Network Functions Virtualization

SIGCOMM Topic Preview 2017

Justine Sherry

Carnegie Mellon University

Before there was NFV, there were middleboxes

"A middlebox is defined as any intermediary device performing functions other than the normal, standard functions of an IP router on the datagram path between a source host and destination host."

— B. Carpenter. RFC 3234. Middleboxes: Taxonomy and Issues

Example: Intrusion Prevention System

Detects anomalous or known-dangerous traffic and **blocks those connections.**

For each connection:

- Looks at port numbers, IP addresses and compares against blacklists.
- Reconstructs connection by stream and scans for malicious terms.
- Logs protocol, IP addresses, time of connection, etc.

Example: Web Proxy

Intercepts HTTP connections and **caches** frequently accessed content, may also **blacklist** certain content.

Maintains dual connections — one to client, one to server!

- If client requests content in cache, serve locally rather than sending request to server.
- If client requests blocked content, deny the request.

2010-2012: Middleboxes were problematic!

- Deployed in enterprises, ISPs and even data centers. Everyone used them.
- **But deploying them was a pain in the neck.**
    - Poor upgradeability: have to buy a new box every few years
    - Hard to install/configure
    - Fixed capacity — can't "scale on demand"
    - Static routes/policies

Around 2012, the networking community looked to cloud computing to improve how we deployed middleboxes.

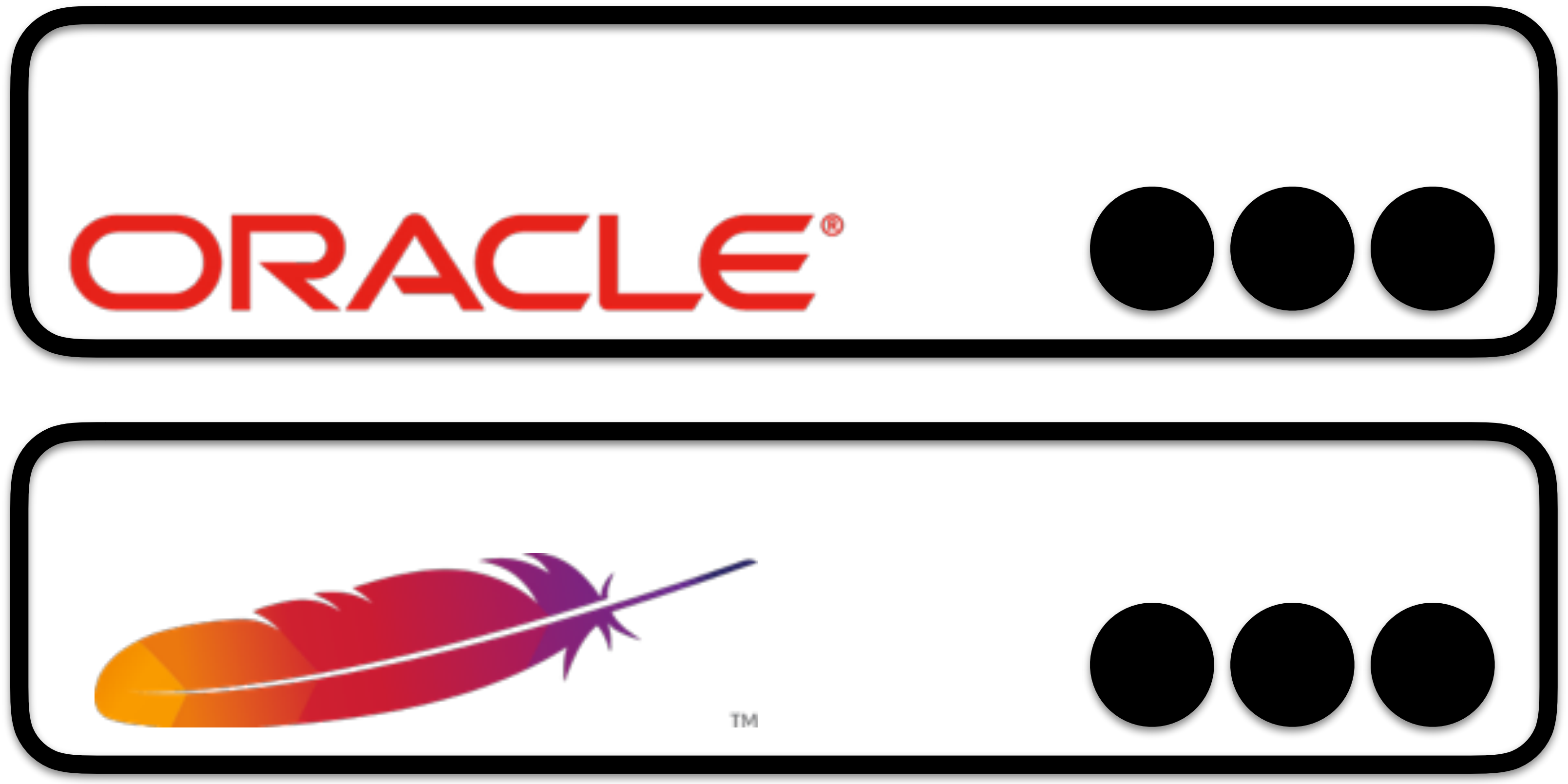
To understand how cloud computing helped middleboxes, let's imagine cloud computing, deployed the way middleboxes were.

Imagine cloud computing if it were deployed like middleboxes.

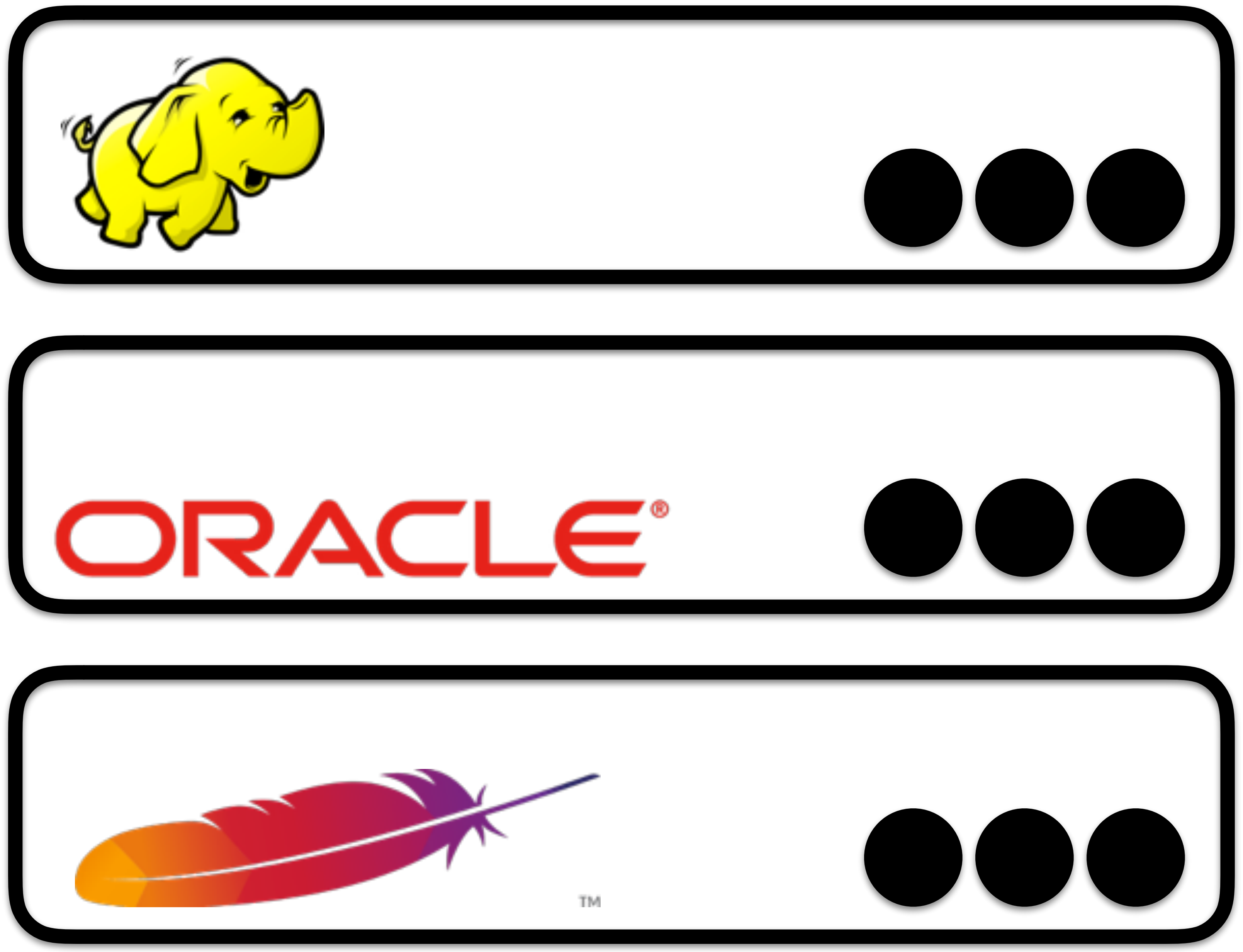
So you want to deploy a web service.

Imagine cloud computing if it were deployed like middleboxes.

So you want to deploy a web service.

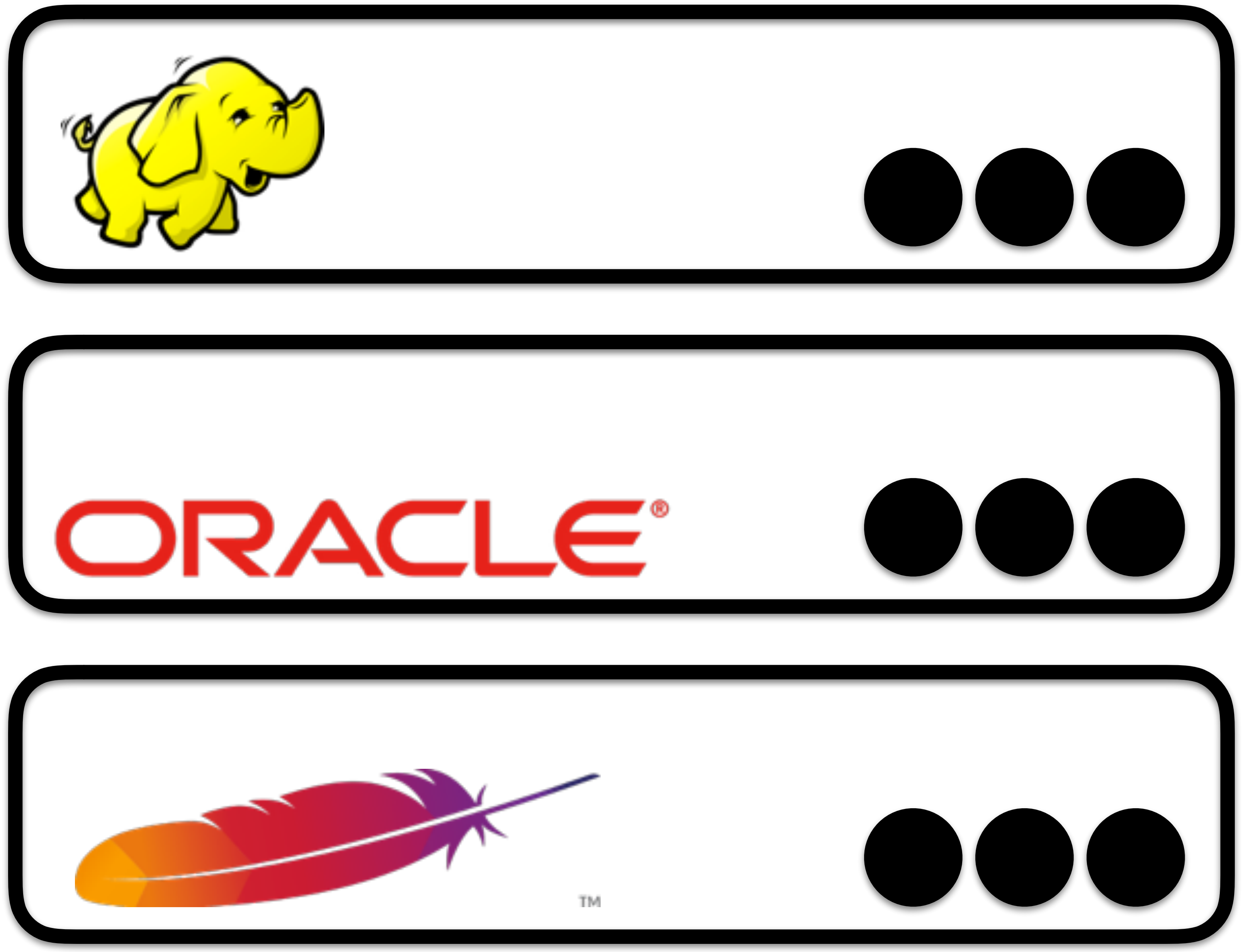


Imagine cloud computing if it were deployed like middleboxes.

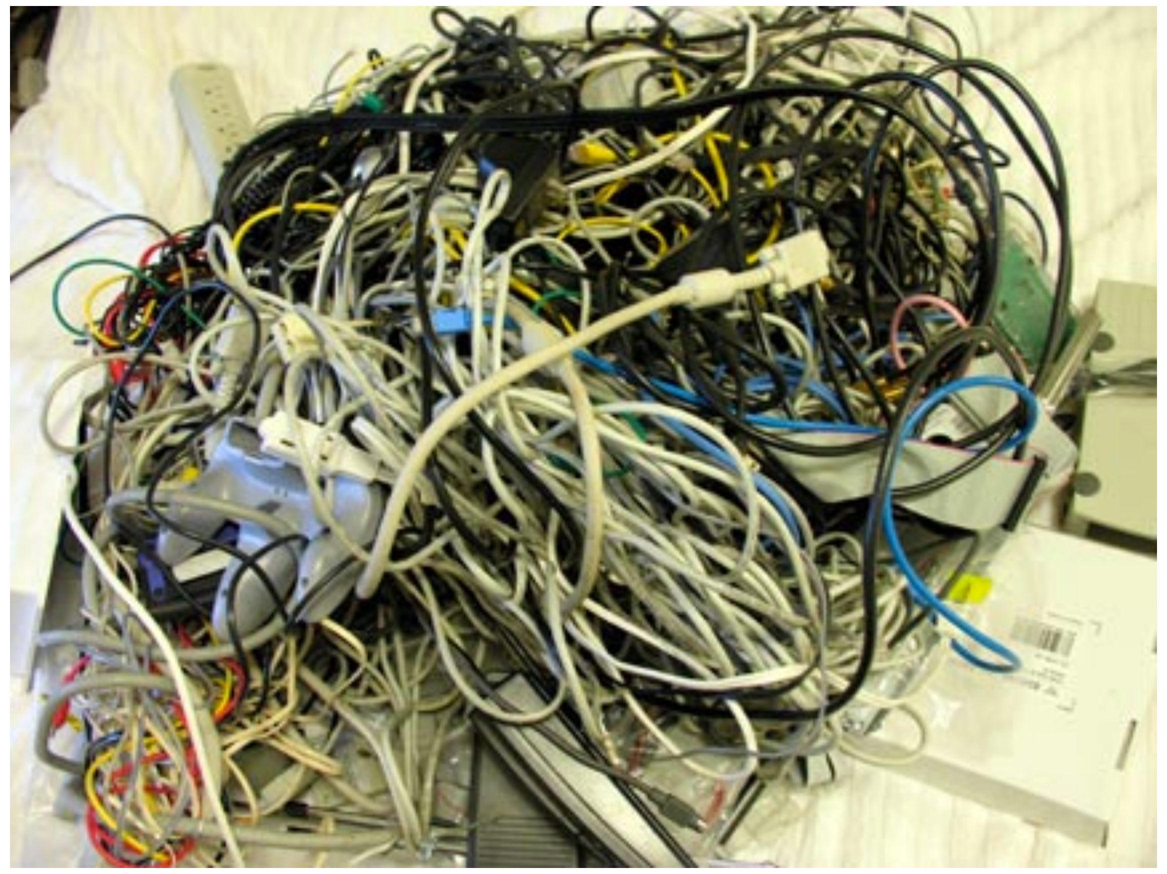


So you want to deploy a web service.

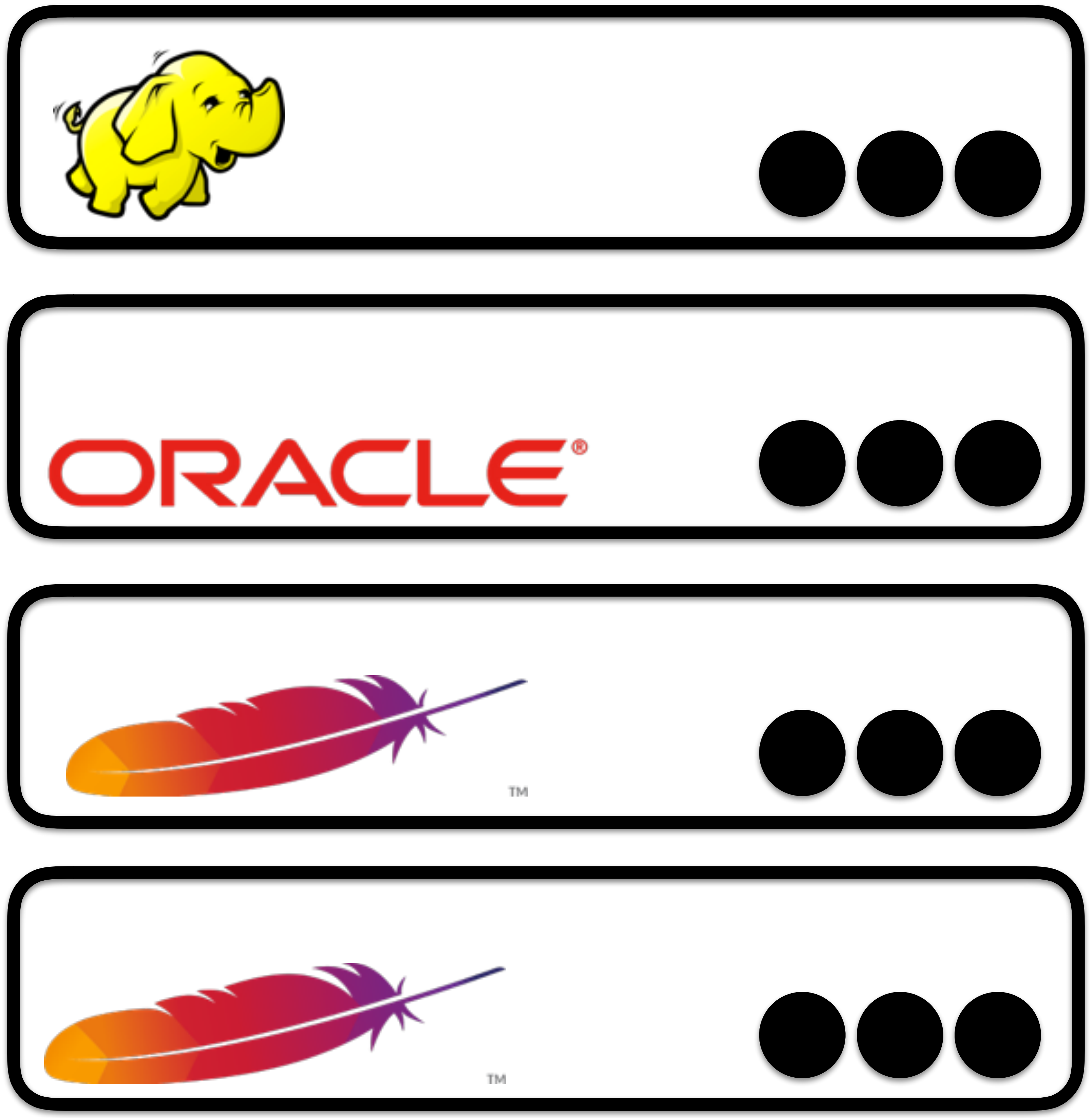
Imagine cloud computing if it were deployed like middleboxes.



So you want to deploy a web service.



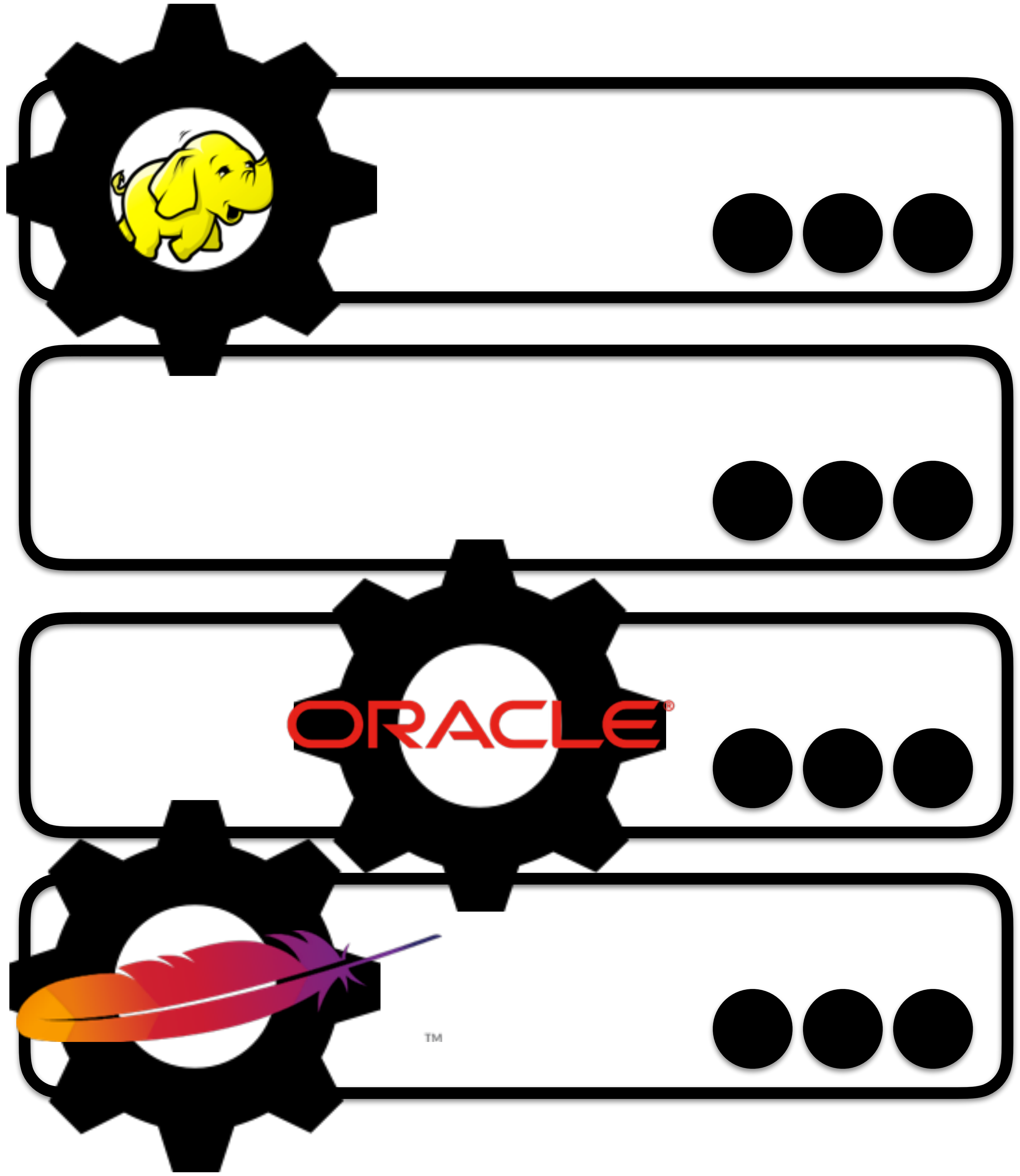
Imagine cloud computing if it were deployed like middleboxes.



So you want to deploy a web service.

This is ridiculous and not what anybody does for cloud services. But it’s what we were doing with middleboxes!

What we actually do in cloud computing.



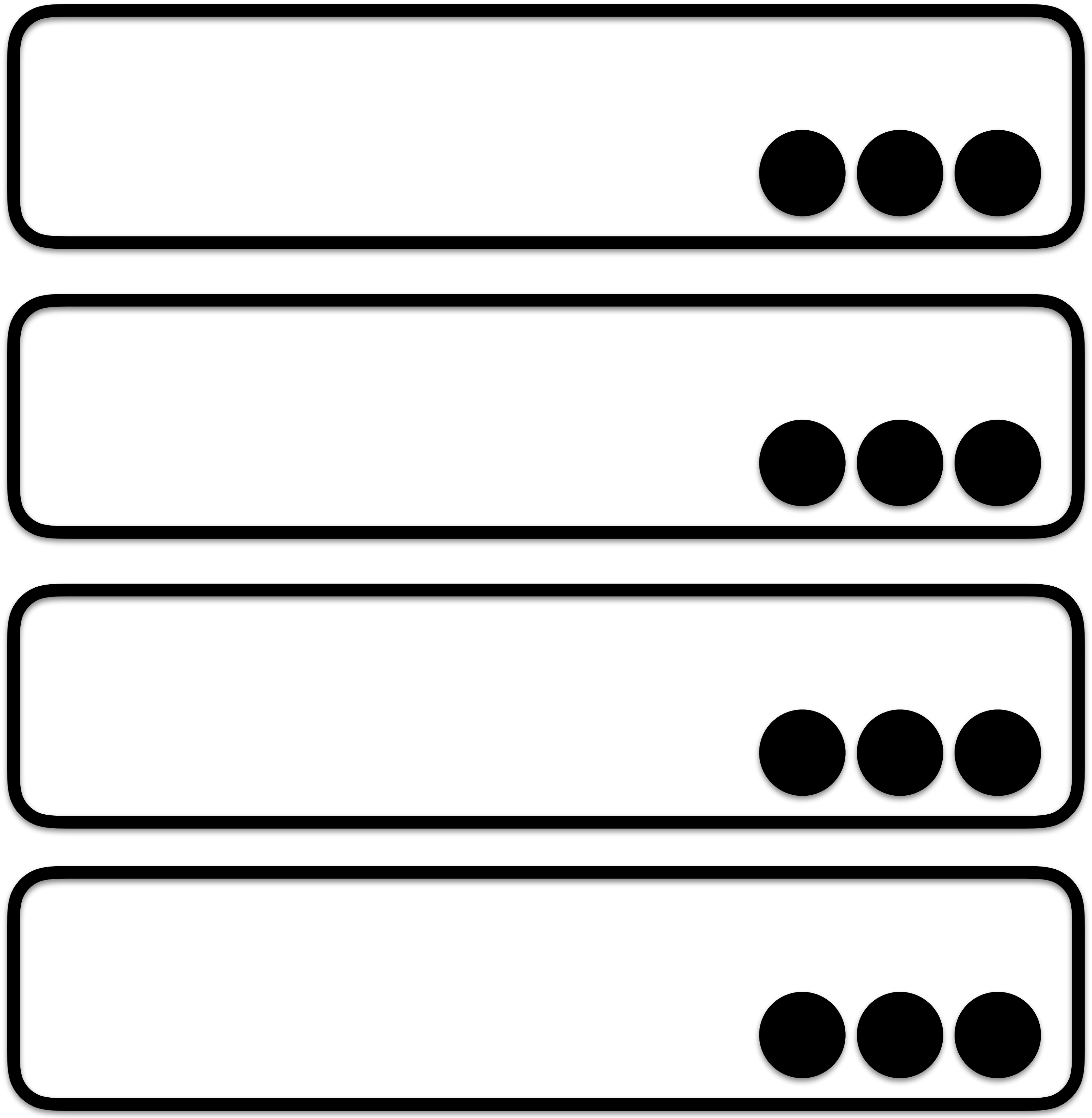
General-purpose hardware.

Services run in software.

Installation is a “click” — no cabling required.

Can re-use infrastructure for different tasks.

2012: ETSI Network Functions Virtualization



Network traffic routed through general-purpose hardware.

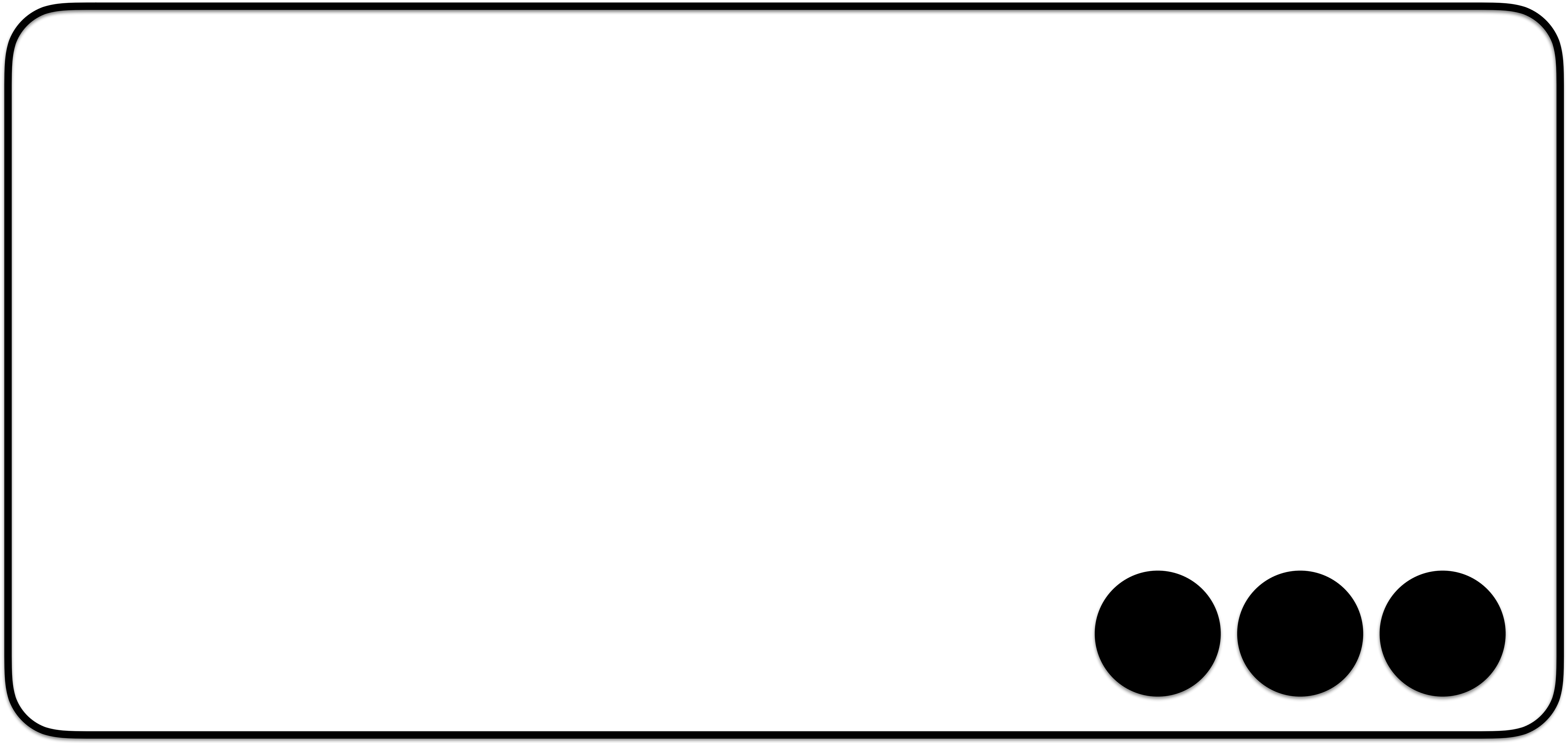
"Network Functions"

Benefits of NFV

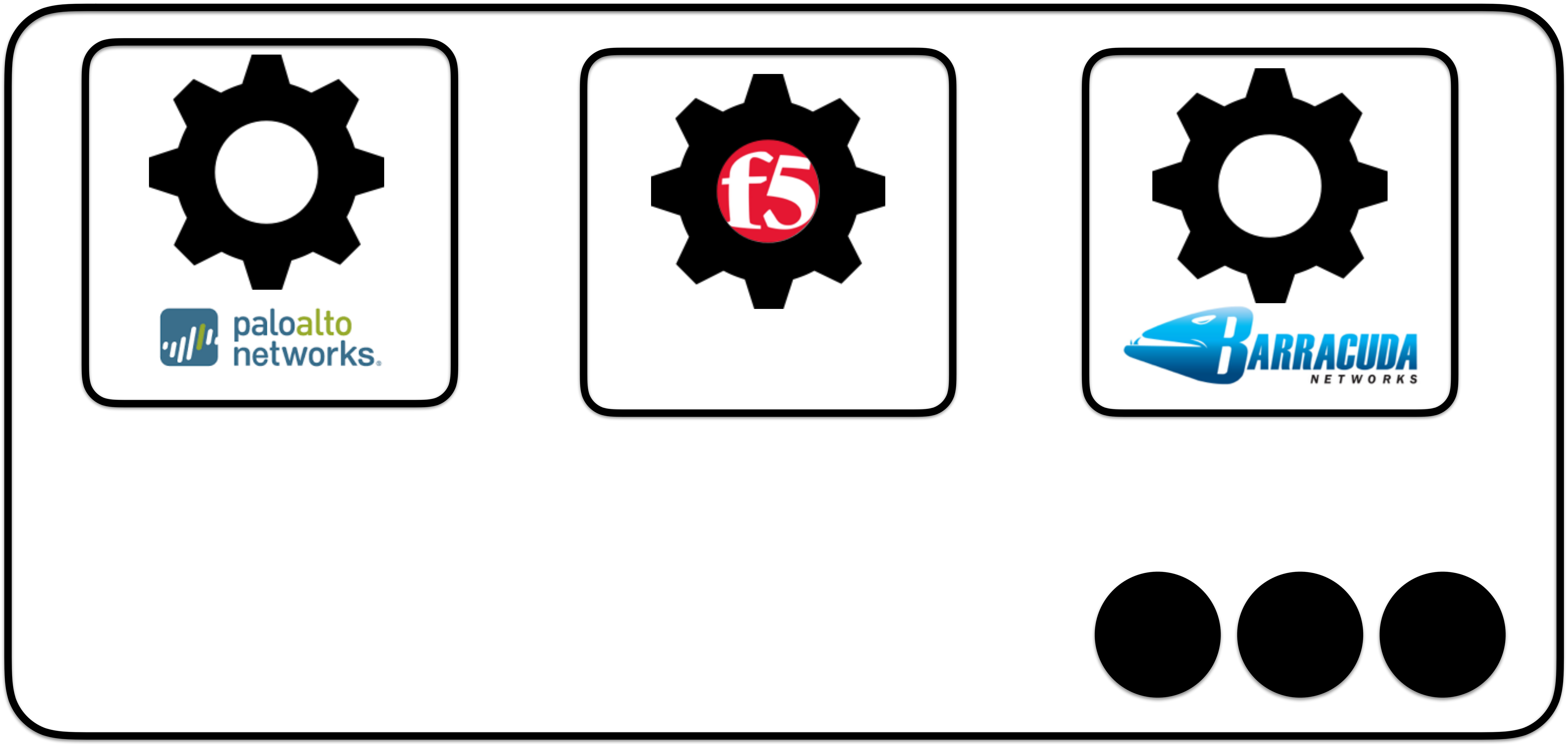
- Re-use hardware resources for many different applications
- “Scale on demand” as load changes
- Easier and more generic management tools
- Fast to upgrade and change software deployments
- Generic hardware usually -> cheaper, too!

Rough NFV System Architecture

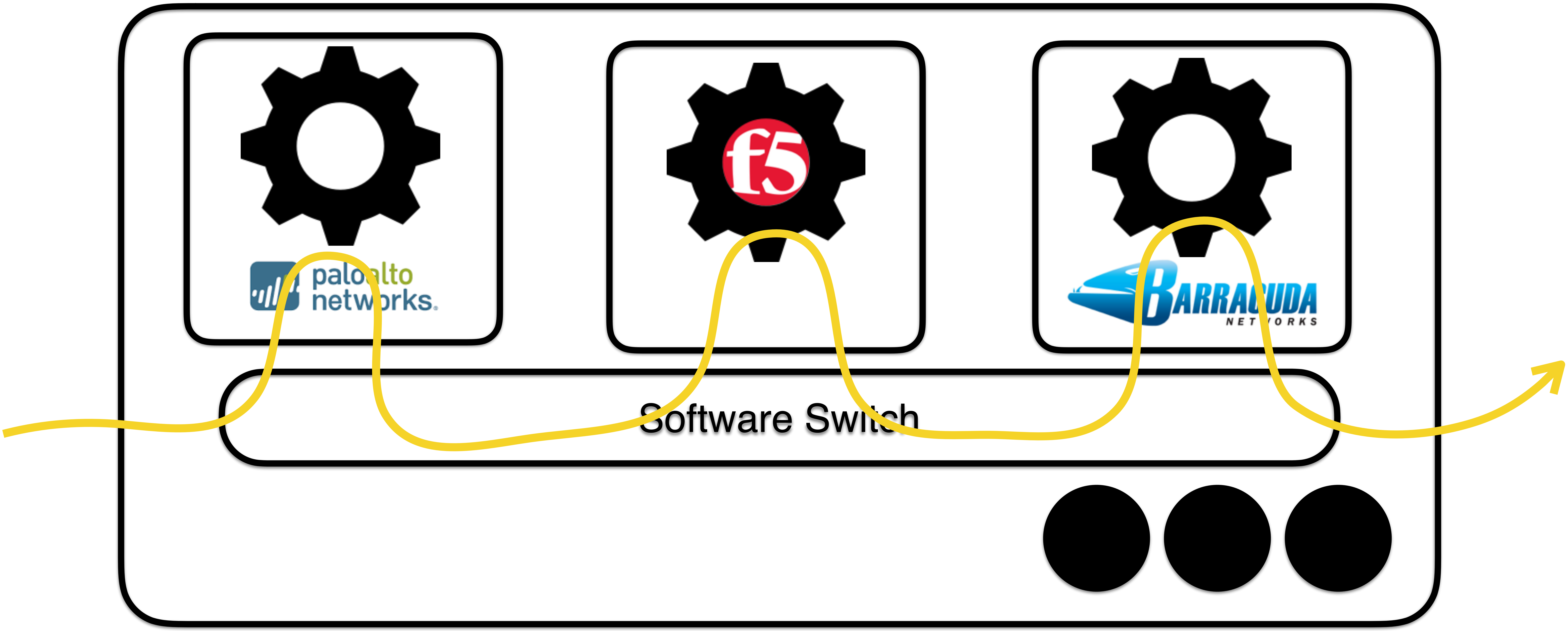
Rough NFV System Architecture



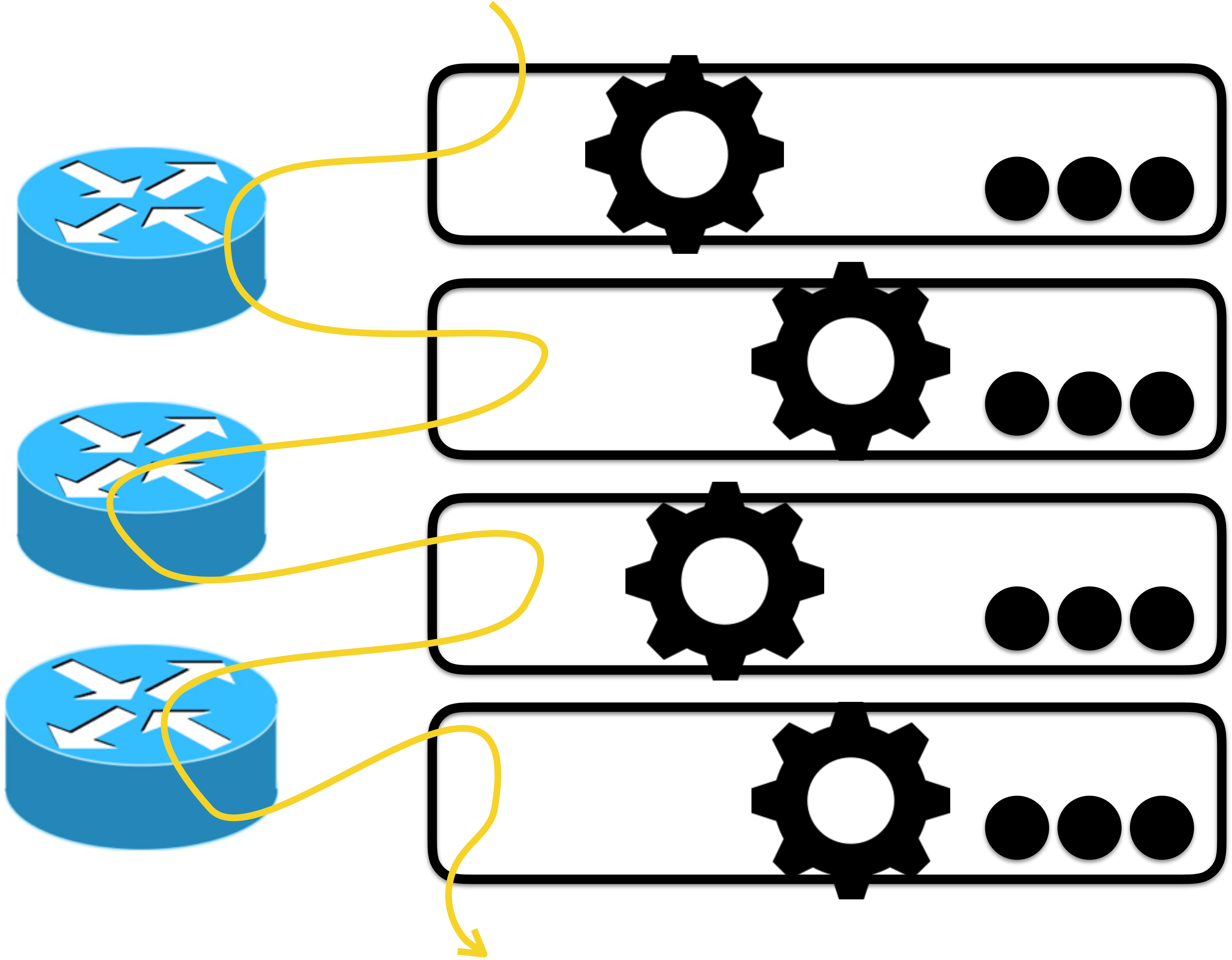
Rough NFV System Architecture



Rough NFV System Architecture



Multi-node NFV Architecture



Somehow we should stitch together multiple servers, too!

Today’s research: How do we actually build this?!

NFP: Enabling Network Function Parallelism in NFV

Chen Sun
Tsinghua University
c-sun14@mails.tsinghua.edu.cn

Jun Bi*
Tsinghua University
junbi@tsinghua.edu.cn

Zhilong Zheng
Tsinghua University
zhengzl15@mails.tsinghua.edu.cn

Heng Yu
Tsinghua University
hengyu1213@163.com

Hongxin Hu
Clemson University
hongxih@clemson.edu

ABSTRACT

Software-based sequential service chains in Network Function Virtualization (NFV) could introduce significant performance overhead. Current acceleration efforts for NFV mainly target on optimizing each component of the sequential service chain. However, based on the statistics from real world enterprise networks, we observe that 53.8% network function (NF) pairs can work in parallel. In particular, 41.5% NF pairs can be parallelized without causing extra resource overhead. In this paper, we present NFP, a high performance framework, that innovatively enables *network function parallelism* to improve NFV performance. NFP consists of three logical components. First, NFP provides a policy specification scheme for operators to intuitively describe sequential or parallel NF chaining intents. Second, NFP orchestrator intelligently identifies NF dependency and automatically compiles the policies into high performance service graphs. Third, NFP infrastructure performs light-weight packet copying, distributed parallel packet delivery, and load-balanced merging of packet copies to support NF parallelism. We implement an NFP prototype based on DPDK in Linux containers. Our evaluation results show that NFP achieves significant latency reduction for real world service chains.

CCS CONCEPTS

• **Networks → Middle boxes / network appliances**; *Network performance analysis*; Network control algorithms;

KEYWORDS

NFV, network function parallelism, service chain

ACM Reference format:
Chen Sun, Jun Bi, Zhilong Zheng, Heng Yu, and Hongxin Hu. 2017. NFP: Enabling Network Function Parallelism in NFV. In *Proceedings of SIGCOMM '17, Los Angeles, CA, USA, August 21–25, 2017,* 14 pages.
https://doi.org/10.1145/3098822.3098826

*Chen Sun, Jun Bi, Zhilong Zheng, and Heng Yu are with Institute for Network Sciences and Cyberspace, Tsinghua University, Department of Computer Science, Tsinghua University, and Tsinghua National Laboratory for Information Science and Technology (TNList). Jun Bi is the corresponding author.

ACM ISBN 978-1-4503-4653-5/17/08…$15.00
https://doi.org/10.1145/3098822.3098826

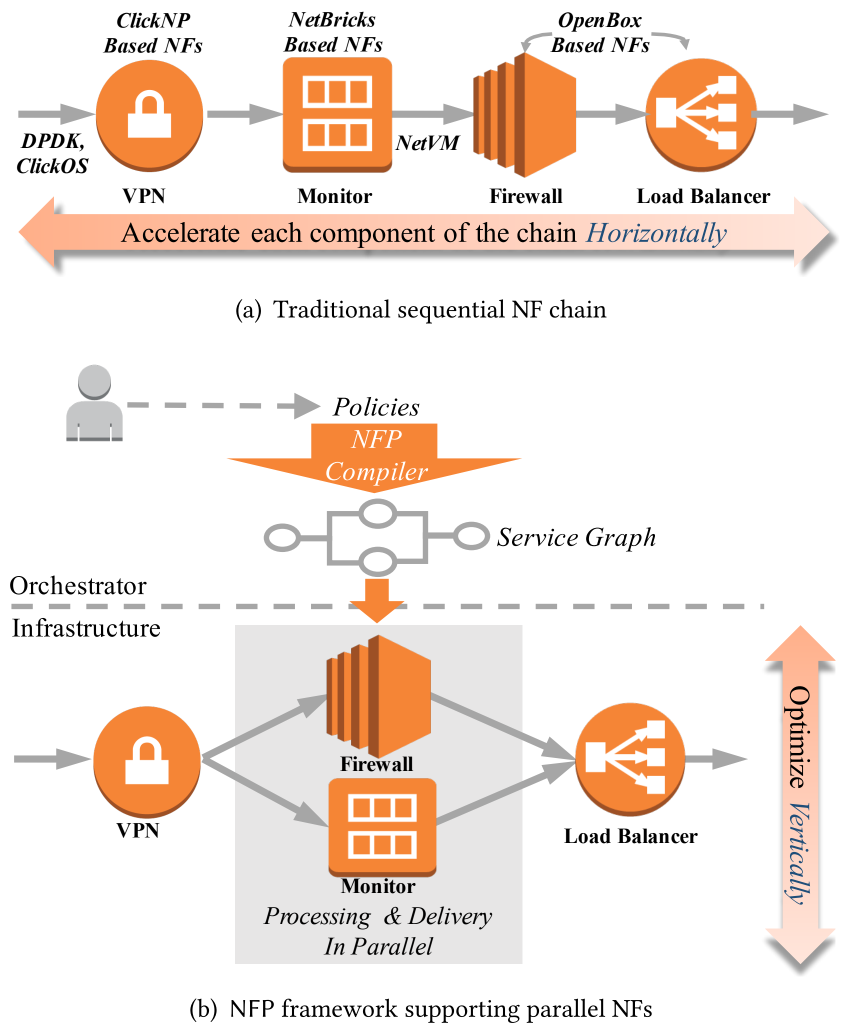


Figure 1: Traditional sequential chain derived from [36] v.s. NFP service graph with parallel NFs

1 INTRODUCTION

Network Functions Virtualization (NFV) addresses the problems of traditional proprietary middleboxes [61] by leveraging virtualization technologies to implement network functions (NFs) on commodity hardware, in order to enable rapid creation, destruction, or migration of NFs [24]. In operator networks [52], data centers [32, 36], mobile networks [25] and enterprise networks [60], network operators often require traffic to pass through multiple NFs in a particular *sequence* (e.g. firewall+IDS+proxy) [7, 26, 50], which is commonly referred to as service chaining. Meanwhile, Software-defined Networking (SDN) is used to steer traffic through appropriate NFs to enforce chaining policies [2, 16, 23, 32, 50]. Together, NFV and SDN can enable flexible and dynamic *sequential* service chaining.

Monday, 2PM Session

"NFP: Enabling Network Function Parallelism in NFV"

- NFs are often designed to run in parallel: when packets are read in, they are processed by one of many cores.
- What if multiple cores could operate on one packet at the same time?

Monday, 2PM Session

“Dynamic Service Chaining with Dysco”

- Remember those arrows the packet followed?
- How should the software and hardware switches know where to “steer” the packets — which NFs should a given NF be processed by?

Dynamic Service Chaining with Dysco

Pamela Zave
AT&T Labs–Research
pamela@research.att.com

Ronaldo A. Ferreira
UFMS
raf@facom.ufms.br

Xuan Kelvin Zou
Google
kelvinzou@google.com

Masaharu Morimoto
NEC Corporation of America
m-morimoto@bc.jp.nec.com

Jennifer Rexford
Princeton University
jrex@cs.princeton.edu

ABSTRACT

Middleboxes are crucial for improving network security and performance, but only if the right traffic goes through the right middleboxes at the right time. Existing traffic-steering techniques rely on a central controller to install *fine-grained forwarding rules* in network elements—at the expense of a large number of rules, a central point of failure, challenges in ensuring all packets of a session traverse the same middleboxes, and difficulties with middleboxes that modify the “five tuple.” We argue that a *session-level protocol* is a fundamentally better approach to traffic steering, while naturally supporting host mobility and multihoming in an integrated fashion. In addition, a session-level protocol can enable new capabilities like *dynamic* service chaining, where the sequence of middleboxes can change during the life of a session, e.g., to remove a load-balancer that is no longer needed, replace a middlebox undergoing maintenance, or add a packet scrubber when traffic looks suspicious. Our Dysco protocol steers the packets of a TCP session through a service chain, and can dynamically reconfigure the chain for an ongoing session. Dysco requires no changes to end-host and middlebox applications, host TCP stacks, or IP routing. Dysco’s distributed reconfiguration protocol handles the removal of proxies that terminate TCP connections, middleboxes that change the size of a byte stream, and concurrent requests to reconfigure different parts of a chain. Through formal verification using Spin and experiments with our Linux-based prototype, we show that Dysco is provably correct, highly scalable, and able to reconfigure service chains across a range of middleboxes.

CCS CONCEPTS

• **Networks** → **Network protocols**; **Middle boxes / network appliances**; *Session protocols*; *Network components*;

KEYWORDS

Session Protocol; NFV; Verification; Spin.

SIGCOMM ’17, August 21–25, 2017, Los Angeles, CA, USA

ACM ISBN 978-1-4503-4653-5/17/08…$15.00
https://doi.org/10.1145/3098822.3098827

ACM Reference format:
Pamela Zave, Ronaldo A. Ferreira, Xuan Kelvin Zou, Masaharu Morimoto, and Jennifer Rexford. 2017. Dynamic Service Chaining with Dysco. In *Proceedings of SIGCOMM ’17, Los Angeles, CA, USA, August 21–25, 2017,* 14 pages. https://doi.org/10.1145/3098822.3098827

1 INTRODUCTION

In the early days of the Internet, end-hosts were stationary devices, each with a single network interface, communicating directly with other such devices. Now most end-hosts are mobile, many are multihomed, and traffic traverses chains of middleboxes such as firewalls, network address translators, and load balancers. In this paper, we argue that the “new normal” of middleboxes warrants a re-examination of approaches, as has happened with mobility [49].

Most existing research proposals for middlebox insertion or “service chaining” use a logically centralized controller to install fine-grained forwarding rules in network elements, to steer traffic through the right sequence of middleboxes [1, 9, 10, 18, 19, 36, 37, 50]. The many weaknesses of these solutions are a direct result of their reliance on forwarding rules for traffic steering:

- They rely on real-time response from the central controller to handle frequent events, including link failures, traffic fluctuations, and the addition of new middlebox instances.
- They need network state that grows with the number of policies, the difficulty of classifying traffic, the length of service chains, and the number of instances per middlebox type.
- Updates to rules due to changes in policy, topology, or load may change the paths of ongoing sessions, yet all packets of a session must traverse the same middleboxes (“session affinity”).
- Fine-grained routing is inherently intra-domain. It is difficult to outsource middleboxes to the cloud [40] or other third-party providers [45], since the controller cannot control the entire path.
- Some middleboxes modify the “five-tuple” of packets in unpredictable ways, so that forwarding rules matching packets going into the middlebox might not match them on the way out.
- Some middleboxes classify packets to choose which middlebox should come next. These middleboxes should be able to select the service chain for their outgoing packets, which forwarding by network elements does not allow them to do.
- Adding middleboxes to a secure session (e.g., TLS) is challenging without cooperation with the end-hosts to exchange the information needed to decrypt and reencrypt the data [25].
- A multihomed host spreads traffic over multiple administrative domains (e.g., enterprise WiFi and commercial cellular network), yet some middleboxes need to see all the data in a TCP session (e.g., for parental controls [38]). In the administrative domain

NFVnice: Dynamic Backpressure and Scheduling for NFV Service Chains

Sameer G Kulkarni*★, Wei Zhang‡, Jinho Hwang§, Shriram Rajagopalan§, K.K. Ramakrishnan†, Timothy Wood‡, Mayutan Arumaithurai* and Xiaoming Fu*

*University of Göttingen, Germany, ‡George Washington University, §IBM T J Watson Research Center, †University of California, Riverside.

ABSTRACT

Managing Network Function (NF) service chains requires careful system resource management. We propose *NFVnice*, a user space NF scheduling and service chain management framework to provide fair, efficient and dynamic resource scheduling capabilities on Network Function Virtualization (NFV) platforms. The NFVnice framework monitors load on a service chain at high frequency (1000Hz) and employs backpressure to shed load early in the service chain, thereby preventing wasted work. Borrowing concepts such as rate proportional scheduling from hardware packet schedulers, CPU shares are computed by accounting for heterogeneous packet processing costs of NFs, I/O, and traffic arrival characteristics. By leveraging cgroups, a user space process scheduling abstraction exposed by the operating system, NFVnice is capable of controlling when network functions should be scheduled. NFVnice improves NF performance by complementing the capabilities of the OS scheduler but without requiring changes to the OS's scheduling mechanisms. Our controlled experiments show that NFVnice provides the appropriate rate-cost proportional fair share of CPU to NFs and significantly improves NF performance (throughput and loss) by reducing wasted work across an NF chain, compared to using the default OS scheduler. NFVnice achieves this even for heterogeneous NFs with vastly different computational costs and for heterogeneous workloads.

CCS CONCEPTS

• **Networks** → **Network resources allocation**; **Network management**; *Middle boxes / network appliances*; *Packet scheduling*;

KEYWORDS

Network Functions (NF), Backpressure, NF-Scheduling, Cgroups.

ACM Reference format:

S.G. Kulkarni, W. Zhang, J. Hwang, S. Rajagopalan, K.K. Ramakrishnan, T. Wood, M. Arumaithurai, X. Fu. 2017. NFVnice: Dynamic Backpressure and Scheduling for NFV Service Chains. In *Proceedings of SIGCOMM '17, Los Angeles, CA, USA, August 21-25, 2017,* 14 pages.
https://doi.org/10.1145/3098822.3098828

★Work done while Sameer was at University of California, Riverside, as part of secondment in the EU FP7 Marie Curie CleanSky ITN Research Project.

SIGCOMM '17, August 21-25, 2017, Los Angeles, CA, USA

ACM ISBN 978-1-4503-4653-5/17/08…$15.00
https://doi.org/10.1145/3098822.3098828

1 INTRODUCTION

Network Function Virtualization (NFV) seeks to implement network functions and middlebox services such as firewalls, NAT, proxies, deep packet inspection, WAN optimization, etc., in software instead of purpose-built hardware appliances. These software based network functions can be run on top of commercial-off-the-shelf (COTS) hardware, with virtualized network functions (NFs). Network functions, however, often are chained together [20], where a packet is processed by a sequence of NFs before being forwarded to the destination.

The advent of container technologies like Docker [34] enables network operators to densely pack a single NFV appliance (VM/bare metal) with large numbers of network functions at runtime. Even though NFV platforms are typically capable of processing packets at line rate, without efficient management of system resources in such densely packed environments, service chains can result in serious performance degradation because bottleneck NFs may drop packets that have already been processed by upstream NFs, resulting in wasted work in the service chain.

NF processing has to address a combination of requirements. Just as hardware switches and routers provide rate-proportional scheduling for packet flows, an NFV platform has to provide a fair processing of packet flows. Secondly, the tasks running on the NFV platform may have heterogeneous processing requirements that OS schedulers (unlike hardware switches) address using their typical fair scheduling mechanisms. OS schedulers, however, do not treat packet flows fairly in proportion to their arrival rate. Thus, NF processing requires a re-thinking of the system resource management framework to address both these requirements. Moreover, standard OS schedulers: a) do not have the right metrics and primitives to ensure fairness between NFs that deal with the same or different packet flows; and b) do not make scheduling decisions that account for chain level information. If the scheduler allocates more processing to an upstream NF and the downstream NF becomes overloaded, packets are dropped by the downstream NF. This results in inefficient processing and wasting the work done by the upstream NF. OS schedulers also need to be adapted to work with user space data plane frameworks such as Intel's DPDK [1]. They have to be cognizant of NUMA (Non-uniform Memory Access) concerns of NF processingand the dependencies among NFs in a service chain. Determining how to dynamically schedule NFs is key to achieving high performance and scalability for diverse service chains, especially in a scenario where multiple NFs are contending for a CPU core[1]

[1]While CPU core counts are increasing in modern hardware, they are likely to remain a bottleneck resource, especially when service chains are densely packed into a single machine (as is often the case with several proposed approaches [23, 52]).

Monday, 2PM Session

"NFVNice: Dynamic Backpressure and Scheduling for NFV Service Chains"

- How should we schedule which NFs run and when?
- One key challenge: avoid wasting work!

Thursday, 8:30AM Session

“A High Performance Packet Core for Cellular Networks”

- Focuses on a particular class of NFs used in ISPs: “Evolved Packet Core” devices.
- Shows how to re-architect them to be more efficient and faster by taking advantage of software!

A High Performance Packet Core for Next Generation Cellular Networks

Zafar Ayyub Qazi, UC Berkeley
Melvin Walls, Nefeli Networks, Inc.
Aurojit Panda, UC Berkeley
Vyas Sekar, Carnegie Mellon University
Sylvia Ratnasamy, UC Berkeley
Scott Shenker, UC Berkeley

ABSTRACT

Cellular traffic continues to grow rapidly making the scalability of the cellular infrastructure a critical issue. However, there is mounting evidence that the current Evolved Packet Core (EPC) is ill-suited to meet these scaling demands: EPC solutions based on specialized appliances are expensive to scale and recent software EPCs perform poorly, particularly with increasing numbers of devices or signaling traffic.

In this paper, we design and evaluate a new system architecture for a software EPC that achieves high and scalable performance. We postulate that the poor scaling of existing EPC systems stems from the manner in which the system is decomposed which leads to device state being duplicated across multiple components which in turn results in frequent interactions between the different components. We propose an alternate approach in which state for a single device is consolidated in one location and EPC functions are (re)organized for efficient access to this consolidated state. In effect, our design “slices” the EPC by user.

We prototype and evaluate *PEPC*, a software EPC that implements the key components of our design. We show that PEPC achieves 3-7× higher throughput than comparable software EPCs that have been implemented in industry and over 10× higher throughput than a popular open-source implementation (OpenAirInterface). Compared to the industrial EPC implementations, PEPC sustains high data throughput for 10-100× more users devices per core, and a 10× higher ratio of signaling-to-data traffic. In addition to high performance, PEPC’s by-user organization enables efficient state migration and customization of processing pipelines. We implement user migration in PEPC and show that state can be migrated with little disruption, e.g., migration adds only up to 4μs of latency to median per packet latencies.

CCS CONCEPTS

• **Networks** → *Network components*; *Middle boxes / network appliances*;

SIGCOMM ’17, August 21-25, 2017, Los Angeles, CA, USA

ACM ISBN 978-1-4503-4653-5/17/08…$15.00
https://doi.org/10.1145/3098822.3098848

KEYWORDS

Cellular Networks, EPC, Network Function

ACM Reference format:
Zafar Ayyub Qazi, Melvin Walls, Aurojit Panda, Vyas Sekar, Sylvia Ratnasamy, and Scott Shenker. 2017. A High Performance Packet Core for Next Generation Cellular Networks. In *Proceedings of SIGCOMM ’17, Los Angeles, CA, USA, August 21-25, 2017*, 14 pages.
https://doi.org/10.1145/3098822.3098848

1 INTRODUCTION

Cellular networks are experiencing explosive growth along multiple dimensions: (i) traffic volumes (e.g., mobile traffic grew by 74% in 2015), (ii) the number and diversity of connected devices (e.g., projections show that by 2020 there will be 11.6 billion mobile connected devices including approximately 3 billion IoT devices [11]), and (iii) signaling traffic (e.g., signaling traffic in the cellular network is reported to be growing 50% faster than data traffic [31]).

These trends impose significant *scaling* challenges on the cellular infrastructure. In particular, there is growing concern regarding the scalability of the cellular evolved packet core (EPC) [21] infrastructure. The EPC is the portion of the network that connects the base stations to the IP backbone and implements cellular-specific processing on user’s data and signaling traffic. Recent industrial and academic studies have provided mounting anecdotal and empirical evidence showing that existing EPC implementations cannot keep up with the projected growth in cellular traffic [10, 18, 19, 23, 37].

We postulate that the poor scaling of existing solutions stems from the manner in which existing EPC systems have been decomposed. More specifically, EPC systems today are factored based on traffic type, with different components to handle signaling and data traffic: the Mobility Management Entity (MME) handles signaling traffic from mobile devices and base stations, while the Serving and Packet Gateways (S-GW and P-GW) handle data traffic. The problem with this factoring is it complicates how *state* is decomposed and managed. As we elaborate on in §2, current designs lead to three problems related to state management:

(1) Duplicated state leads to frequent synchronization across components. In current EPCs, per-user state is often duplicated between components. For example, a user request to establish a cellular connection is processed by the MME which instantiates user state and then communicates this addition to the S-GW which in turn locally instantiates similar per-user state. A similar interaction takes place when user state is updated after mobility events. This duplication introduces complexity (e.g., implementing the protocols

Thanks!

Isn’t this a cute dog?