

E2: A Framework for NFV Applications

Shoumik Palkar
UC Berkeley

Chang Lan
UC Berkeley

Aurojit Panda
UC Berkeley

Sylvia Ratnasamy
UC Berkeley

Keon Jang
Intel Labs

Luigi Rizzo
Università di Pisa

Sangjin Han
UC Berkeley

Scott Spencer
UC Berkeley
ICSI

by
Jean-Philippe Gauthier



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.

Primarily deployed for security and performance benefits.



Firewalls

Intrusion Detection Systems (IDS)

Intrusion Prevention Systems (IPS)



Proxy/Caches

WAN Optimizers

Protocol Accelerators

Middleboxes

- ❖ As more and more companies started to rely on middleboxes, more and more of them started to realize the problems they bring
- ❖ Expensive hardware
- ❖ Complex management



Network Function Virtualization

- ❖ The goal is to bring greater openness and agility to network dataplanes
- ❖ Inspired by cloud computing, NFV advocates moving Network Functions (NFs) out of their dedicated physical boxes to virtualized software applications that can be run on commodity hardware
- ❖ However, the current trend is to replace, on a one-to-one basis, the monolithic hardware by a monolithic software
- ❖ It fails to provide a coherent management solution for middleboxes
 - ❖ Operators still need to cope with NF-specific management systems
 - ❖ Developers may need to invent their own solutions for non-trivial tasks (scaling, fault-tolerance, ...)


Network Function Virtualization

- ❖ NFV needs a framework that implements general techniques for common issues. Similar to data analytics frameworks (Hadoop, Spark, Map Reduce)
- ❖ Placement
- ❖ Elastic scaling
- ❖ Service composition
- ❖ Resource isolation
- ❖ Fault tolerance
- ❖ Energy management
- ❖ ...

Elastic Edge (E2)

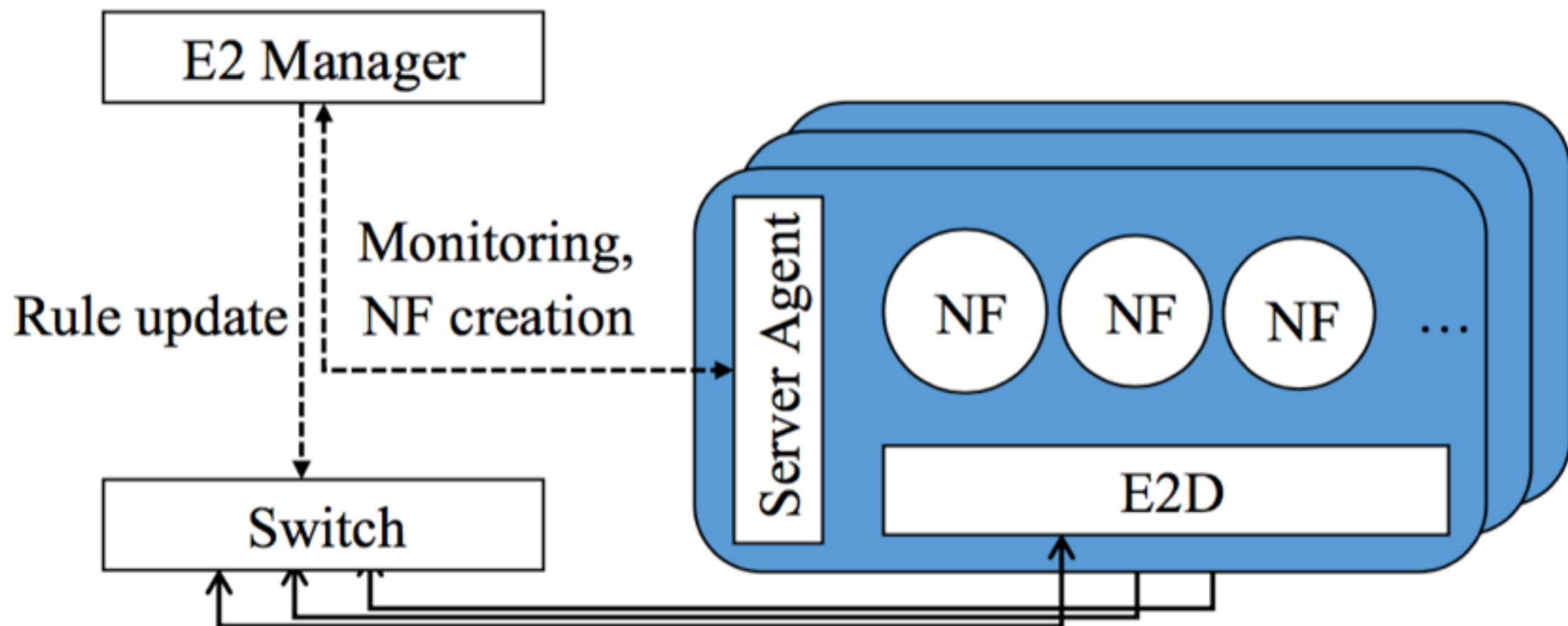
- ❖ A NFV management framework that
 - ❖ Allows developers to focus on their core application logic
 - ❖ Simplifies the operators responsibilities
- ❖ The framework will provide solutions for
 - ❖ Automate placement
 - ❖ Service interconnection
 - ❖ Dynamic scaling

Outline

- 
- Design of the system
 - E2 Dataplane
 - E2 Control Plane
 - Evaluation
 - Conclusion

System architecture

- ❖ E2 Manager orchestrates overall operations
- ❖ Server Agent manages operations within each server
- ❖ E2D acts as software traffic processing layer



Design considerations

- ❖ E2 is designed for a hardware infrastructure composed of general-purpose servers connected by commodity switches
- ❖ E2 is responsible for managing system resources
 - ❖ Must avoid over-booking CPU and NIC resources
 - ❖ Must avoid over-loading the switch capacity
 - ❖ Must avoid excessive use of the flow table

System workflow

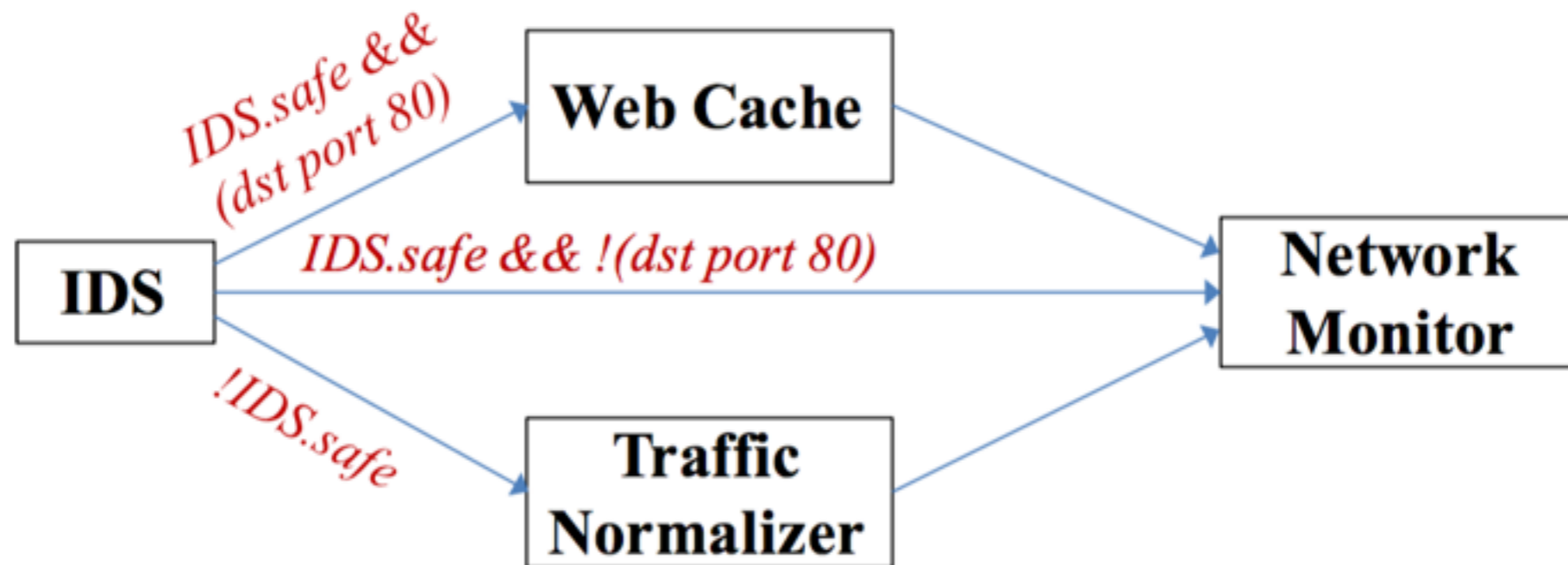
- ❖ An operator defines a set of network policies (called pipelets) to the global SDN controller
- ❖ The SDN controller hands the E2 Manager a set of pipelets
- ❖ The E2 Manager is responsible for executing these pipelets on the E2 cluster
- ❖ The Server Agent takes care of the local configurations
- ❖ NFs and E2D report back to the Server Agent (hardware failure, overload, ...)
- ❖ Server Agents report back to the E2 Manager which reports back to the SDN controller

Pipelets

- ❖ Pipelets are defined by the operators using a declarative language
- ❖ A pipelet describes how a particular traffic class should be processed and a corresponding directed acyclic graph (DAG)
- ❖ Traffic classes are defined in term of packet header fields and physical ports on the switch
- ❖ A node in the pipelet's DAG represents a NF or a physical port on the switch and edges describe the traffic between nodes
- ❖ Edges may be annotated with one or more traffic filters
- ❖ A traffic filter is a boolean expression that defines what subset of traffic from the source node should reach the destination node

Pipelets

- ❖ Conceptually, pipelets can be viewed as DAG, as called as policy graph (pGraph)



Pipelets

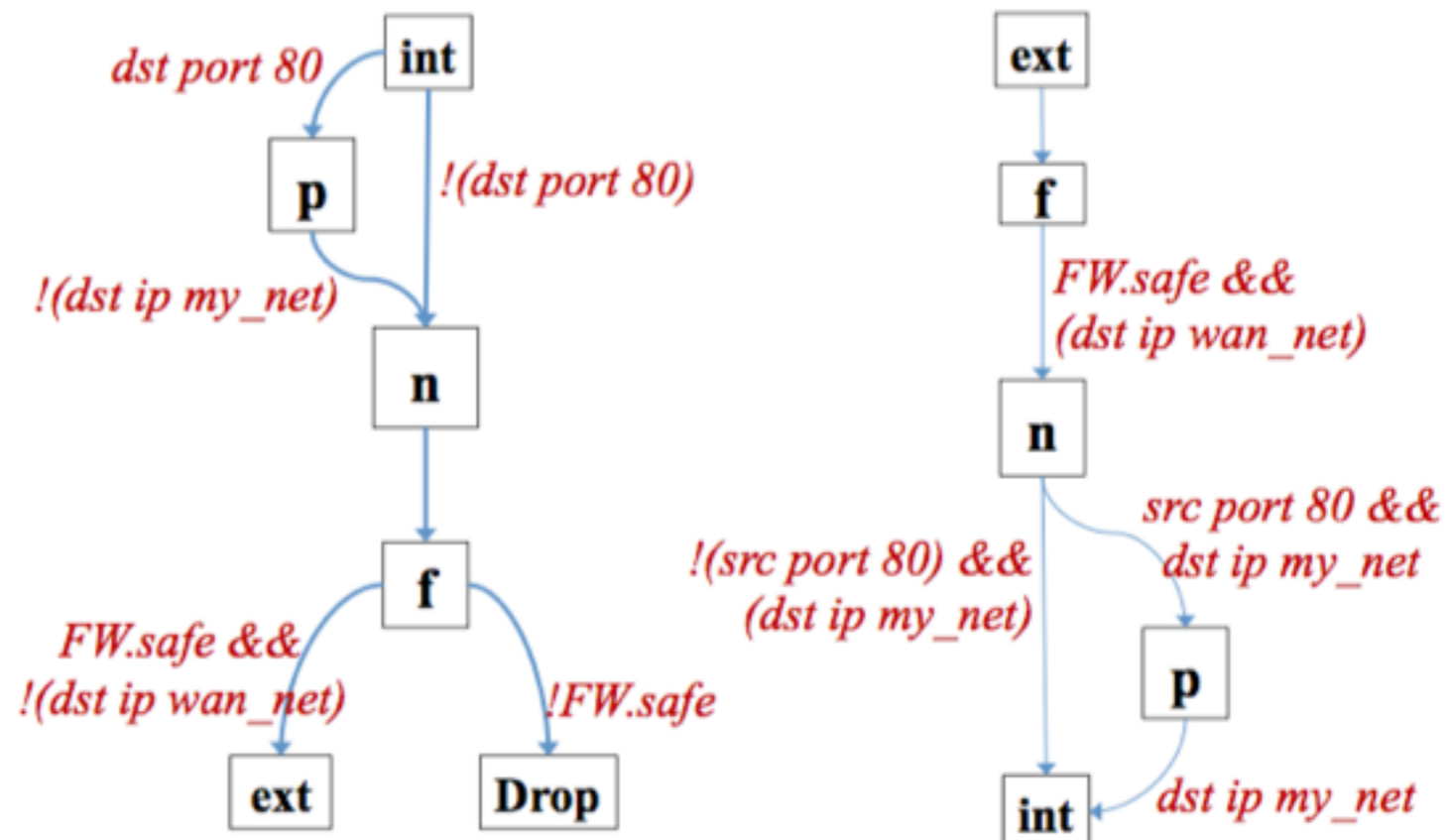
- ❖ Technically, pipelets are defined with a declarative policy language

```
// First, instantiate NFs from application types.
Proxy p;
NAT n;
FW f;
Port<0-7> int; // internal customer-facing ports
Port<8-15> ext; // external WAN-facing ports

// subnet declarations, to simplify filter writing
Address my_net 10.12.30.0/24; // private IP addr
Address wan_net 131.114.88.92/30; // public IP addr

pipelet { // outbound traffic
  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;
}

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



Pipelet for outbound traffic

Pipelet for inbound traffic

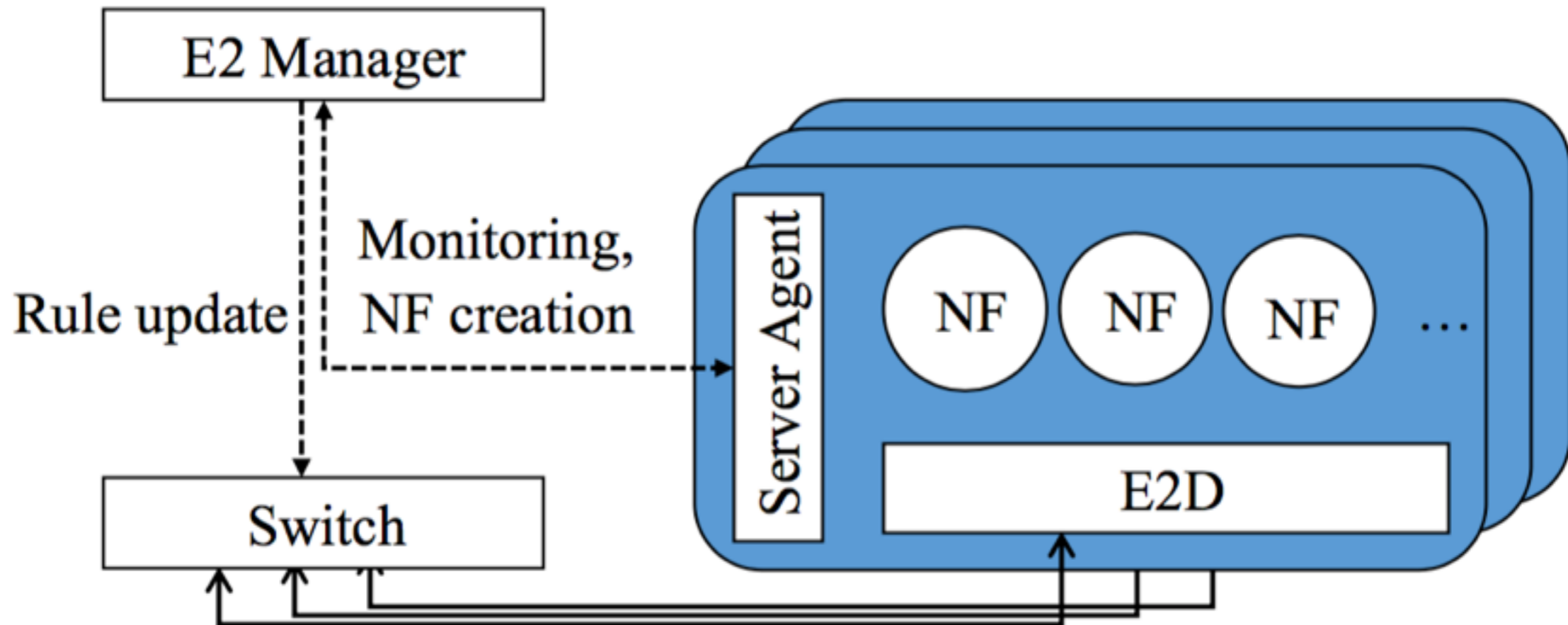
NF Description

- ❖ In addition to pipelets, E2 takes a NF description that guides the framework in configuring each NF
 - ❖ Native vs legacy
 - ❖ Attribute-Method binding
 - ❖ Scaling constraints
 - ❖ Affinity constraints
 - ❖ NF performance

Hardware description

- ❖ It also takes a hardware description that tells the framework what resources are available
- ❖ Number of cores available
- ❖ Network I/O bandwidth
- ❖ Number of switch ports
- ❖ Number of entries in the switch flow table
- ❖ Available switch actions
- ❖ ...

The E2 Dataplane (E2D)



The E2 Dataplane (E2D)

- ❖ The goal of E2D is to provide flexible yet efficient plumbing across NF instances in the pGraph
- ❖ The implementation is based on SoftNIC (a software-based NIC developed by researchers from Berkeley)
- ❖ It allows arbitrary packet processing modules to be configured as data flow graph
- ❖ E2 doesn't use OVS because NFV doesn't share many design considerations. It would have required a lot of work to adapt and possibly breaking changes.

SoftNIC

- ❖ SoftNIC exposes virtual NIC ports (vports) to NF instances
- ❖ Between vports and pports (physical ports), packet processing modules can be configured as a data flow graph
- ❖ SoftNIC uses Intel DPDK (Data Plane Development Kit) for low-overhead I/O to hardware NICs and uses batch processing
- ❖ It also runs on a small number of dedicated processors for high throughput and low latency (better use of CPU caches and removes context-switches)
- ❖ Can process up to 40 Gbps of data per core

Extending SoftNIC for E2D

- ❖ New modules were developed for E2D (load monitoring, flow tracking, packet classification, load balancing, tunnelling across NFs)
- ❖ Development of a native API that NFs can use to improve system wide performance and modularity
- ❖ Provides support for zero-copy packet transfer over vports
- ❖ Rich message abstraction
 - ❖ Reconstructed TCP bytestream (remove redundant overhead)
 - ❖ Per-packet metadata tags that accompany the packet
 - ❖ Inter NF notifications (notify to block traffic from IPS to FW)

Extending SoftNIC for E2D

- ❖ Exposes a control API to E2's Server Agent
- ❖ Dynamically create or destroy vports
- ❖ Add/remove modules in E2D's packet processing pipeline
- ❖ Receive notifications of NF overload or failure from E2D

The E2 Control Plane

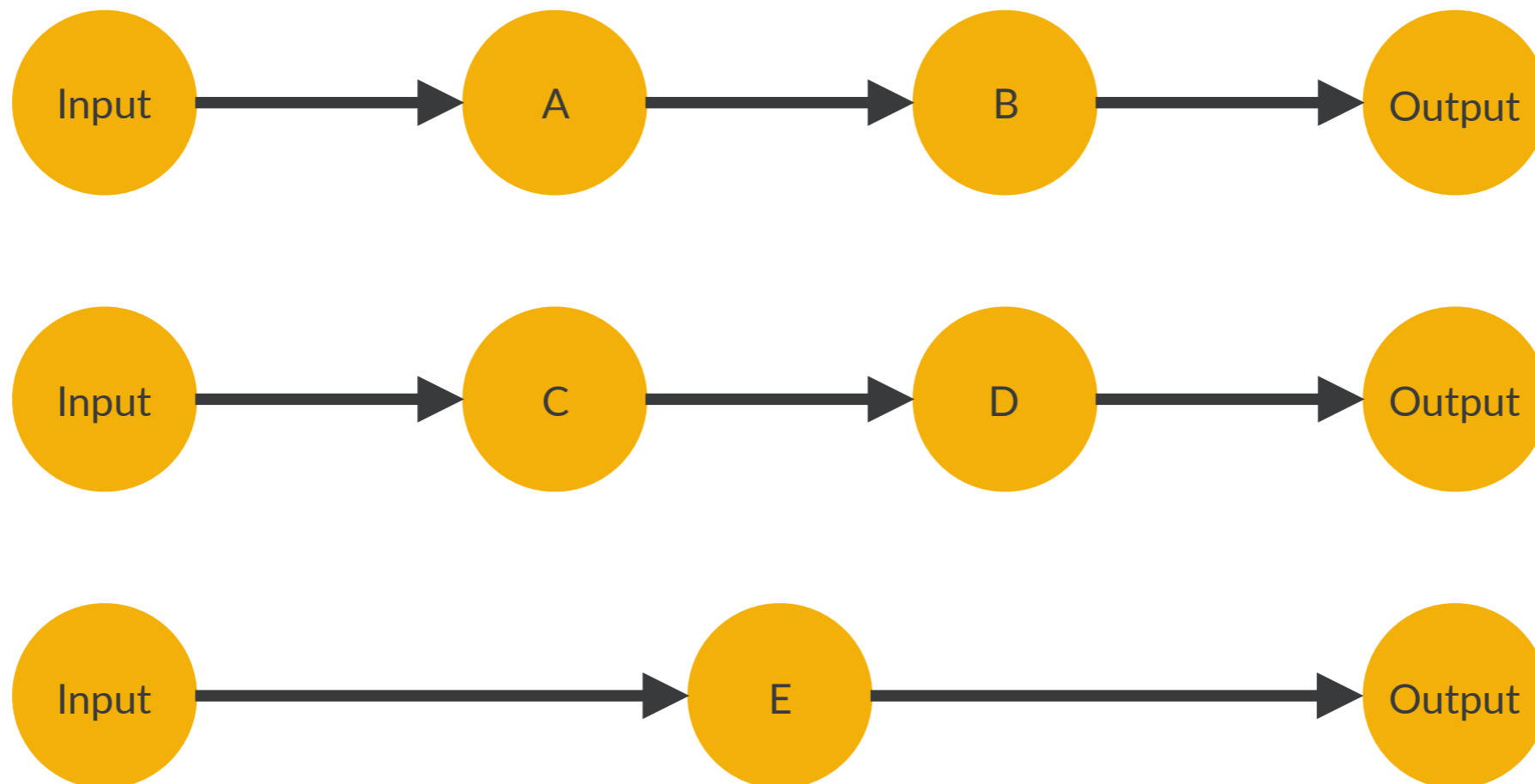
- ❖ In charge of
 - ❖ Placement (instantiation of pipelets on servers)
 - ❖ Interconnection (setting up and configuring the interconnections between NFs)
 - ❖ Scaling (dynamically adapting the placement decisions depending on load variations)
 - ❖ Ensuring affinity constraints of NFs

NF Placement

- ❖ The initial placement of NFs involves 5 steps
 1. Merging pipelets into a single policy graph
 2. Sizing
 3. Converting the pGraph to an iGraph
 4. Instance placement
 5. Offloading to hardware switch

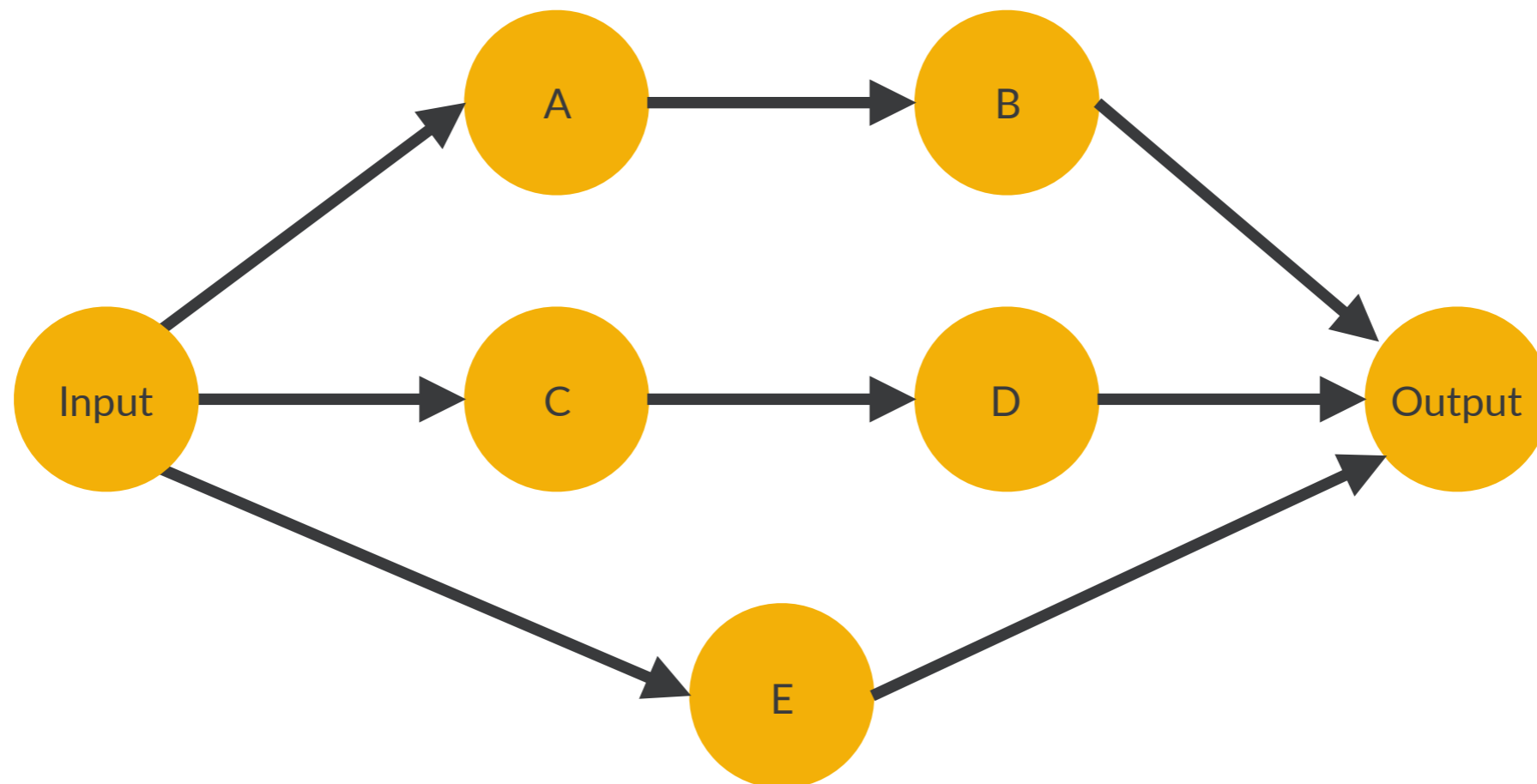
NF Placement - Step 1 (Merging pipelets)

- ❖ Combine the set of input pipelets into a single policy graph (pGraph)
- ❖ Simply the union of all the pipelets



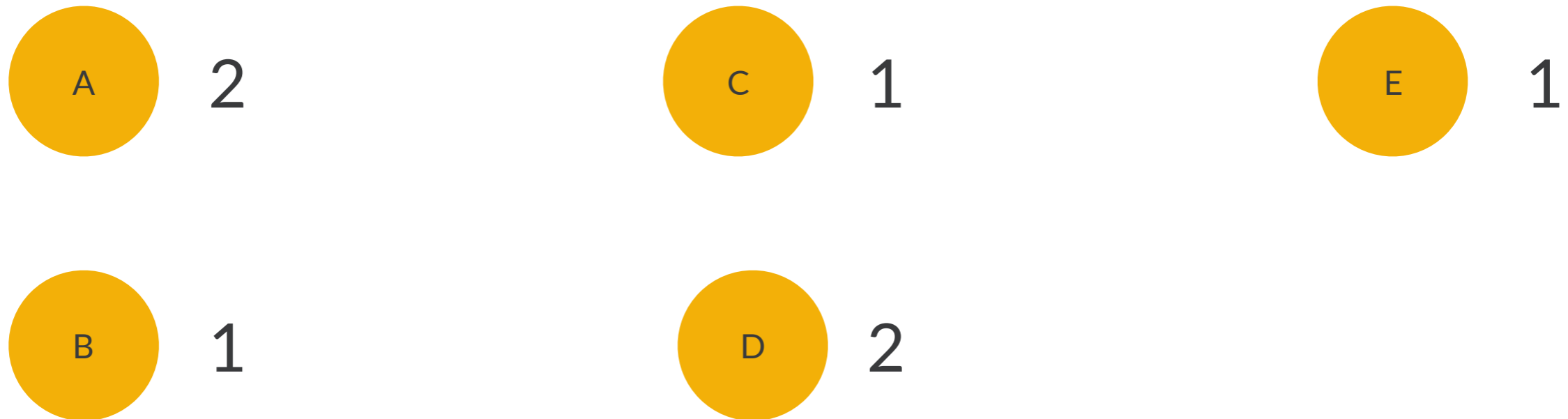
NF Placement - Step 1 (Merging pipelets)

- ❖ Combine the set of input pipelets into a single policy graph (pGraph)
- ❖ Simply the union of all the pipelets



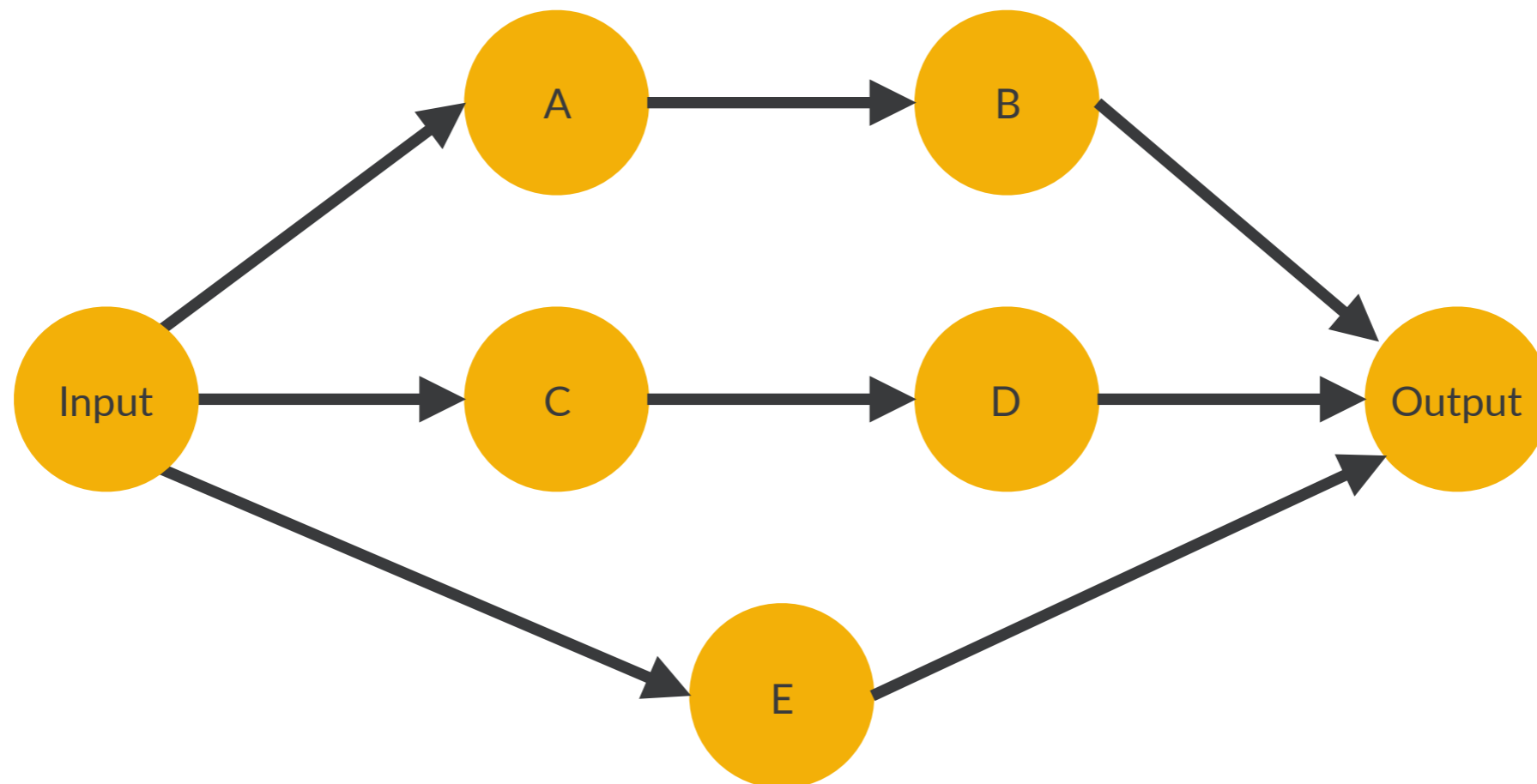
NF Placement - Step 2 (Sizing)

- ❖ Use the initial estimate of the load on a NF and its per-core capacity from the NF description to find how many instances (running on separate cores) should be allocated to it
- ❖ The sizing step doesn't need to be accurate, we only need a starting point and the dynamic scaling will take care of the rest



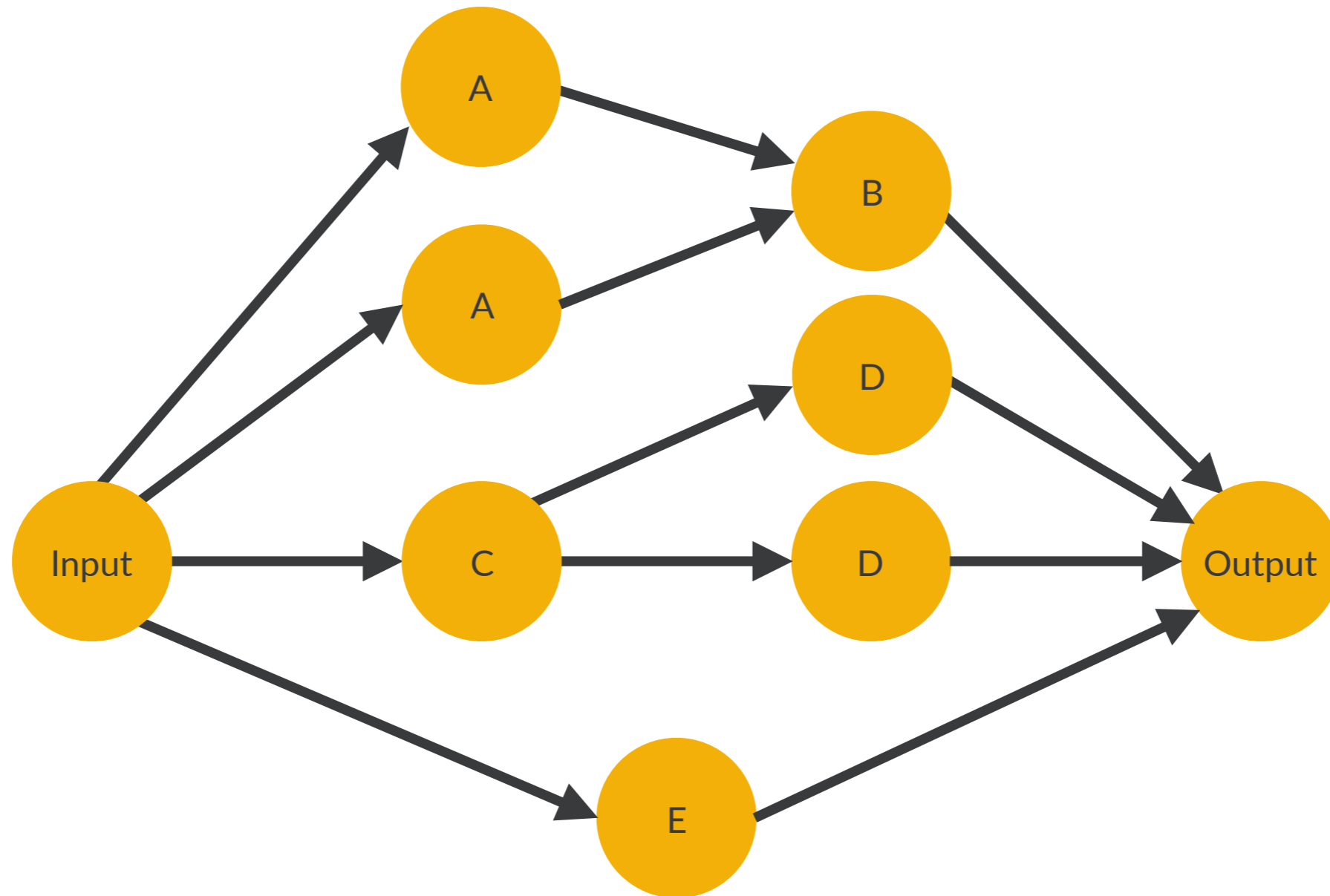
NF Placement - Step 3 (pGraph to iGraph)

- ❖ Transform the policy graph (pGraph) into an instance graph (iGraph)
- ❖ Uses the size of each NFs from step 2



NF Placement - Step 3 (pGraph to iGraph)

- ❖ Transform the policy graph (pGraph) into an instance graph (iGraph)
- ❖ Uses the size of each NFs from step 2



NF Placement - Step 4 (Instance Placement)

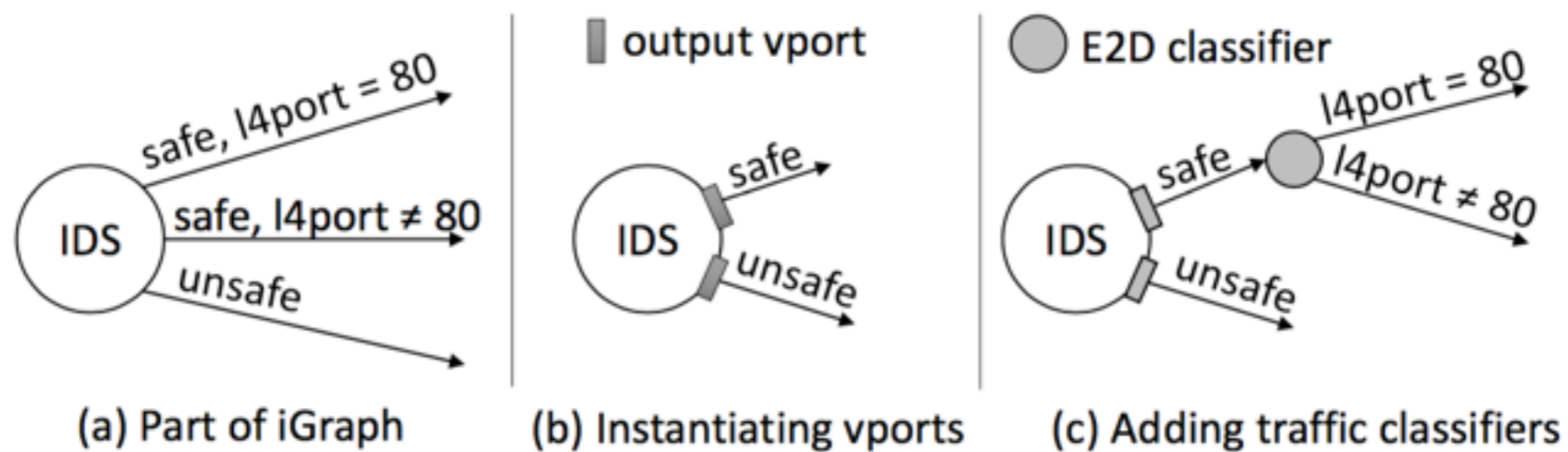
- ❖ Maps each NF instance to a particular server
- ❖ The goal is to minimize inter-server traffic
 - ❖ Forwarding within a single server incurs lower delay and consumes less processor cycles
 - ❖ Link bandwidth between servers and the switch is a limited resource
- ❖ Optimization problem → Graph Partition Problem → NP-hard
- ❖ Iterative local searching algorithm (modified form of Kernighan-Lin)

NF Placement - Step 5 (Offloading to the hardware switch)

- ❖ Commodity switch ASICs implement various low-level features
- ❖ Possibility to offload some functions, but resources are limited
- ❖ E2 uses an opportunistic approach
 - ❖ NF is considered as a candidate to offloading to the switch only if, at the end of the placement, that NF is adjacent to a switch port, and the switch has available resources to run it
- ❖ The current prototype doesn't focus on offloading

Service Interconnection

- ❖ Recall that edges in pGraph are annotated with filters
- ❖ Three stages
 - ❖ Instantiating NF's ports
 - ❖ Adding traffic filters
 - ❖ Configuring the switch and E2D

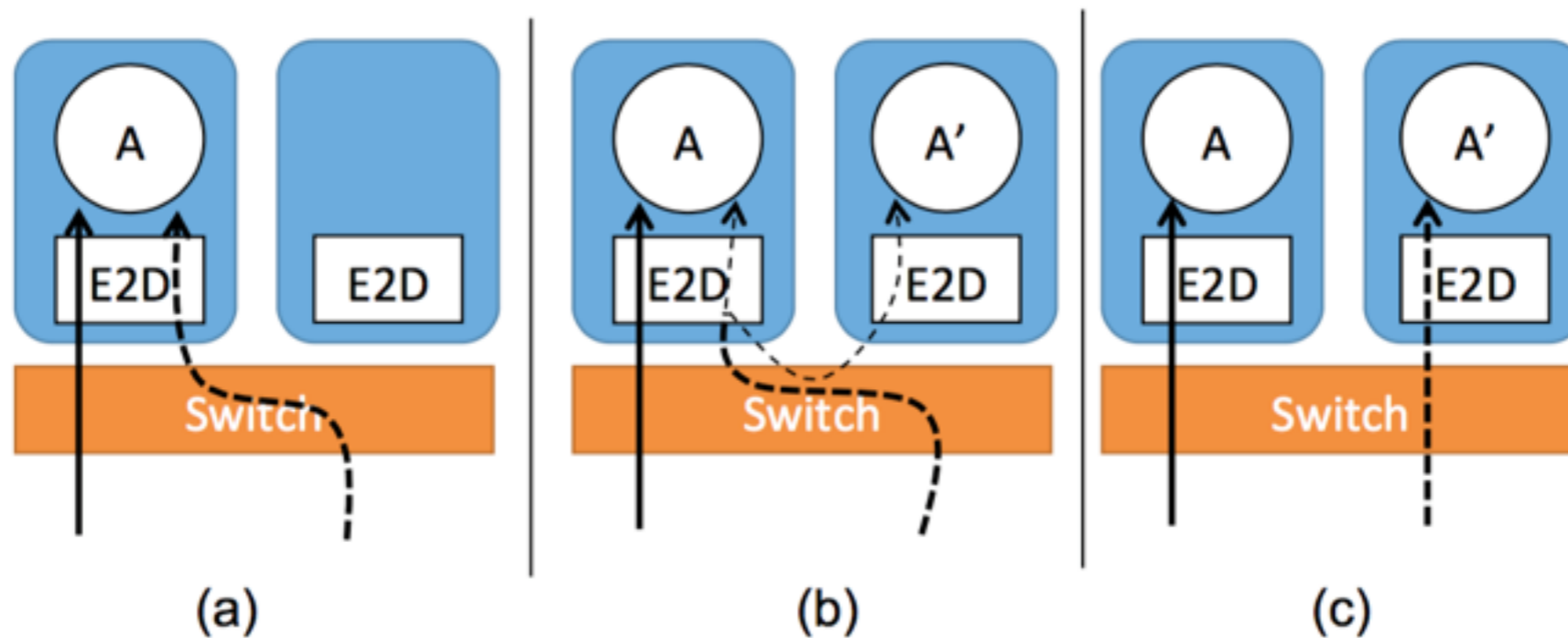


Dynamic scaling

- ❖ Some NFs are stateful and require affinity (traffic for a given flow must reach the instance that holds the flow's state)
- ❖ Traditional approaches
 - ❖ State migration (moving the state to another instance)
 - ❖ Expensive and incompatible with legacy applications
- ❖ E2 uses a novel migration avoidance strategy in which hardware and software act in concert to maintain affinity

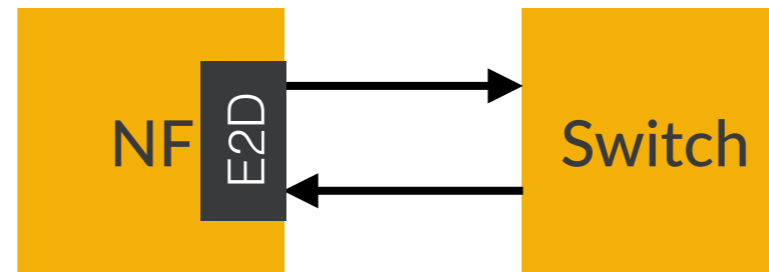
Migration avoidance for flow affinity

- ❖ Each flow can be mapped to a flowID (hash function on relevant header fields)
- ❖ Each instance as a range ($[X, Y)$)
 - ❖ When overloaded, the range is split ($[X, M)$ and $[M, Y)$)
- ❖ Need to take care of the old flows (exception flows)

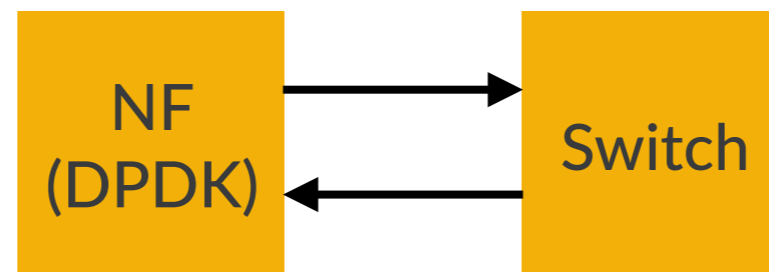


E2D Performance Evaluation

- ❖ A simple forwarding test



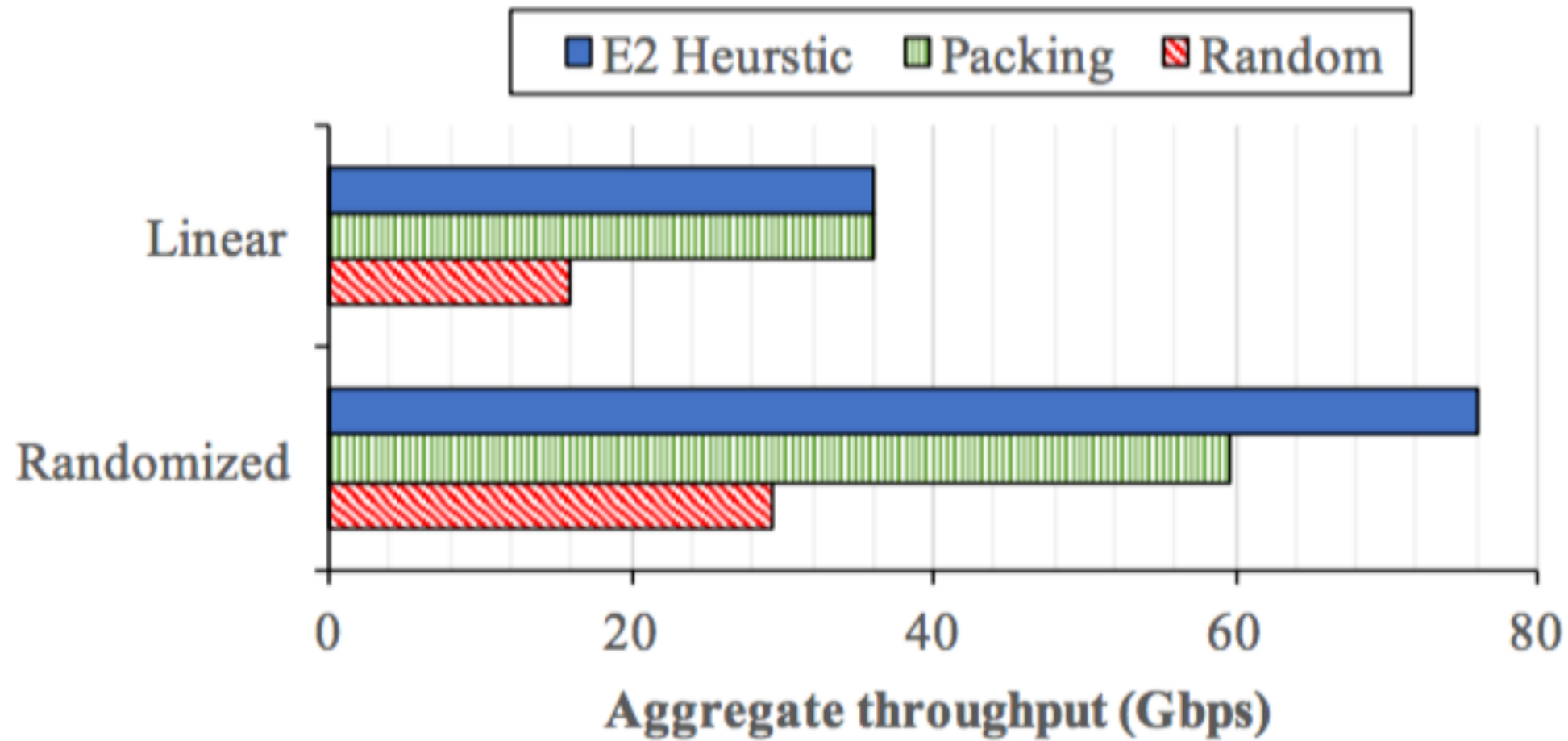
Average of $4.91\mu\text{s}$



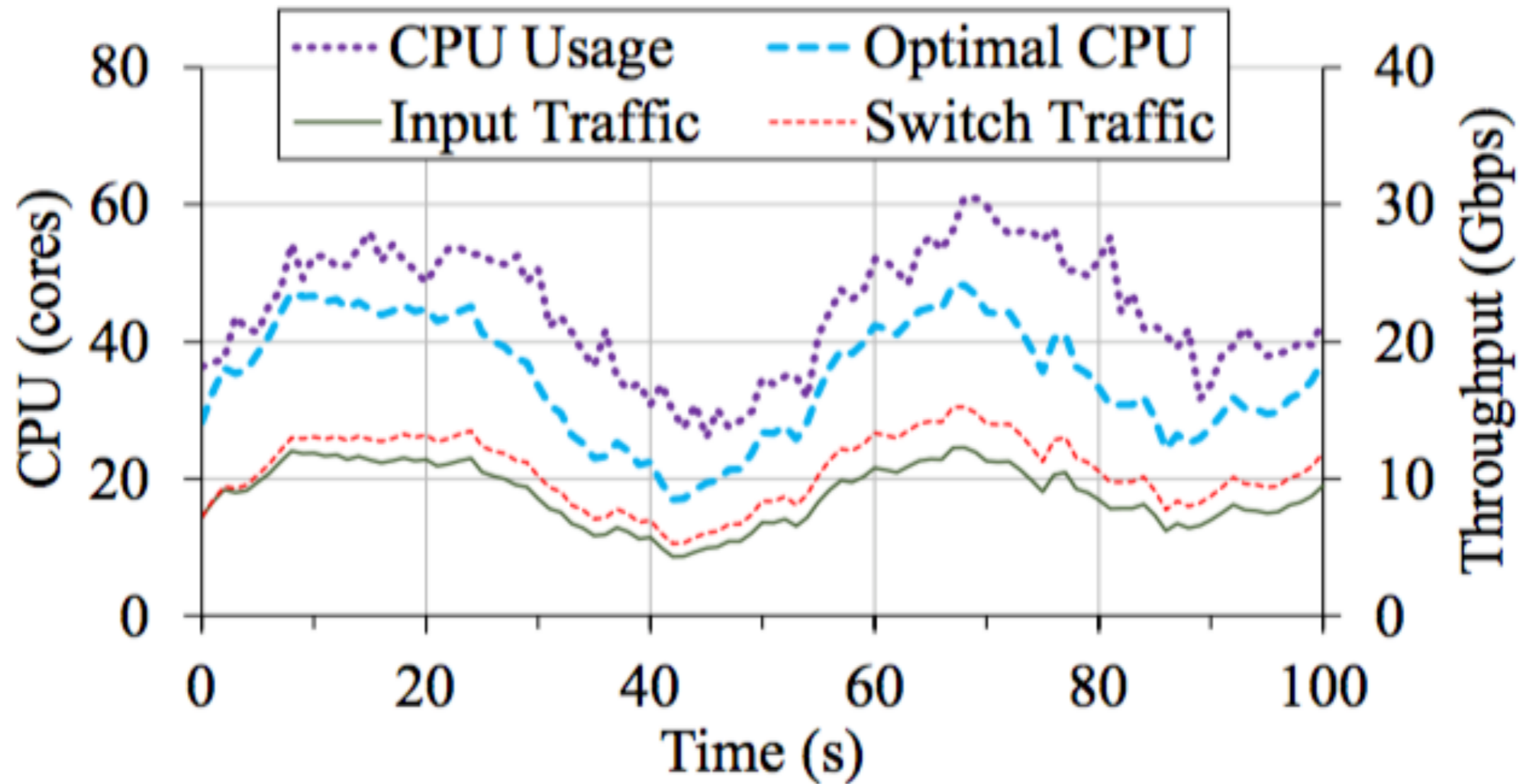
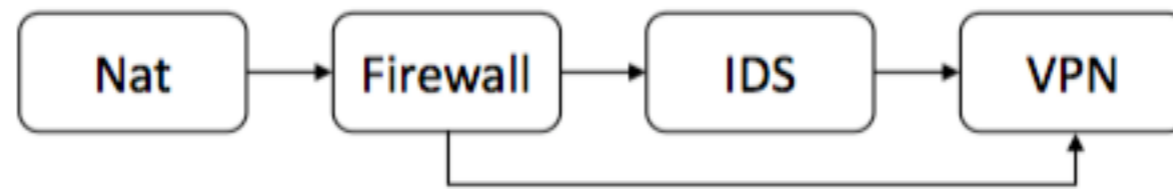
Average of $4.61\mu\text{s}$

- ❖ Overhead of $0.15\mu\text{s}$ (for each direction)

Placement algorithm performance evaluation



Overall performance evaluation



Conclusion

- ❖ E2, a management framework for NFV
- ❖ It provides the operator with a single coherent management system
- ❖ It takes care of the placement, scaling, service interconnection and other functionalities
- ❖ E2 doesn't impose undue overheads and enable flexible and efficient interconnection between NFs.
- ❖ The placement algorithm performs better than traditional approaches
- ❖ The migration avoidance strategy has a lot of potential