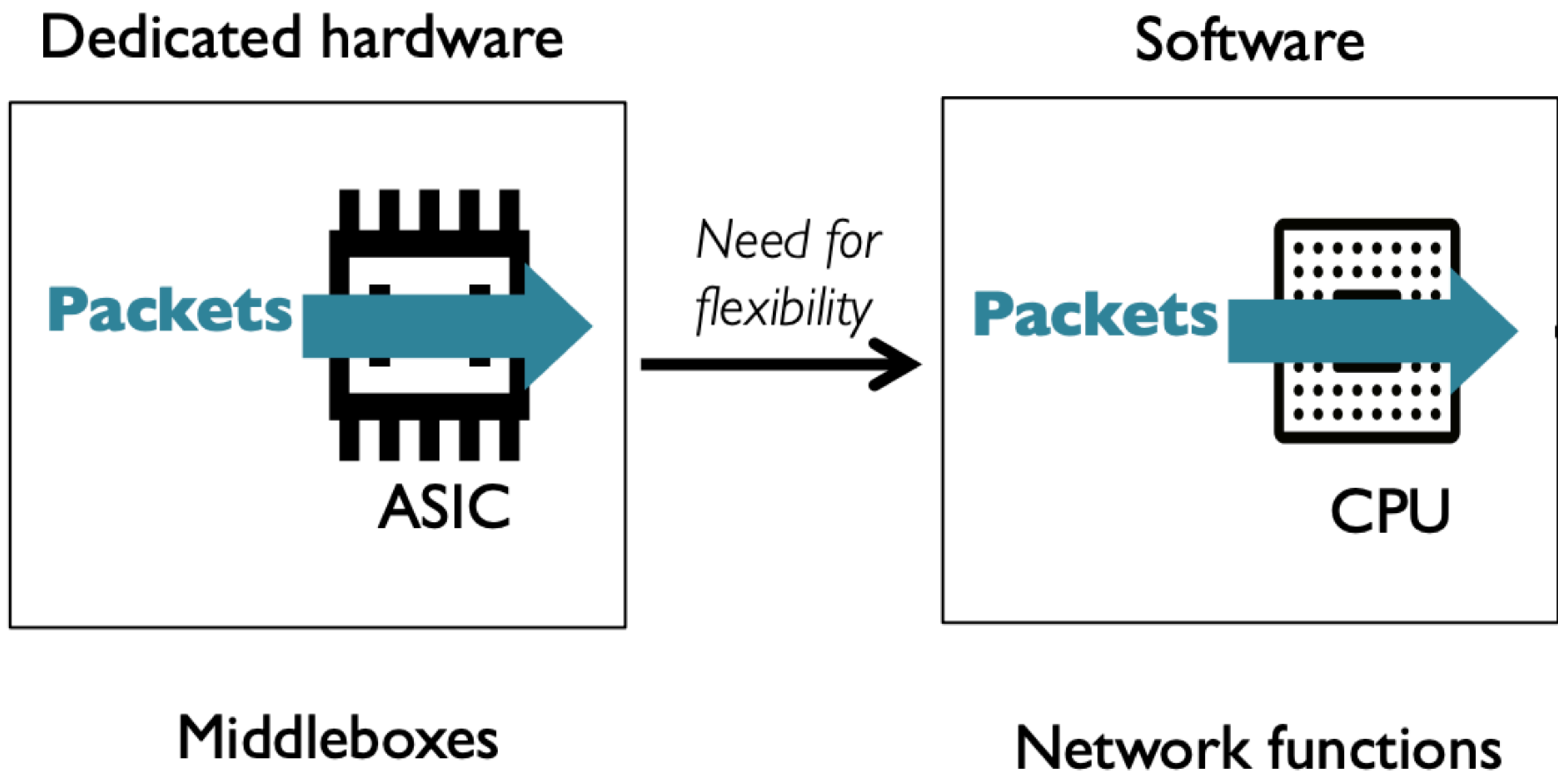
E2: A Framework for NFV Applications

Shoumik Palkar*	Chang Lan*	Sangjin Han
UC Berkeley sppalkar@berkeley.edu	UC Berkeley clan@eecs.berkeley.edu	UC Berkeley sangjin@eecs.berkeley.edu
Keon Jang	Aurojit Panda	Sylvia Ratnasamy
Intel Labs keon.jang@intel.com	UC Berkeley apanda@cs.berkeley.edu	UC Berkeley sylvia@eecs.berkeley.edu

Luigi Rizzo
Università di Pisa
rizzo@iet.unipi.it

Scott Shenker
UC Berkeley and ICSI
shenker@icsi.berkeley.edu

Evolution of Middleboxes



Thanks!