DevoFlow: Scaling Flow Management for High-Performance Networks

Andy Curtis
Jeff Mogul
Jean Tourrilhes
Praveen Yalagandula
Puneet Sharma
Sujata Banerjee

WATERLOO
CHERITON SCHOOL OF
COMPUTER SCIENCE

Software-defined networking

Software-defined networking

- Enables programmable networks

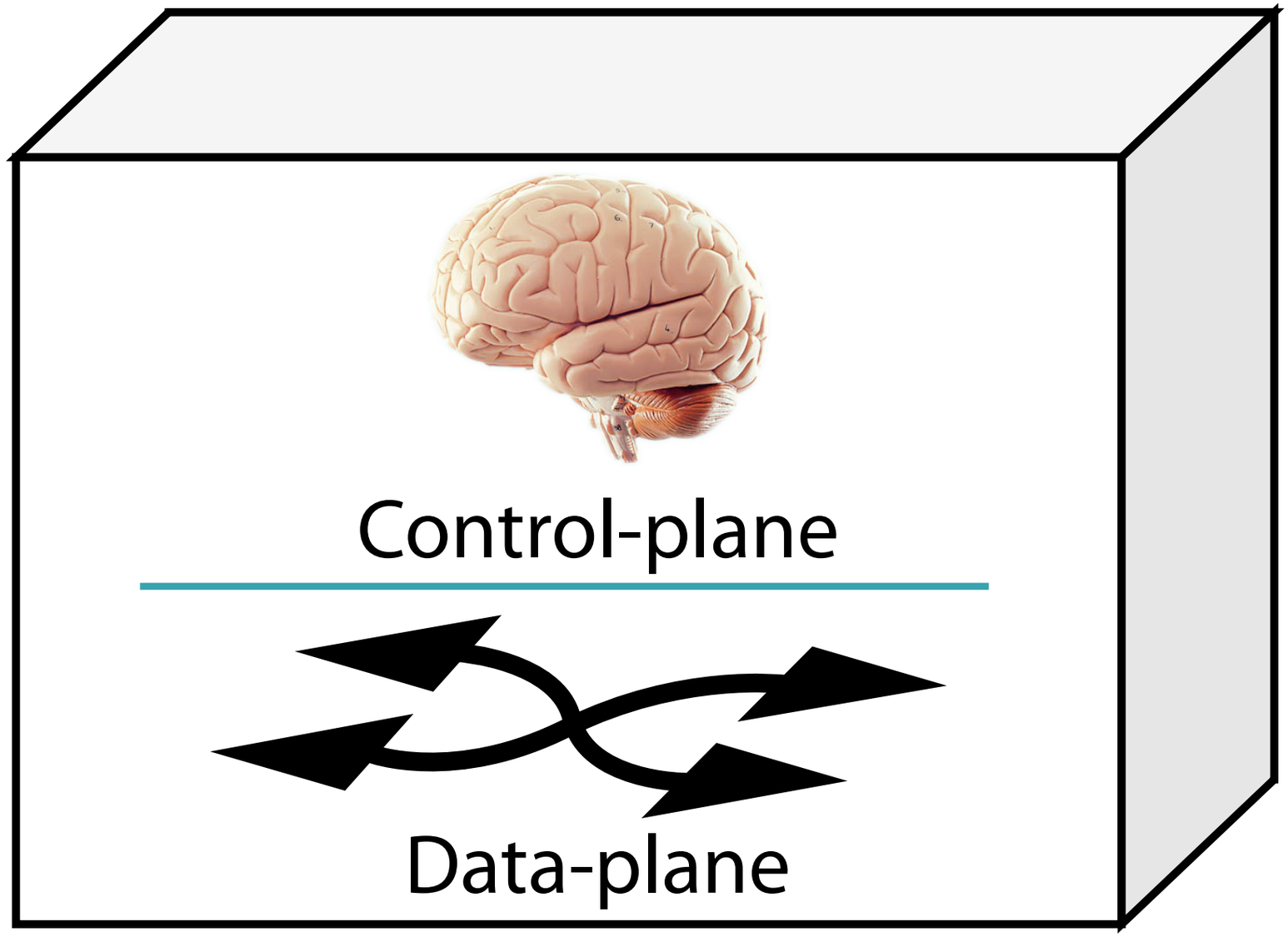
Software-defined networking

- Enables programmable networks
- Implemented by OpenFlow

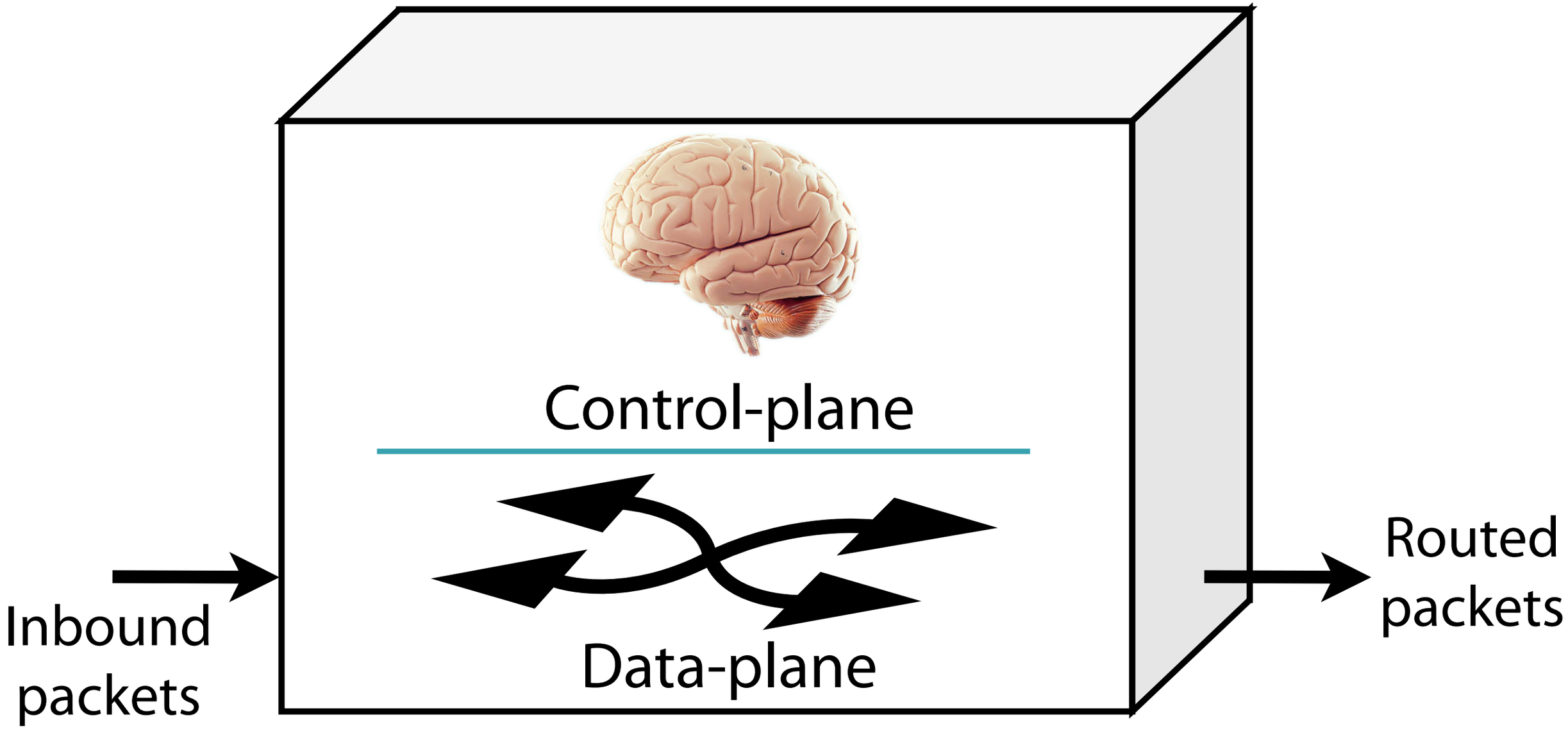
Software-defined networking

- Enables programmable networks
- Implemented by OpenFlow
- OpenFlow is a great concept, but...
  - its original design imposes excessive overheads

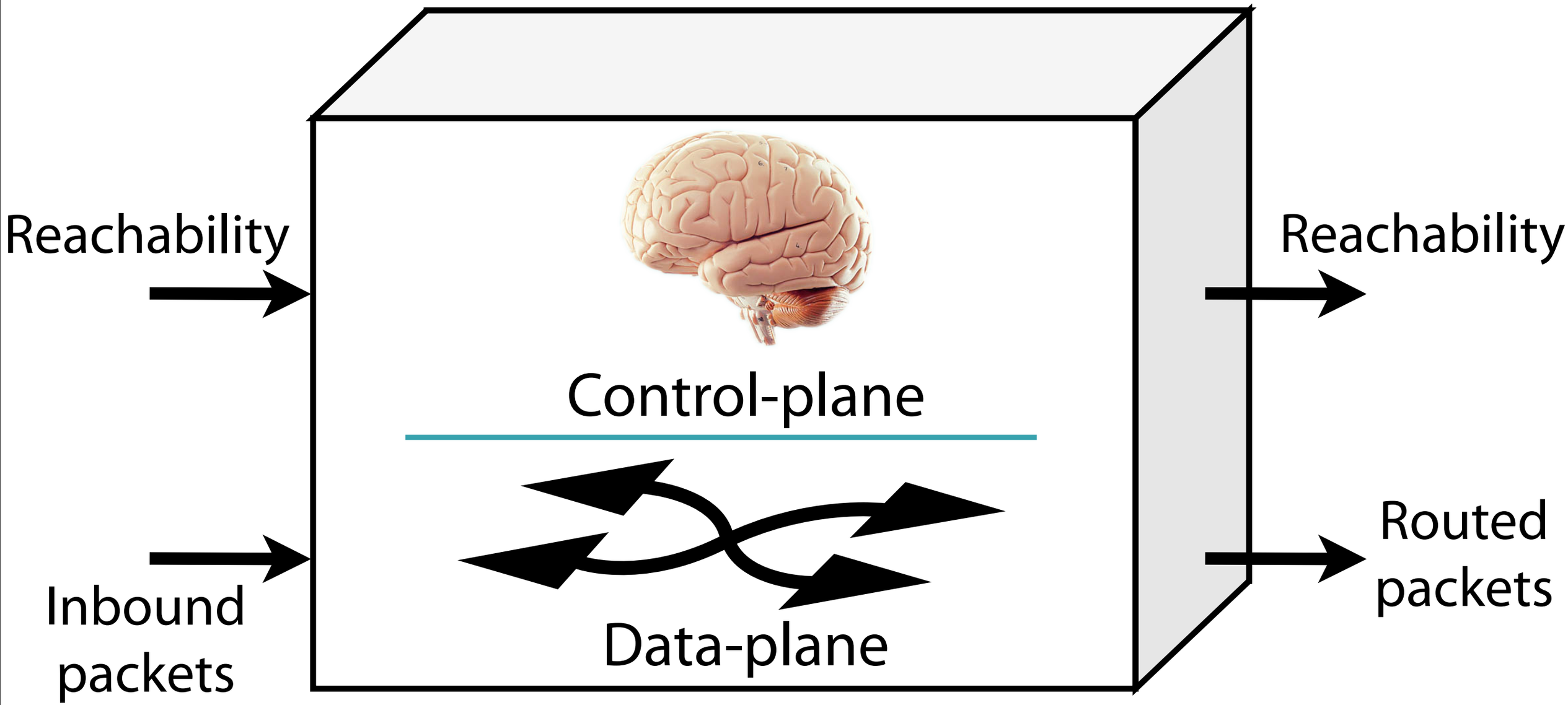
Traditional switch

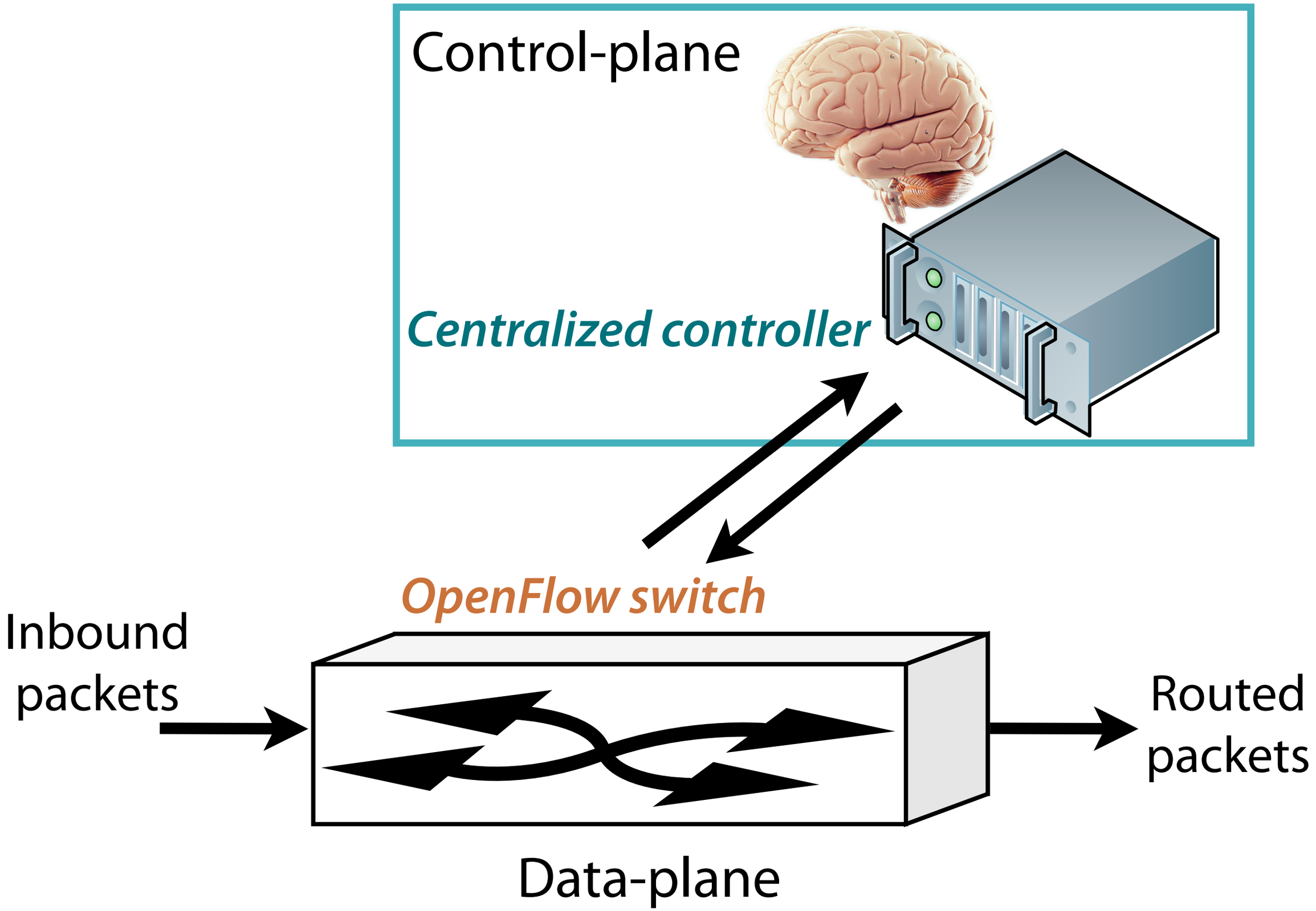


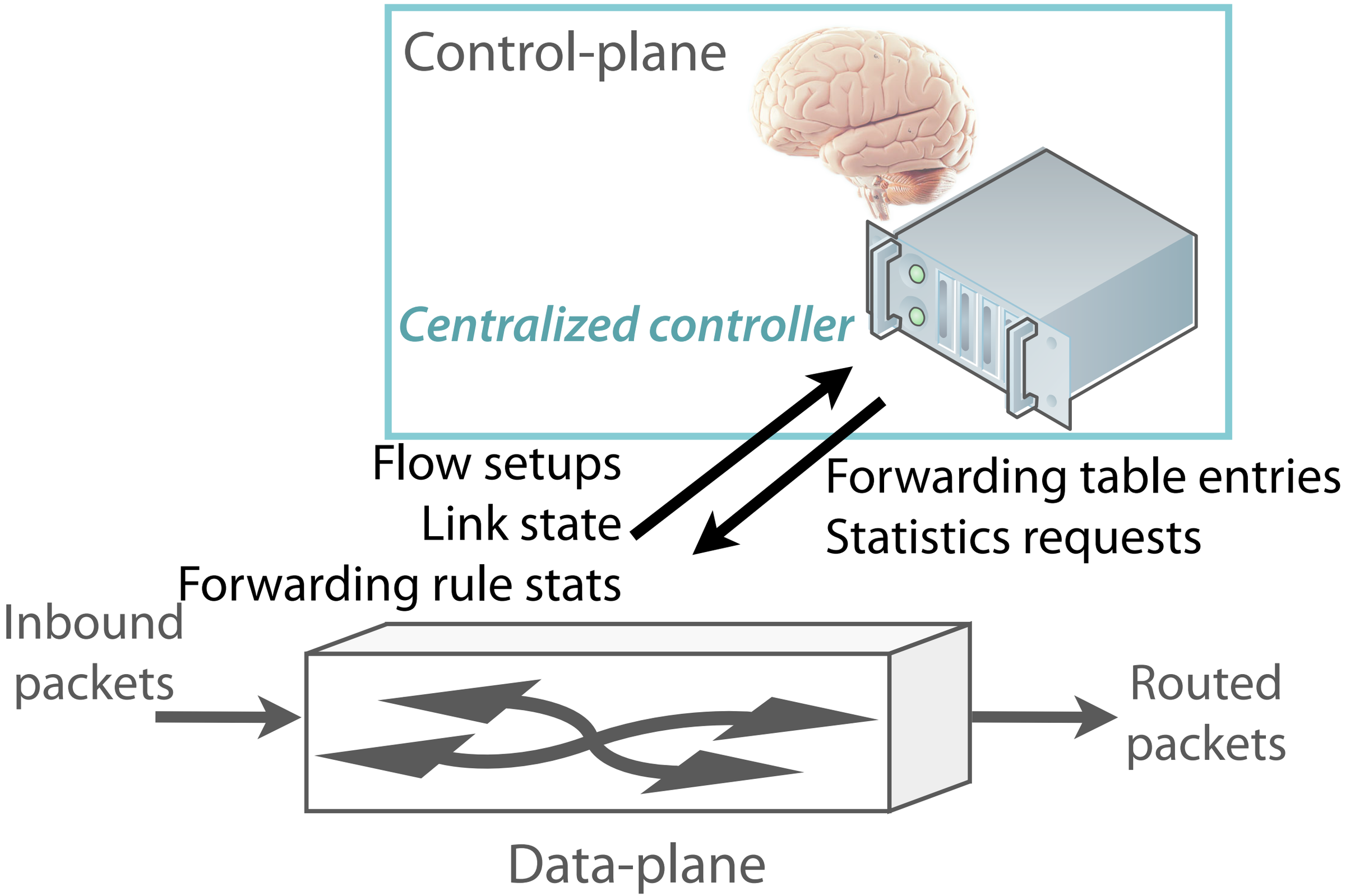
Traditional switch



Traditional switch




Control-plane
Centralized controller
OpenFlow switch
Inbound packets
Routed packets
Data-plane


Control-plane
Centralized controller
Flow setups
Link state
Forwarding rule stats
Forwarding table entries
Statistics requests
Inbound packets
Routed packets
Data-plane

OpenFlow enables innovative management solutions

OpenFlow enables innovative management solutions

- Consistent routing and security policy enforcement [Ethane, SIGCOMM 2007]
- Data center network architectures like VL2 and PortLand [Tavakoli et al. Hotnets 2009]
- Client load-balancing with commodity switches [Aster*x, ACLD demo 2010; Wang et al., HotICE 2011]
- Flow scheduling [Hedera, NSDI 2010]
- Energy-proportional networking [ElasticTree, NSDI 2010]
- Automated data center QoS [Kim et al., INM/WREN 2010]

But OpenFlow is not perfect...

- Scaling these solutions to data center-sized networks is challenging

Contributions

- Characterize overheads of implementing OpenFlow in hardware
- Propose DevoFlow to enable cost-effective, scalable flow management
- Evaluate DevoFlow by applying it to data center flow scheduling

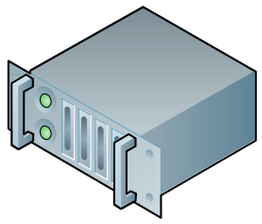
Contributions

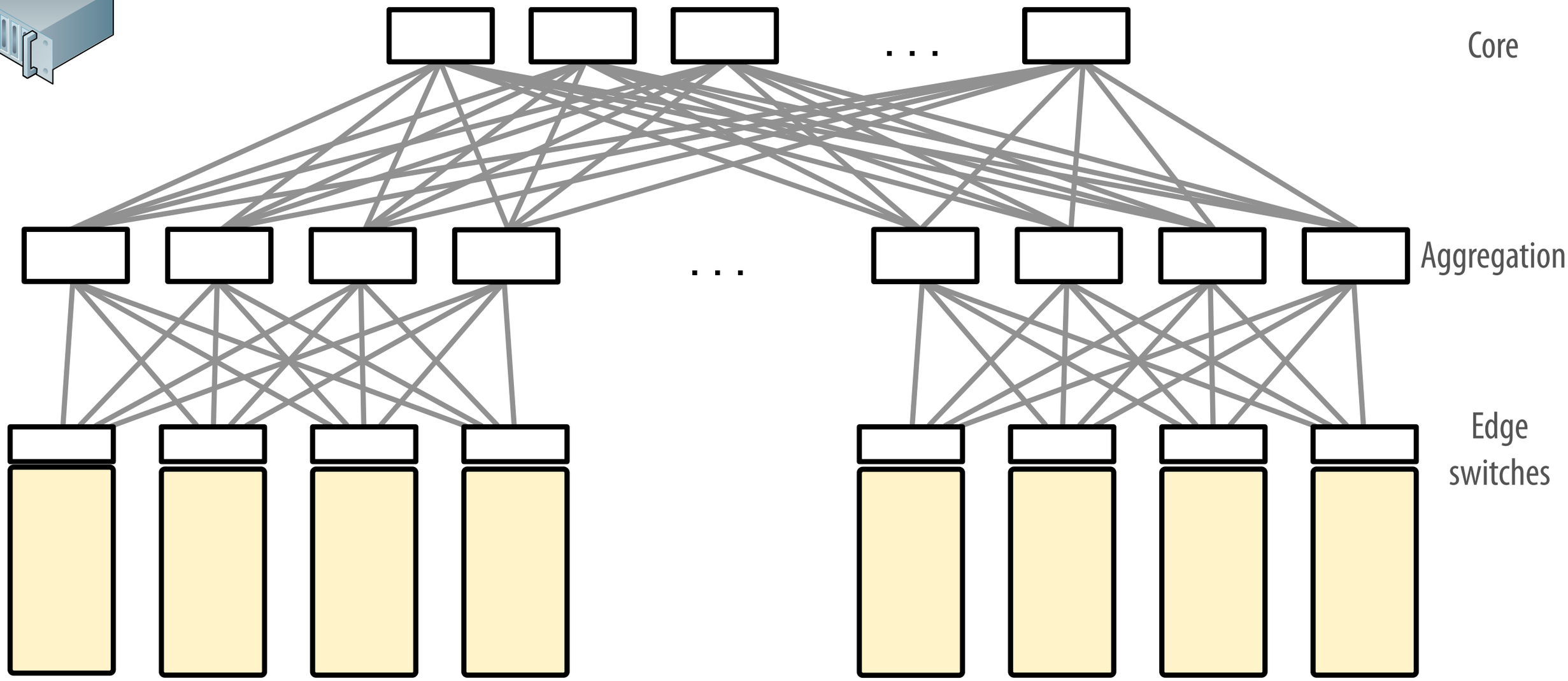
- ***Characterize overheads of implementing OpenFlow in hardware***

Experience drawn from implementing OpenFlow on HP ProCurve switches

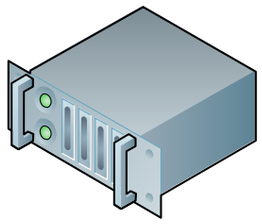
OpenFlow couples flow setup with visibility

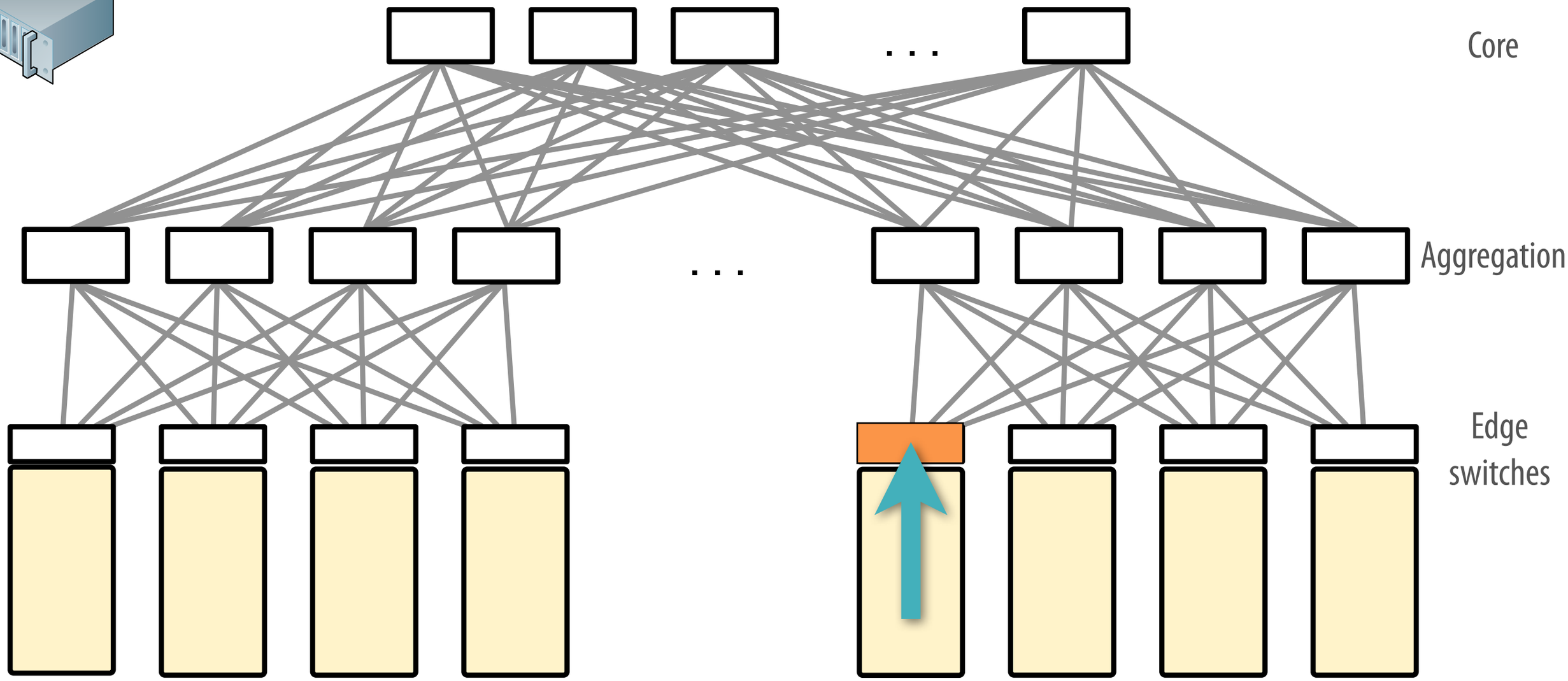
Central controller





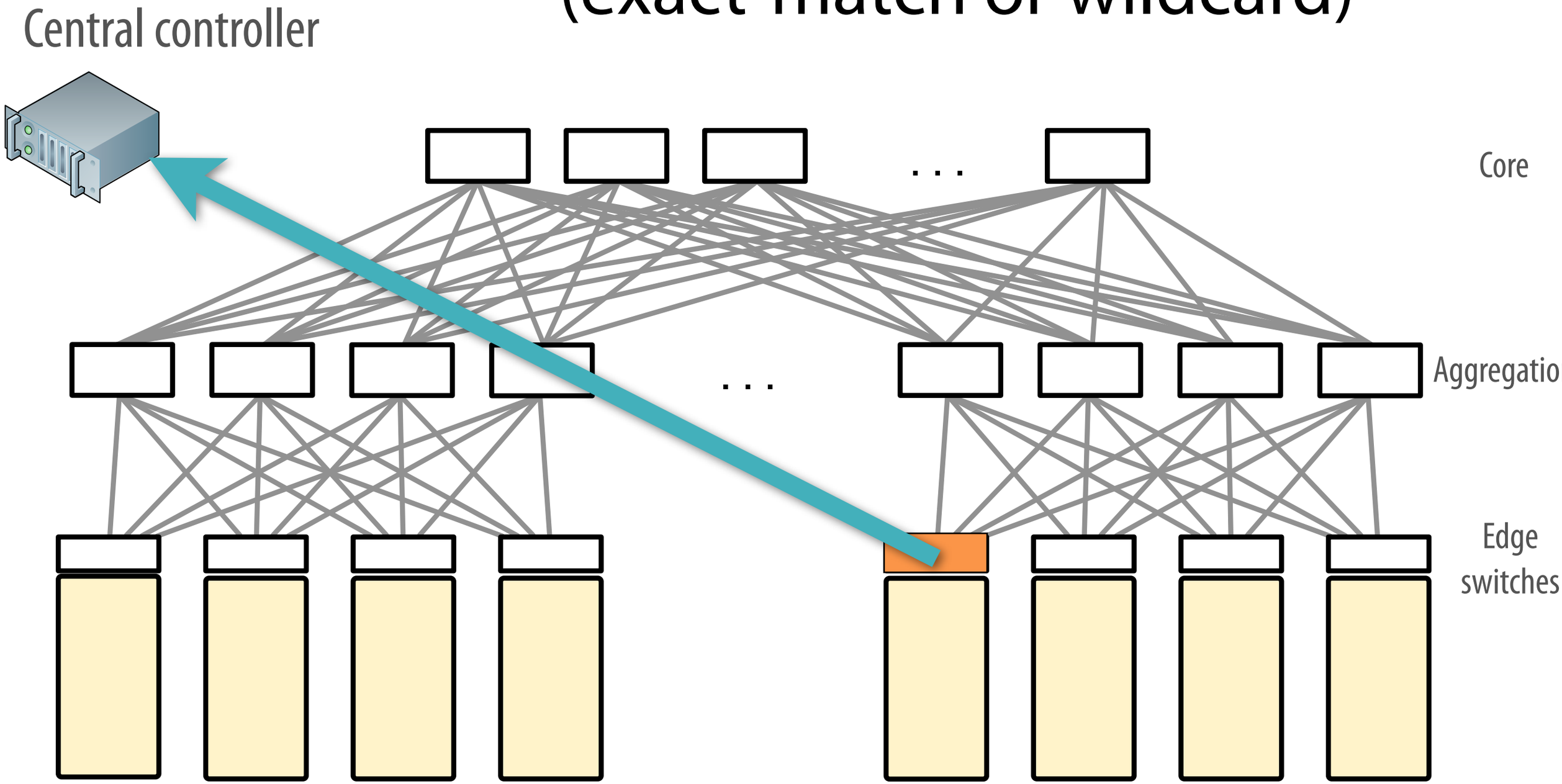
Central controller



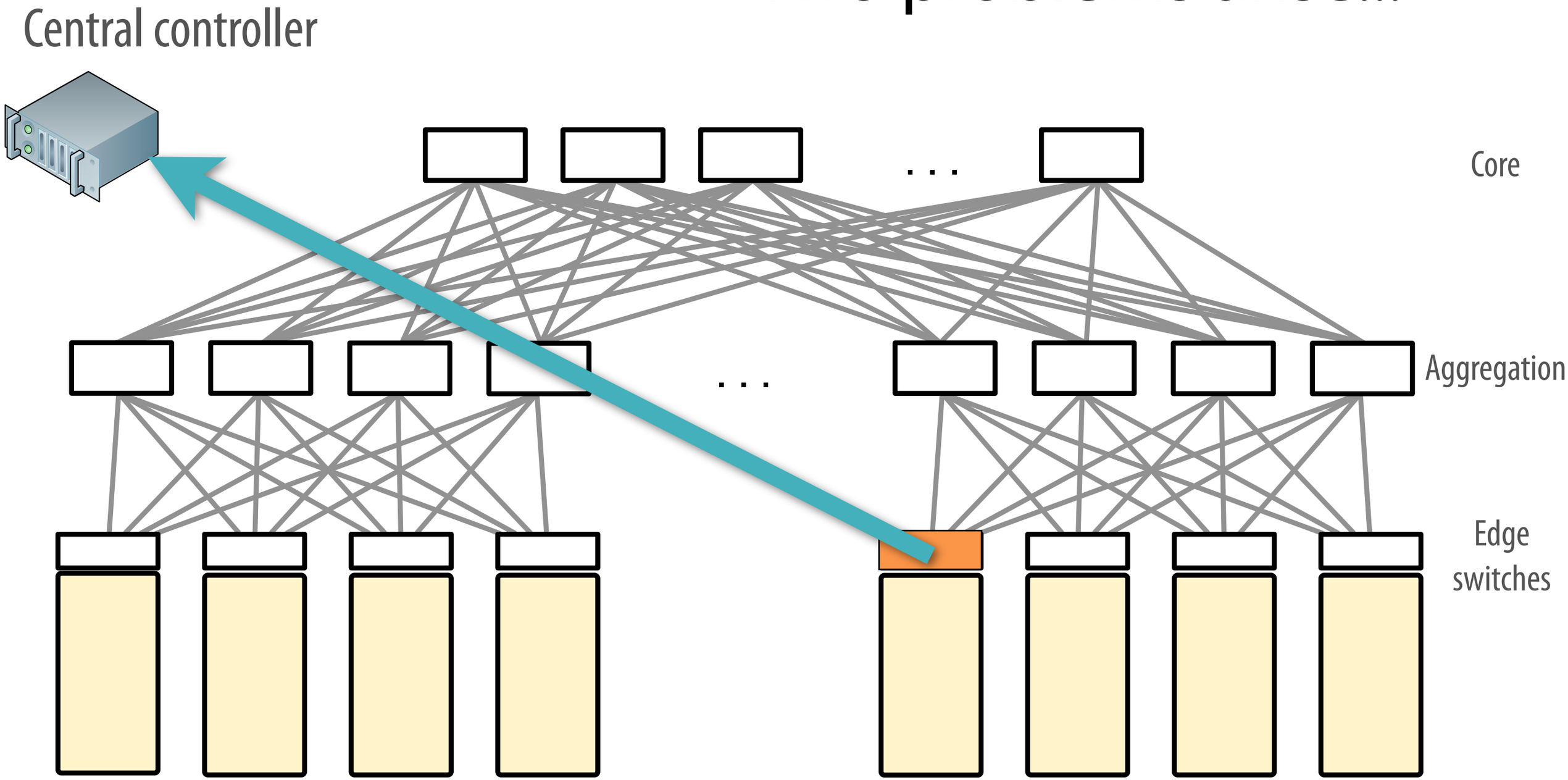


Flow arrival

If no forwarding table rule at switch (exact-match or wildcard)

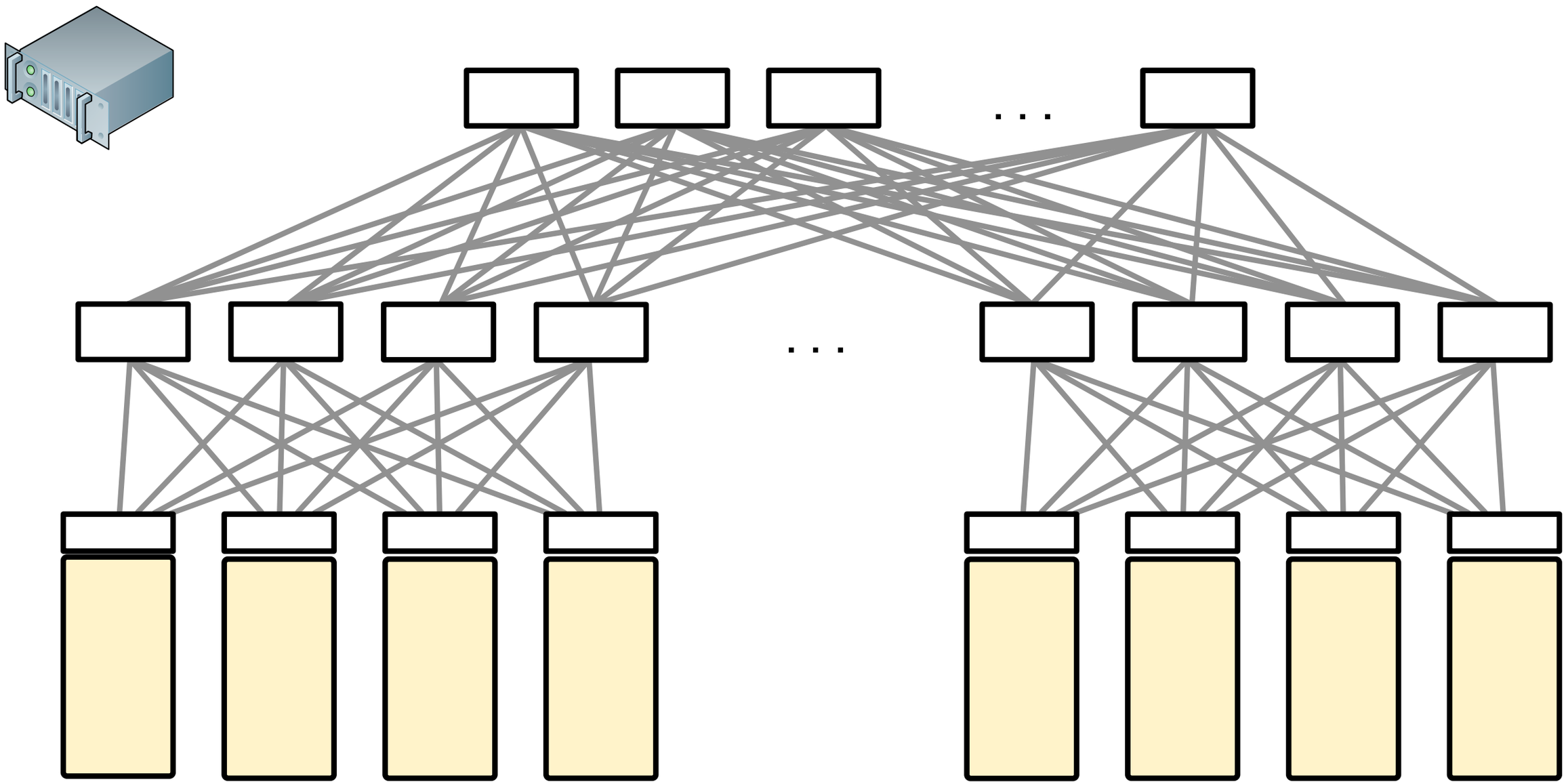


Two problems arise...



problem 1:

bottleneck at controller

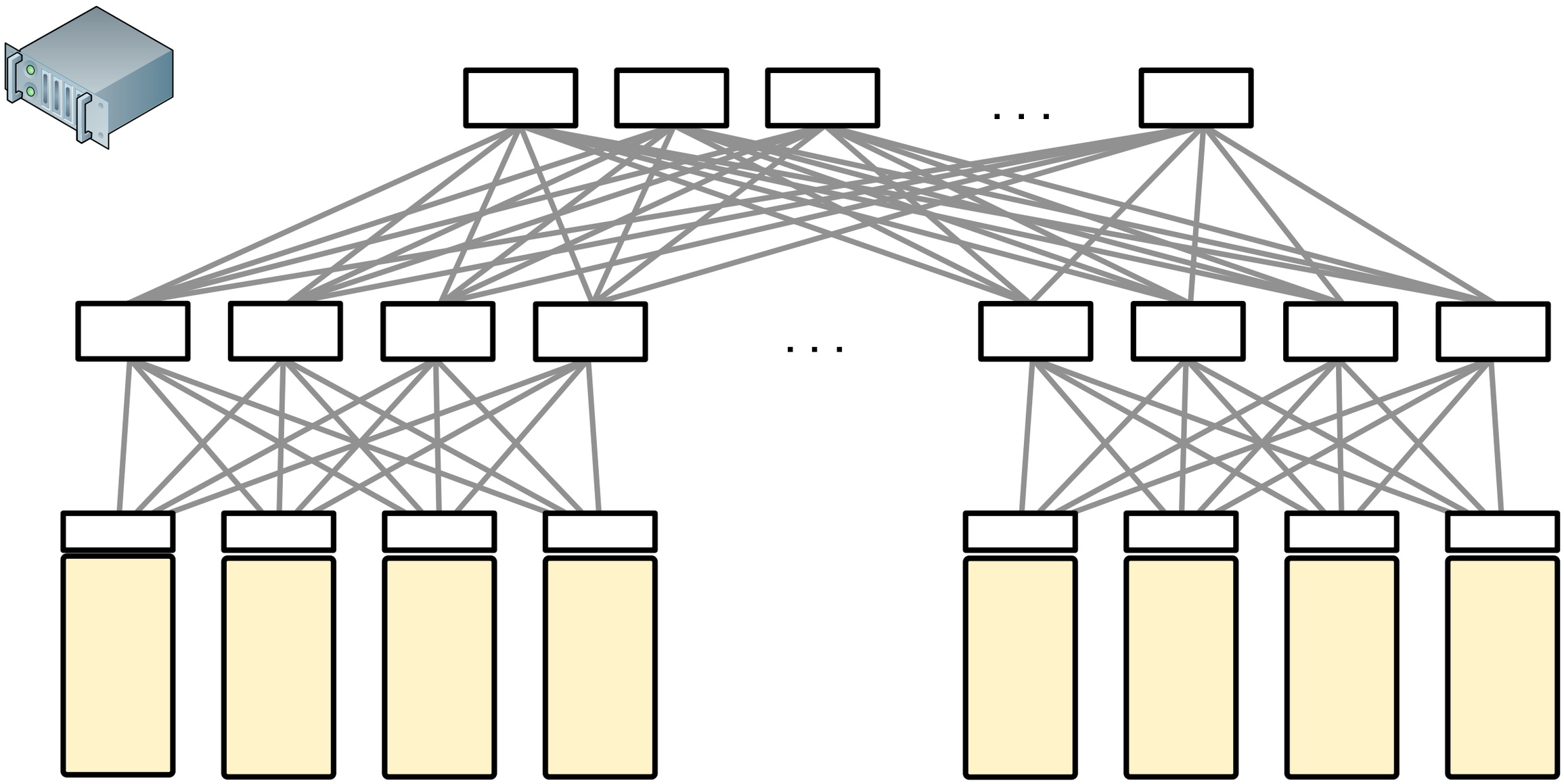


problem 1:

bottleneck at controller

Up to 10 million new flows per second in data center with 100 edge switches

[Benson et al. IMC 2010]



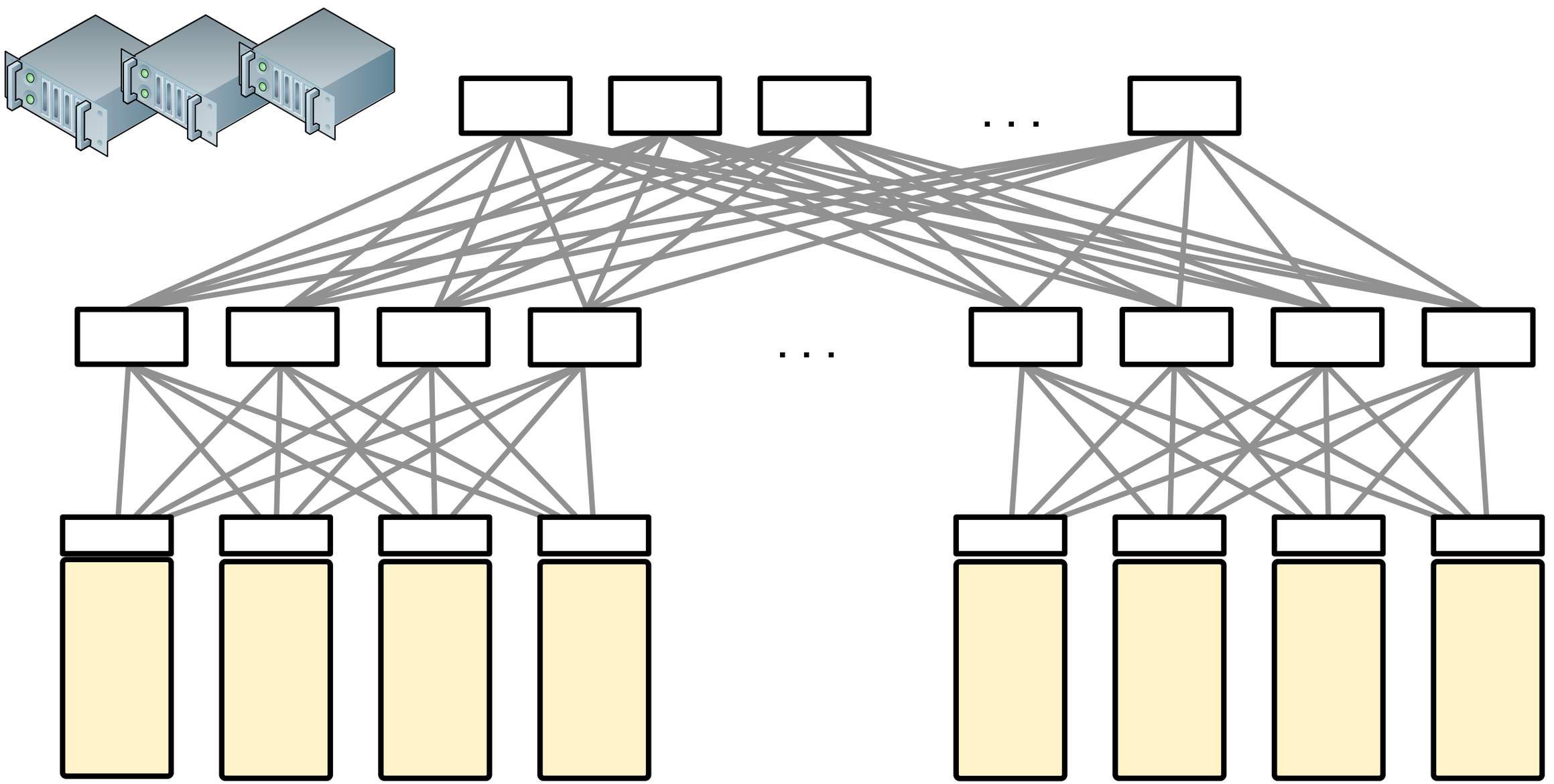
problem 1:

bottleneck at controller

Up to 10 million new flows per second in data center with 100 edge switches

[Benson et al. IMC 2010]

If controller can handle 30K flow setups/ sec. then, we need at least 333 controllers!

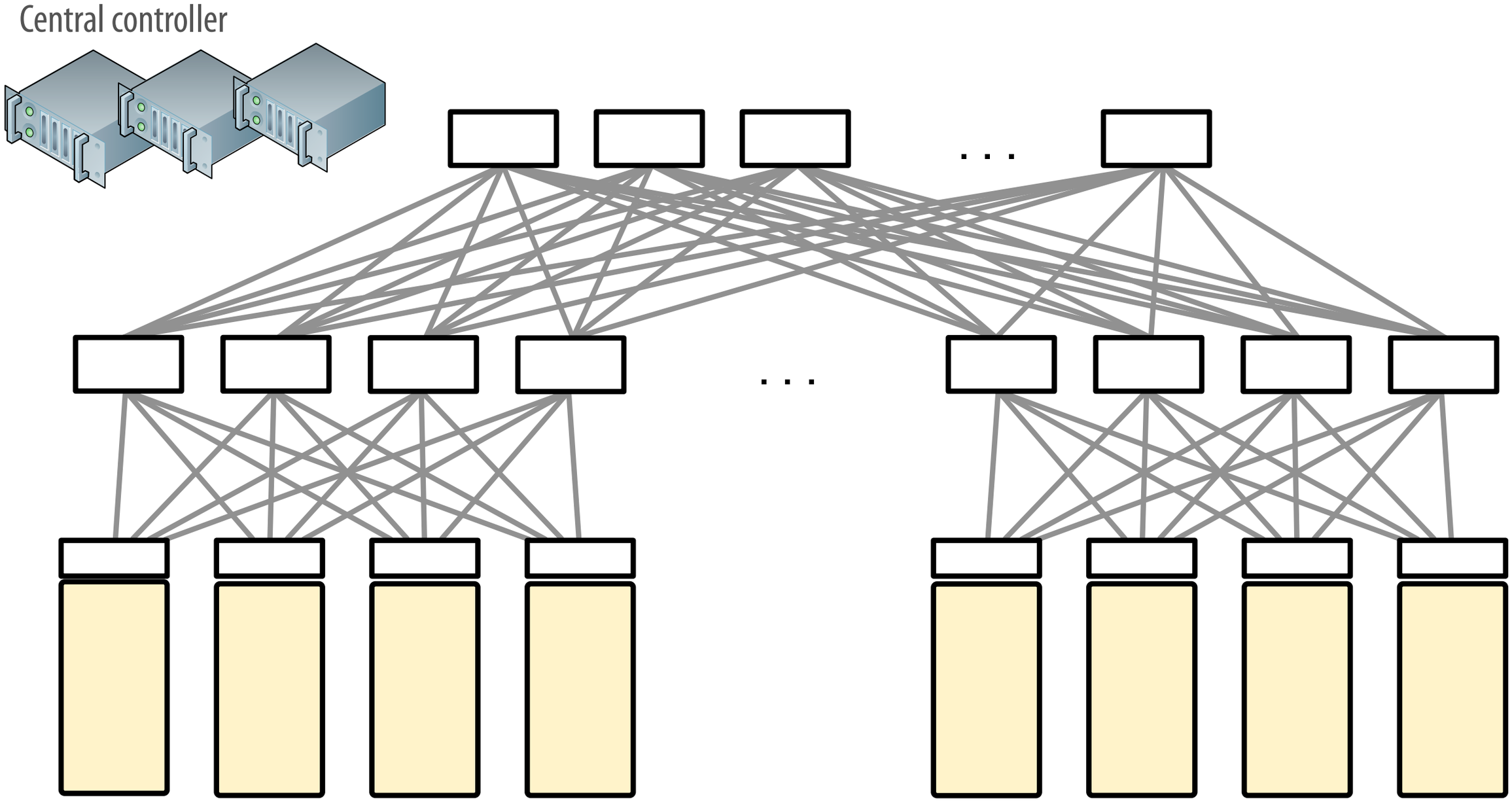


Onix [Koponen et al. OSDI 2010]

Maestro [Cai et al. Tech Report 2010]

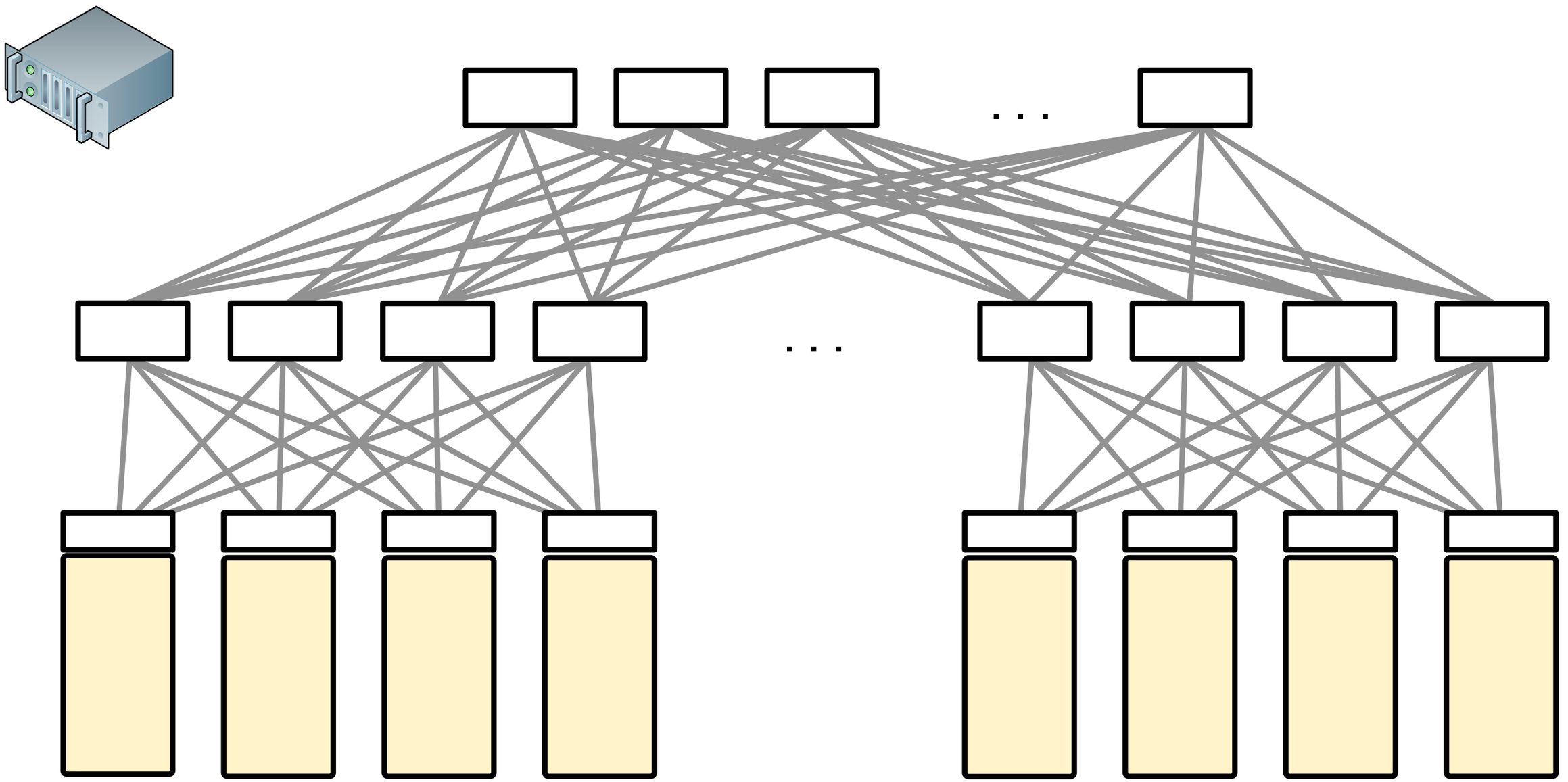
HyperFlow [Tootoonchian and Ganjali, WREN 2010]

Devolved controller [Tam et al. WCC 2011]

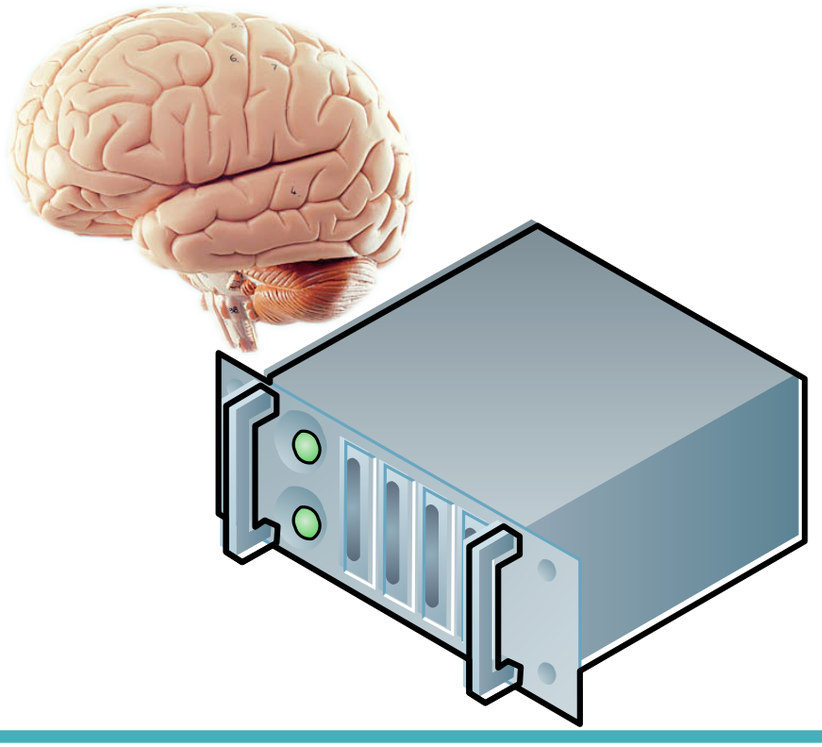


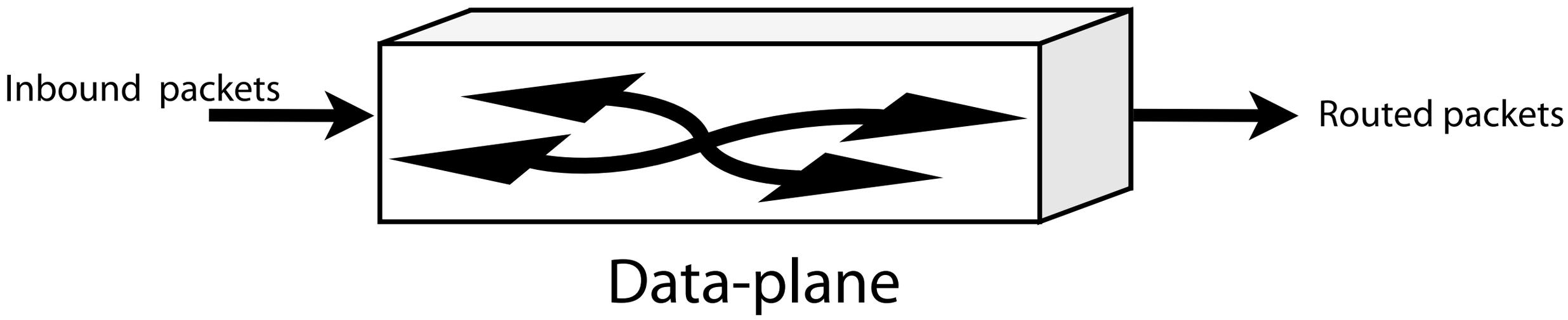
problem 2:

stress on switch control-plane

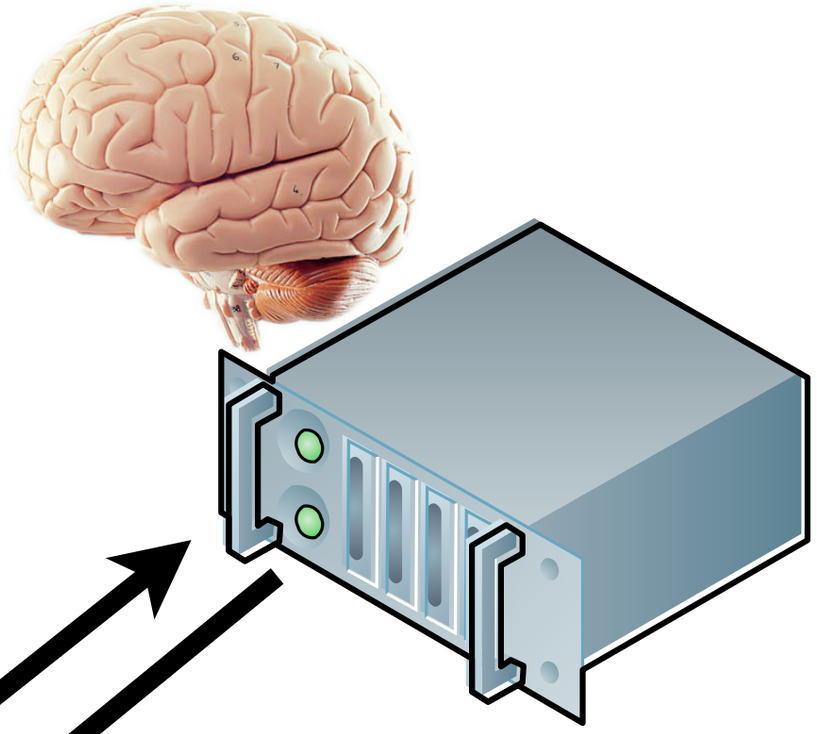


Control-plane

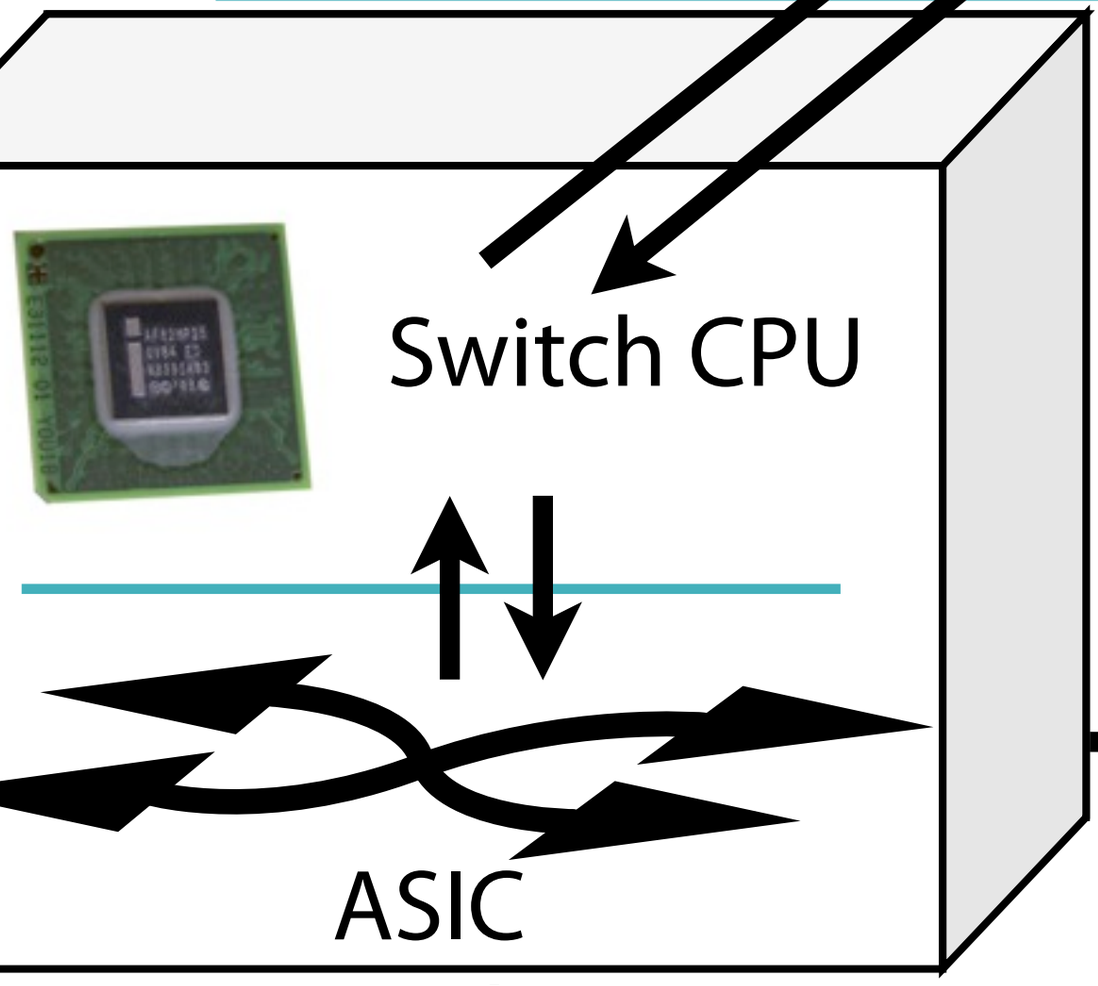




Control-plane



Switch
control-plane



Switch CPU

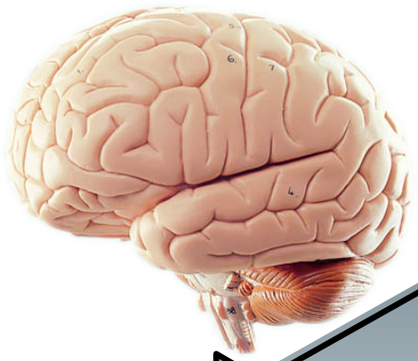
Inbound packets

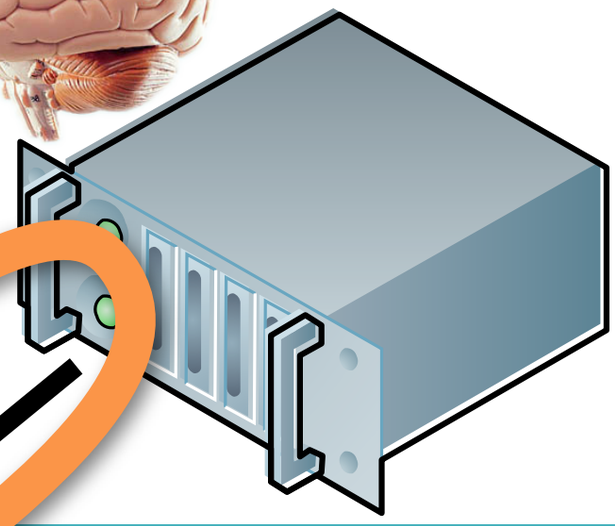
Routed packets

ASIC

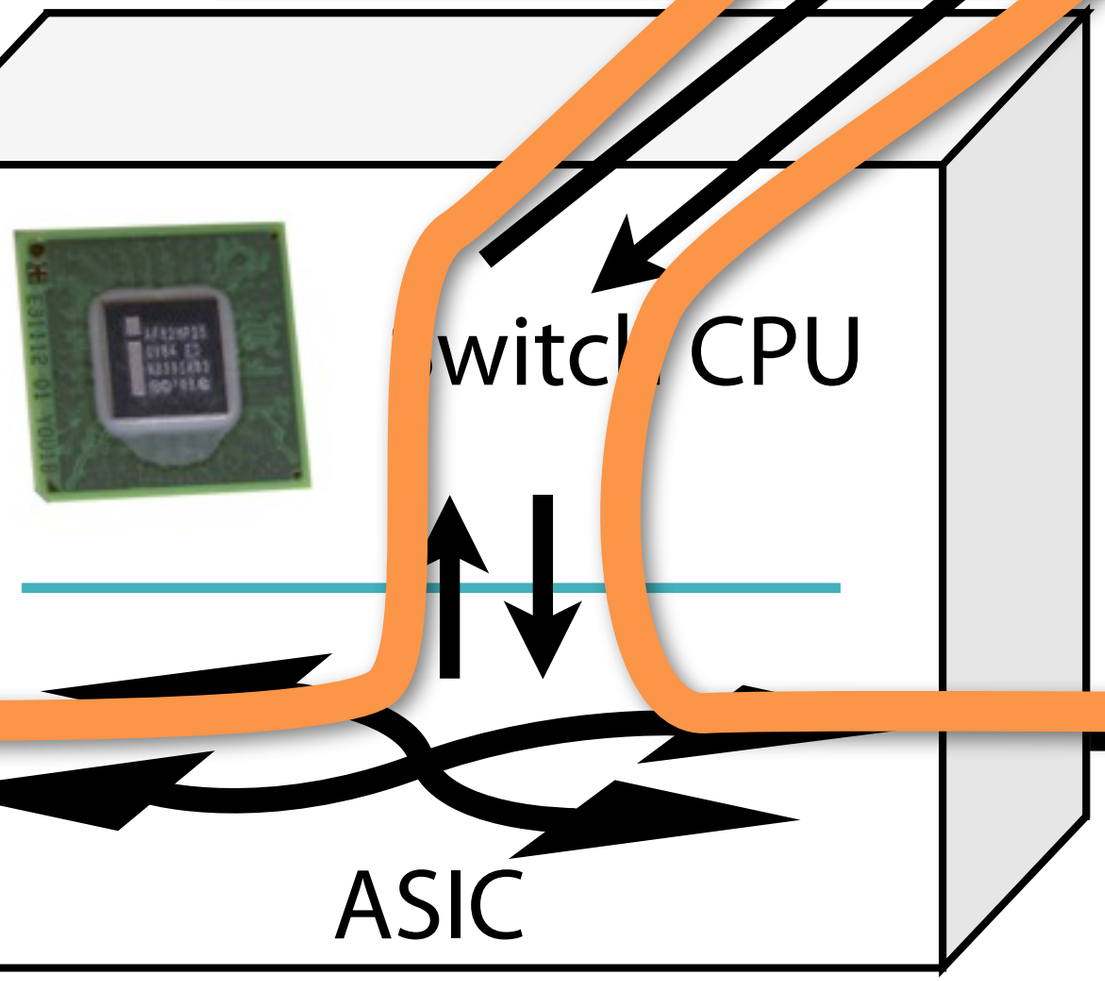
Data-plane

Control-plane





Switch
control-plane



Switch CPU

Inbound packets

Routed packets

ASIC

Data-plane

Scaling problem: switches

- Inherent overheads
- Implementation-imposed overheads

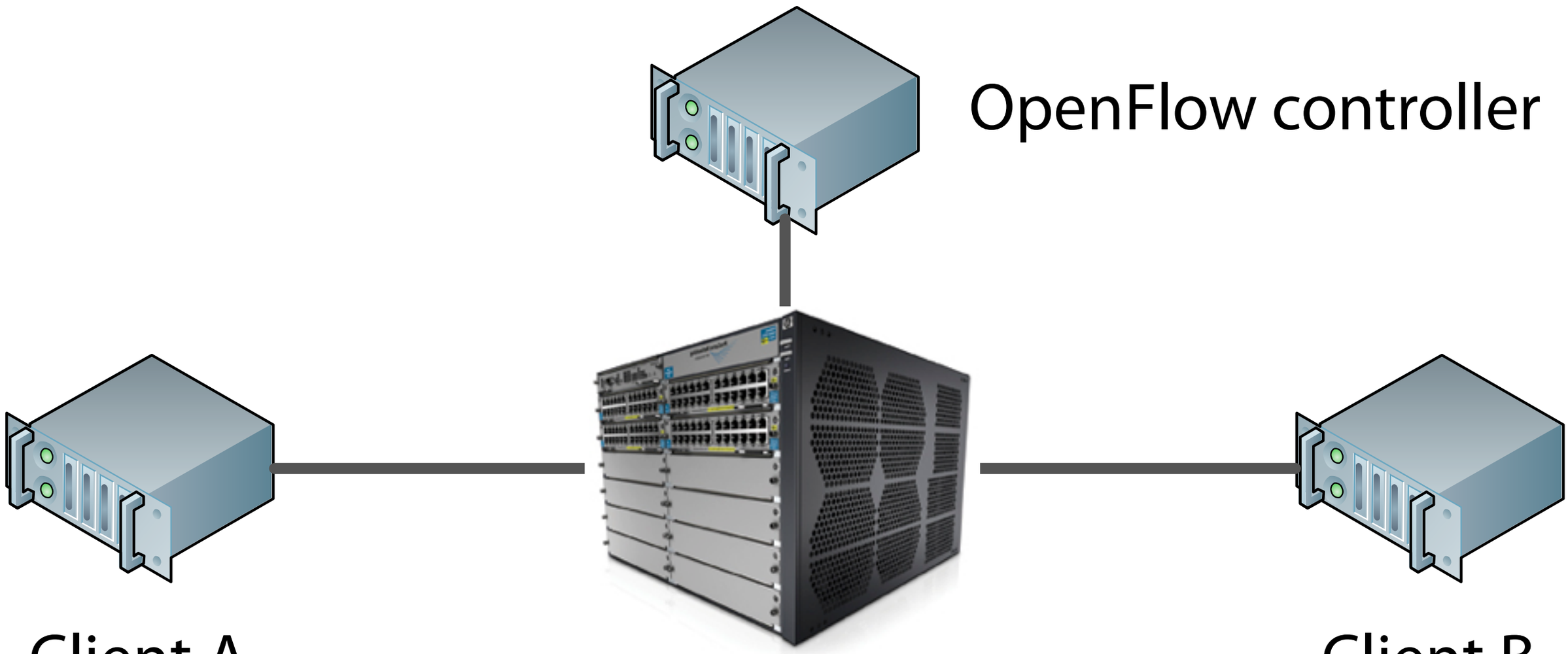
Scaling problem: switches

- Inherent overheads
  - Bandwidth
    - OpenFlow creates too much control traffic
      ~1 control packet for every 2–3 data packets
  - Latency
- Implementation-imposed overheads

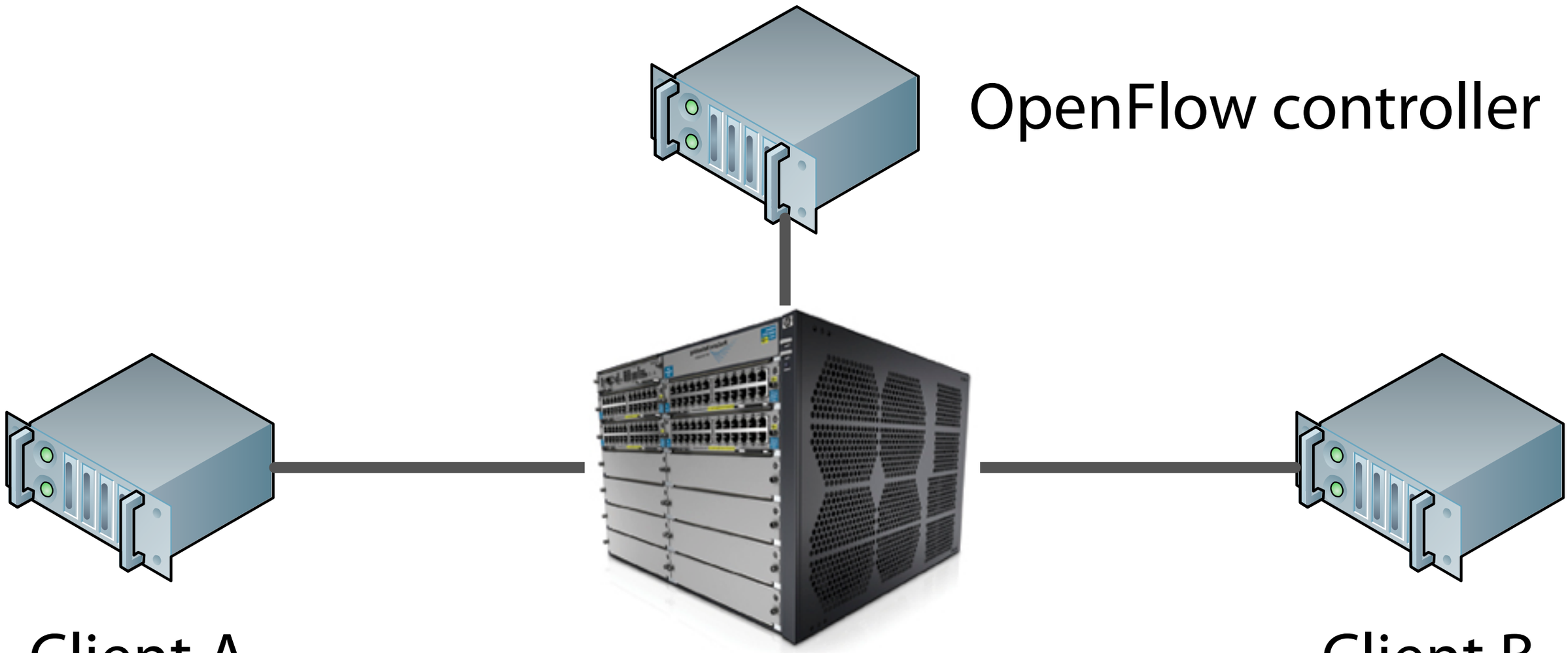
Scaling problem: switches

- Inherent overheads
- Implementation-imposed overheads
  - Flow setup
  - Statistics gathering
  - State size (see paper)

Flow setup


OpenFlow controller
Client A
Client B
ProCurve 5406 zl switch

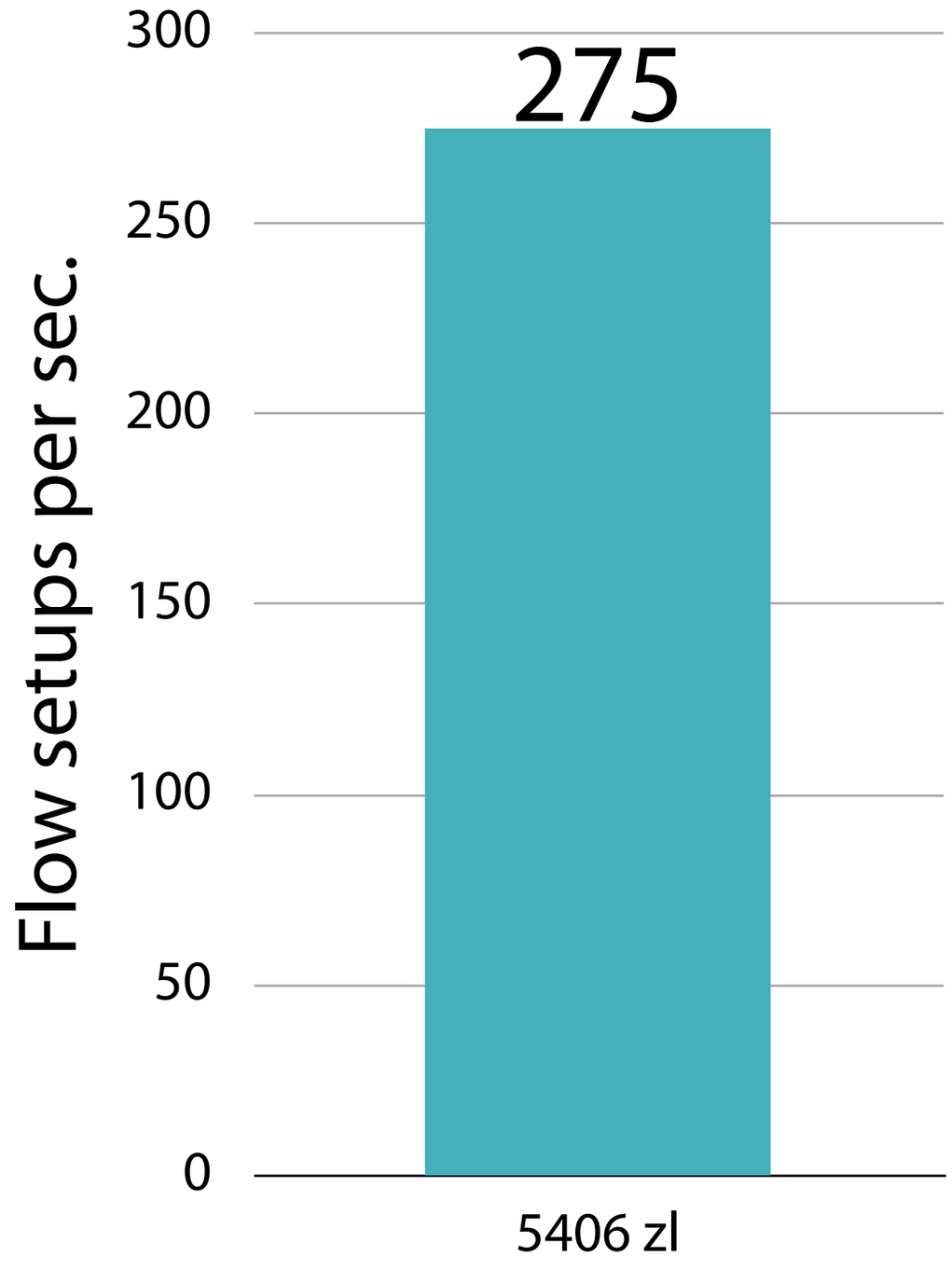
Flow setup



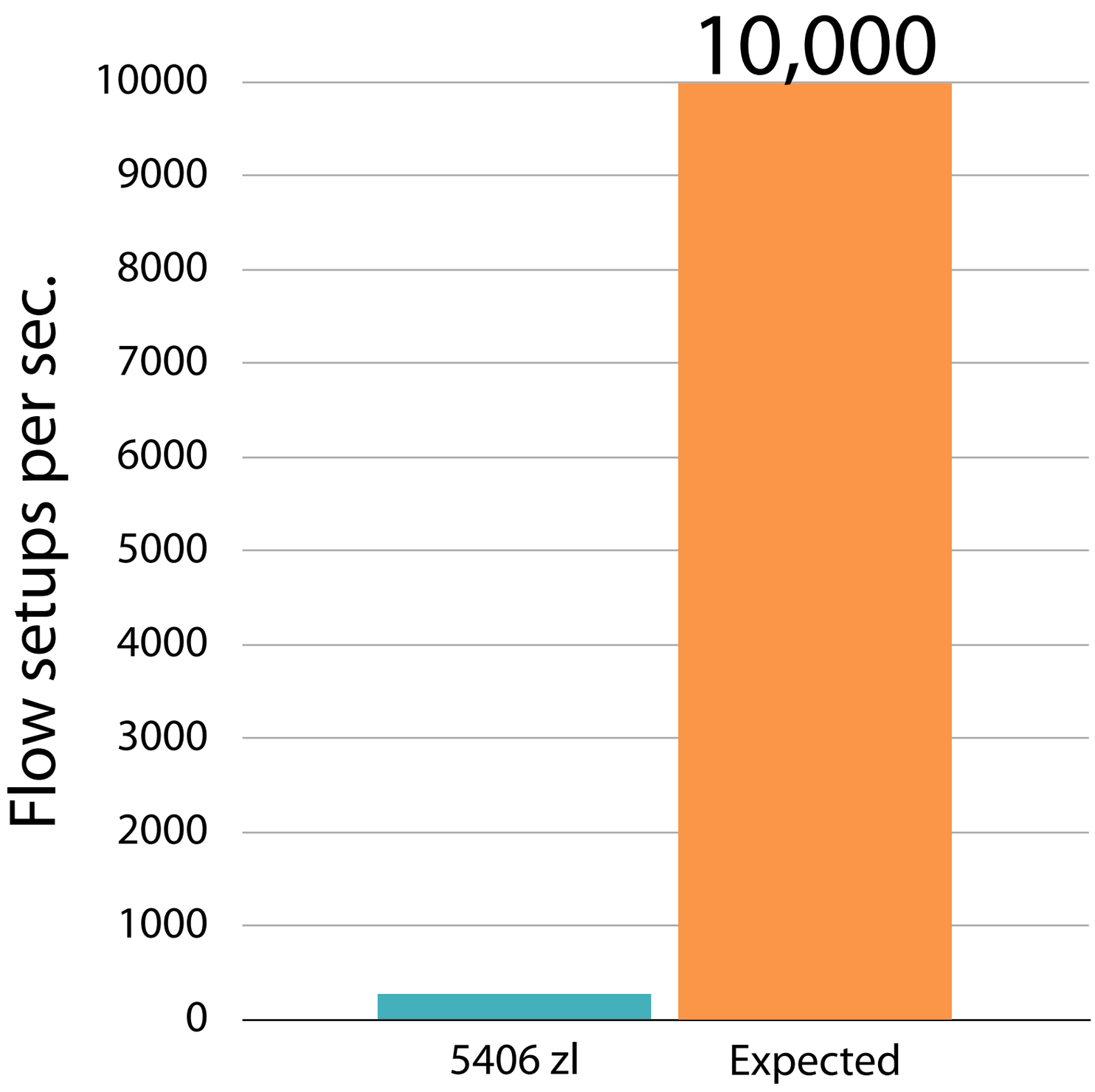
ProCurve 5406 zl switch

We believe our measurement numbers are representative of the current generation of OpenFlow switches

Flow setup



Flow setup

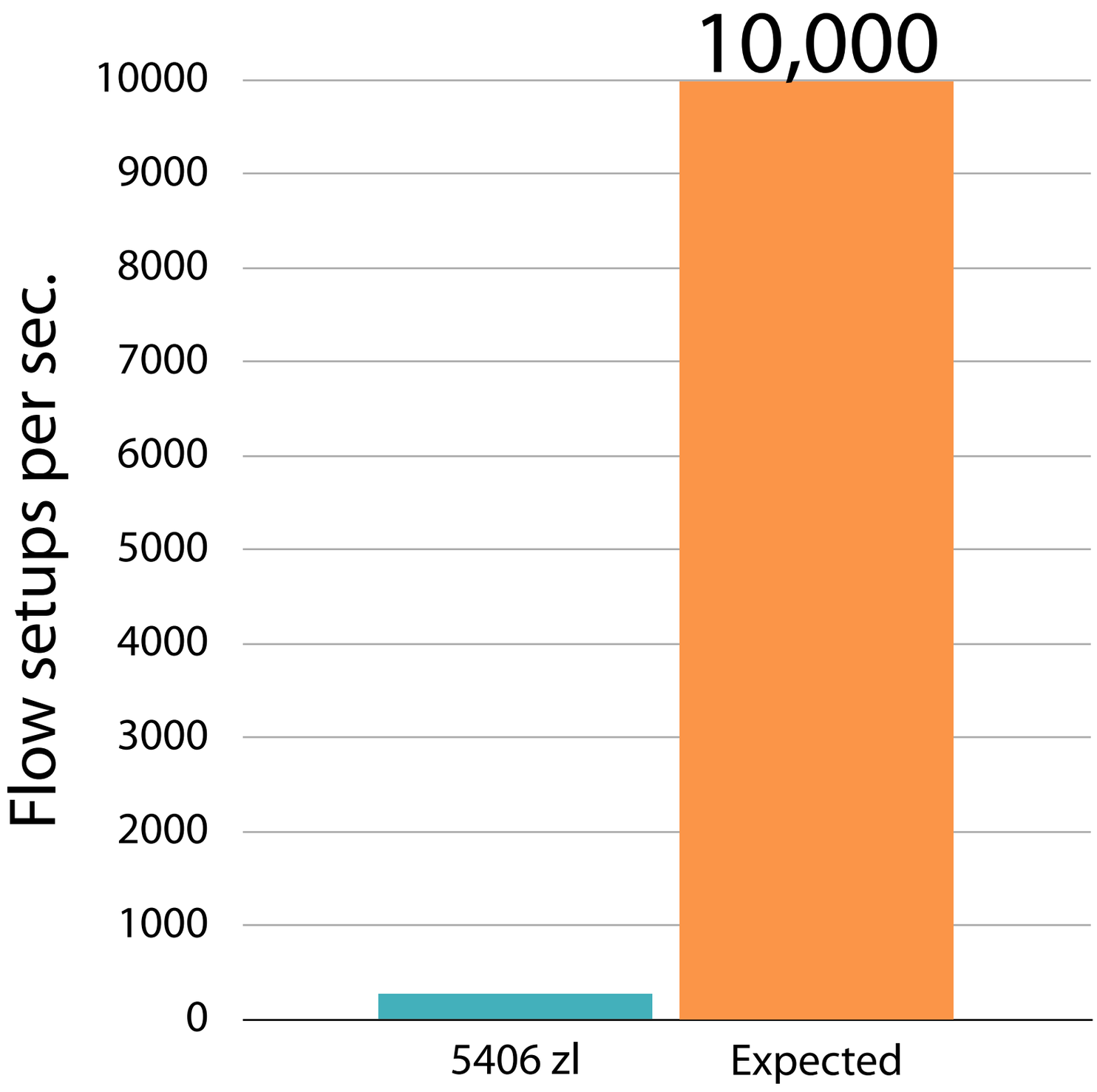


We can expect up to 10K flow arrivals / sec.

[Benson et al. IMC 2010]

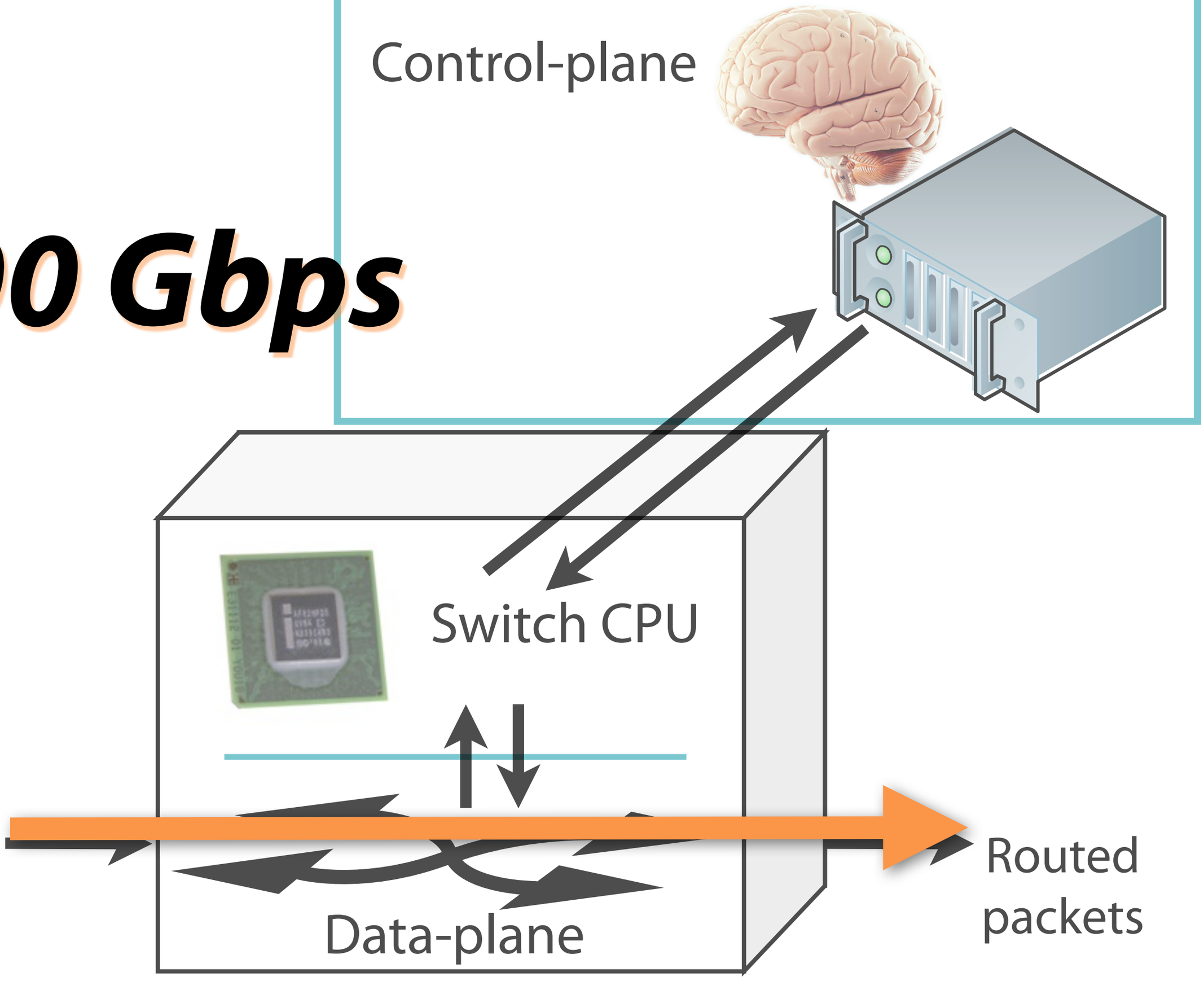
40x difference!

Flow setup

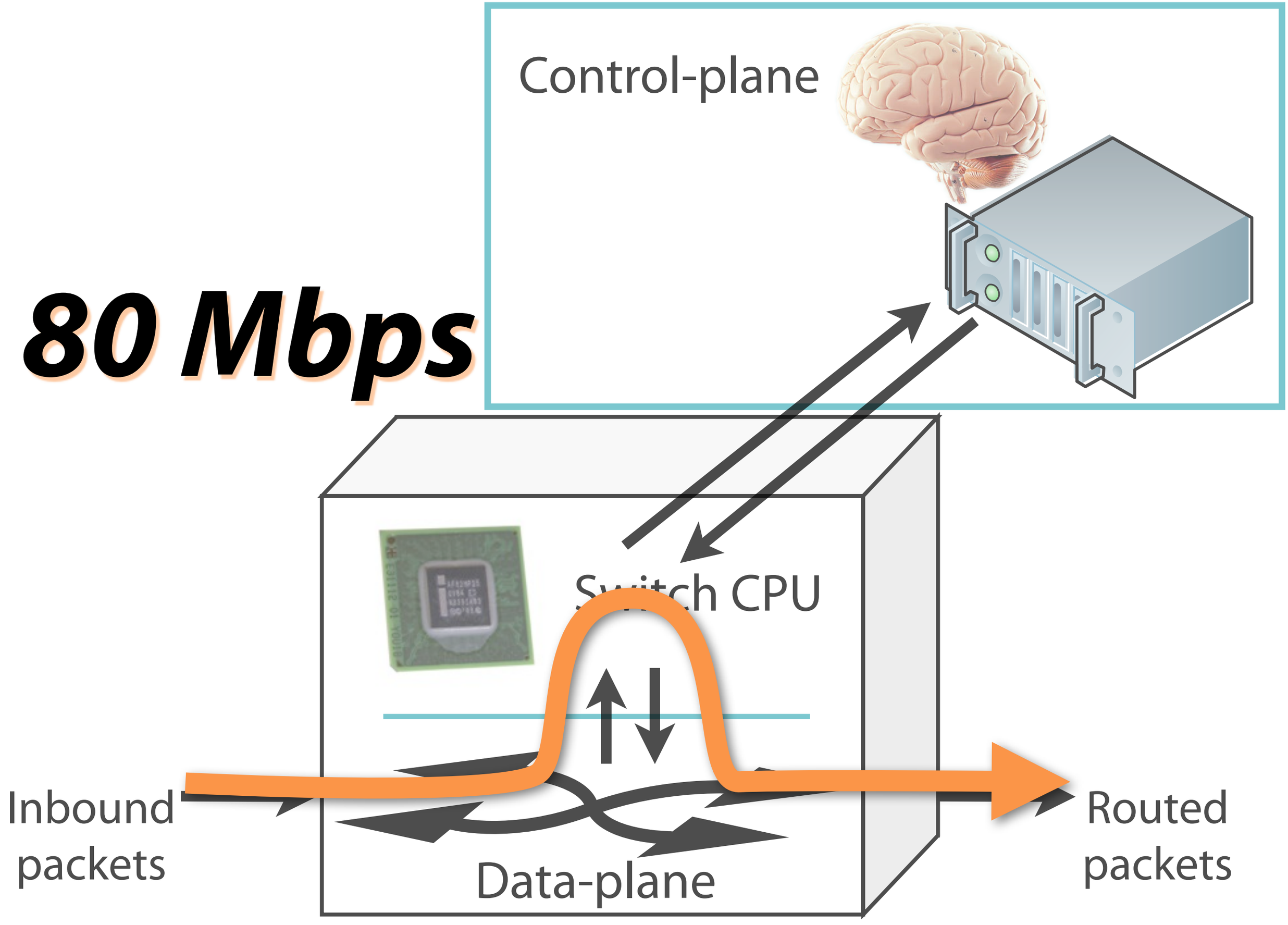


Too much latency: adds 2ms to flow setup

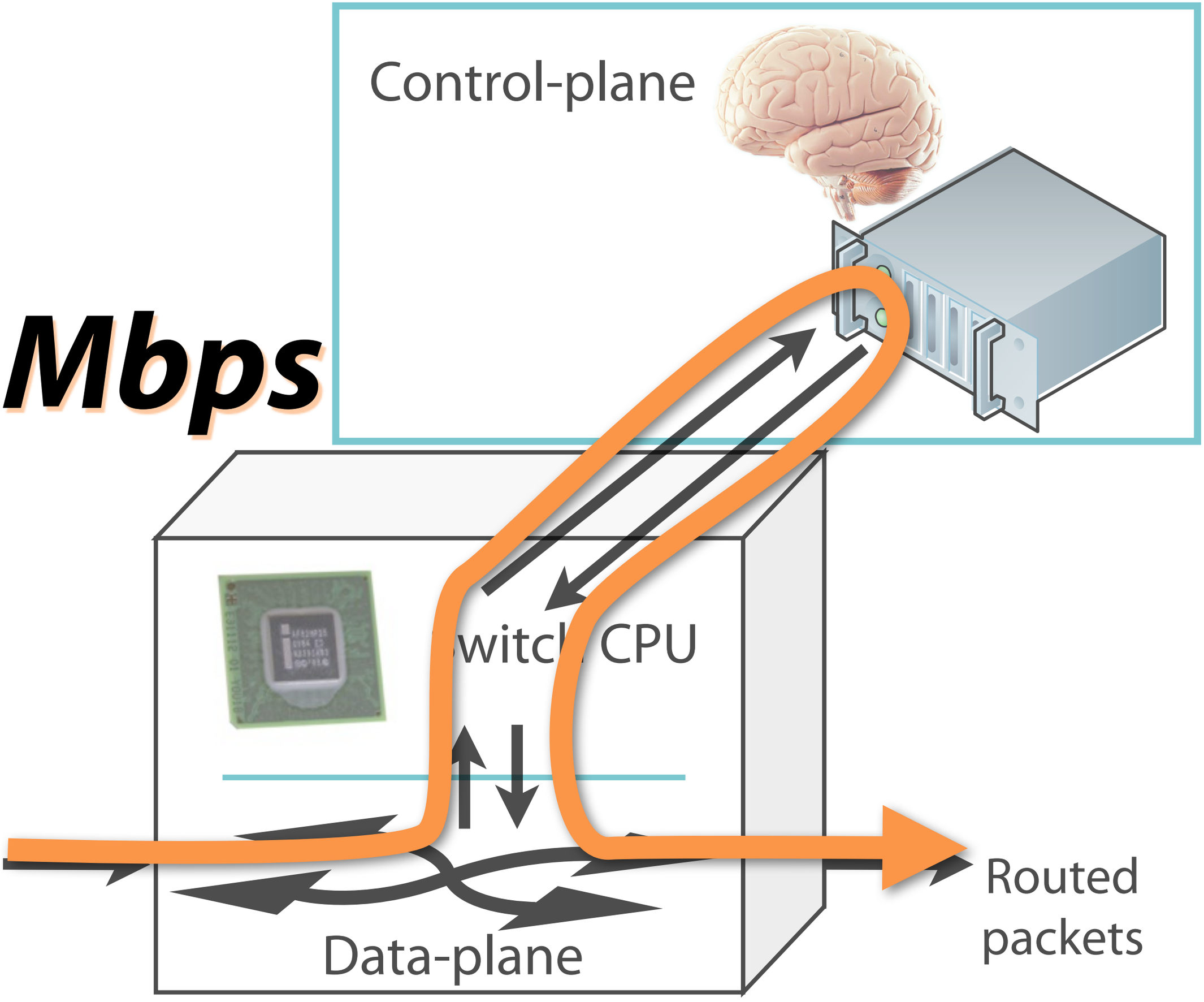
300 Gbps


Control-plane
Switch CPU
Inbound packets
Routed packets
Data-plane

80 Mbps


Control-plane
Switch CPU
Inbound packets
Routed packets
Data-plane

17 Mbps

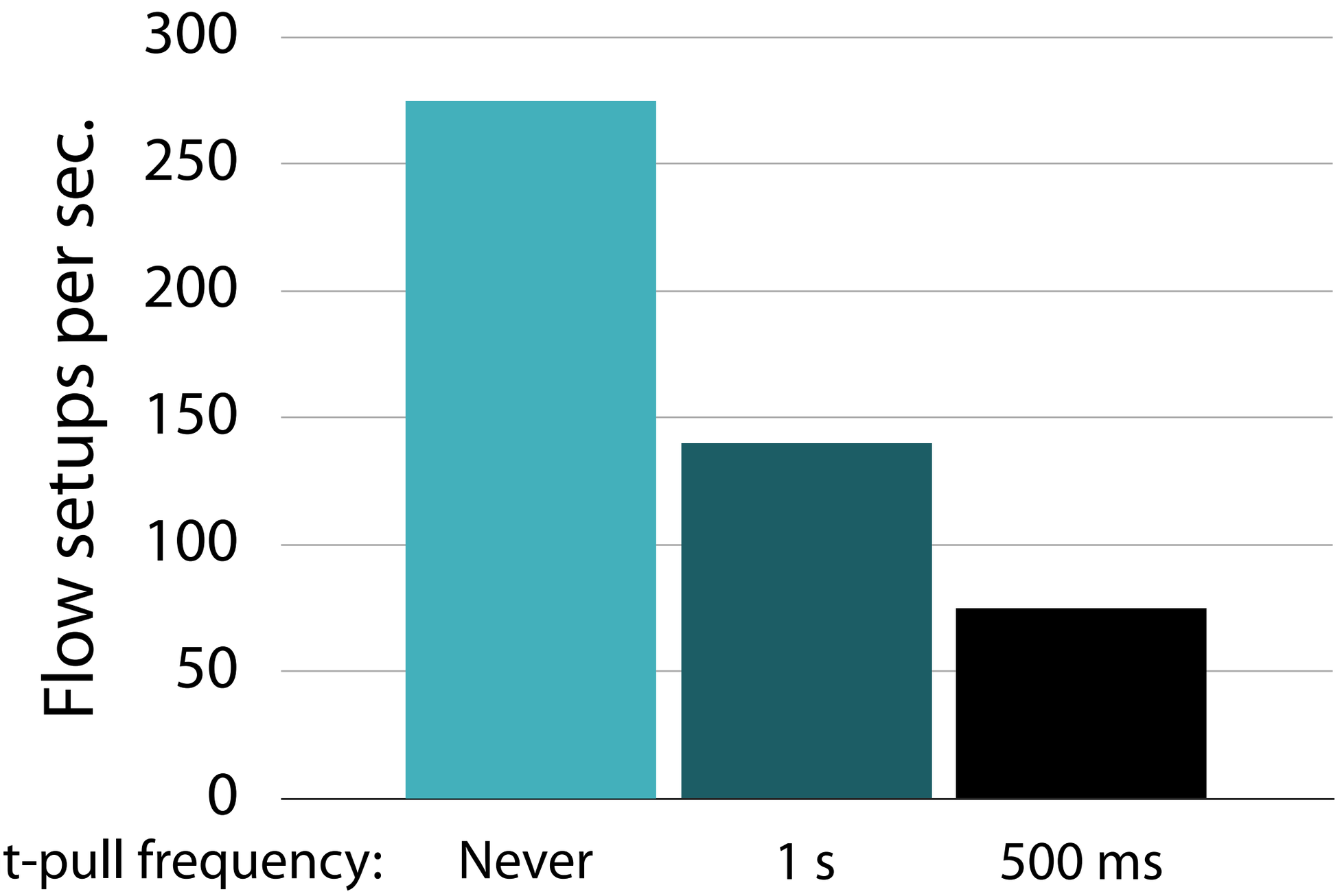

Control-plane
witch CPU
Inbound packets
Routed packets
Data-plane

Stats-gathering

- Flow setups and stat-pulling compete for this bandwidth

Stats-gathering

- Flow setups and stat-pulling compete for this bandwidth



Stats-gathering

- Flow setups and stat-pulling compete for this bandwidth
  - 2.5 sec. to collect stats from the average data center edge switch

Can we solve the problem with more hardware?

- Faster CPU may help, but won't be enough
  - Control-plane datapath needs at least two orders of magnitude more bandwidth
- Ethernet speeds accelerating faster than CPU speeds
- OpenFlow won't drive chip-area budgets for several generations

Contributions

- Characterize overheads of implementing OpenFlow in hardware
- ***Propose DevoFlow to enable cost-effective, scalable flow management***
- Evaluate DevoFlow by applying it to data center flow scheduling

Devolved OpenFlow

We devolve control over most flows
back to the switches

DevoFlow design

- Keep flows in the data-plane
- Maintain just enough visibility for effective flow management
- Simplify the design and implementation of high-performance switches

DevoFlow mechanisms

- Control mechanisms
- Statistics-gathering mechanisms

DevoFlow mechanisms

- Control mechanisms
  - Rule cloning
    - ASIC clones a wildcard rule as an exact match rule for new microflows

DevoFlow mechanisms

- Control mechanisms
  - Rule cloning

wildcard rules

src	dst	src port	dst Port
*	129.100.1.5	*	*

exact-match rules

src	dst	src port	dst Port

DevoFlow mechanisms

- Control mechanisms
  - Rule cloning

wildcard rules

src	dst	src port	dst Port
*	129.100.1.5	*	*

exact-match rules

src	dst	src port	dst Port

DevoFlow mechanisms

- Control mechanisms
  - Rule cloning

wildcard rules

src	dst	src port	dst Port
*	129.100.1.5	*	*

exact-match rules

src	dst	src port	dst Port

DevoFlow mechanisms

- Control mechanisms
  - Rule cloning

wildcard rules

src	dst	src port	dst Port
*	129.100.1.5	*	*

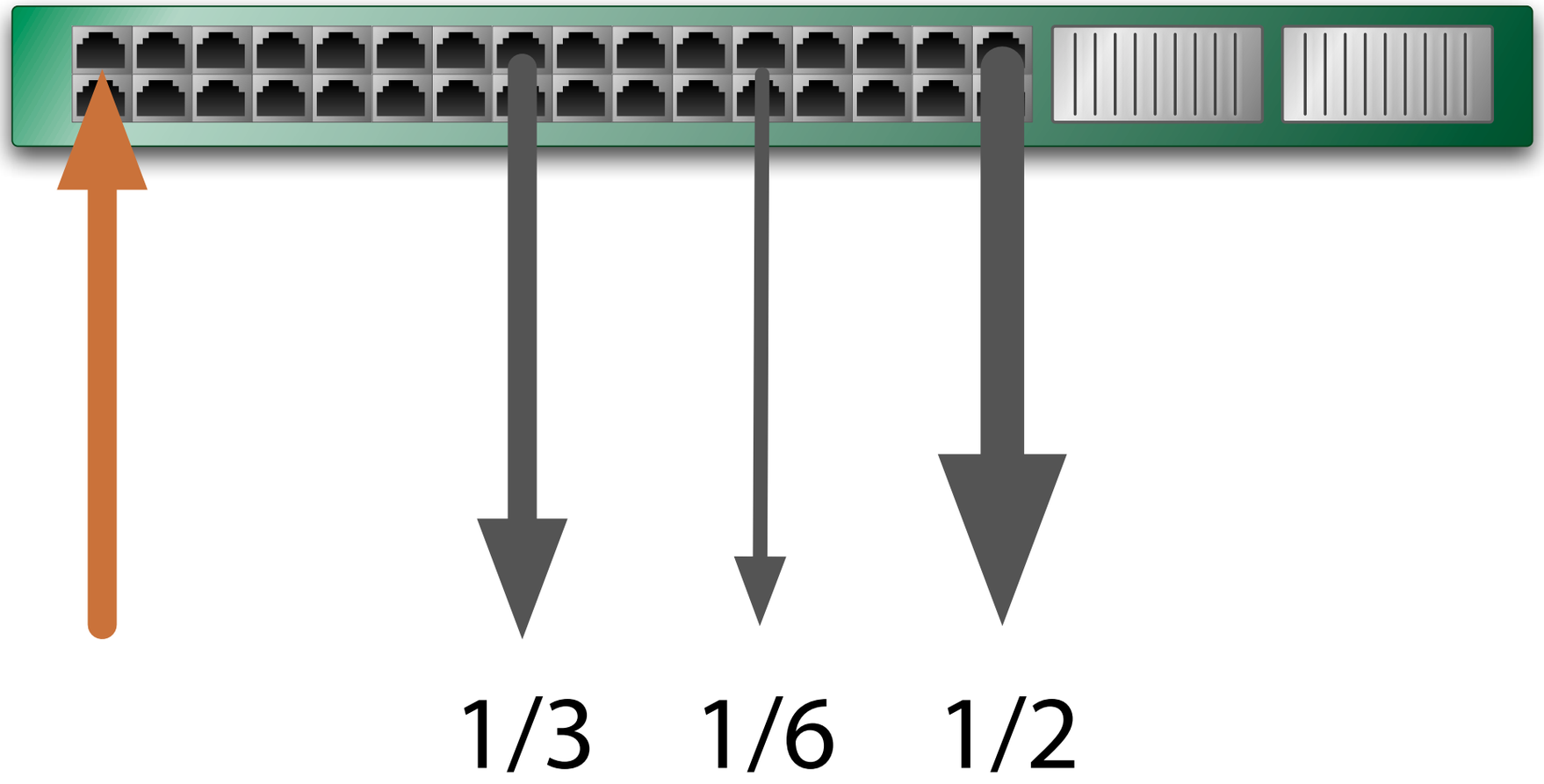
exact-match rules

src	dst	src port	dst Port
129.200.1.1	129.100.1.5	4832	80

DevoFlow mechanisms

- Control mechanisms
  - Rule cloning
    - ASIC clones a wildcard rule as an exact match rule for new microflows
  - Local actions
    - Rapid re-routing
      - Gives fallback paths for when a port fails
    - Multipath support

Control-mechanisms



- Multipath support

Statistics-gathering mechanisms

Statistics-gathering mechanisms

- Sampling
  - Packet header is sent to controller with 1/1000 probability

Statistics-gathering mechanisms

- Sampling
  - Packet header is sent to controller with 1/1000 probability
- Triggers and reports
  - Can set a threshold per rule; when threshold is reached, flow is setup at central controller

Statistics-gathering mechanisms

- Sampling
  - Packet header is sent to controller with 1/1000 probability
- Triggers and reports
  - Can set a threshold per rule; when threshold is reached, flow is setup at central controller
- Approximate counters
  - Tracks all flows matching a wildcard rule

Implementing DevoFlow

- Have not implemented in hardware
- Can reuse existing functional blocks for most mechanisms

Using DevoFlow

- Provides tools to scale your SDN application, but scaling is still a challenge
- Example: flow scheduling
  - Follows Hedera's approach [Al-Fares et al. NSDI 2010]

Using DevoFlow: flow scheduling

Using DevoFlow: flow scheduling

- Switches use multipath forwarding rules for new flows

Using DevoFlow: flow scheduling

- Switches use multipath forwarding rules for new flows
- Central controller uses sampling or triggers to detect elephant flows
  - Elephant flows are dynamically scheduled by the central controller
  - Uses a bin packing algorithm, see paper

Evaluation

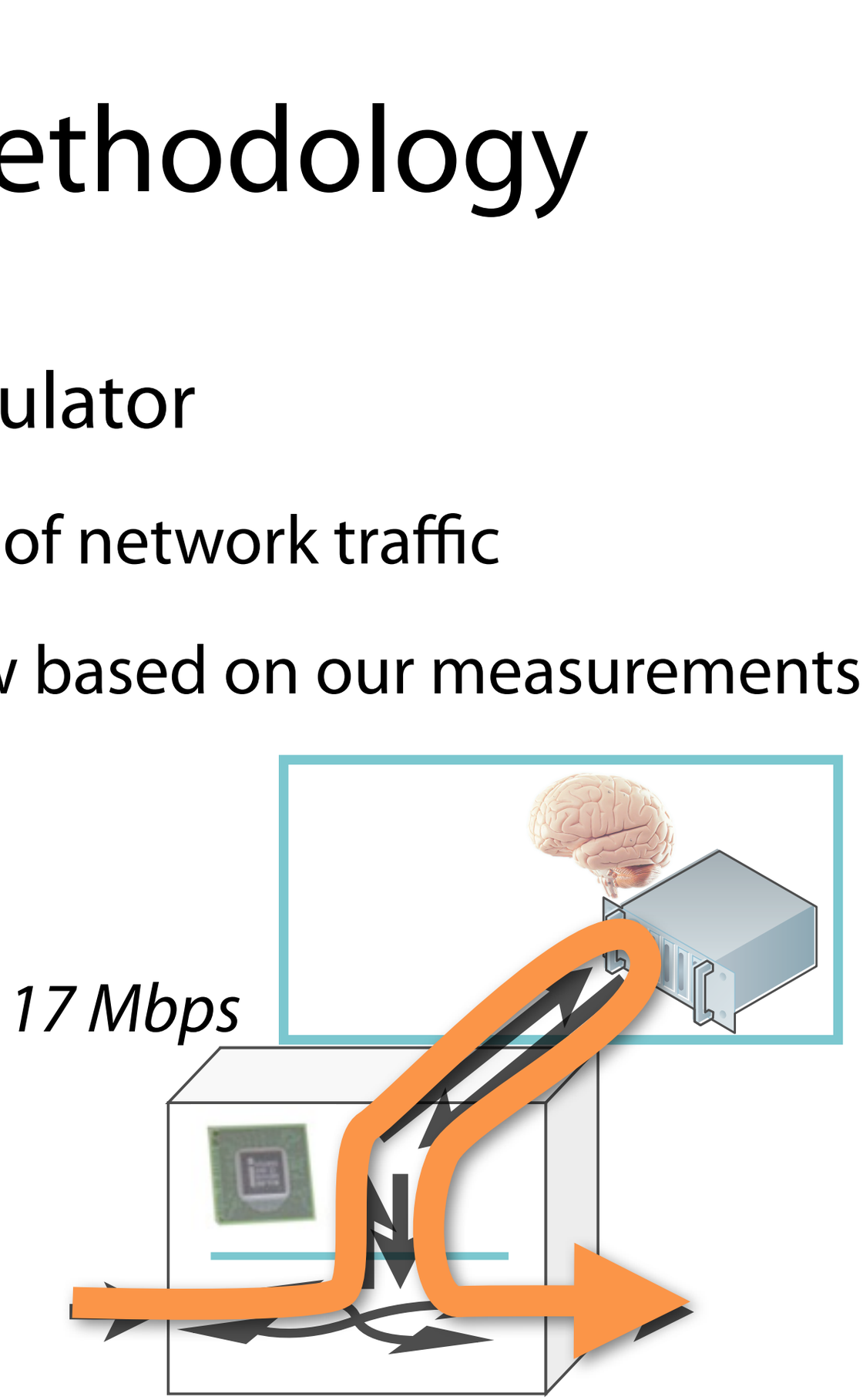
- How much can we reduce flow scheduling overheads while still achieving high performance?

Evaluation: methodology

- Custom built simulator
  - Flow-level model of network traffic
  - Models OpenFlow based on our measurements of the 5406 zl

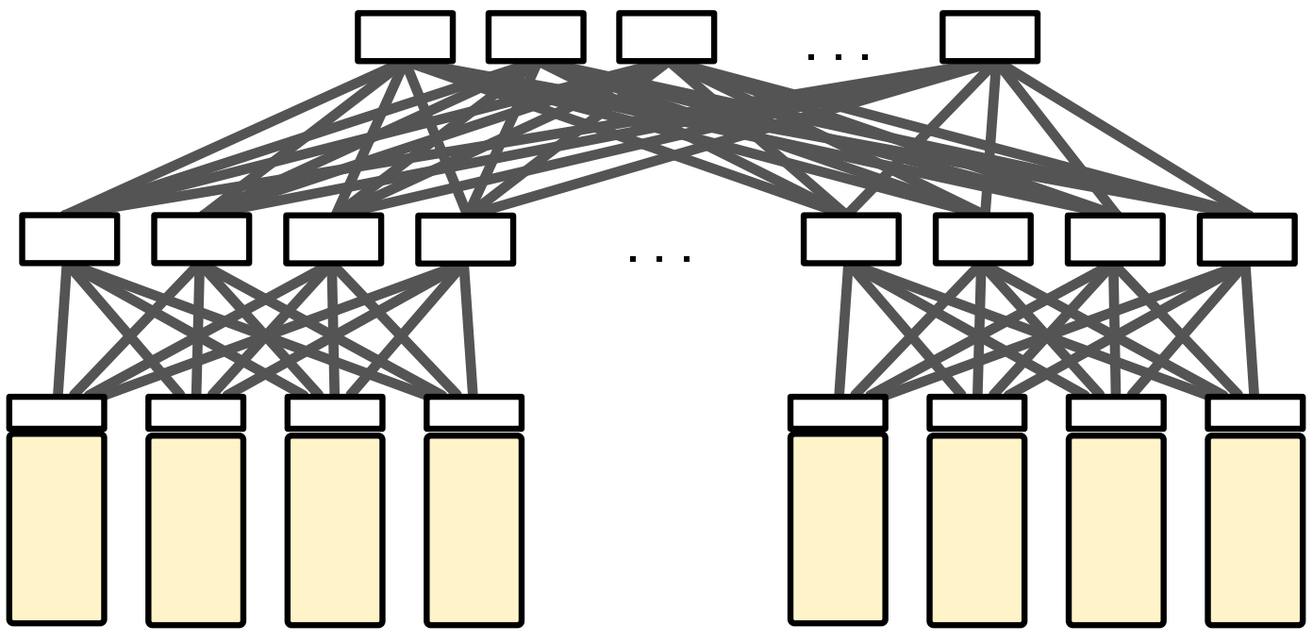
Evaluation: methodology

- Custom built simulator
  - Flow-level model of network traffic
  - Models OpenFlow based on our measurements of the 5406 zl

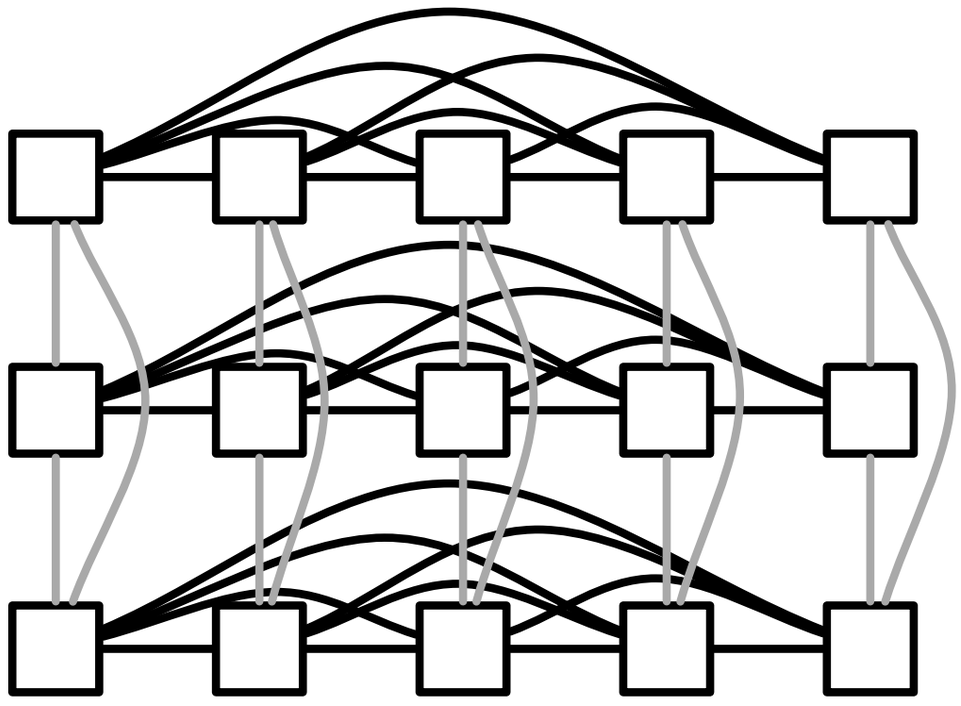


Evaluation: methodology

- Clos topology
  - 1600 servers
  - 640 Gbps bisection bandwidth
  - 20 servers per rack



- HyperX topology
  - 1620 servers
  - 405 Gbps bisection bandwidth
  - 20 servers per rack



Evaluation: methodology

- Workloads
  - Shuffle, 128 MB to all servers, five at a time
  - Reverse-engineered MSR workload [Kandula et al. IMC 2009]

Evaluation: methodology

- Workloads
  - Shuffle, 128 MB to all servers, five at a time
  - Reverse-engineered MSR workload [Kandula et al. IMC 2009]
    - Based on two distributions: inter-arrival times and bytes per flow

Evaluation: methodology

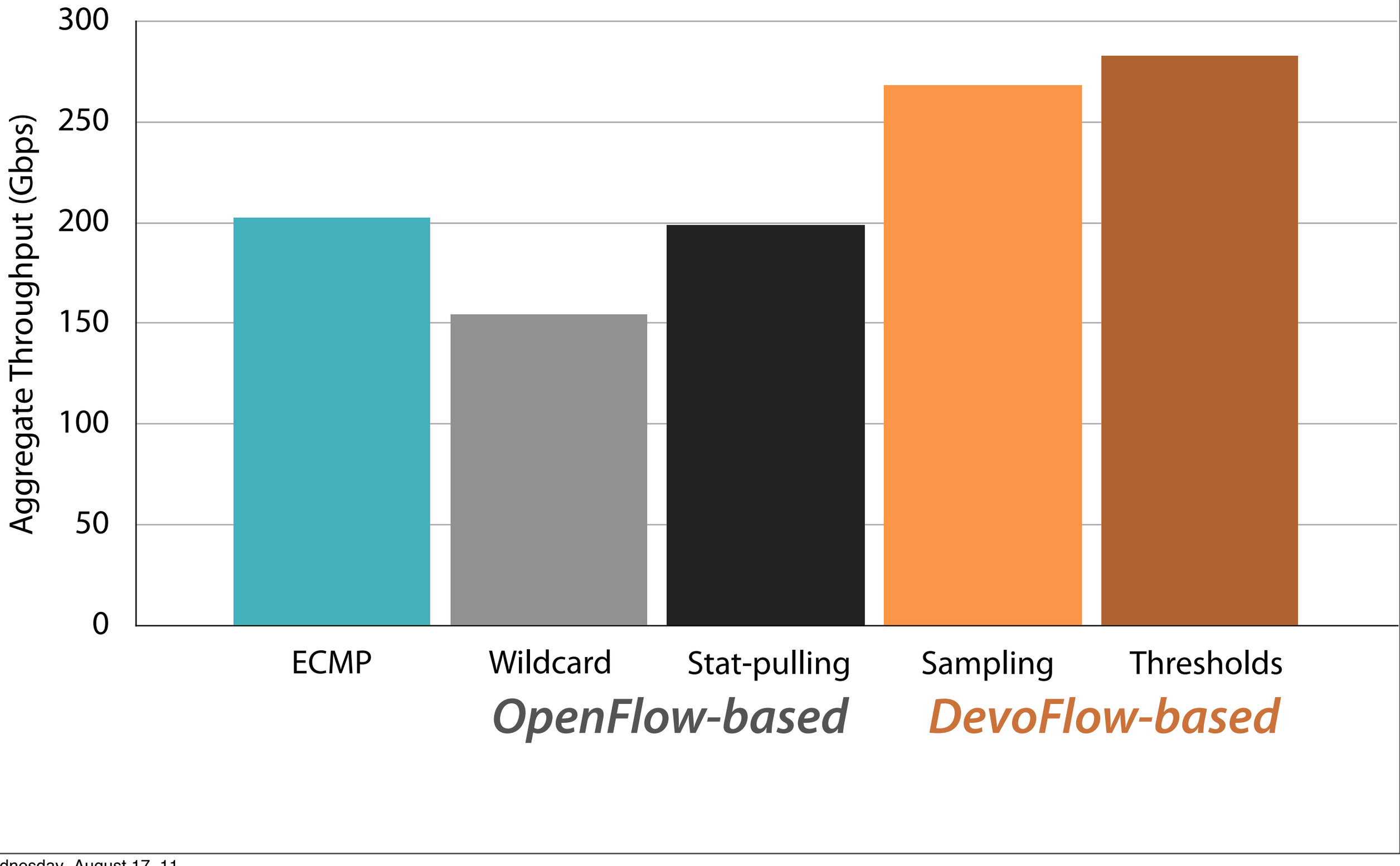
- Schedulers
  - ECMP
  - OpenFlow
    - Coarse-grained using ***wildcard rules***
    - Fine-grained using ***stat-pulling***
      (i.e., Hedera [Al-Fares et al. NSDI 2010])
  - DevoFlow
    - Statistics via ***sampling***
    - Triggers and reports at a specified ***threshold*** of bytes transferred

Evaluation: metrics

- Performance
  - Aggregate throughput
- Overheads
  - Packets/sec. to central controller
  - Forwarding table size

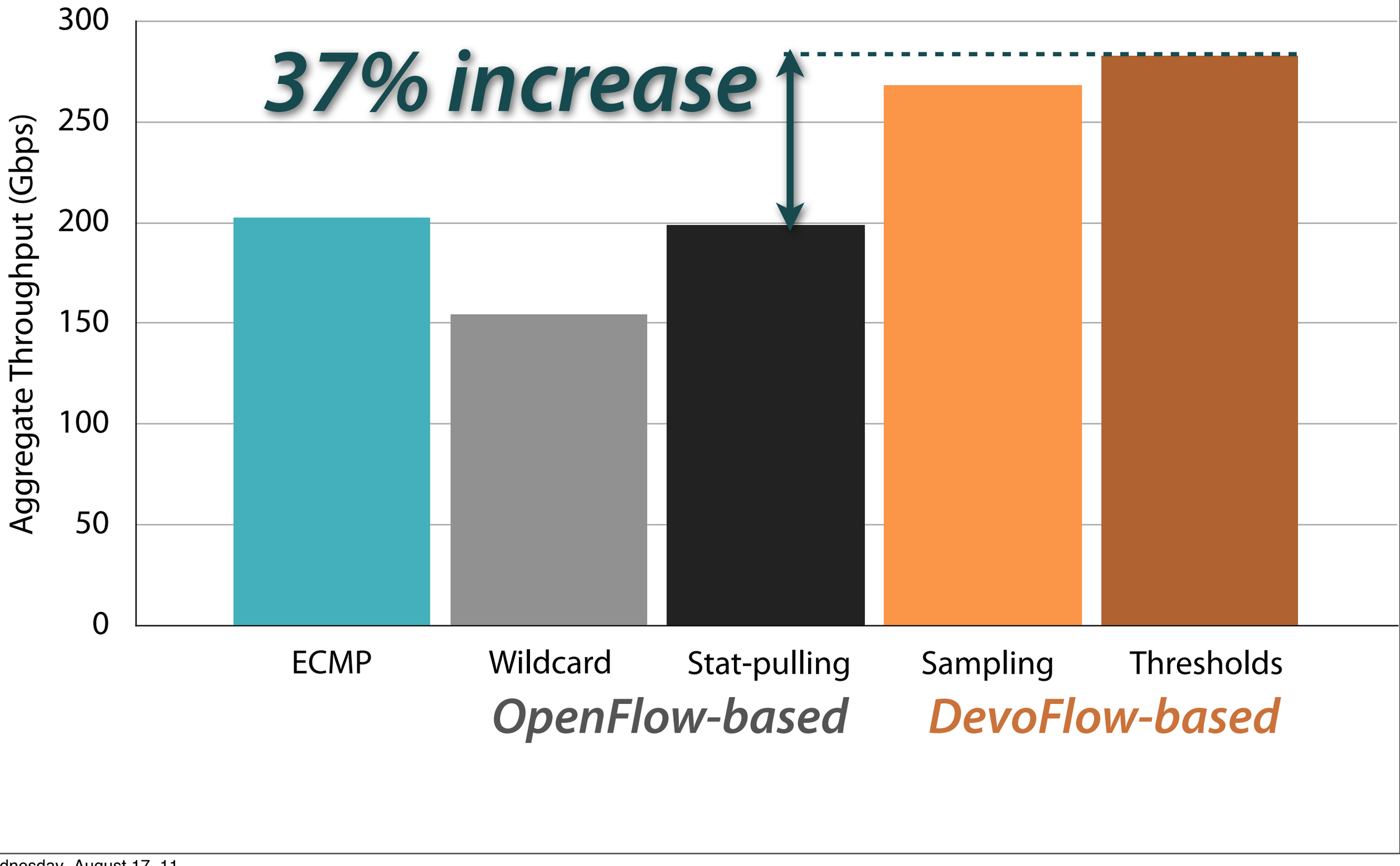
Performance: Clos topology

Shuffle with 400 servers

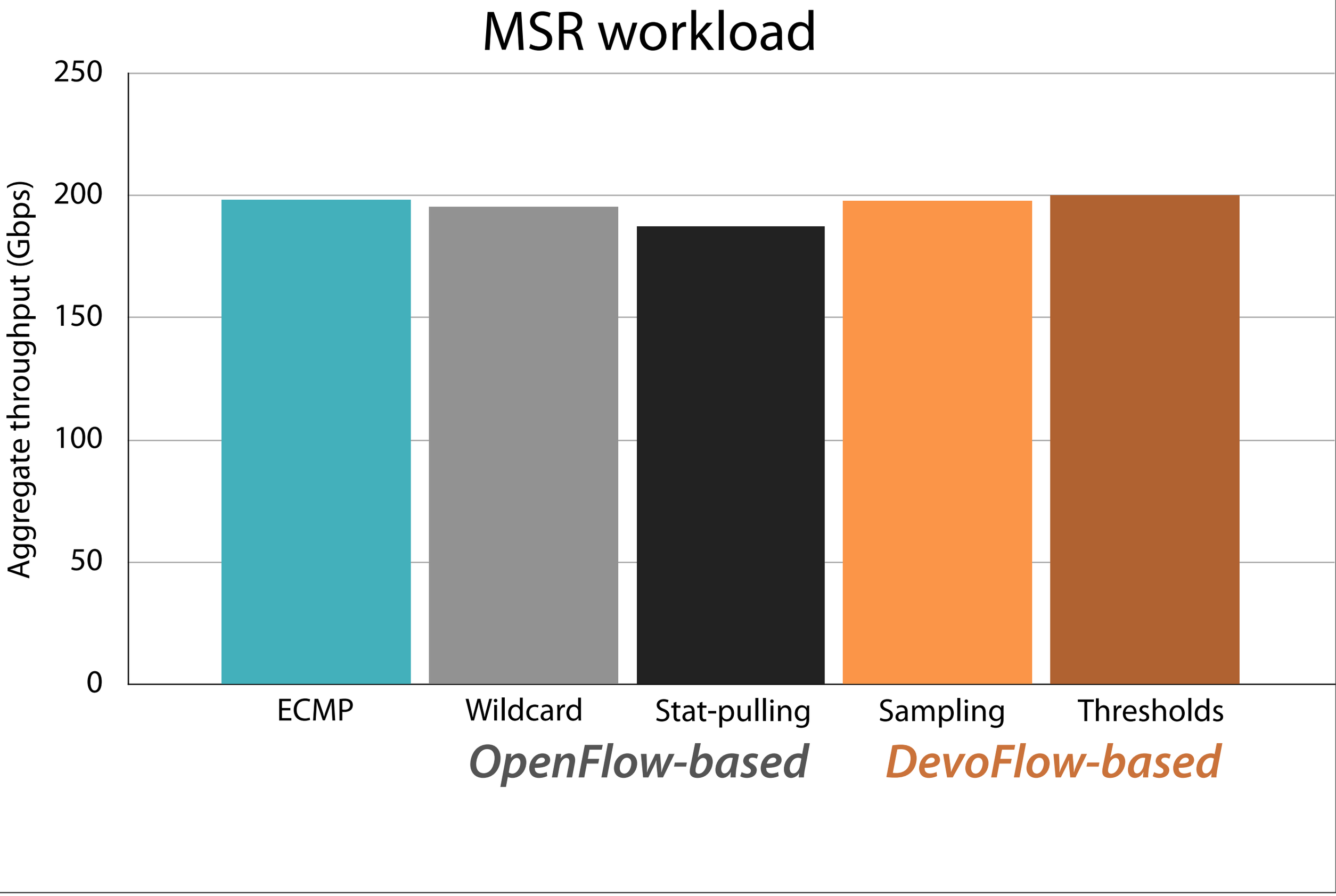


Performance: Clos topology

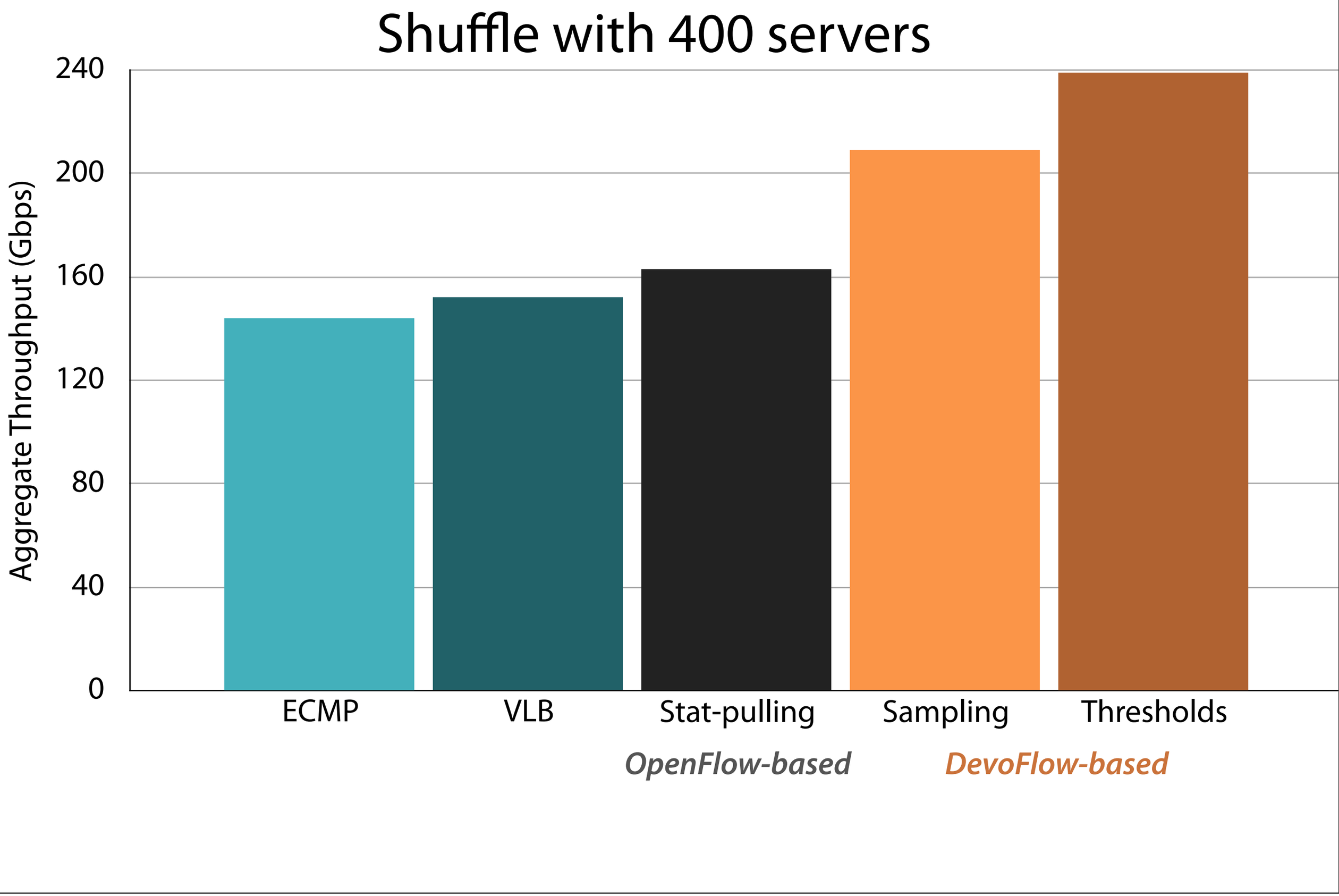
Shuffle with 400 servers



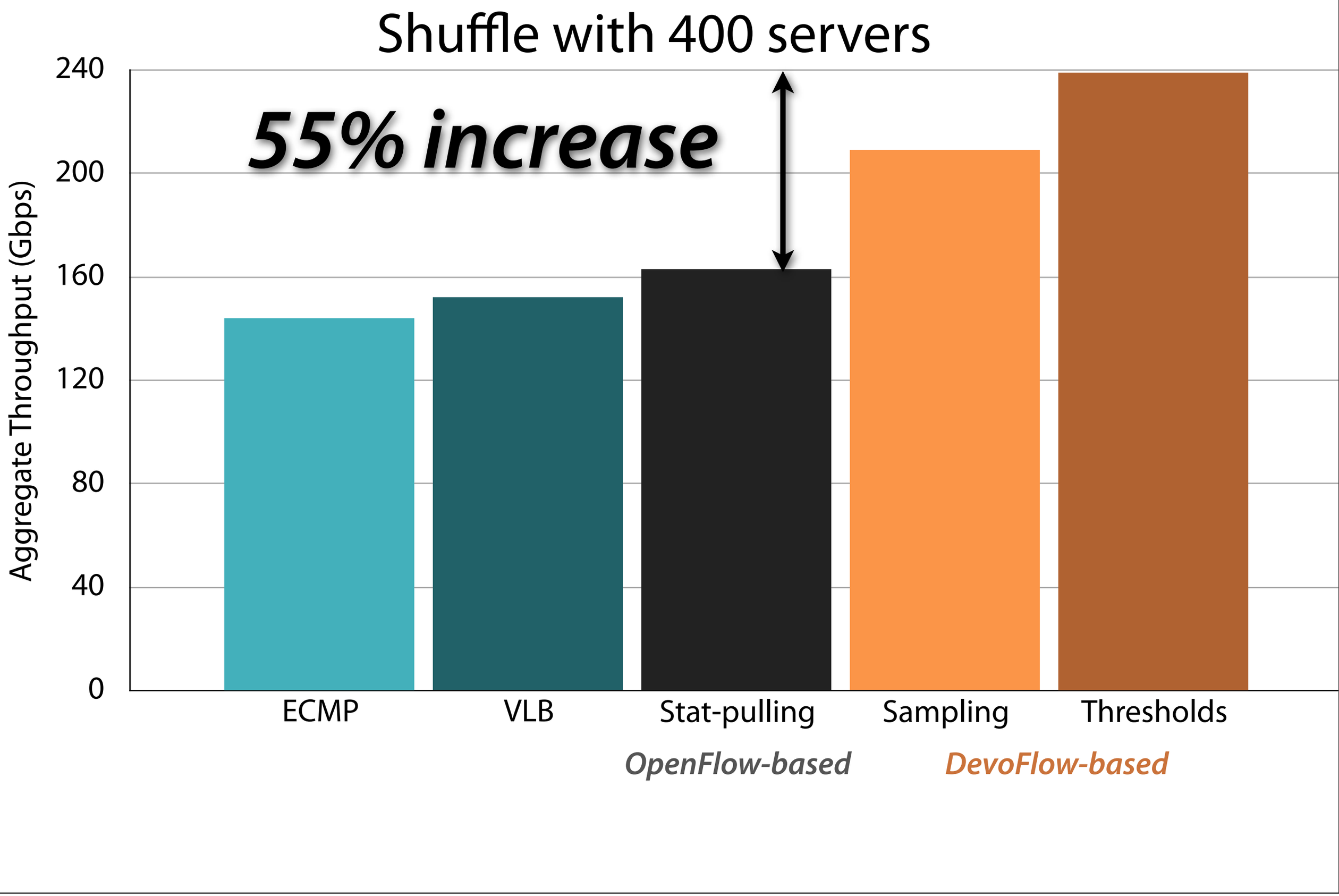
Performance: Clos topology



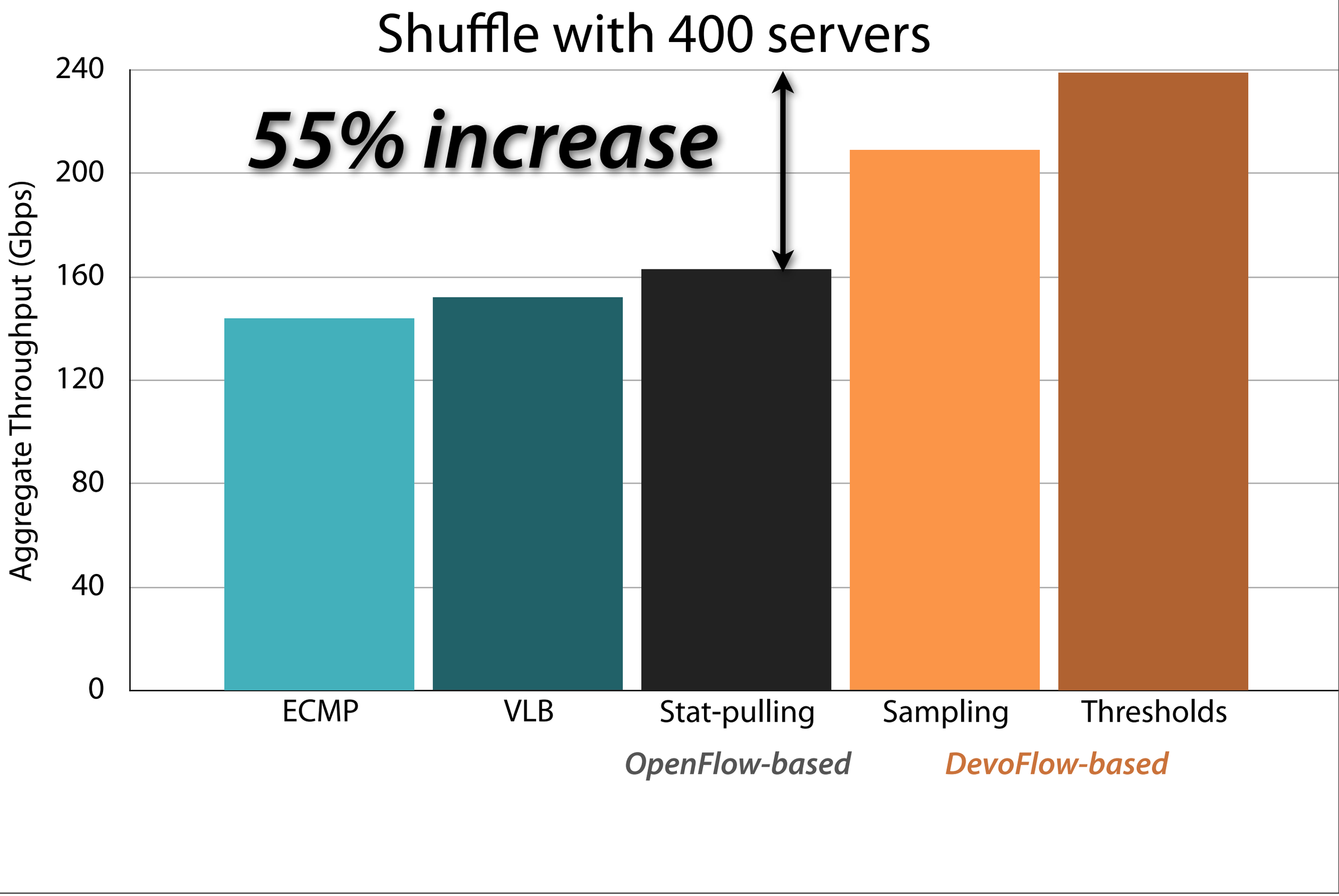
Performance: HyperX topology



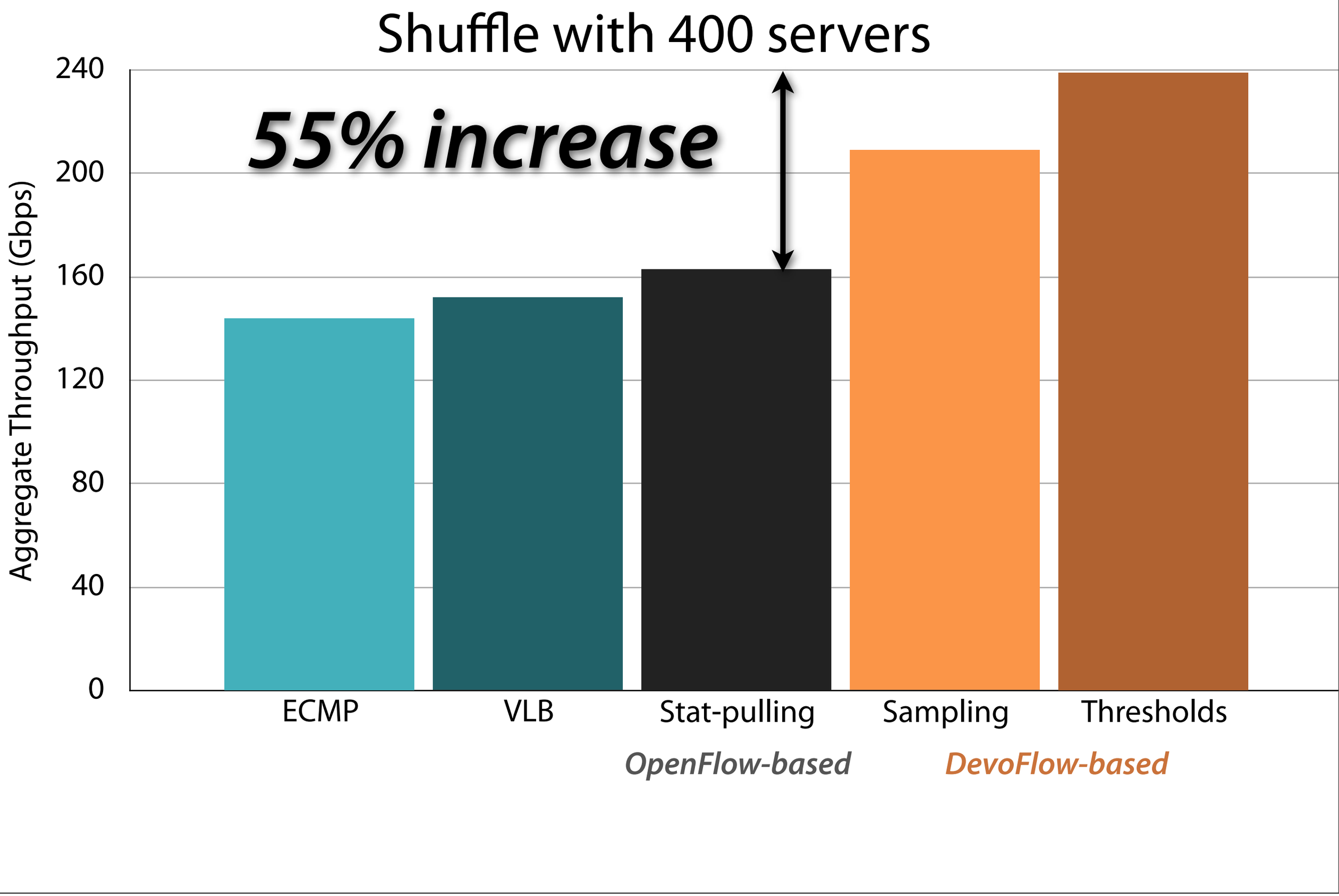
Performance: HyperX topology



Performance: HyperX topology

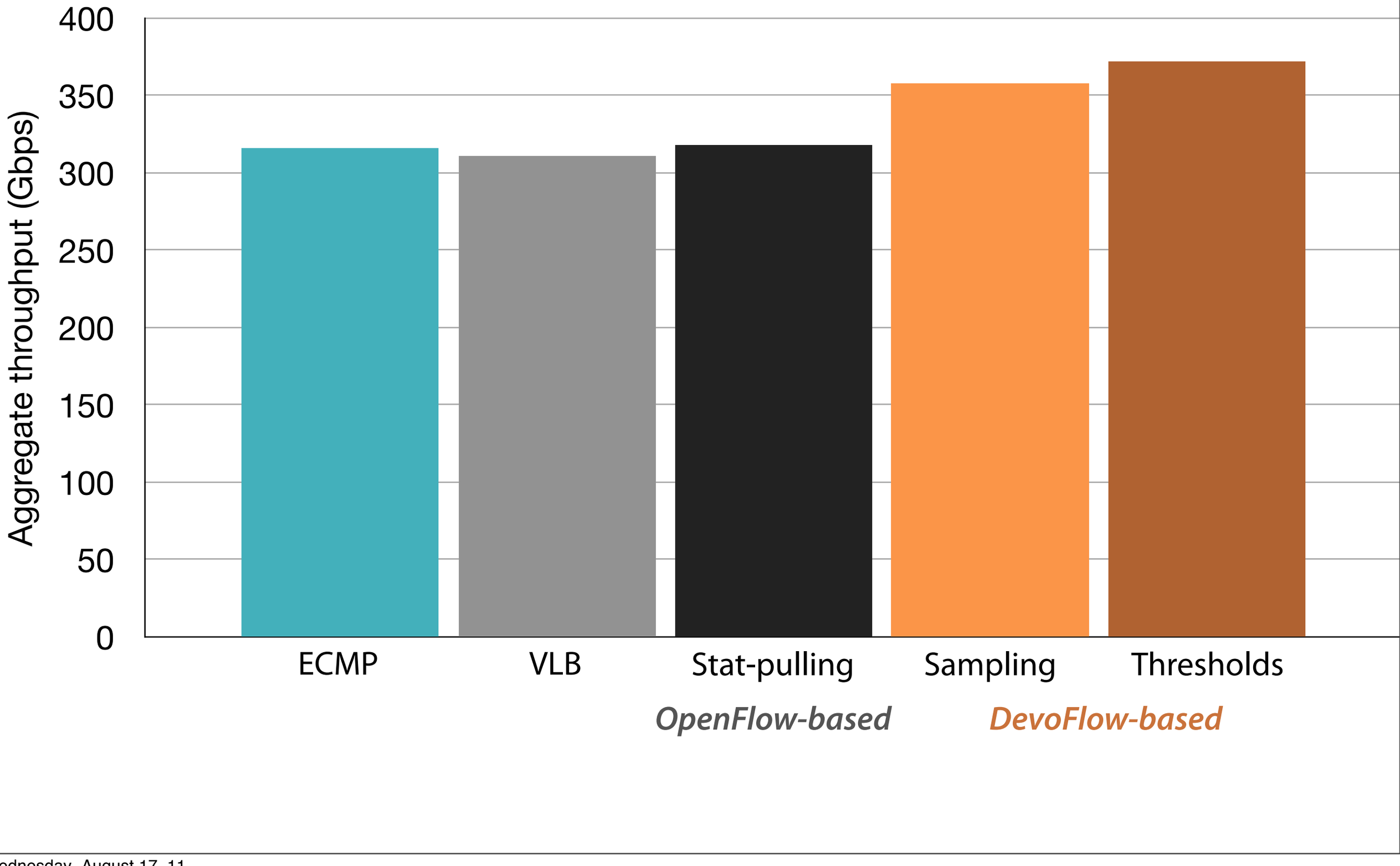


Performance: HyperX topology



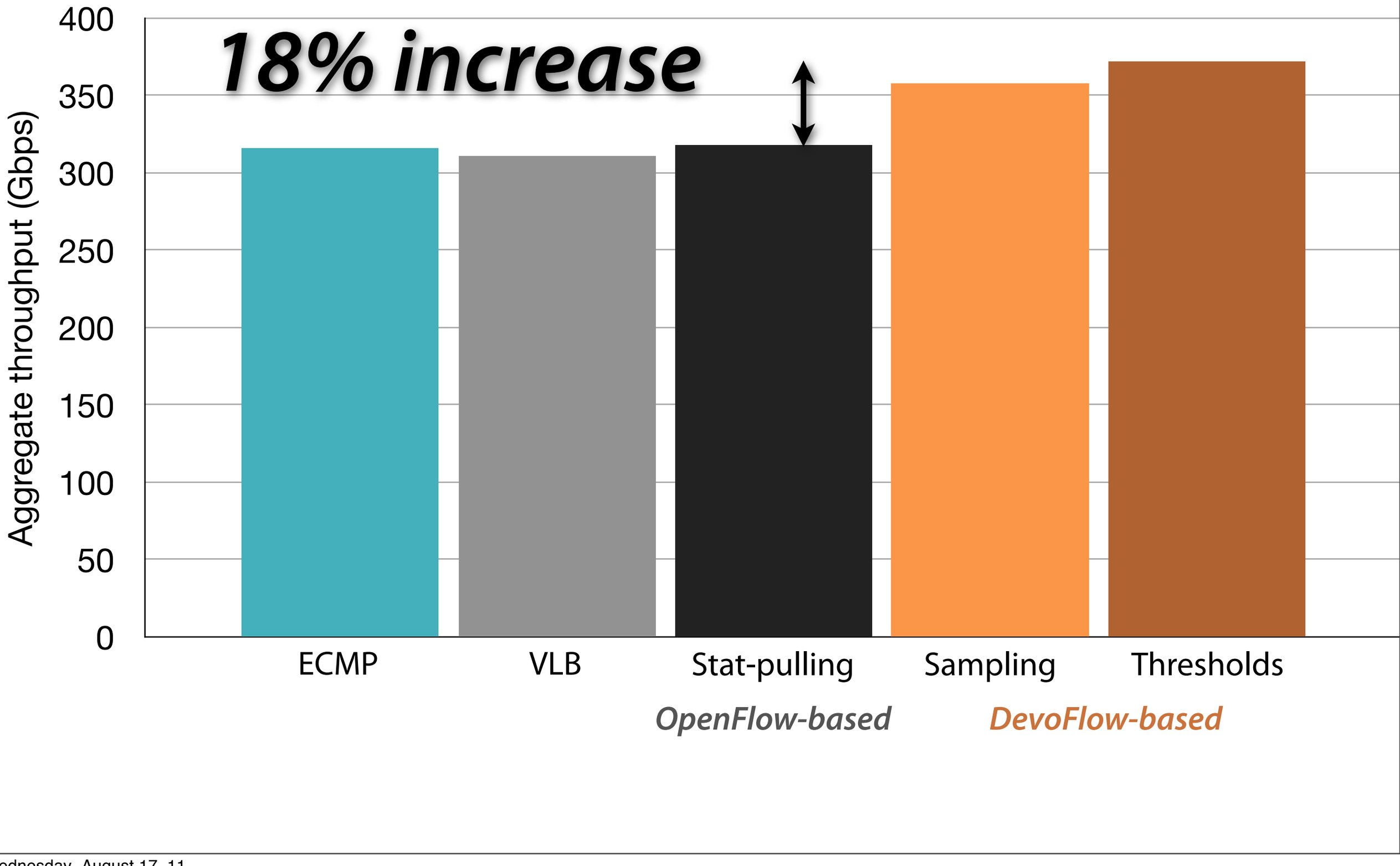
Performance: HyperX topology

Shuffle + MSR workload

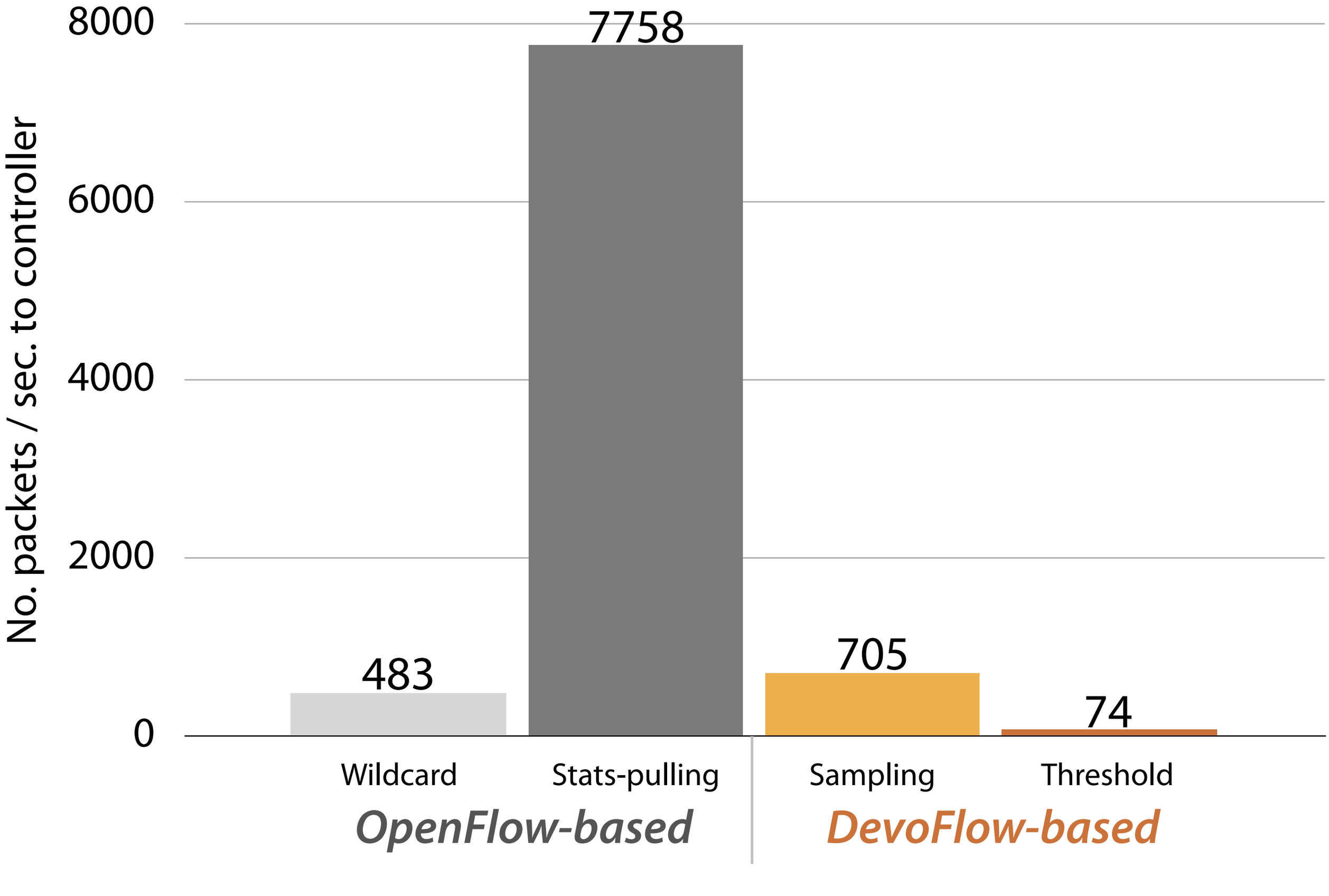


Performance: HyperX topology

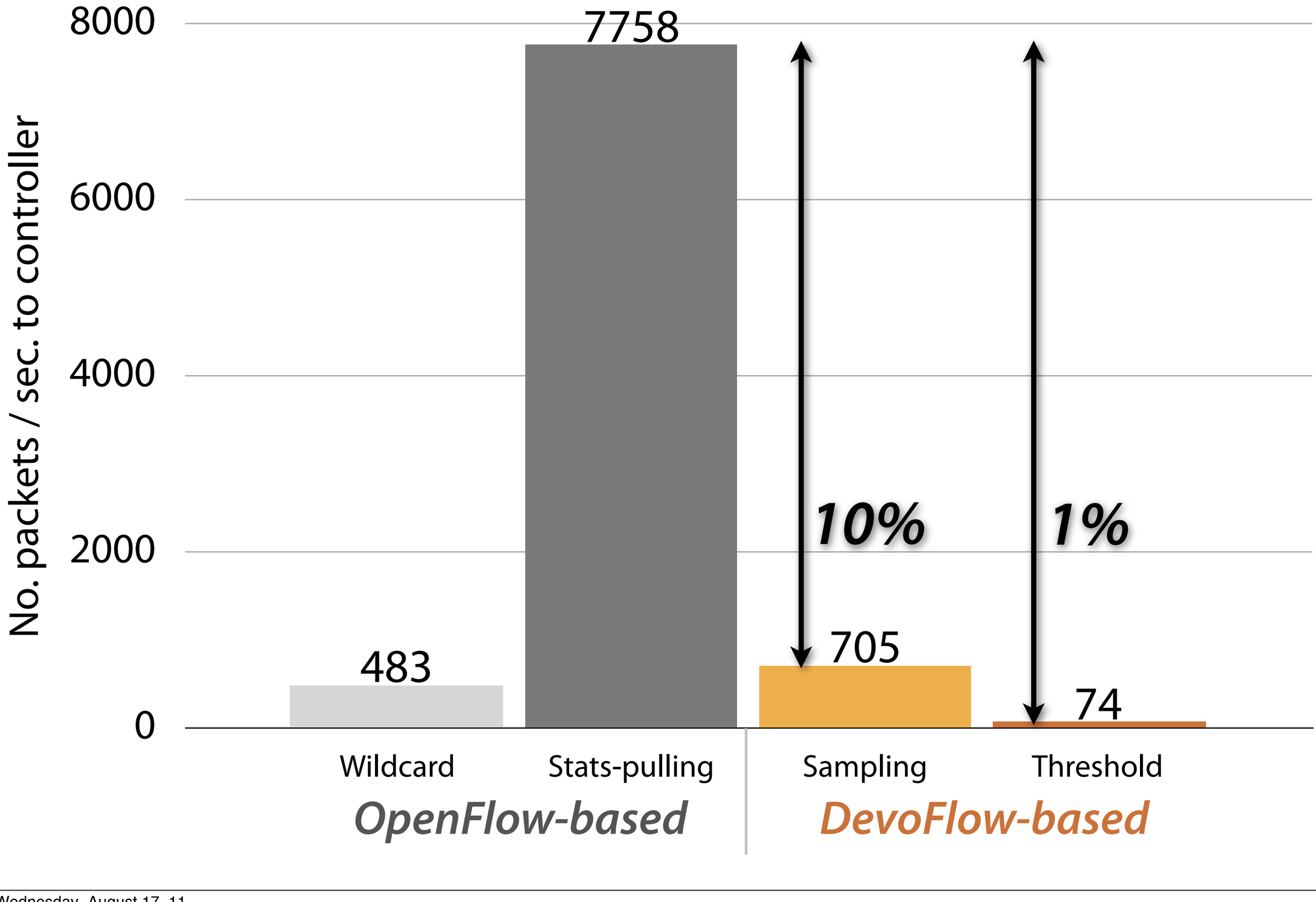
Shuffle + MSR workload



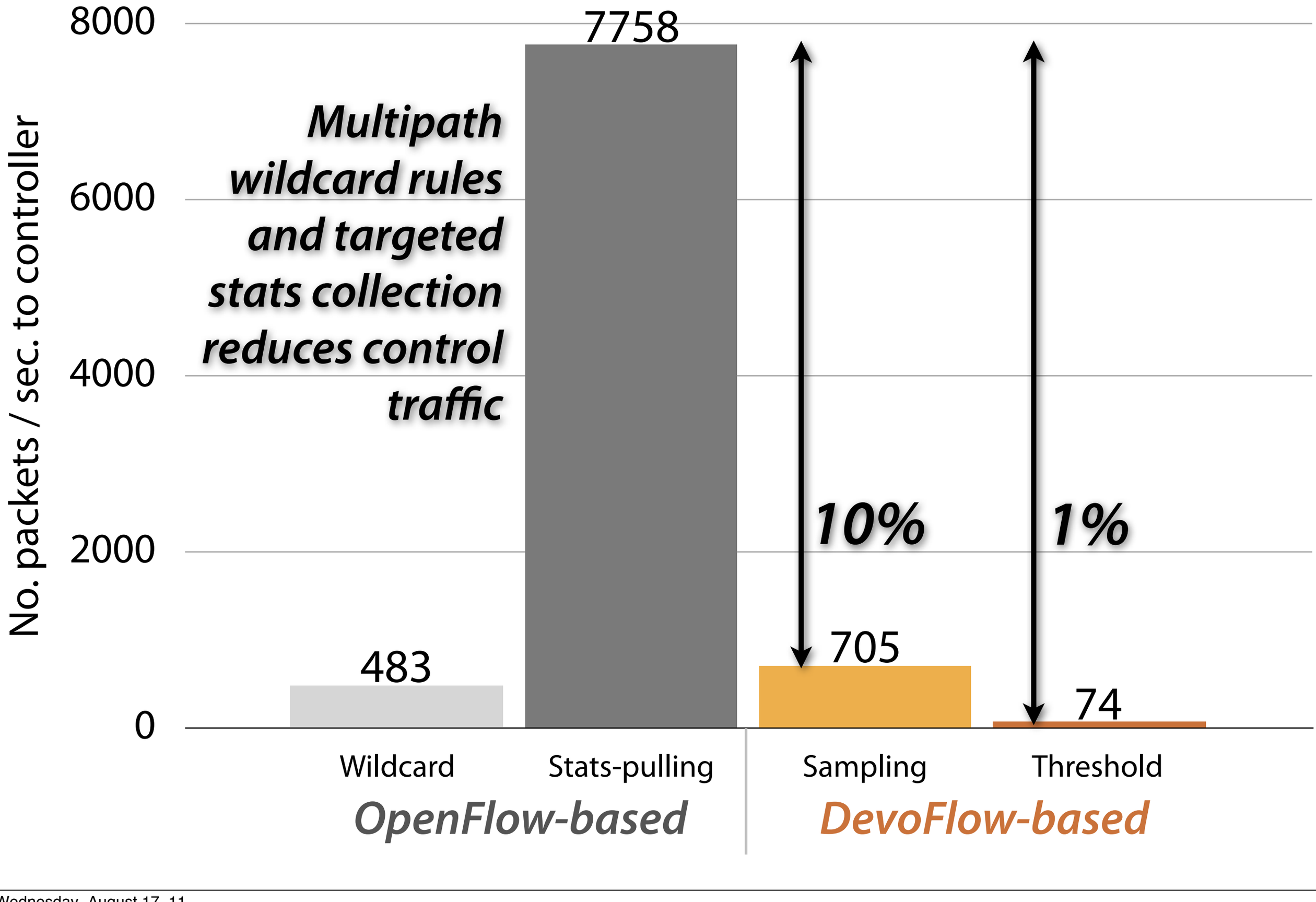
Overheads: Control traffic



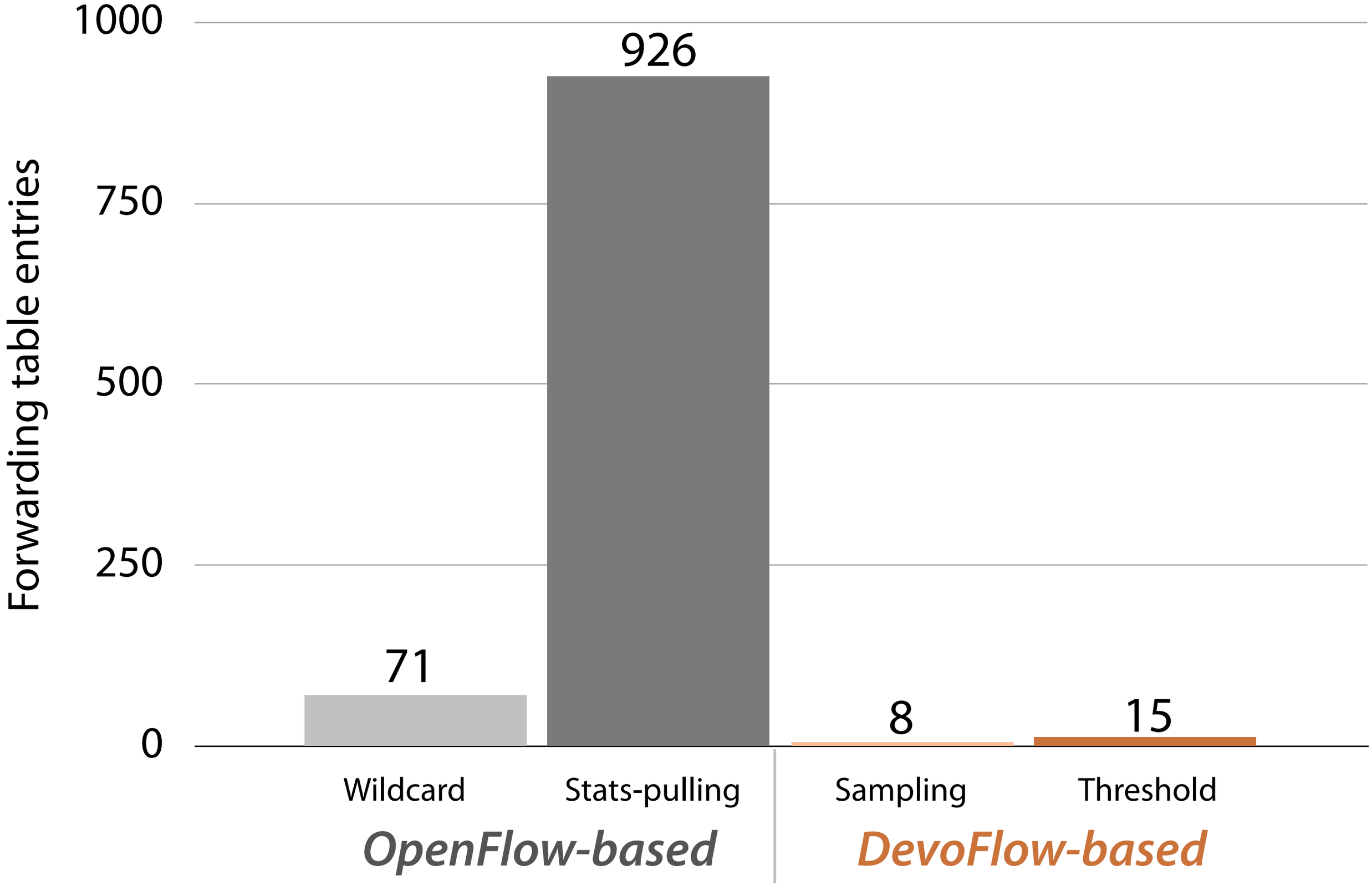
Overheads: Control traffic



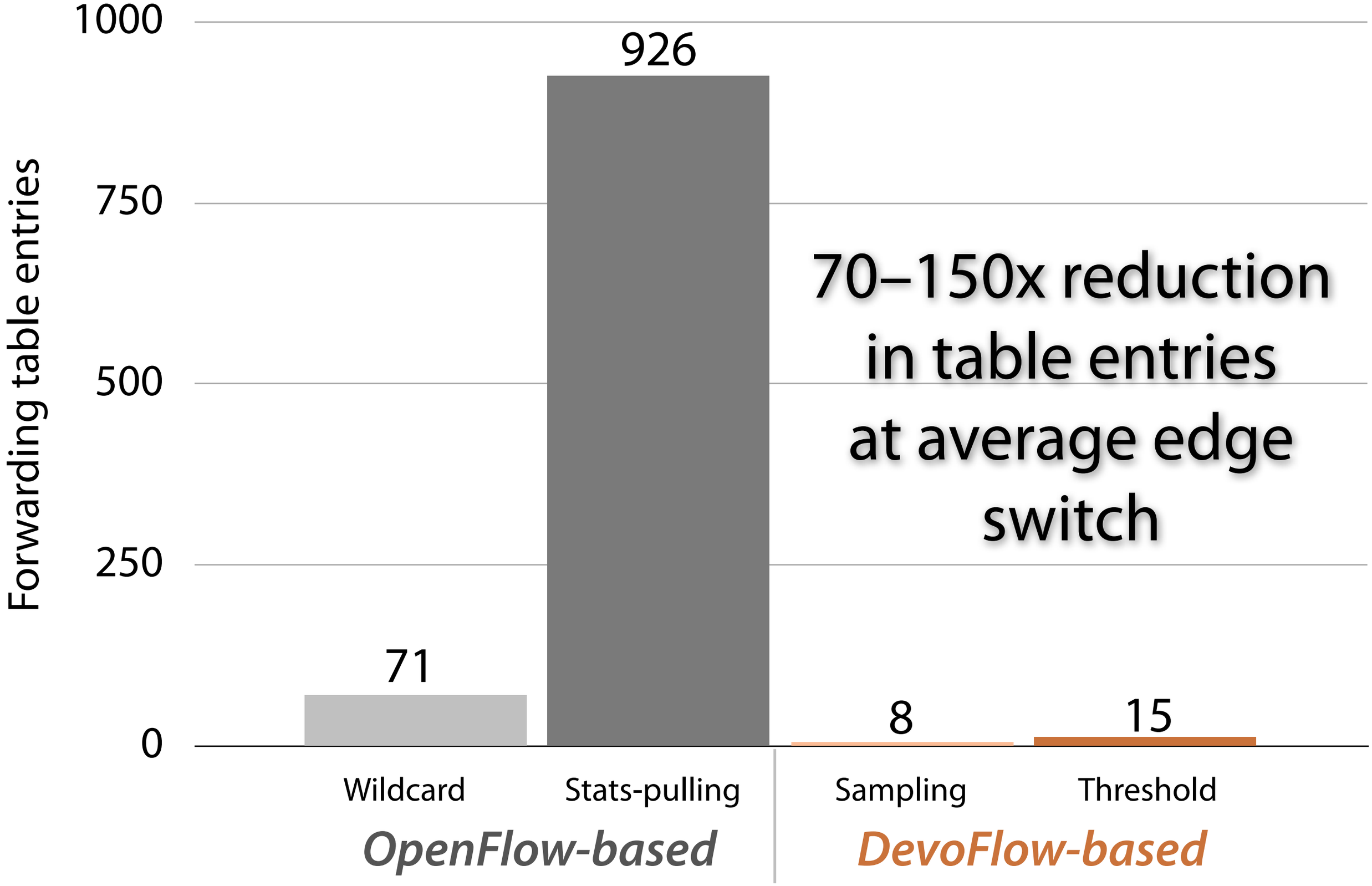
Overheads: Control traffic



Overheads: Flow table entries

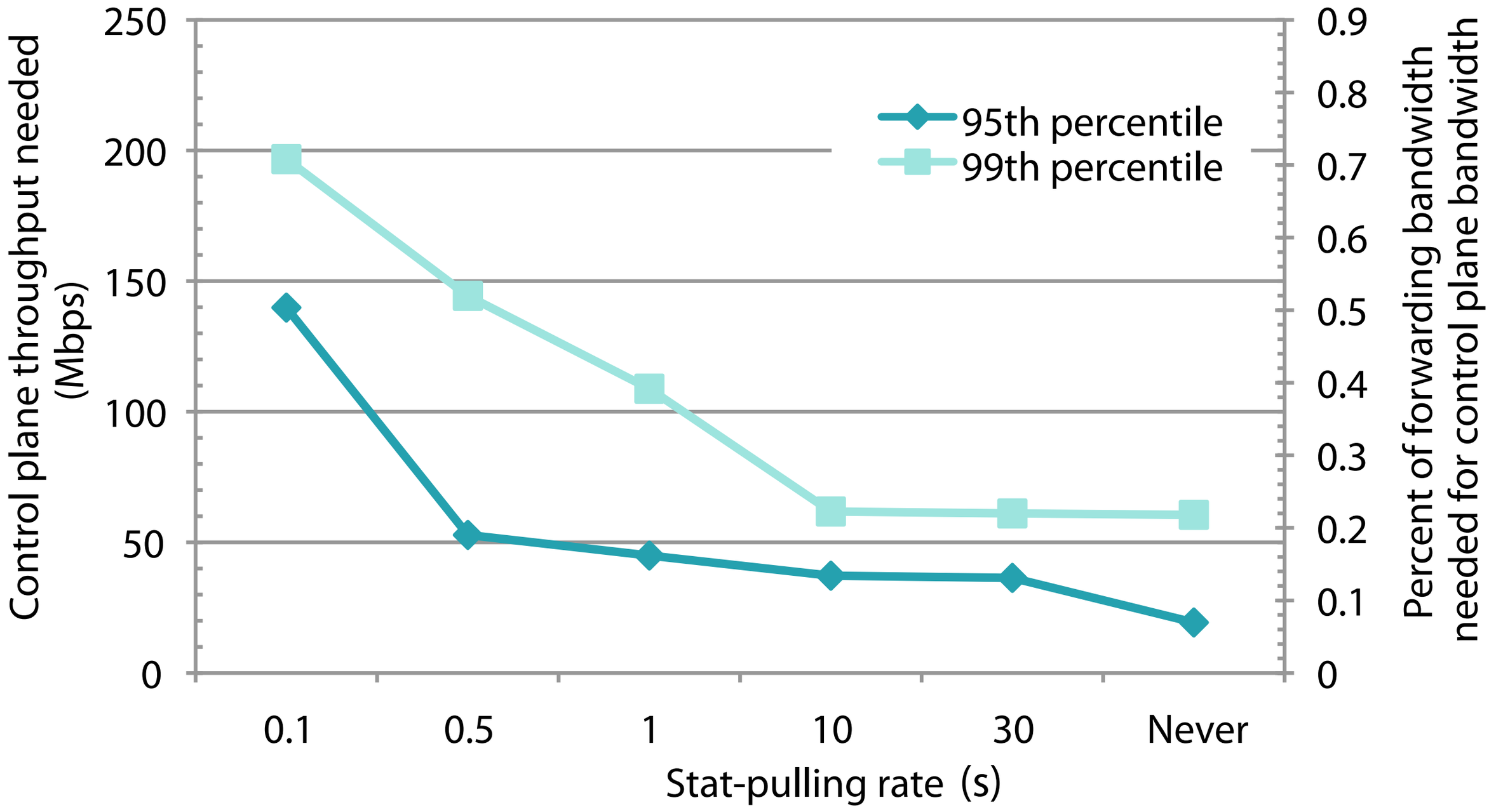


Overheads: Flow table entries



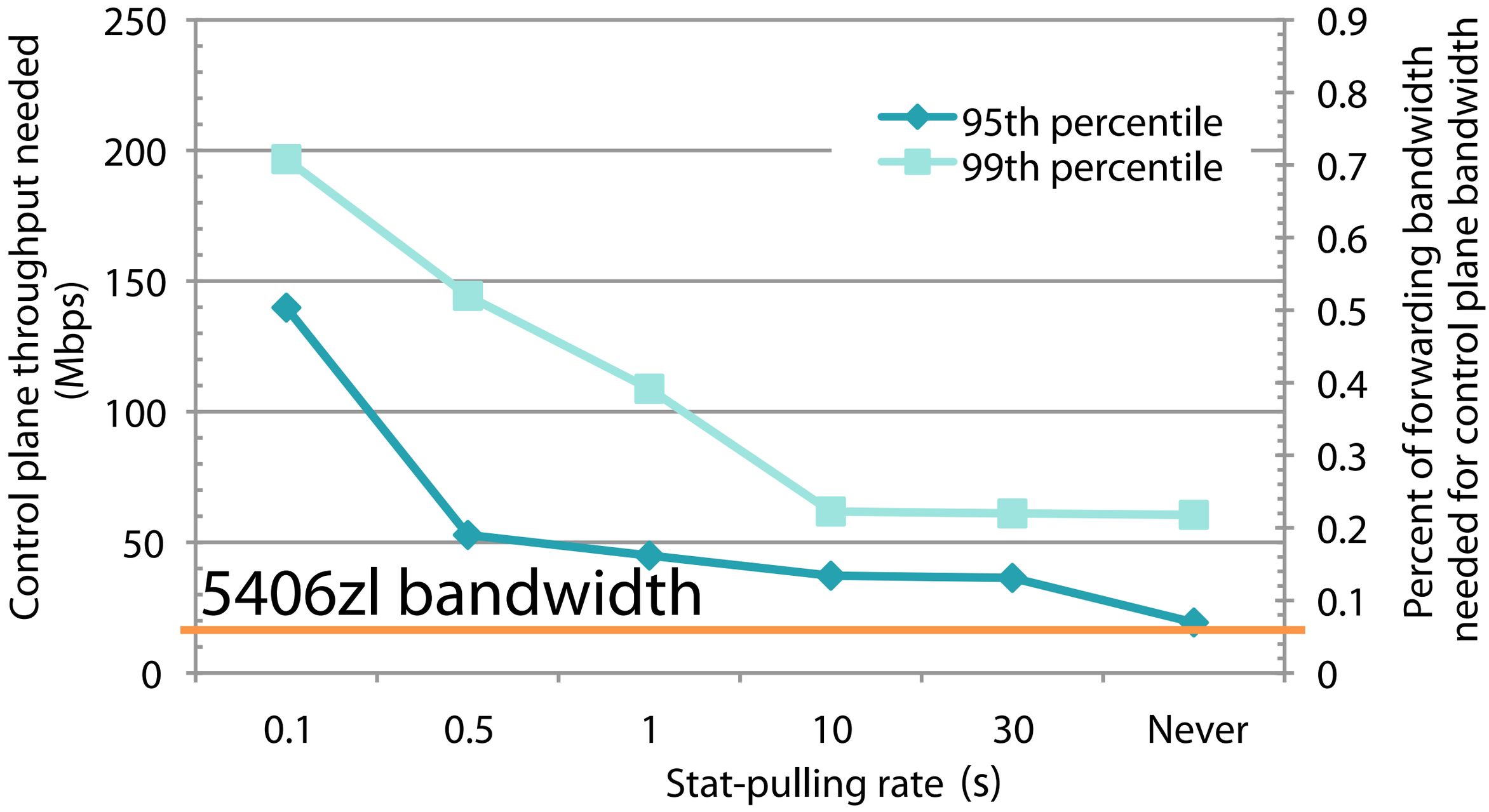
Evaluation: overheads

Control-plane bandwidth needed



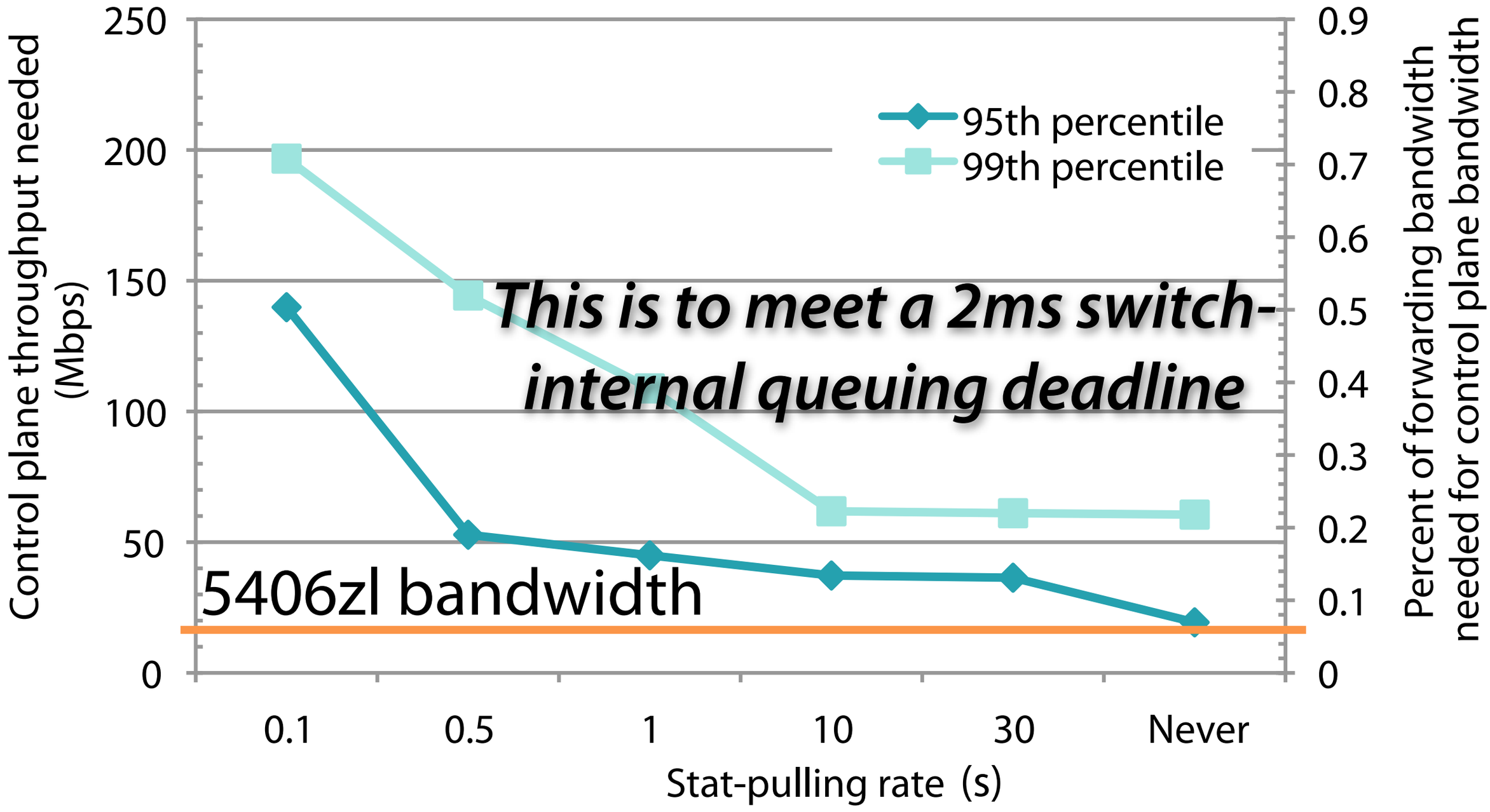
Evaluation: overheads

Control-plane bandwidth needed



Evaluation: overheads

Control-plane bandwidth needed



Conclusions

- OpenFlow imposes high overheads on switches
- Proposed DevoFlow to give tools to reduce reliance on the control-plane
- DevoFlow can reduce overheads by 10–50x for data center flow scheduling

Other uses of DevoFlow

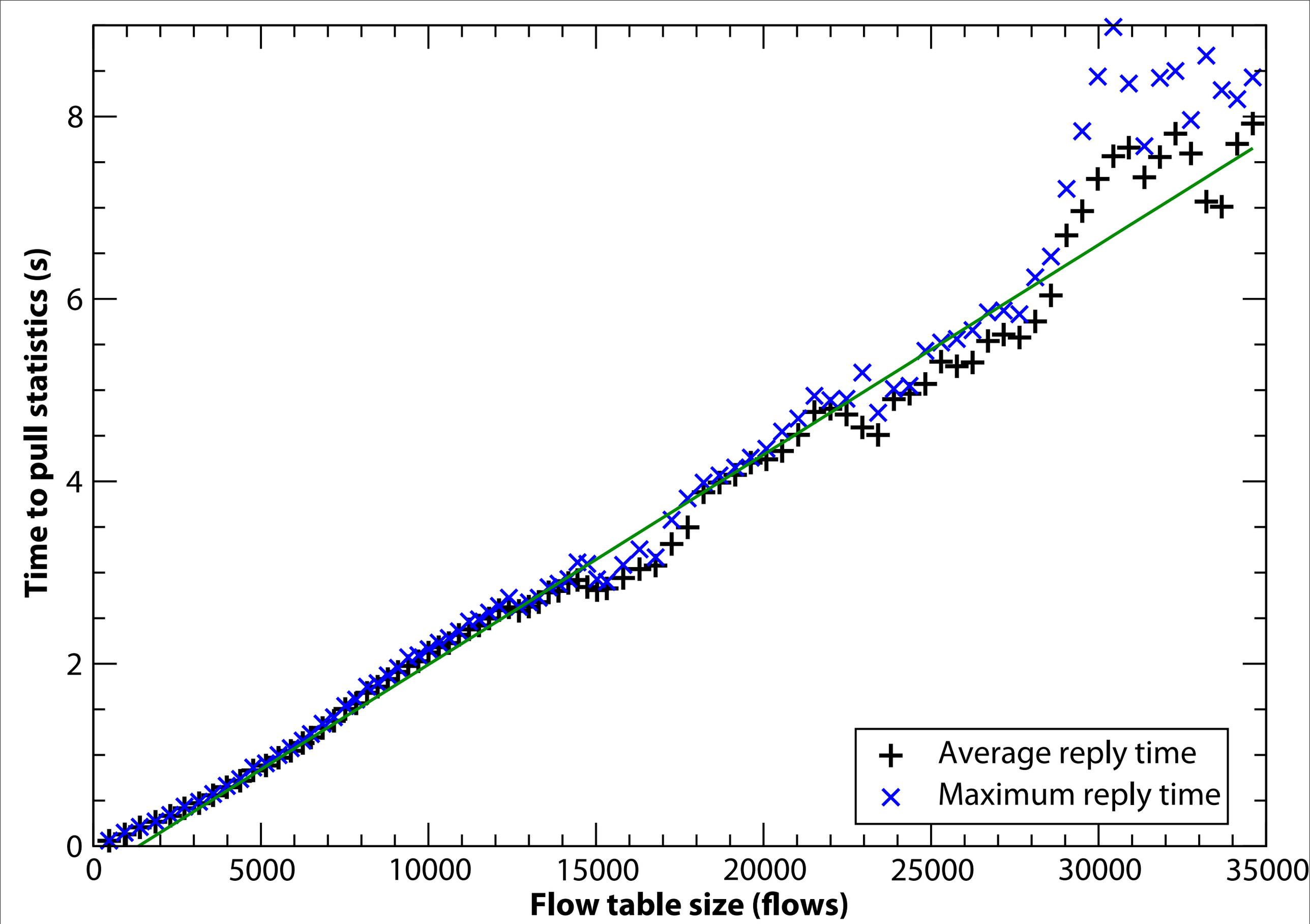
- Client load-balancing (Similar to Wang et al. HotICE 2011)
- Network virtualization [Sherwood et al. OSDI 2010]
- Data center QoS
- Multicast
- Routing as a service [Chen et al. INFOCOM 2011]
- Energy-proportional routing [Heller et al. NSDI 2010]

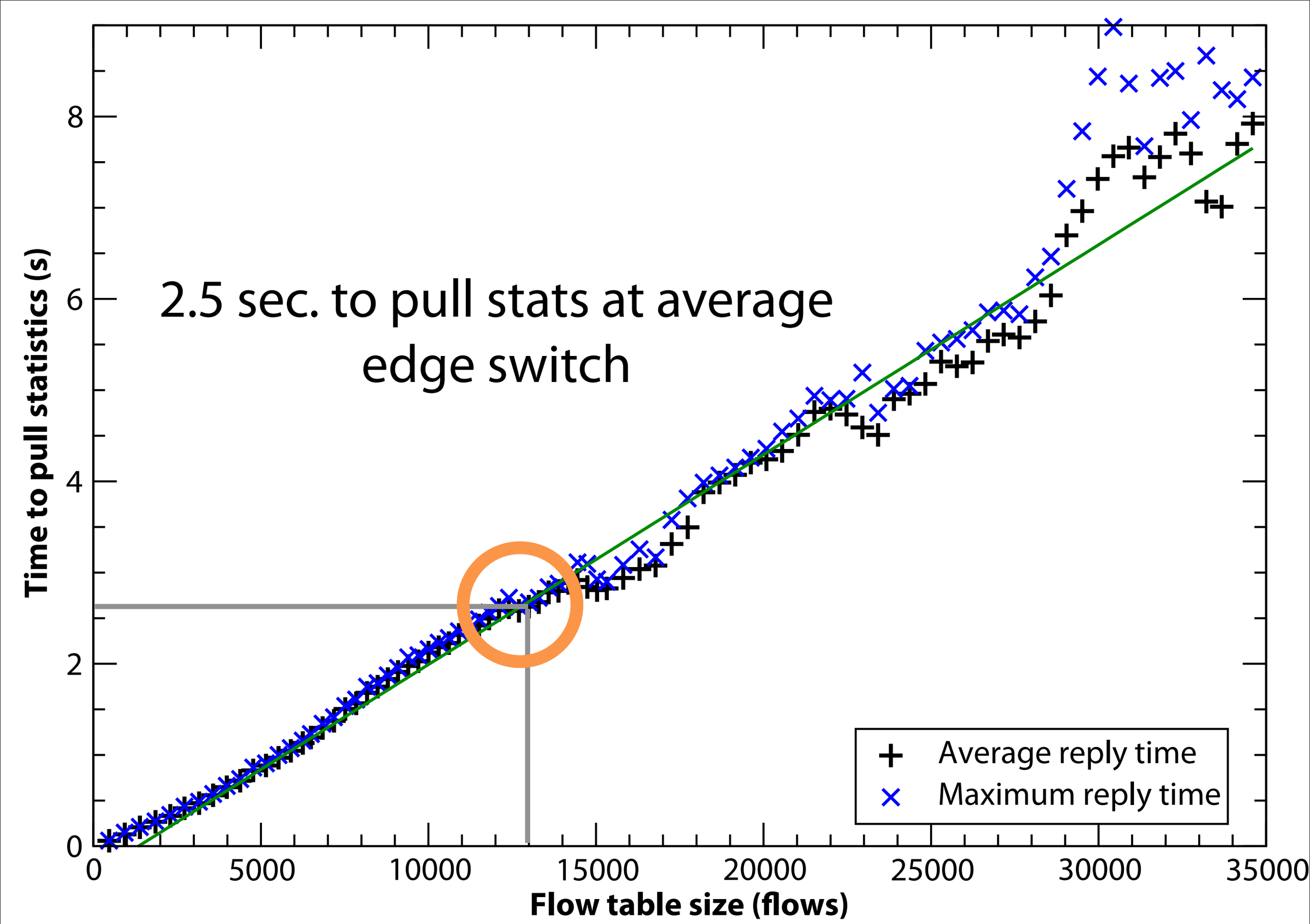
Implementing DevoFlow

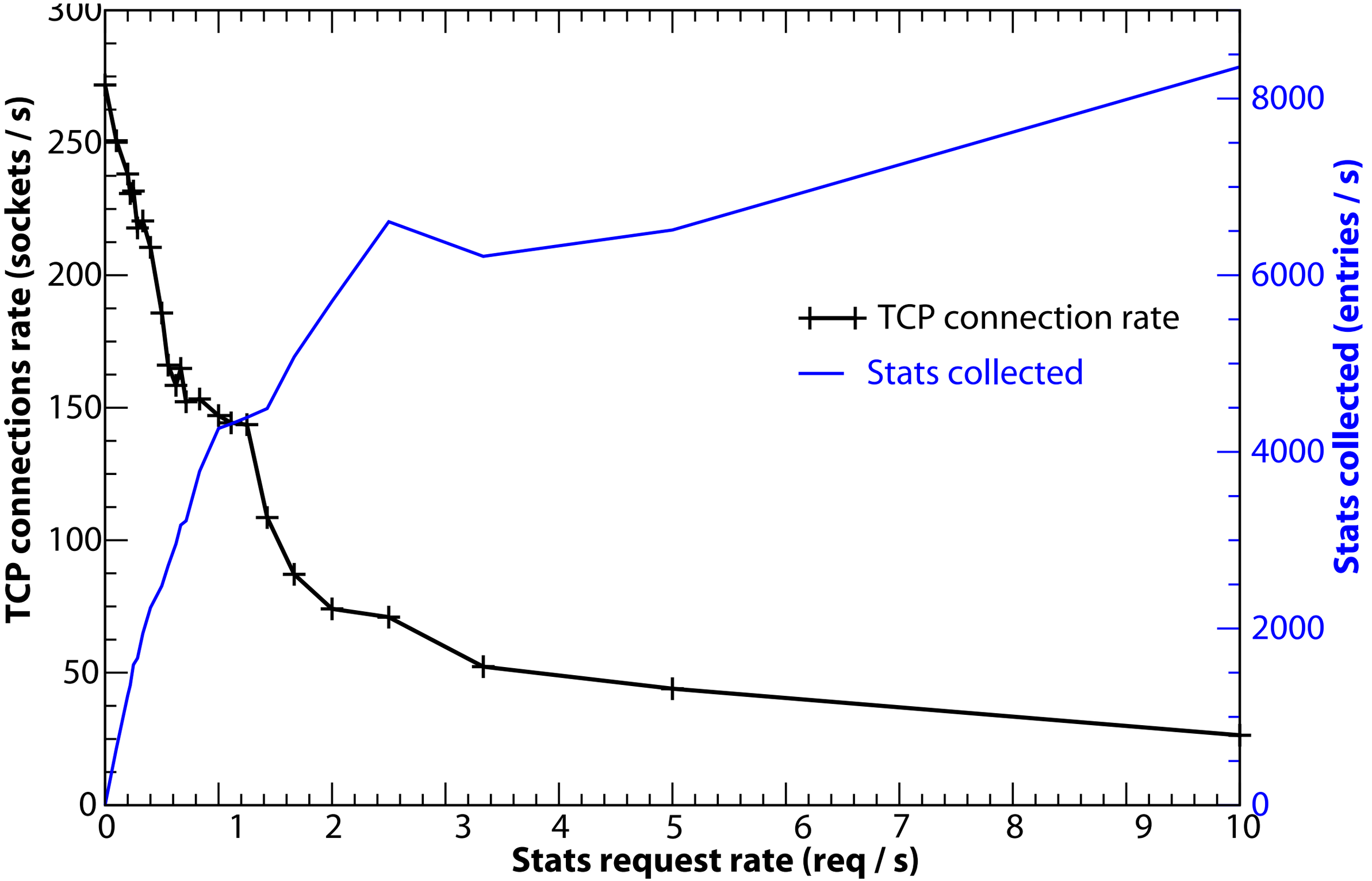
- Rule cloning:
  - May be difficult to implement on ASIC. Can definitely be done with use of switch CPU
- Multipath support:
  - Similar to LAG and ECMP
- Sampling:
  - Already implemented in most switches
- Triggers:
  - Similar to rate limiters

Flow table size

- Constrained resource
  - Commodity switches:
    32K–64K exact match entries
    ~1500 TCAM entries
- Virtualization may strain table size
  - 10s of VMs per machine implies >100K table entries


Time to pull statistics (s)
0
2
4
6
8
0
5000
10000
15000
20000
25000
30000
35000
Flow table size (flows)
Average reply time
Maximum reply time

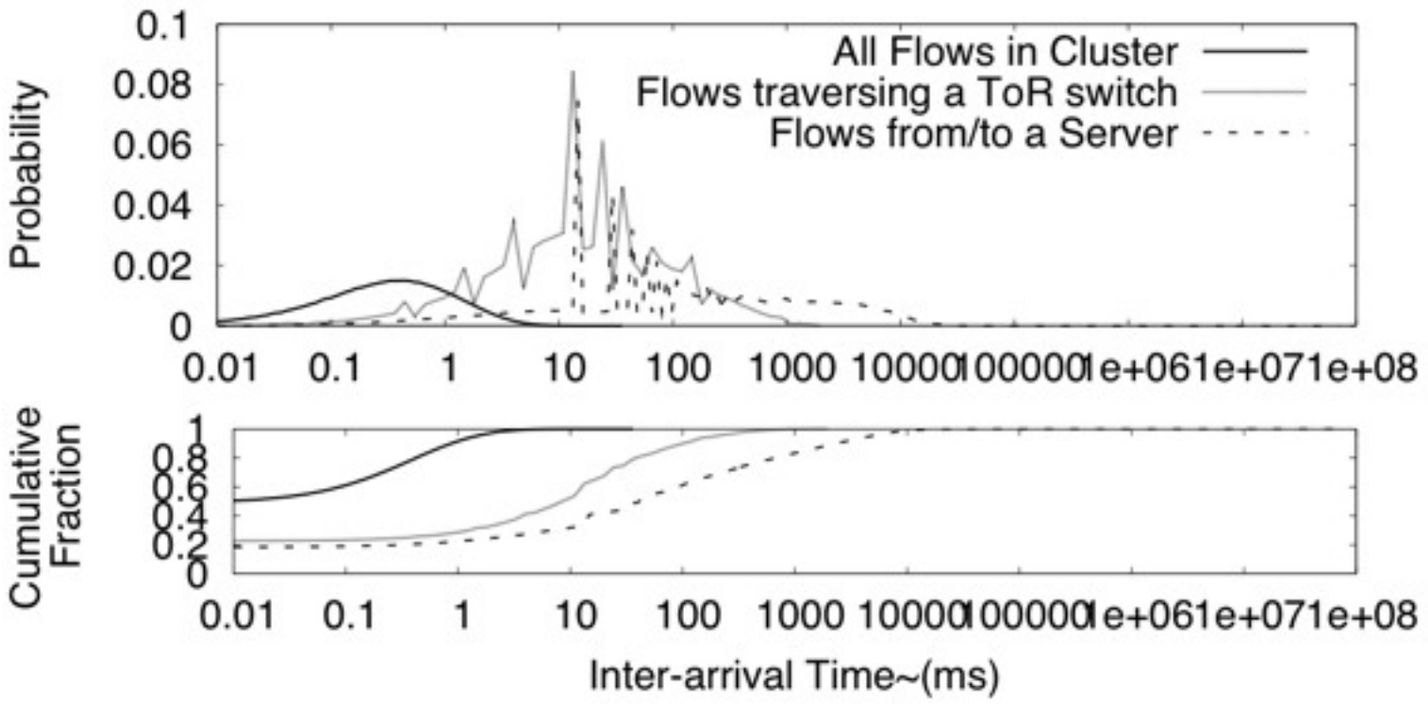

2.5 sec. to pull stats at average edge switch
Time to pull statistics (s)
0
2
4
6
8
0
5000
10000
15000
20000
25000
30000
35000
Flow table size (flows)
Average reply time
Maximum reply time


TCP connections rate (sockets / s)
Stats collected (entries / s)
TCP connection rate
Stats collected
Stats request rate (req / s)
0
50
100
150
200
250
300
0
2000
4000
6000
8000
0
1
2
3
4
5
6
7
8
9
10

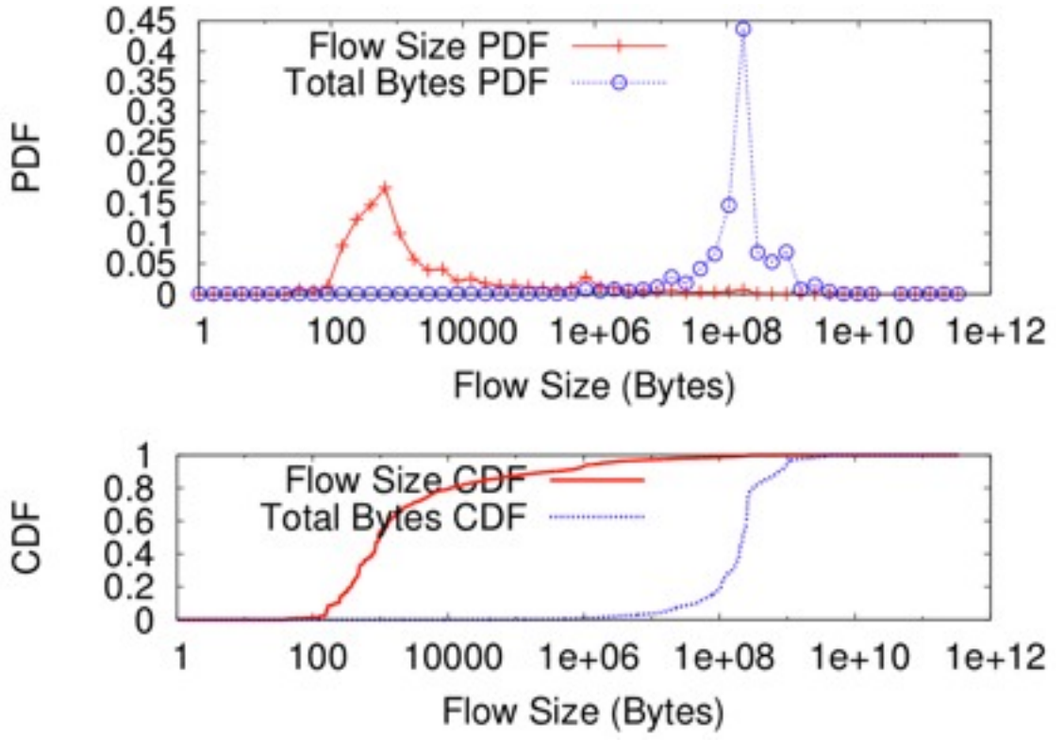
Evaluation: methodology

- Workloads
  - Reverse-engineered MSR workload [Kandula et al. IMC 2009]

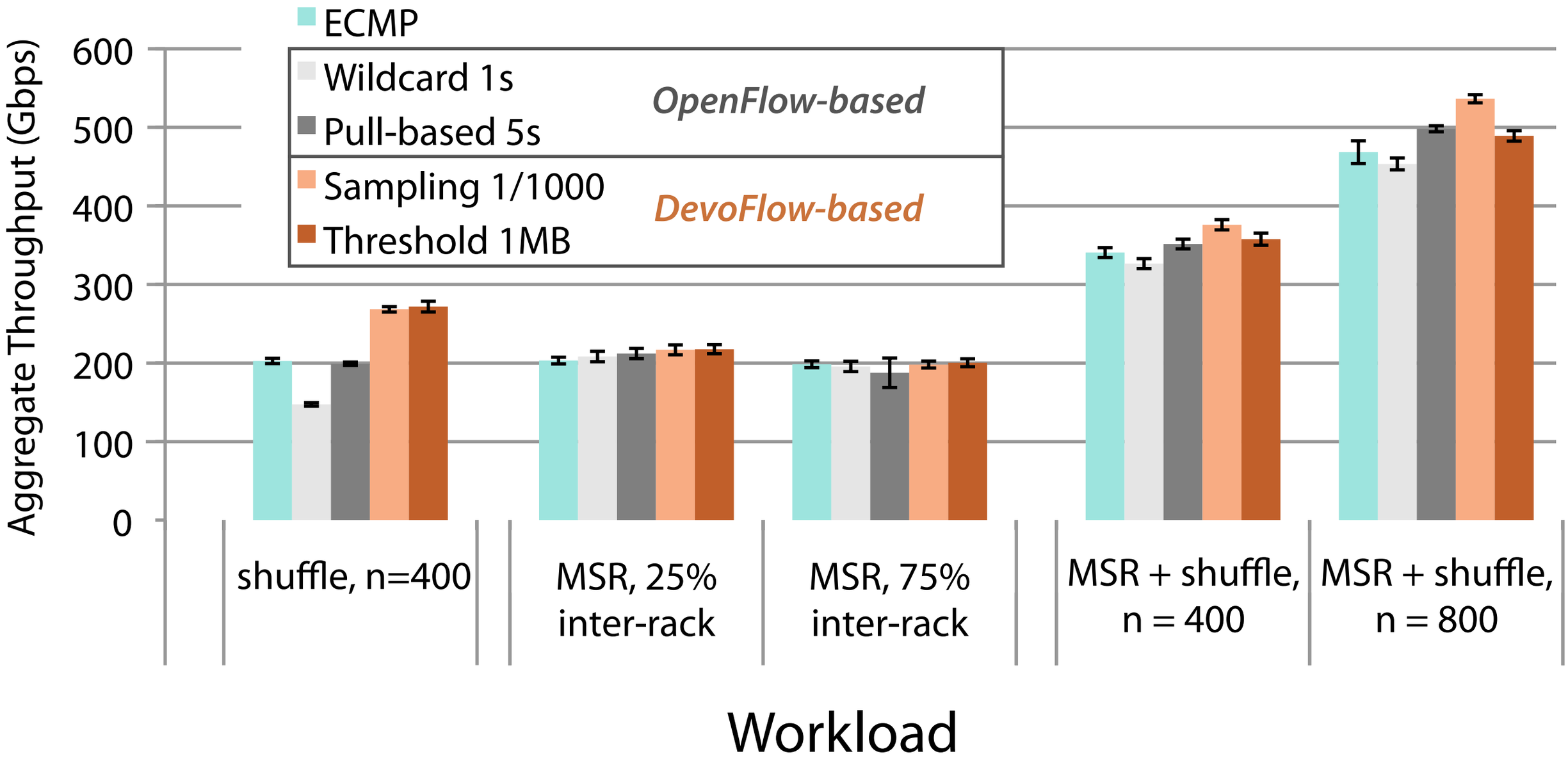
Inter-arrival times



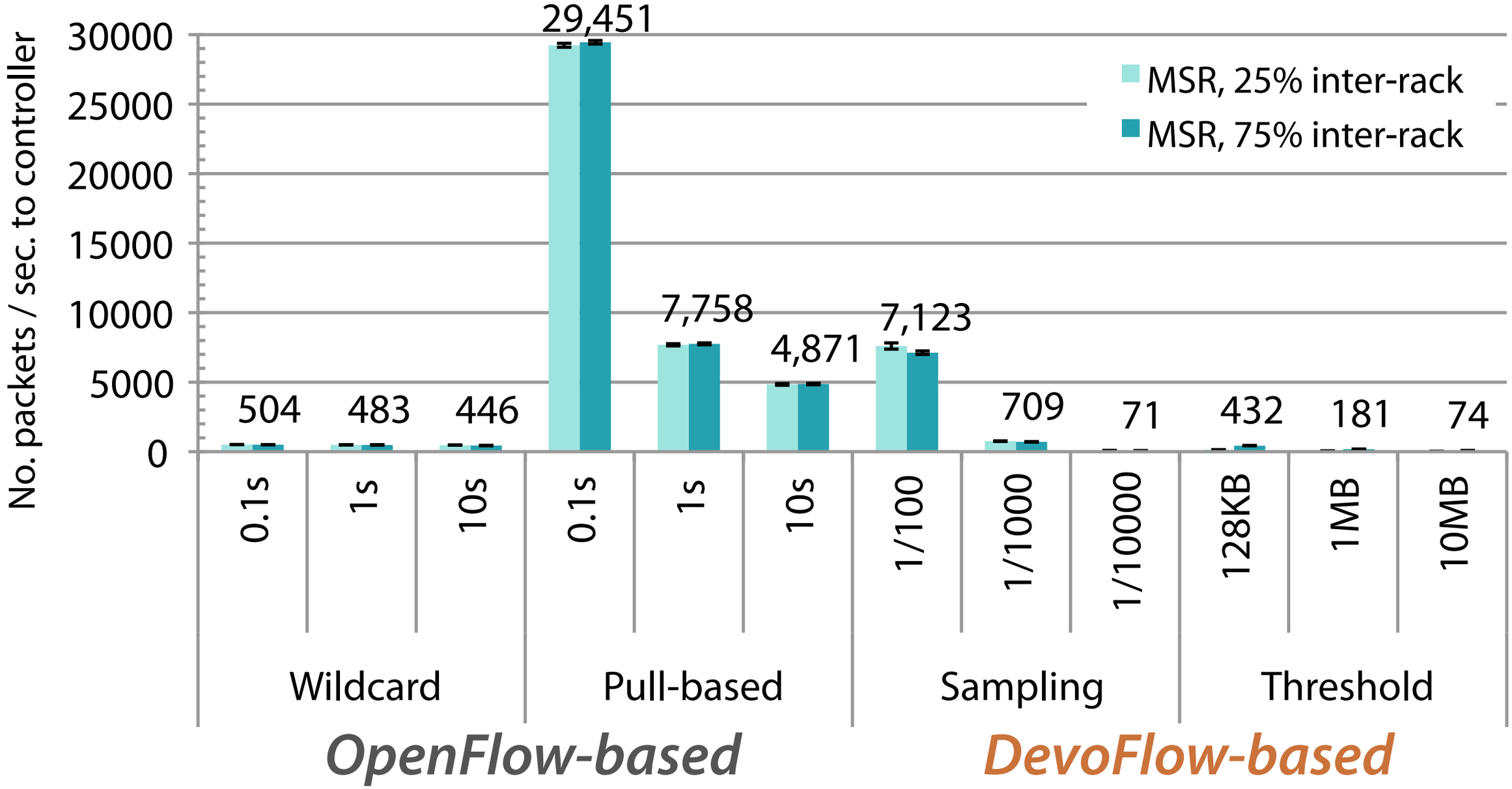
Flow sizes



Evaluation: performance
Clos topology

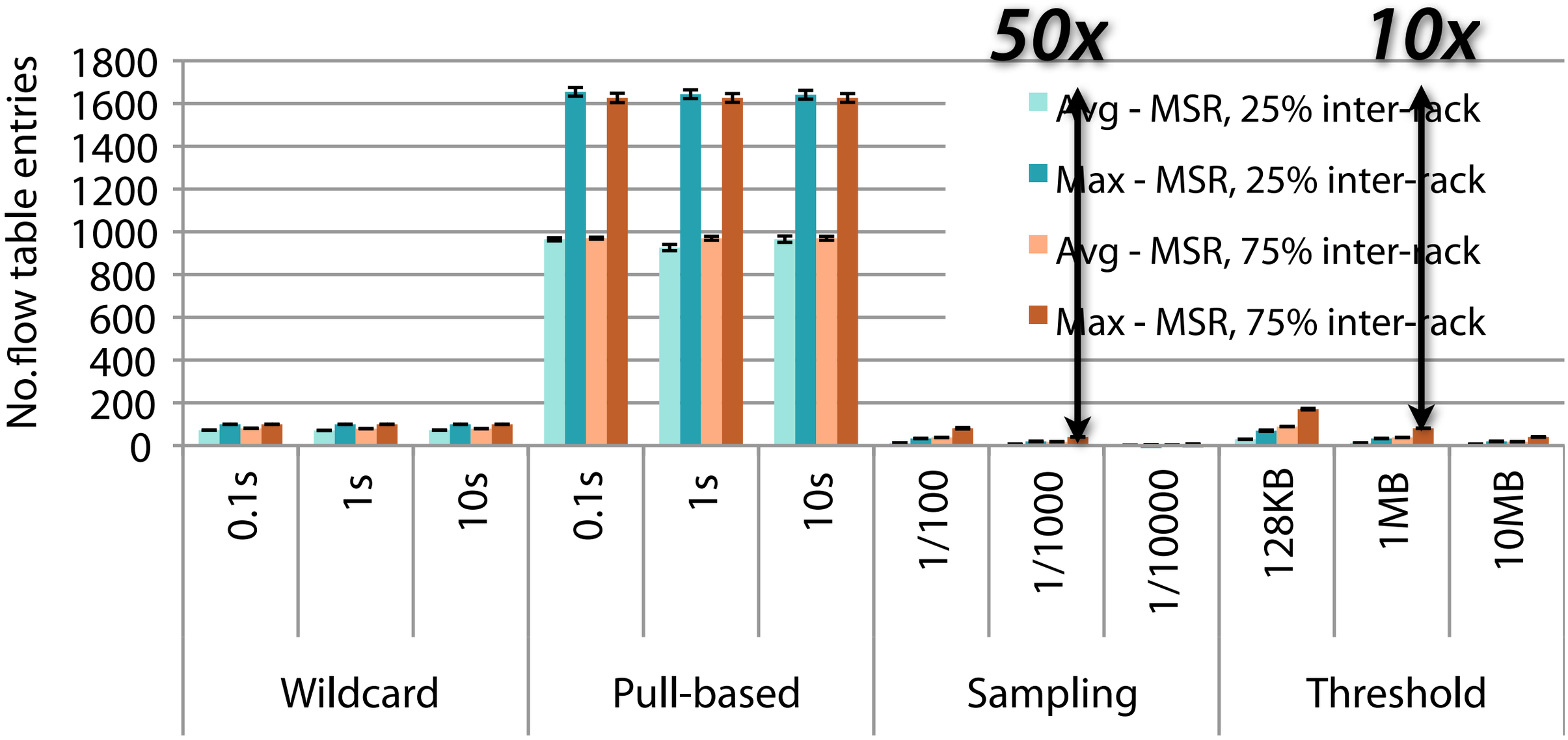


Evaluation: overheads
Control traffic



Evaluation: overheads

Flow table entries



OpenFlow-based *DevoFlow-based*

DevoFlow aggressively uses multipath wildcard rules