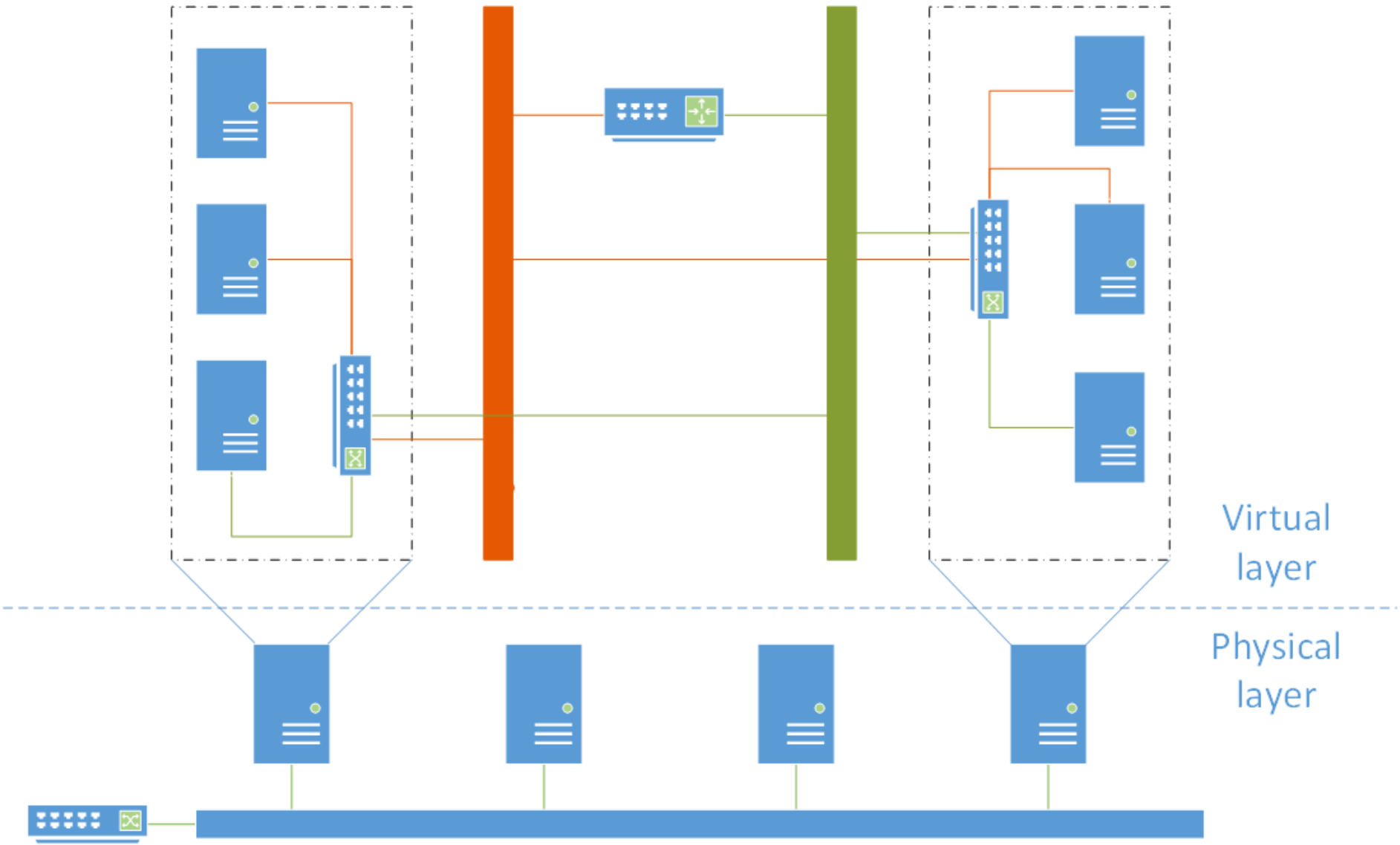
E2: A Framework for NFV Applications

Shoumik Palkar, Chang Lan, Sangjin Han, Keon Jang, Aurojit Panda,
Sylvia Ratnasamy, Luigi Rizzo, Scott Shenker

Presentation by Piotr Zalas

Basic terms



Overview of typical Central Office

Definition of CO

A CO is a facility commonly located in a metropolitan area to which residential and business lines connect

CO's hardware:

- Broadband Network Gateways (BNGs) – connect broadband users to the carrier's IP backbone
- Evolved Packet Core (EPC) gateways – connect cellular users to the IP backbone
- Devices supporting content caching, Deep Packet Inspection, parental controls, WAN and application acceleration, traffic scrubbing for DDoS prevention, firewalls, IPTV multicast, DHCP, VPN, Hierarchical QoS, NAT, ...

Reasons for migration to NFV

Problems:

- the capital and operational expenses incurred by a carrier's COs are very high – there are many COs, each of non-trivial scale; e.g., AT&T reports 5,000 CO locations in the US alone, with 10-100K subscribers per CO
- standalone devices with proprietary internals and vendor-specific management APIs
- new business models based on opening up infrastructure to 3rd party services to exploit their physical proximity to users

Expected outcome:

- uniform architecture based on commodity hardware
- efficiency through statistical multiplexing
- centralized management across CO locations
- flexibility and portability of software services

Problems introduced by NFV solutions

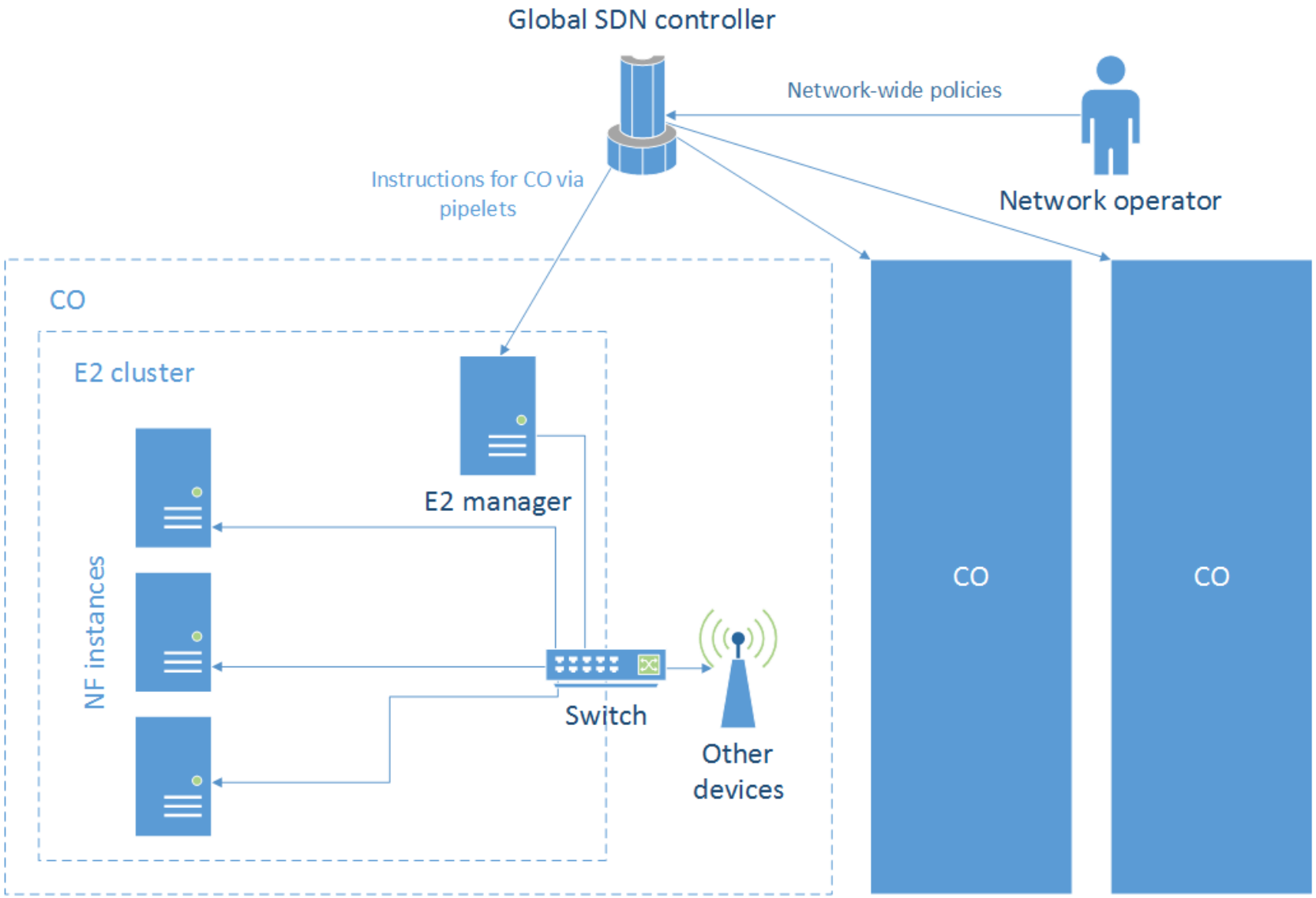
- NFV products and prototypes tend to be merely virtualized software implementations of products that were previously offered as dedicated hardware appliances
- Each software middlebox still comes as a closed implementation bundled with a custom management solution
- The operator must cope with many NF-specific management systems
- NF developers must invent their own solutions to common but non-trivial problems such as dynamic scaling and fault tolerance

E2 – Elastic Edge

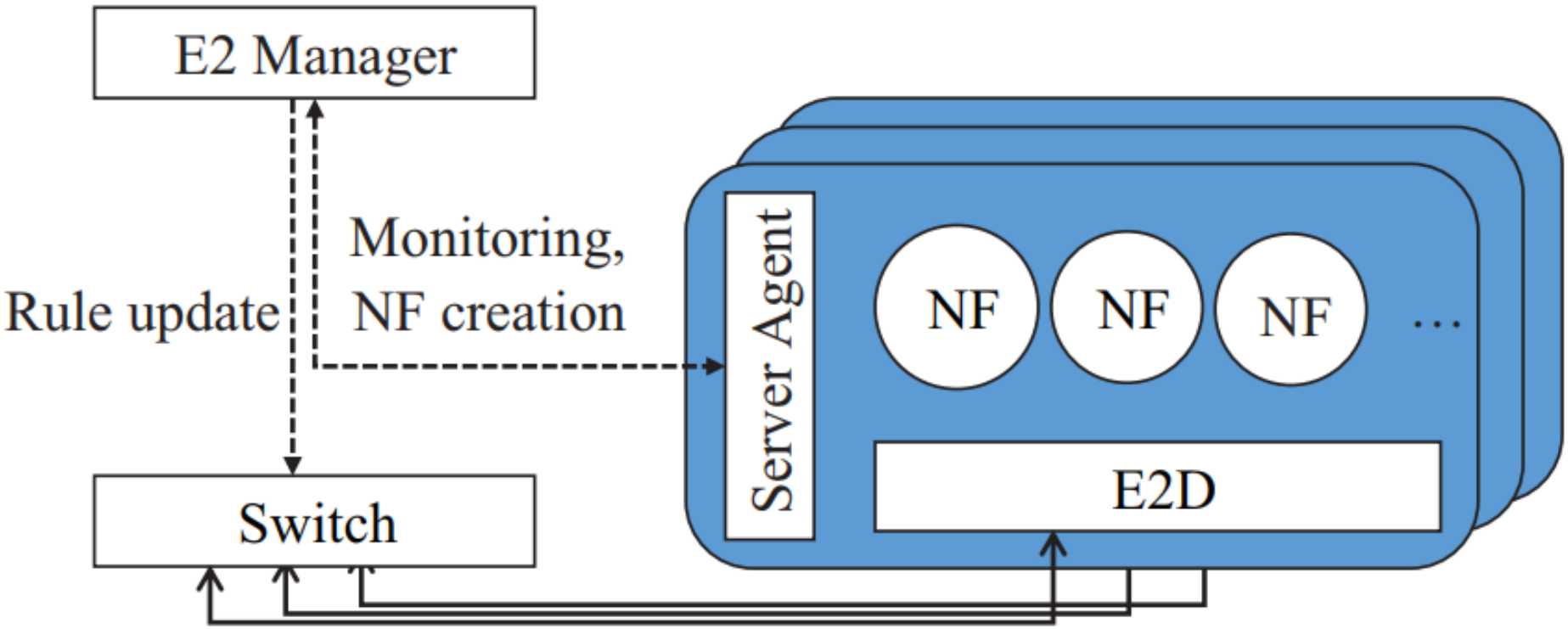
A framework for packet-processing applications that implements general techniques for common issues:

- placement (which NF runs where)
- elastic scaling (adapting the number of NF instances and balancing load across them)
- service composition
- resource isolation
- fault-tolerance
- energy management
- monitoring

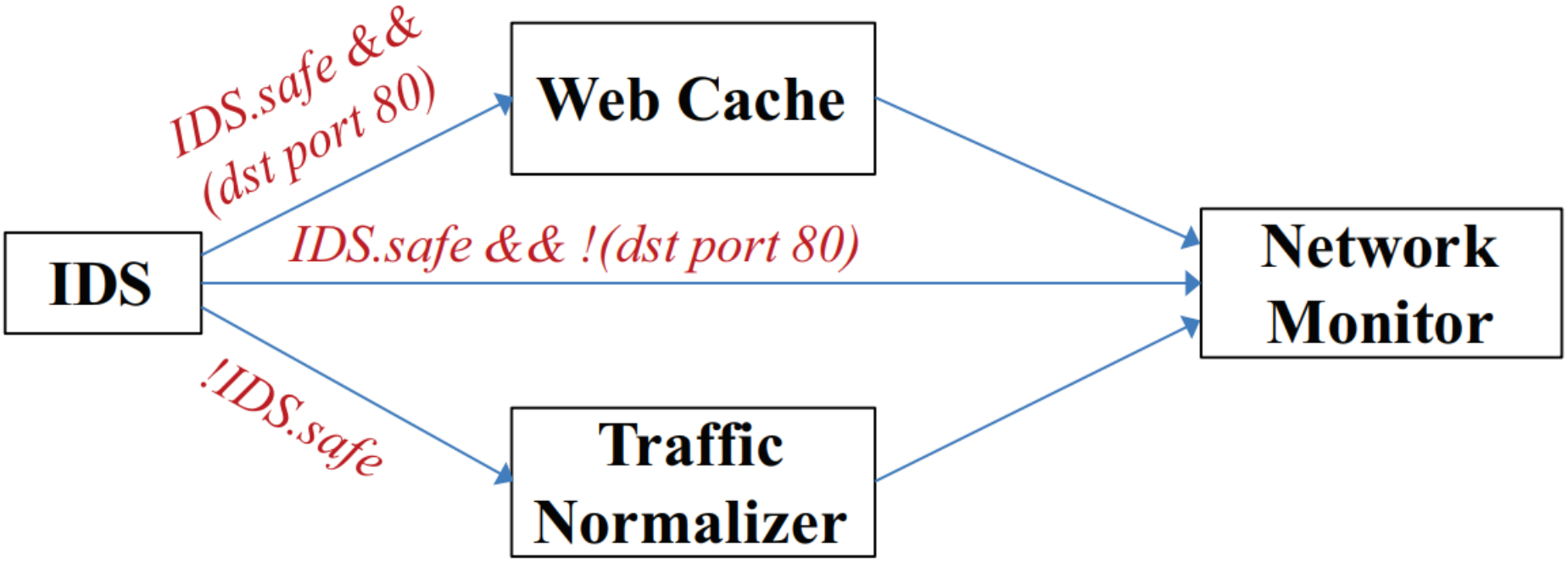
Design overview



E2 cluster architecture



System Inputs – pipelets

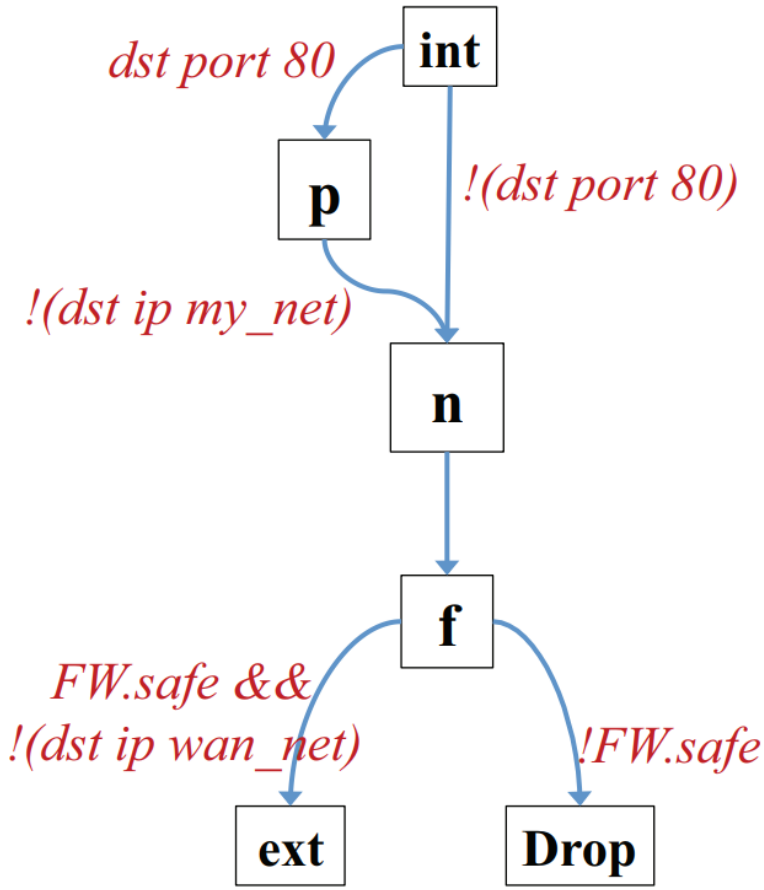


- node - NF or a physical port on the switch
- edges - describe the traffic between nodes
- filter - a boolean expression that defines what subset of the traffic from the source node should reach the destination node

Example of pipelet

```
Proxy p; NAT n; FW f;
Port<0-7> int;
Port<8-15> ext;

pipelet {
 int [dst port 80] -> p;
 int [!(dst port 80)] -> n;
 p [!(dst ip my_net)] -> n;
 n -> f;
 f [FW.safe &&
  !(dst ip wan_net)] -> ext;
 f [!FW.safe] -> Drop;
}
```



System Inputs – resources

NF description:

- Native or Legacy API - native give better performance
- Attribute-Method bindings - attribute $\rightarrow$ {port, per-packet metadata}
- Scaling constraints - tell whether the application can be scaled across servers/cores or not
- Affinity constraints - how to split traffic across NF instances (e.g., "all packets with a particular TCP port")
- NF performance - an estimate of the per-core, per-GHz traffic rate that the NF can sustain

Hardware description:

- the number of cores (and speed) and the network I/O bandwidth per server
- the number of switch ports and entries in the switch flow table

E2 Dataplane – E2D

Characteristics:

- Modular architecture based on SoftNIC
- Highly efficient (uses Intel DPDK)

Extensions to SoftNIC:

- Basic modules utilized to implement E2's components for NF placement, interconnection, and dynamic scaling (e.g. modules for load monitoring, flow tracking, load balancing, packet classification, and tunneling across NFs)
- Native API – for better performance and richer message abstraction
- Control API exposed to E2's Server Agent

Why OVS is not suitable

[Its] expressiveness and functionality are limited by the flow-table semantics and predefined actions of OpenFlow

E2 Control Plane

The E2 control plane is in charge of:

- Placement – instantiating the pipelets on servers
- Interconnection – setting up and configuring the interconnections between NFs
- Scaling – dynamically adapting the placement decisions depending on load variations
- Ensuring affinity constraints of NFs

E2 Control Plane – NF placement and interconnection

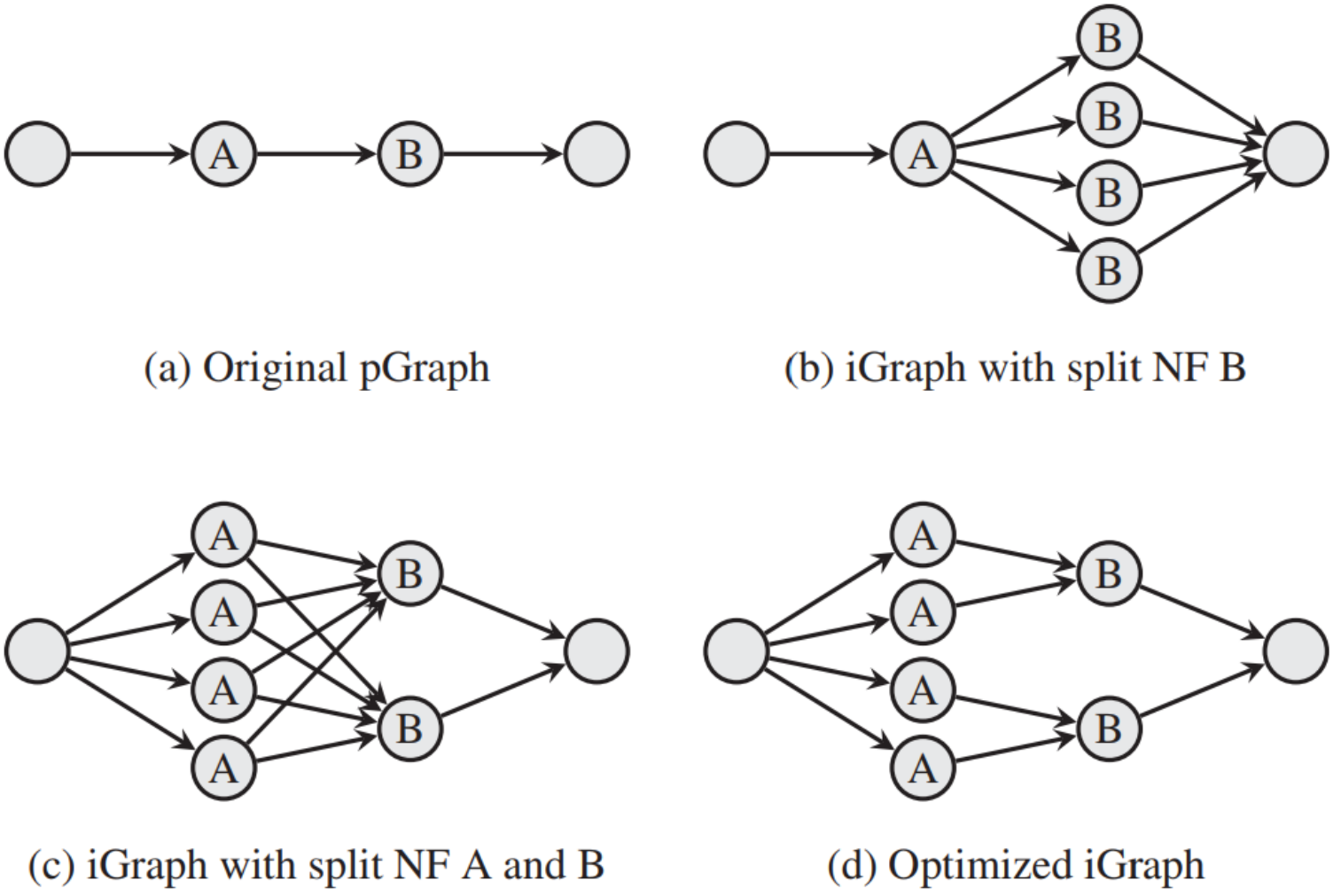
Stages of NF placement:

- Step 1: Merging pipelets into a single policy graph (pGraph)
- Step 2: Sizing
- Step 3: Converting the pGraph to an iGraph (instance graph)
- Step 4: Instance placement - use Kernighan-Lin heuristic for minimizing inter-server traffic
- Step 5: Offloading low-level functions such as L2/L3-forwarding, VLAN/tunneling, and QoS packet scheduling to the hardware switch

Service Interconnection:

- Instantiating NFs' ports
- Adding traffic filters
- Configuring the switch and the E2D

NF placement example



(a) Original pGraph

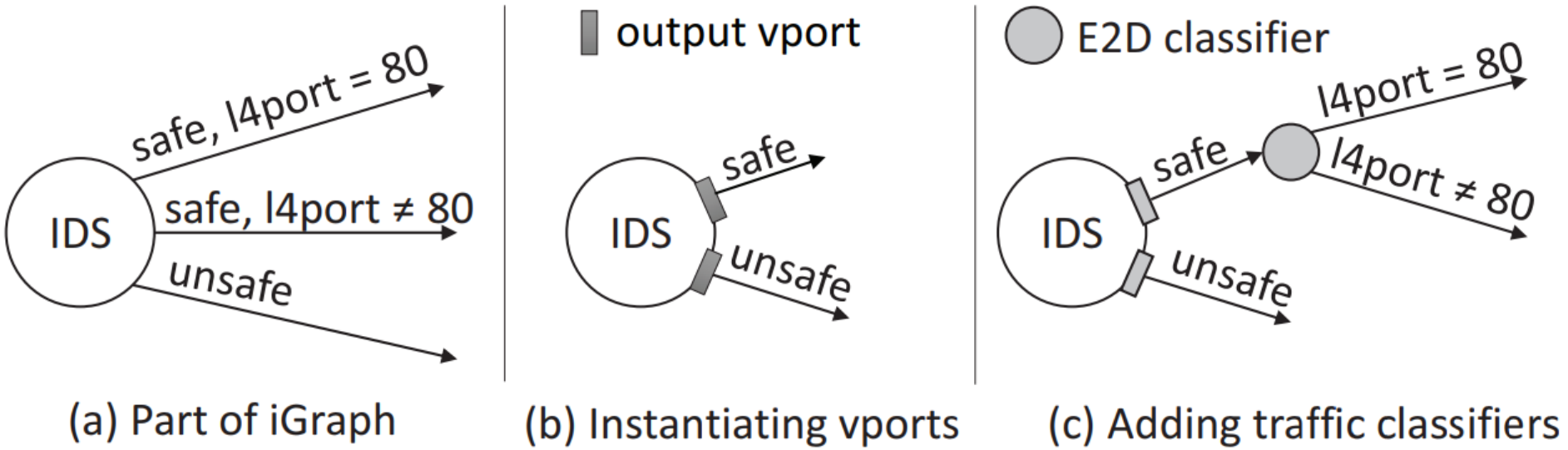
(b) iGraph with split NF B

(c) iGraph with split NF A and B

(d) Optimized iGraph

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

Service interconnection example



(a) Part of iGraph (b) Instantiating vports (c) Adding traffic classifiers

E2 converts edge annotations on an iGraph (a) into output ports (b) that the applications write to, and then adds traffic filters that the E2D implements (c)

E2 Control Plane – dynamic scaling

Scaling algorithm:

- NFs report on their instantaneous load, and the E2D itself detects overloads based on queues and processing delays
- When a node signals overload the Server Agent notifies the E2 Manager
- E2 Manager updates estimates, recalculates iGraph and places new NF instances
- Flow is split between old and new NFs

Flow splitting is problematic

Usually traffic for a given flow must reach the instance that holds that flow's state. Prior solutions depend on state migration techniques, which is both expensive and incompatible with legacy applications, or require large rule sets in hardware switches.

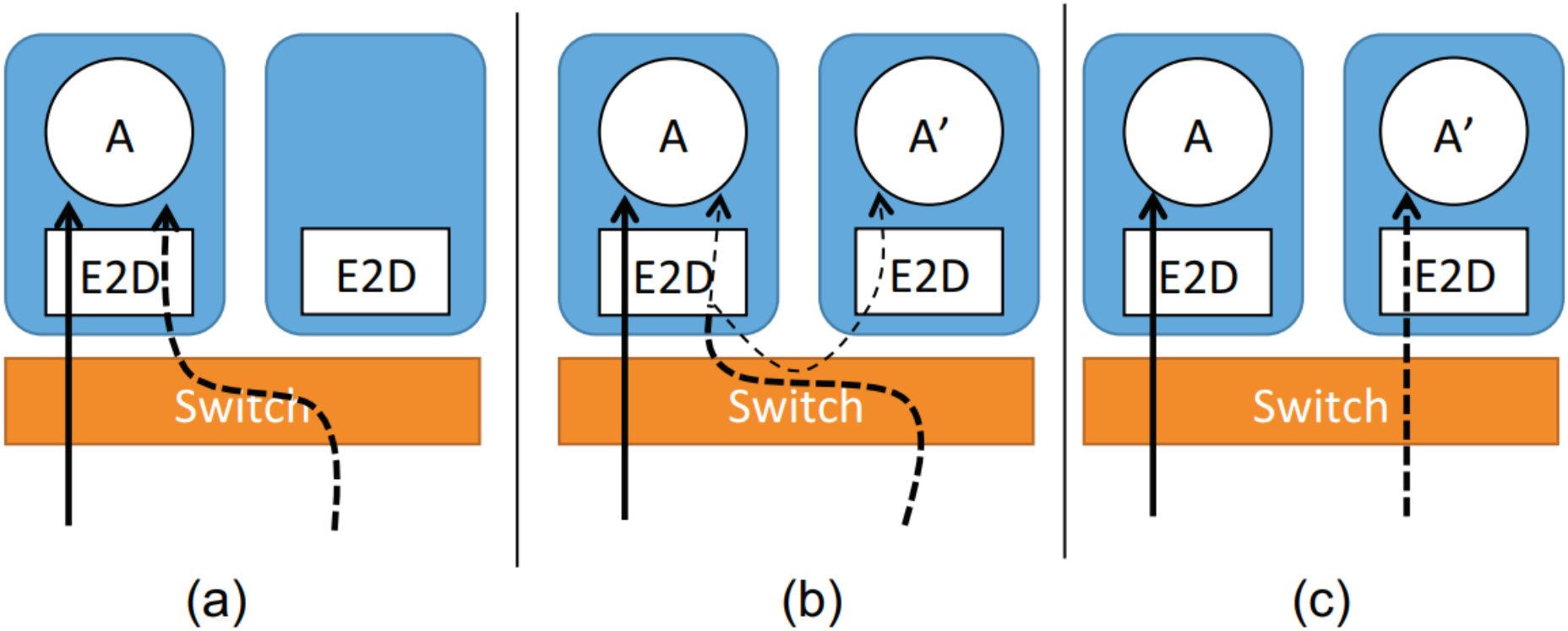
E2 Control Plane – dynamic scaling

Migration avoidance algorithm:

- Upon splitting, the range filter on the hardware switch is initially unchanged, and the new filters (two new ranges plus exceptions) are installed in the E2D of the server that hosts old NF
- As old flows in gradually terminate, the corresponding exception rules can be removed
- When the number of exceptions drops below some threshold, the new ranges and remaining exceptions are pushed to the switch, replacing the original rule

The trade-off is the additional latency to new flows being punted between servers (but this overhead is small and for a short period of time) and some additional switch bandwidth.

E2 Control Plane – dynamic scaling



(a) Flows enter a single NF instance. (b) Migration avoidance partitions the range of Flow IDs and punts new flows to a new replica using the E2D. Existing flows are routed to the same instance. (c) Once enough flows expire, E2 installs steering rules in the switch.

Evaluation

Hardware:

- 1 x Intel Xeon E5-2680 v2 CPU with 10 cores in each of 2 sockets
- 3 x Intel Xeon E5-2650 v2 CPU with 8 cores in each of 2 sockets
- Each server is connected to the switch via one 10 Gbps link
- Intel FM6000 Seacliff Trail Switch with 48 10 Gbps ports and 2,048 flow table entries
- Four 10 G links from external ports on switch to traffic generator
- Traffic generator – server with four 10G NICs and two Intel Xeon E5-2680 v2 CPUs

Configuration:

- On each server, we dedicate one core to run the E2D layer
- E2 Manager runs on a standalone server that connects to each server and to the management port of the switch on a separate 1 Gbps control network.

E2D Performance

Loopback test:

- NF generates packets and then absorbs them ($NF \rightarrow E2D \rightarrow NF$)
- E2D incurs 0.3 μs delay (or 0.15 μs for each direction)
- Forwarding through E2D on a single core fully saturates the server's 10 Gbps link

Native API – Zero-copy vports performance:

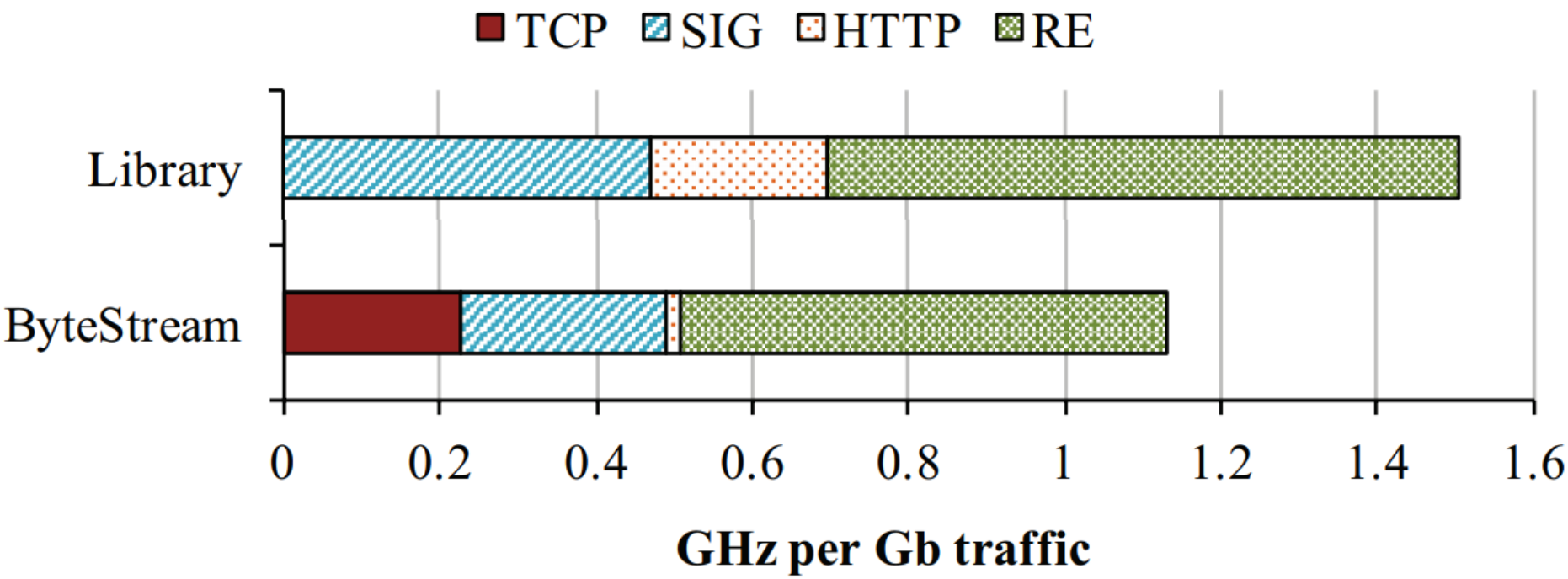
- $NF_1 \rightarrow E2D \rightarrow NF_2$, both NFs are on same server
- Reduces latency of NF-to-NF communication by over 2.5x on average
- Increases throughput by over 26x

Native API – Bytestream vports performance:

- Pipeline of 3 NFs performing DPI on TCP stream
- E2D guarantees reliable transfer of all messages between NFs that use bytestream vports, with much less overhead than full TCP
- Save 25% of processing cycles, for the same amount of input traffic

E2D Performance

Comparison of CPU cycles for three DPI NFs, without and with bytestream vports. The both cases use the native API.



TCP – NF for unpacking stream; SIG – signature matching with the Aho-Corasick algorithm; HTTP – HTTP parser; RE – redundancy elimination using Rabin fingerprinting.

E2D Performance

Native API – Metadata Tags overhead:

- The inter-NF throughput using our zero-copy native API under two scenarios is measured
- In Header-Match, the E2D checks a particular header field against a configured value; no metadata tags are attached to packets.
- In Metadata-Match, the source NF adds tag for each packet; then the E2D then checks the tag against a configured value.
- Adding metadata lowers throughput by 5.7%.

Native API – URL tag test:

- Each packet must be forwarded to corresponding CDN NF based on URL in their header
- First case: tag packet with URL of CDN and let E2D resolve it
- Second case: add NF for splitting traffic to each CDN
- Second approach increase CPU load by 41%.

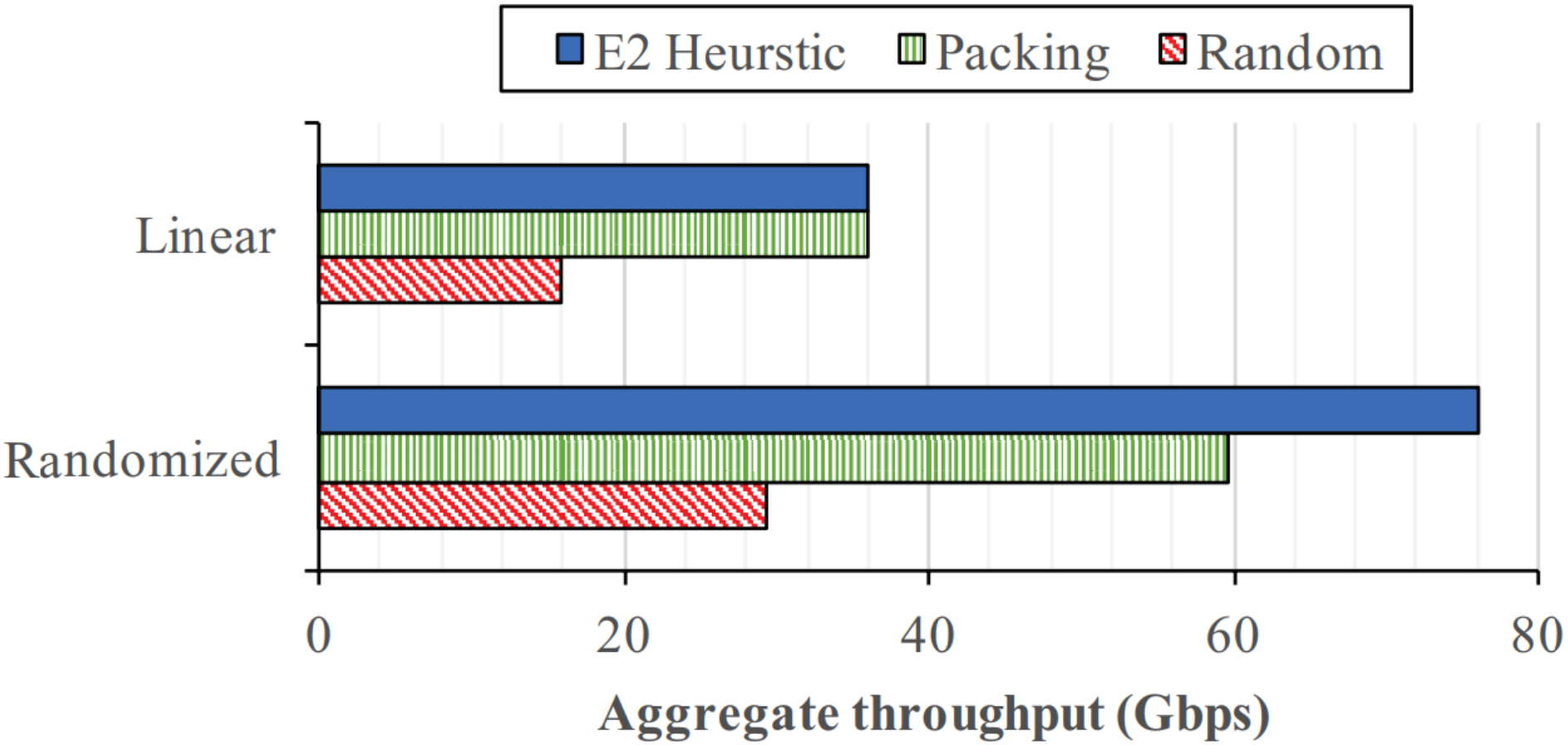
Placement algorithm performance

- Evaluated on 24 servers and 24 external ports.
- It takes 14.6ms to compute an initial placement for a 100-node iGraph
- Controller has a response time of 1.76ms when handling 68 split requests per second (which represents the aggressive case of one split request per core per second).
- Conclusion: centralized controller is unlikely to be a performance bottleneck in the system.

Placement algorithm – network performance

Types of iGraphs:

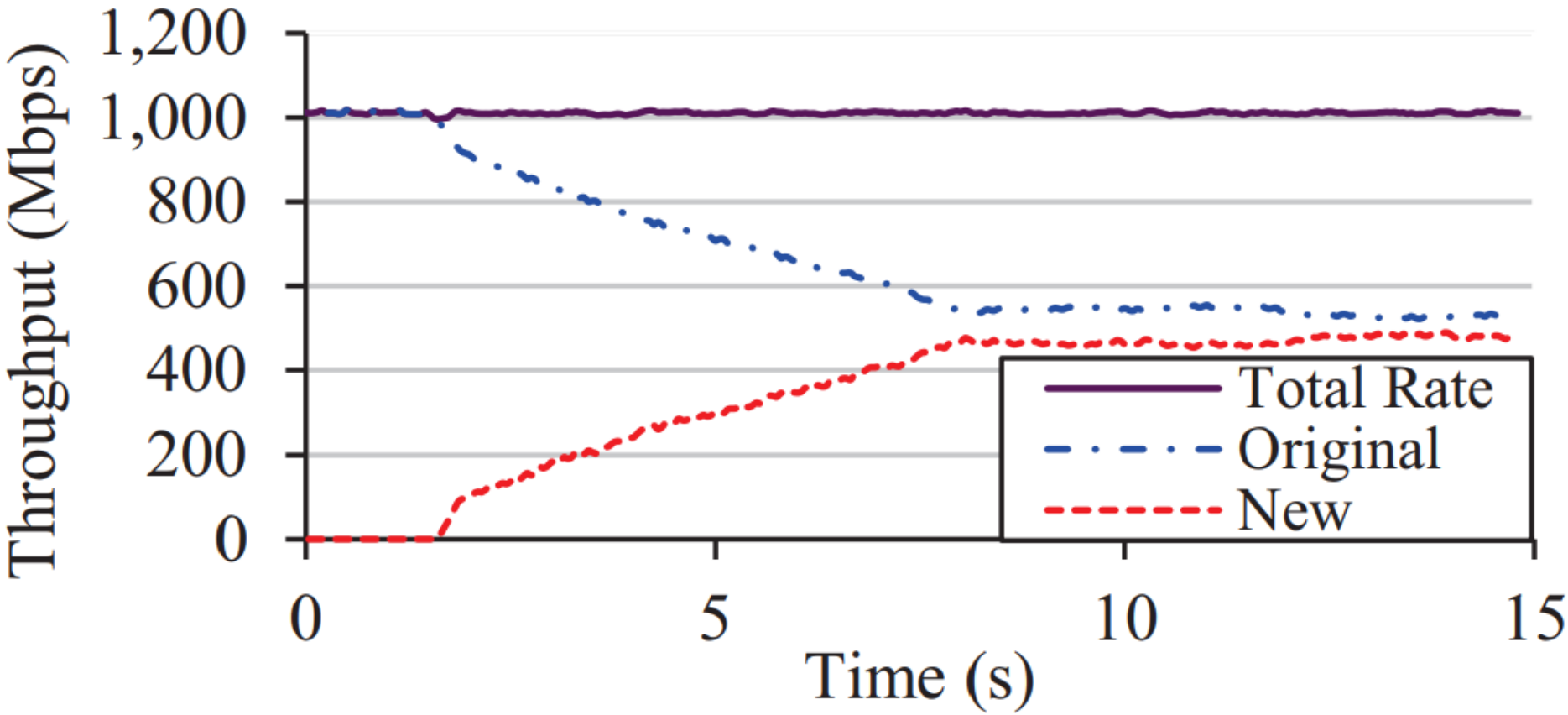
- Linear – linear chain of 5 NFs
- Randomized – 10 NFs with random edges



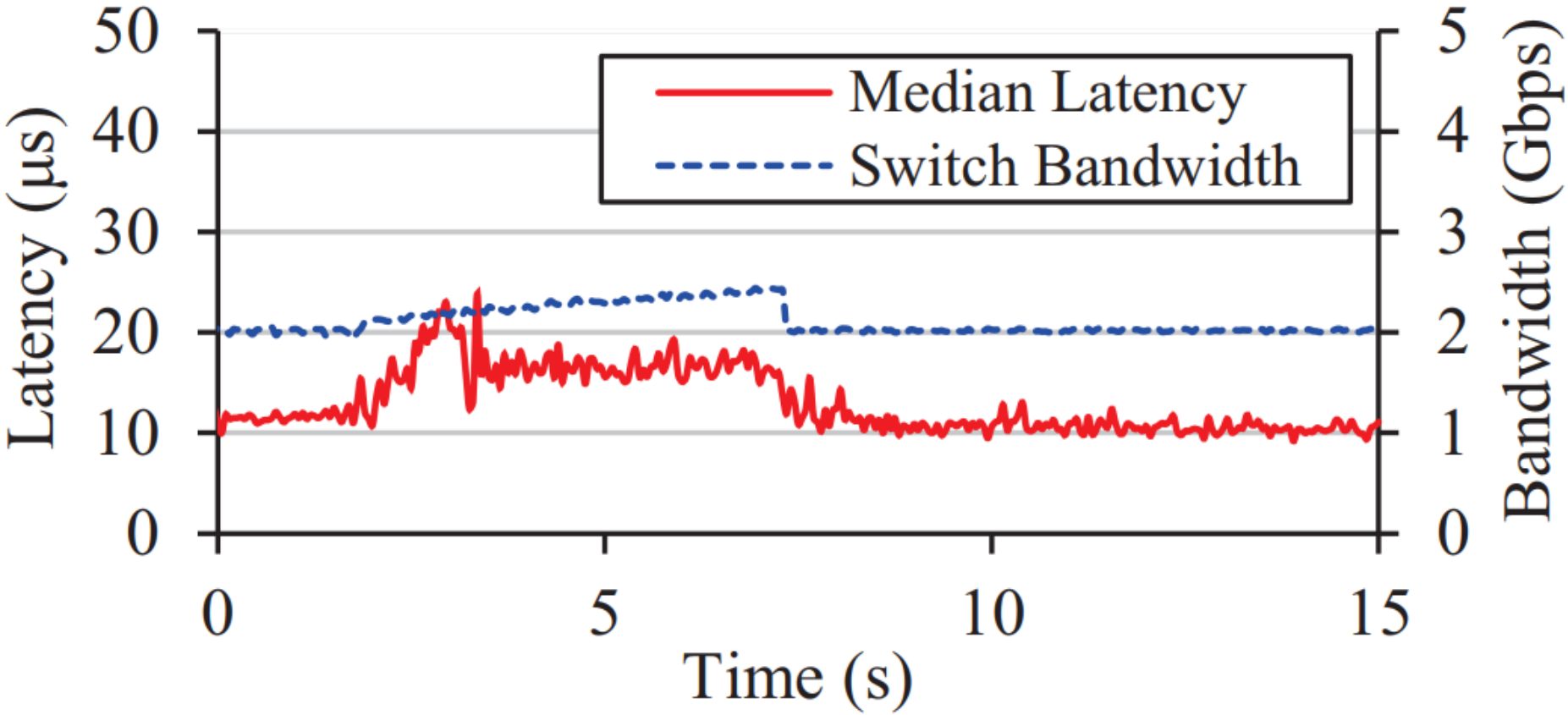
Performance of dynamic scaling

- 1 Gbps of input traffic
- 2000 new flows arriving each second on average
- 1 NF at the beginning of the test
- During test NF is split and flows are distributed

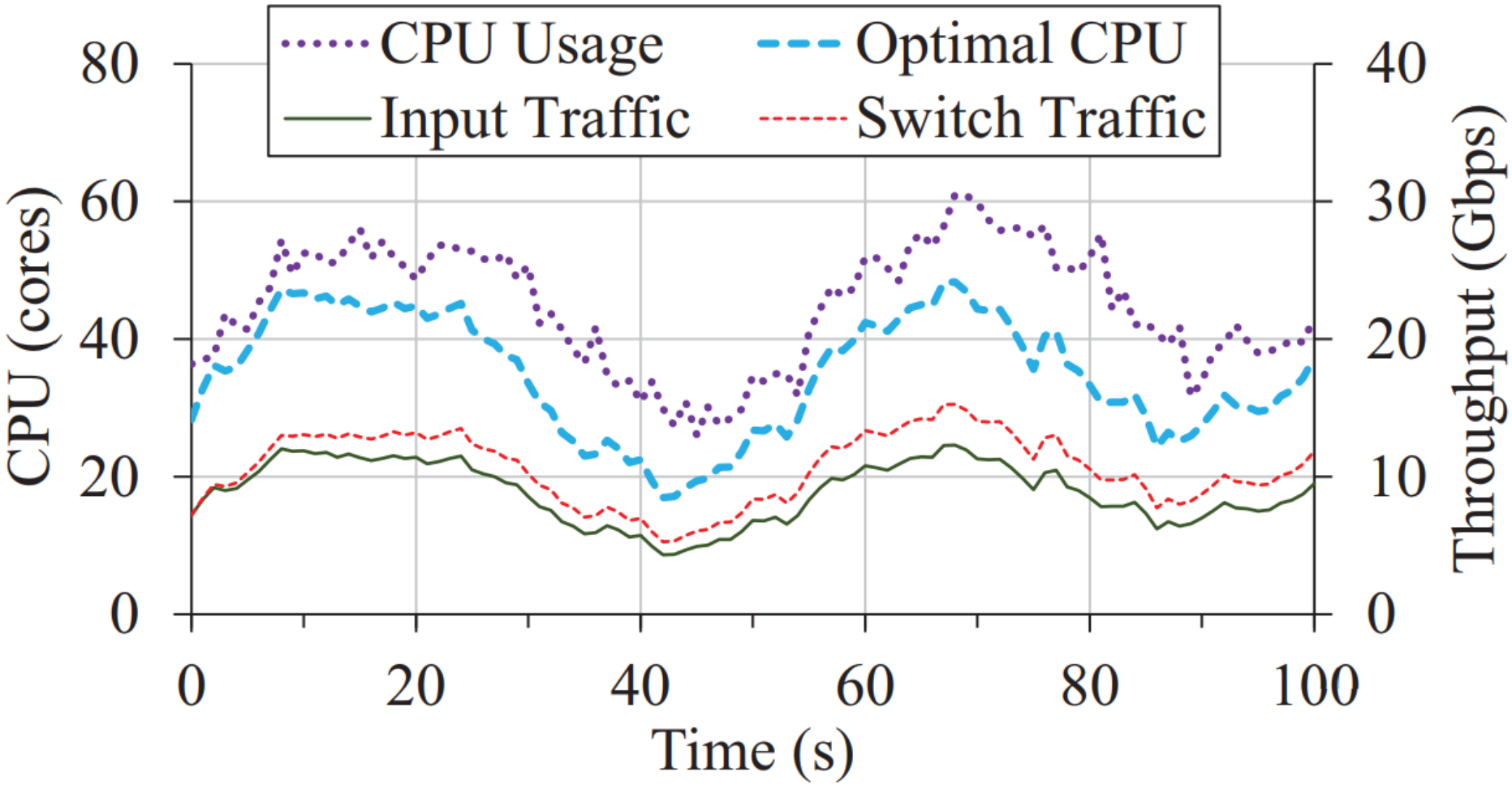
Performance of dynamic scaling



Performance of dynamic scaling



Overall performance – E2 under dynamic workload



Conclusion: E2's resource consumption scales dynamically to track the trend in input load. Avg CPU gap: 22.7%, net gap 16.4%.

Conclusions

- E2 provides the operator with a single coherent system for managing NFs
- E2 relieve developers from having to develop per-NF solutions for placement, scaling, fault-tolerance, and other functionality
- E2 did not impose undue overheads, and enabled flexible and efficient interconnection of NFs
- our placement algorithm performed substantially better than random placement and bin-packing
- our approach to splitting NFs with affinity constraints was superior to the competing approaches

Thanks for Your attention

Link to paper:
http://delivery.acm.org/10.1145/2820000/2815423/p121-palkar.pdf