Software-Defined Networking

Protocols for Data Networks

(aka Advanced Topics in Networking)

Fernando M. V. Ramos, fvramos@ciencias.ulisboa.pt, PRD-RAC

Lecture plan

An introduction to Software-Defined Networking

[Onix]

The first production-level SDN controller

[B4]

Google's WAN SDN

[SWAN]

Microsoft's WAN SDN

Lecture plan

An introduction to Software-Defined Networking

[Onix]

The first production-level SDN controller

[B4]

Google's WAN SDN

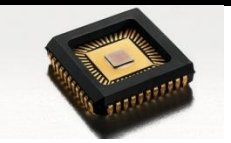
[SWAN]

Microsoft's WAN SDN

Key characteristics of SDN

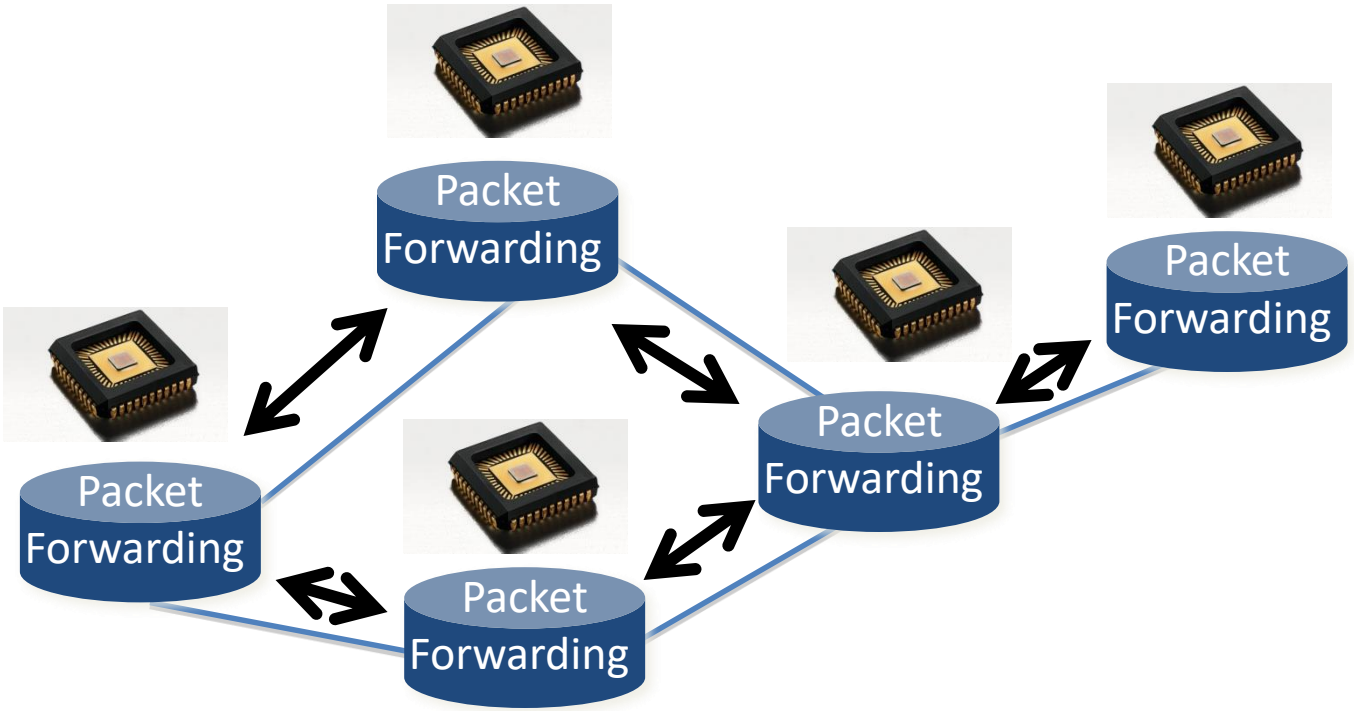
- Control and data planes decoupled
  - The enabler

Traditional networking

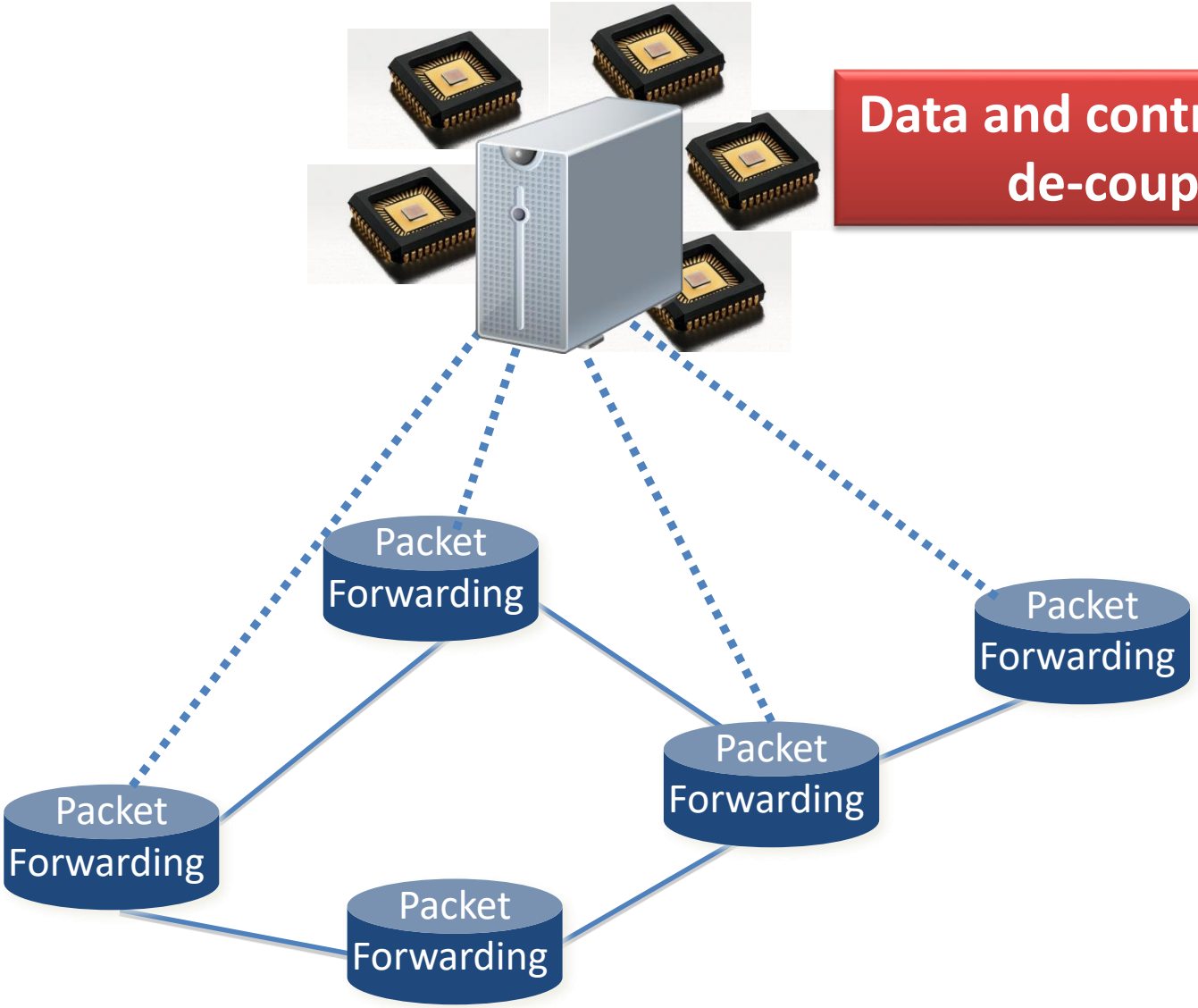


Software control

Data and control planes coupled



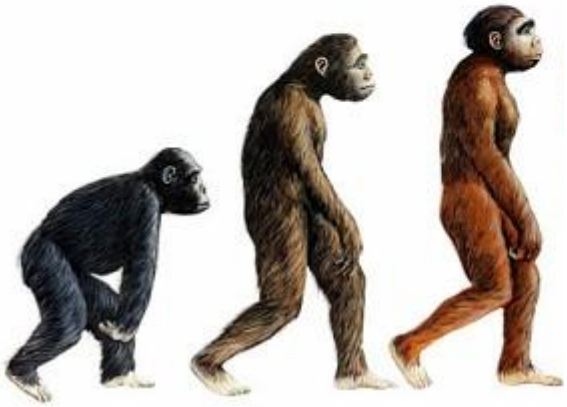
SDN


Data and control planes de-coupled
Packet Forwarding
Packet Forwarding
Packet Forwarding
Packet Forwarding
Packet Forwarding

Key characteristics of SDN

- Control and data planes decoupled
  - The enabler
- Logical centralization of network control
  - Easier to observe/infer and reason about network behavior
  - Network-wide visibility, network-wide and direct control of network traffic
- Flow-based forwarding
  - More flexibility
- Ability to program the network control plane
  - Instead of configuring it (in a tedious, error-prone process)

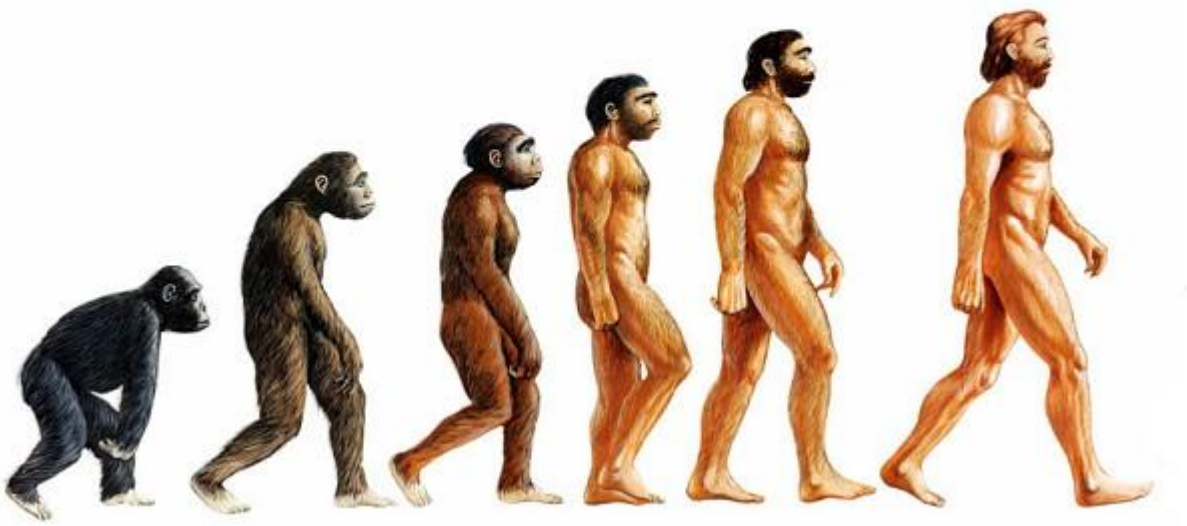
SDN => Change of paradigm



Configurable networks

Programmable networks

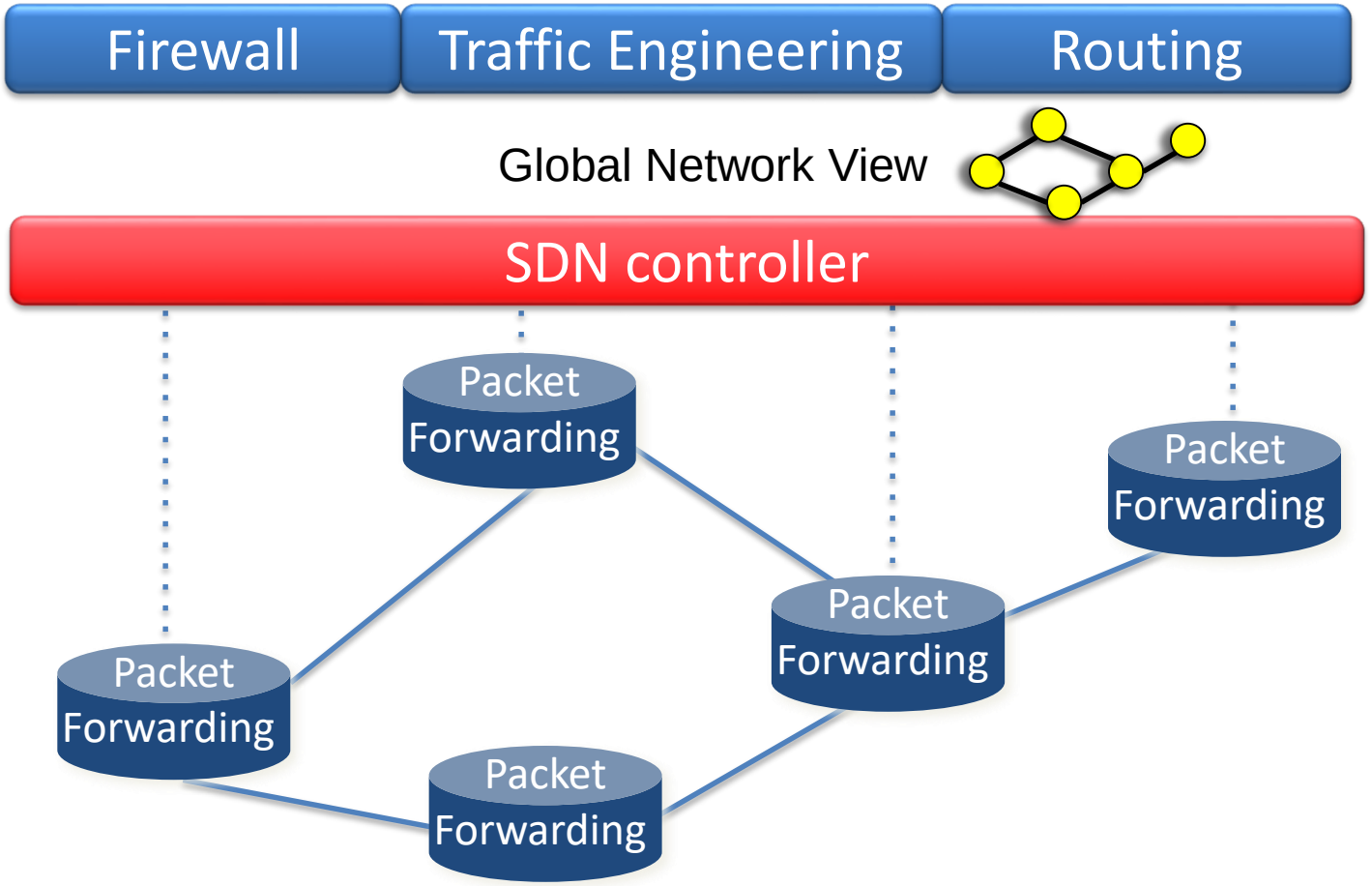
SDN => Change of paradigm



Configurable networks

Programmable networks

Build your net control app!



SDN: an industry change

- Two key aspects made this change possible:
  - Availability of merchant switching chips
    - What is the difference between "merchant silicon" and "custom silicon"?
  - Creation of open interfaces
    - Starting with Openflow

- In what ways can this be comparable to the revolution in computing?
  - Merchant silicon chips ➔ Microprocessor
  - Openflow ➔ x86 instruction set

Analogy with computing industry

Open Interface

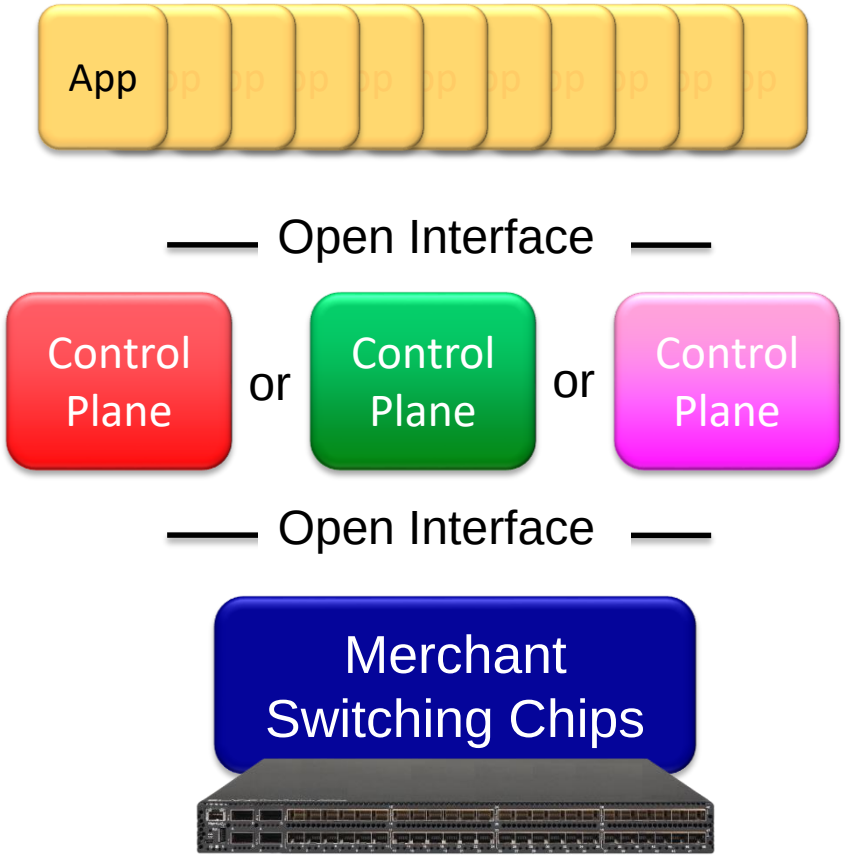
Windows (OS) or Linux or Mac OS

Open Interface

Vertically integrated
Closed, proprietary
Slow innovation
Small industry

Horizontal
Open interfaces
Rapid innovation
Huge industry

Networking industry change



Vertically integrated
Closed, proprietary
Slow innovation

Horizontal
Open interfaces
Rapid innovation

Issue #1: scalability

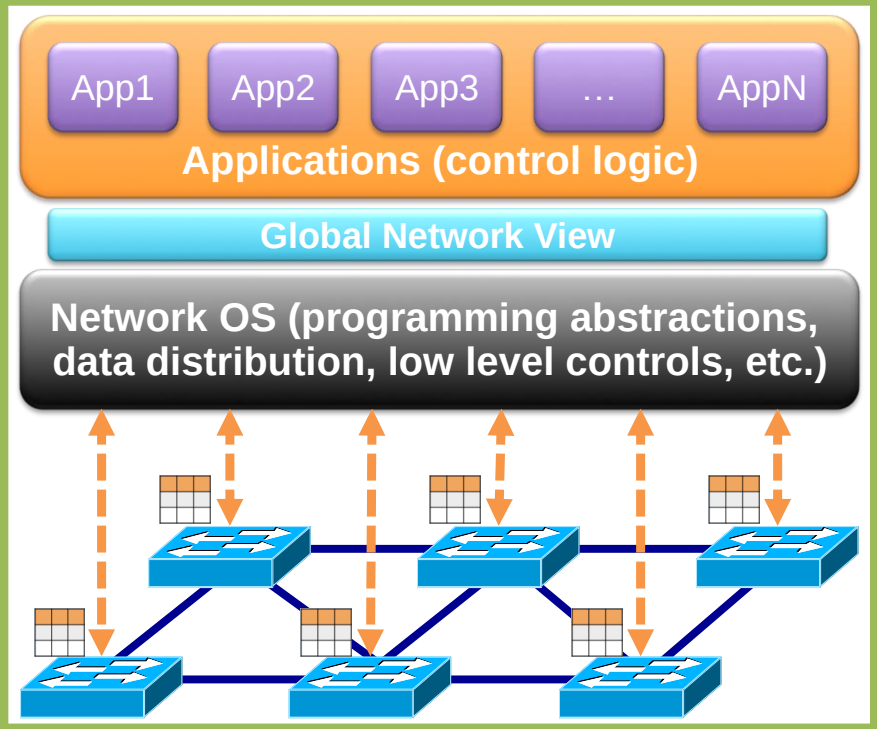
- "It's centralized!"
  - No, it's "logically centralized"

- Discussion
  - Centralized (e.g., NOX, Floodlight) vs distributed control (e.g., ONIX, ONOS)
  - Hierarchical controller designs (e.g., Kandoo)
  - Proactive vs reactive
  - Distribution of work
    - "what should do what?"
    - in a network of controllers, switches (programmable and fixed-function), middleboxes, smartNICs, NetFPGAs, etc.

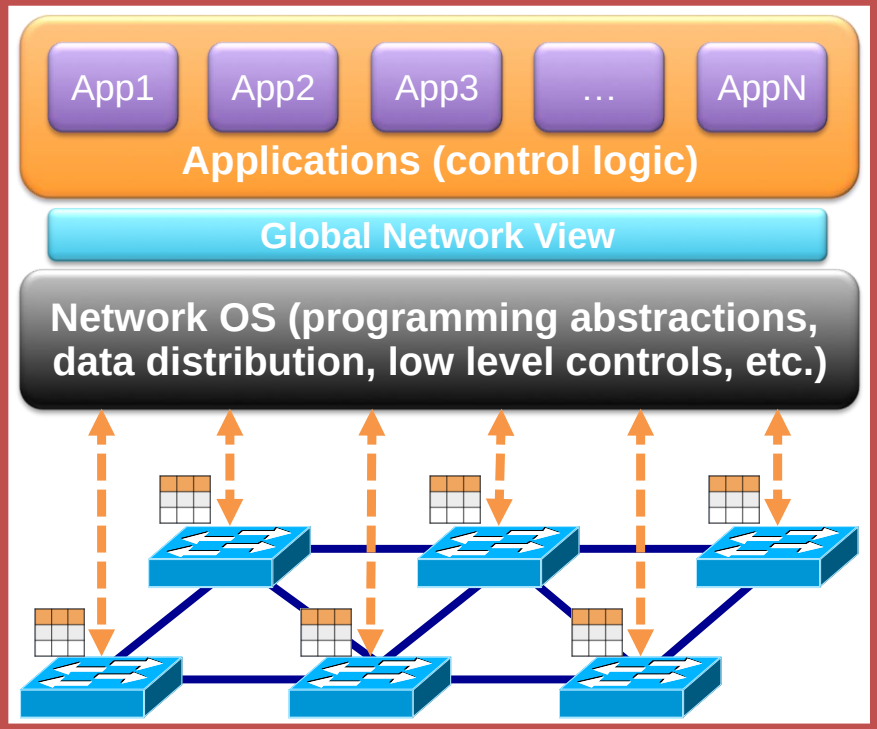
Issue #2: security

- Is SDN secure?

Excellent, now we can program the network!



Wait, now others can program the network!

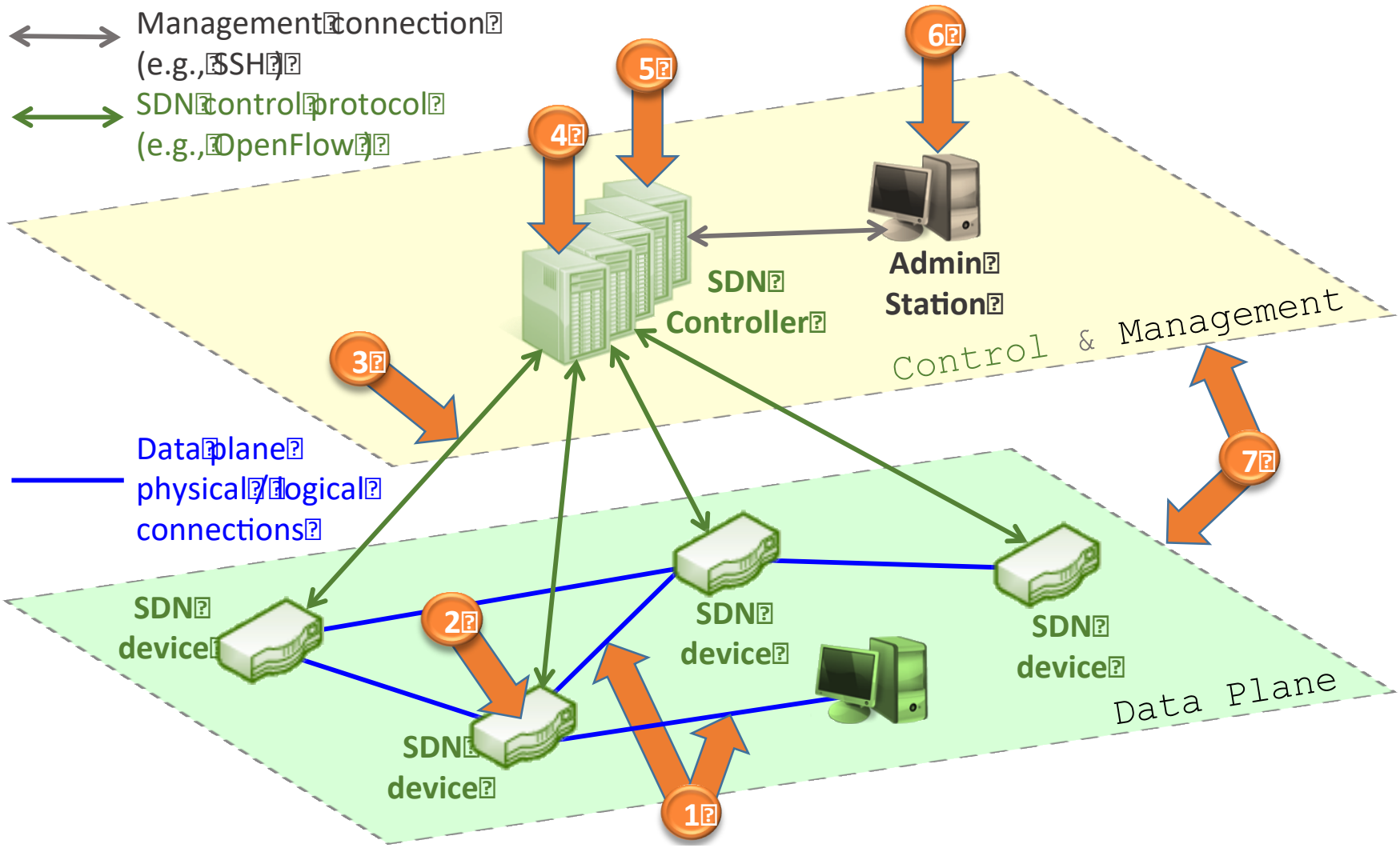


so long, natural protections…

- Traditional networks have "natural protections" against common threats and vulnerabilities…
  - closed (proprietary) nature of network devices
  - heterogeneity of software
  - decentralized nature of the control plane

- …that SDNs in principle do not.

SDN THREAT VECTORS



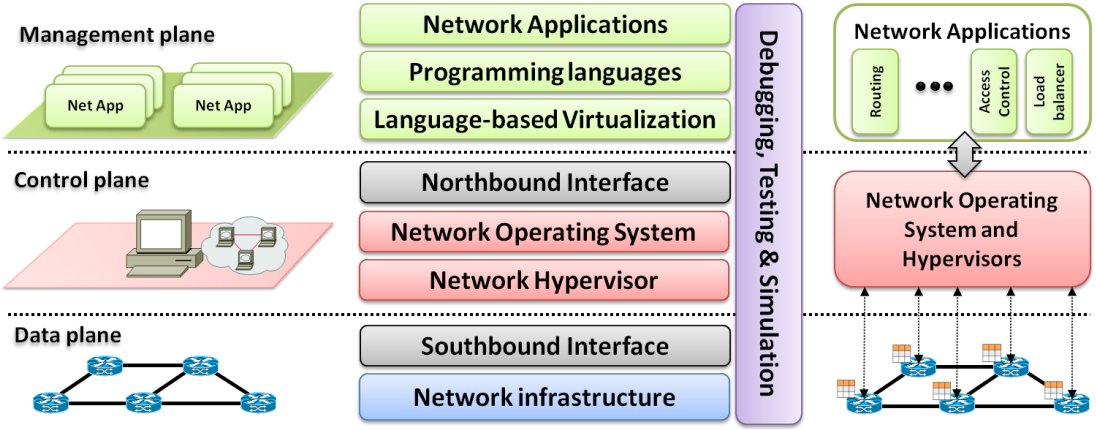
D. Kreutz et al., "Towards secure and dependable software-defined networks", HotSDN'13

Discussion

- Is SDN secure?
  - Not yet, but we are working on it! ☺
- Security
  - D. Kreutz et al., "ANCHOR: logically-centralized security for Software-Defined Networks," Transaction on Privacy and security, 2019
  - D. Kreutz et al., "The KISS principle in Software-Defined Networking: a framework for secure communications," IEEE Security and Privacy, 2018
  - M. Alaluna et al., "Secure and Dependable Multi-Cloud Network Virtualization", XDOM0 workshop, 2017
  - R. Costa and F.M.V. Ramos, "An SDN-based approach to enhance BGP security", NSDI'16 (poster)
  - D. Kreutz et al., "Towards secure and dependable software-defined networks", HotSDN'13
- Dependability
  - F. Botelho et al., "Design and Implementation of a Consistent Datastore for a Distributed SDN Control Plane", EDCC'16
  - F. Botelho et al., "On the design of practical fault-tolerant SDN controllers", EWSDN'14
  - F. Botelho et al., "On the Feasibility of a Consistent and Fault-Tolerant Data Store for SDNs", EWSDN'13

Opportunities for you

- Future networks will rely more and more on software
  - "softwarization" of networking



- According to recent reports (e.g., [IEEE]), these are the skills needed:
  - Incorporate know-how from the IT and network domains
  - Programming skills (mastery of software architecture and open-source software)
  - Expertise in cybersecurity

[IEEE] http://theinstitute.ieee.org/static/special-report-software-defined-networks

Drop me an e-mail if you are interested in these topics.

Lecture plan

An introduction to Software-Defined Networking

[Onix]

The first production-level SDN controller

[B4]

Google's WAN SDN

[SWAN]

Microsoft's WAN SDN

Context and motivation

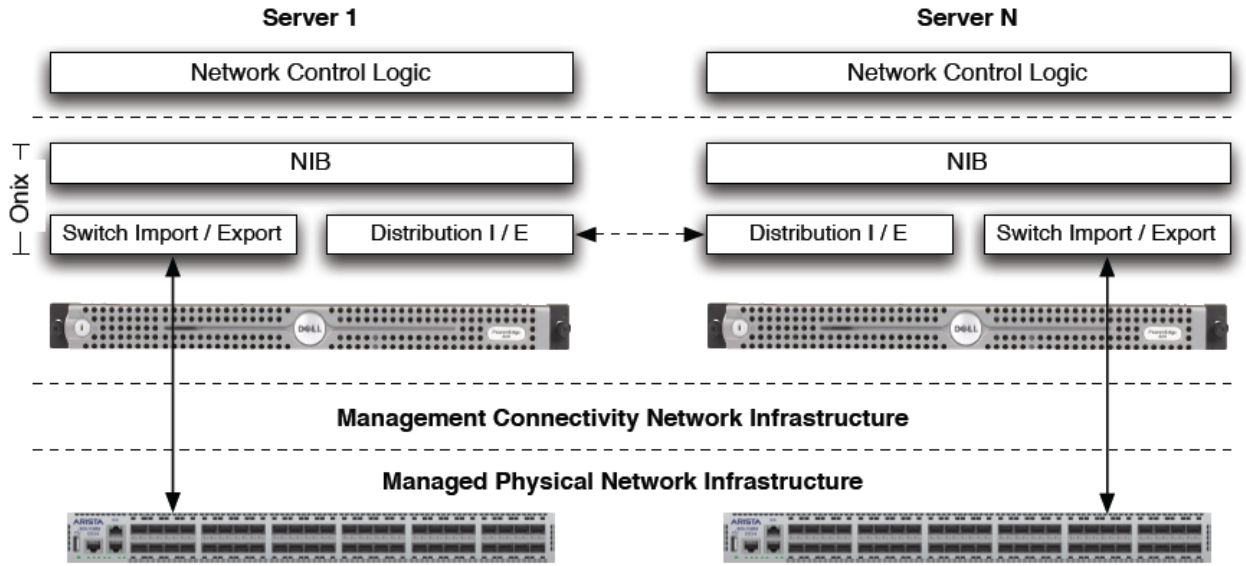
- Network technology has improved dramatically
  - In fact, the data plane…
  - …the control plane has advanced at a much slower pace
- Recently proposed solution: Software-Defined Networking
  - Separation of the control and data planes
- Crucial enabler of the SDN paradigm: the control platform
  - runs on server(s) overseeing a set of simple switches
  - collects info from switches and distributes control state to them
  - provides a programmatic interface to developers of management applications

Proposal

- Onix: a production-quality control platform for SDNs

- Challenges
  - Generality and simplicity
    - A general-purpose API that allows applications to deliver a wide range of functionality in a variety of contexts
    - Does Onix achieve this?
      - Yes
        - » they show several different apps (Ethane, virtualized datacenter, etc.)…
        - » …used in a variety of contexts (WAN, public cloud, enterprise, etc.)
      - The platform is flexible
        - » Developer can make performance/scalability tradeoffs (e.g., relax consistency for higher performance)
  - Scalability and reliability
    - How is this achieved?
      - Distribution and replication
  - Control plane performance

Onix design: components



- **Switches, routers, and middleboxes**
  - must support interface that allows Onix to read and write network element state (e.g., Openflow)
- **Connectivity infrastructure**
  - Communication may be in-band or out-of-band
    - differences?
  - Can use standard routing protocols (OSPF, etc.)
- **Onix**
- **Control logic**
  - determines desired network behavior

Onix design: API

- Main contribution of Onix

- In what consists the Onix API?
  - A data model that represents the network infrastructure
  - Each network element corresponds to one or more data objects
  - The control logic can:
    - Read and alter the current state of an object
    - Register for notifications of state changes

- Where is the network state stored?
  - In the NIB (Network Information Base), a graph of all network entities

- Why is the NIB important:
  - for developers?
    - Network apps are implemented by reading and writing to the NIB
  - for the system?
    - Resilience and scalability are provided by replicating and distributing the NIB, respectively

Scalability

- What strategies does Onix use to improve scaling?

1. Partitioning
   - A particular Onix instance keeps only a subset of the NIB in memory and up-to-date (scales memory)
   - A particular Onix instance has connection only to a subset of the network elements (fewer events; scales processing)
2. Aggregation
   - One instance of Onix can expose a subset of the elements of its NIB as an aggregate element to another instance
     - E.g., expose a single node for a campus building instead of all its network elements
3. Consistency and durability
   - Does Onix dictate the consistency requirements?
     - No, this is left to the control logic. Onix only provides a programmatic framework to assist applications.
     - Is this good or bad?
   - Onix offers (by default) two options
     - Replicated transactional database for state that needs strong consistency (e.g., switch and link inventory, security policies)
     - Memory-based one-hop DHT for state that tolerates inconsistencies (e.g., link utilization levels)

Reliability

- **Network elements and link failures**
  - **Control logic has to steer traffic around the failures**
  - **May use conventional techniques, such as backup paths with fast-failover mechanisms in the network elements**
- **Onix failures**
  - **Onix provides coordination facilities for detecting and reacting to instance failures**
  - **Control logic has to handle lost update race conditions**
- **Connectivity infrastructure failures**
  - **Use dedicated out-of-band channel running traditional protocols (e.g. OSPF) to deal with failures, for instance**

Applications (I)

- **Ethane**
  - Network management app that enforces network security policies
    - Flows need to be approved before entering the network
  - What is the main scalability concern?
    - Flow processing
  - What Onix techniques can be used to scale?
    - Partitioning of flow processing

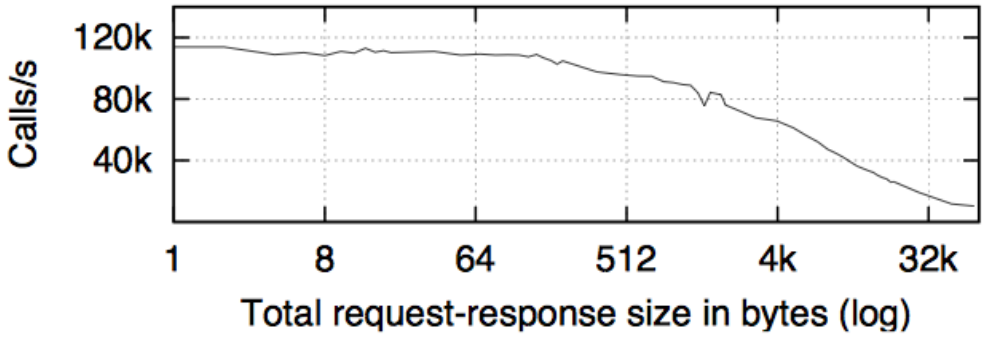
- **Distributed Virtual Switch**
  - Logical switch abstraction over which policies (QoS, ACLs) are declared over the logical switch ports (bound to virtual machines)
    - Note that in virtualized environments (e.g., clouds) the network edge is a virtual L2 switch within hypervisors
  - Assists in VM migration, with network policies following the VMs

Applications (II)

- Multi-tenant virtualized data center
  - Handle end-host dynamics and enforce addressing and resource isolation between tenant networks (network virtualization)
    - Say, different tenants may use the same MAC or IP addresses
  - How does the control logic isolates the different L2 networks?
    - By encapsulating tenant's packets at the edge, and decapsulating them when they enter another hypervisor or are released to the Internet
    - The control logic establishes tunnels pair-wise between all the hypervisors running VMs attached to the tenant virtual network
  - What is the main scalability concern?
    - Pair-wise tunnels needed between all hypervisors: $O(N^2)$ tunnels.
  - What Onix techniques can be used to scale?
    - Partitioning: single Onix instance deals only with a subset of hypervisors
- In a few weeks we will analyze in detail a paper describing this network virtualization solution…

Evaluation

- Let's focus our attention on the multi-node performance
- Memory-based DHT is limited by the Onix RPC stack



  - With 5 Onix instances, 24k small DHT value updates per second
- The transactional database "comes with severe performance limitations"

Queries/trans	1	10	20	50	100
Queries/s	49.7	331.9	520.1	541.7	494.4

Reducing the cost of consistency

- By noting the "severe performance limitations" of Onix transactional database, we proposed a strongly consistent, fault-tolerant SDN control framework that achieves acceptable performance
  - The central element of our architecture is a highly-available, strongly consistent data store (built using a state-of-the-art state machine replication library, BFT-SMaRt)

	Workload Data				Data store Performance	
Application	Workload	OF Requests	Reads (size)	Writes (size)	Thr. (kReq/s)	Latency (ms)
Learning Switch	**w1** - Mapping host-port	ARP Req. ARP Reply ICMP Echo Req.	0 1 (77) 1 (77)	1 (113) 1 (113) 1 (113)	22.7 ± 3.3	9 ± 3
Load Balancer	**w2** - Balancing a request	ARP Req. ICMP Echo Req. ICMP Echo Reply	2 (509) 3 (366) 1 (106)	0 1 (395) 0	19.3 ± 3	12 ± 7
Device Manager	**w3** - Known devices	ICMP Echo Req. ICMP Echo Reply	2 (1680) 2 (1680)	1 (3458) 1 (3458)	4.7 ± 0.5	41 ± 6
	w4 - Unknown devices	ARP Req. ARP Reply ICMP Echo Req.	2 (0) 3 (560) 2 (1680)	4 (1092) 4 (1092) 1 (3458)	3.6 ± 0.3	52 ± 18

- For more details, check:
  - F. Botelho, F. M. V. Ramos, D. Kreutz, A. Bessani, "On the feasibility of a consistent and fault-tolerant data store for SDNs", EWSDN 2013

What you said

This API generality makes the operation more effective in a wider range of different contexts and allows users to make their own trade-offs regarding consistency, durability, and scalability

Daniel, Sérgio, Francisco

Na distribuição do estado pelo cluster creio estar o ponto mais "fraco" de todo o sistema. Os autores tomam uma perspetiva optimista (conflictos irão ocorrer raramente) e empurram para a aplicação a resolução de possíveis conflitos resultantes de race conditions

Diogo

Lecture plan

An introduction to Software-Defined Networking

[Onix]

The first production-level SDN controller

[B4]

Google's WAN SDN

[SWAN]

Microsoft's WAN SDN

Context

- Modern Wide Area Networks are critical to Internet performance and reliability
  - But individual WAN links are expensive and WAN routers are specialized equipment that place a premium on high availability
  - Plus, WANs typically treat all bits the same
    - When a failure occurs all applications are treated equally
- For these reasons, WAN links are typically provisioned to 30-40% utilization
  - Overprovisioning works, but requires 2-3x more bandwidth and high-end routing gear
- Google operates 2 distinct WANS
  - A user-facing network peering with other Internet domains
  - B4, a WAN that connects Google's data centers
    - This is the network that carries more traffic and has the higher traffic growth rate

Motivation

- Overprovisioning for the substantial bandwidth requirements of B4 is becoming prohibitively expensive

- So Google explored an SDN alternative to B4, based on the unique characteristics of their data center WAN:
  - Massive bandwidth requirements deployed to a modest number of sites
  - Elastic traffic demand that seeks to maximize average bandwidth
    - i.e. apps can adapt
  - Full control over edge servers and network, enabling rate limiting and demand measurement at the edge

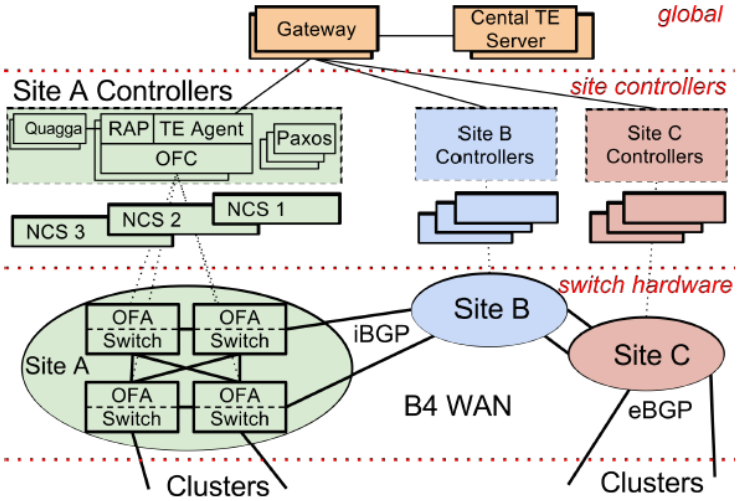
- Why SDN?
  - Software-based control plane runs that runs in commodity servers
    - These are much faster (and cheaper) than the embedded processors in most switches
  - Decoupling of hardware and software
    - Control plane software becomes simpler and evolves quickly
    - Data plane hardware performance continues to evolve independently
  - Opportunity to reason about global state
    - Simplifies management and capacity planning (traffic engineering)

Contribution

- The authors present their experience in deploying Google's WAN – B4 – using SDN principles and Openflow
  - Among the first and largest SDN/Openflow deployments

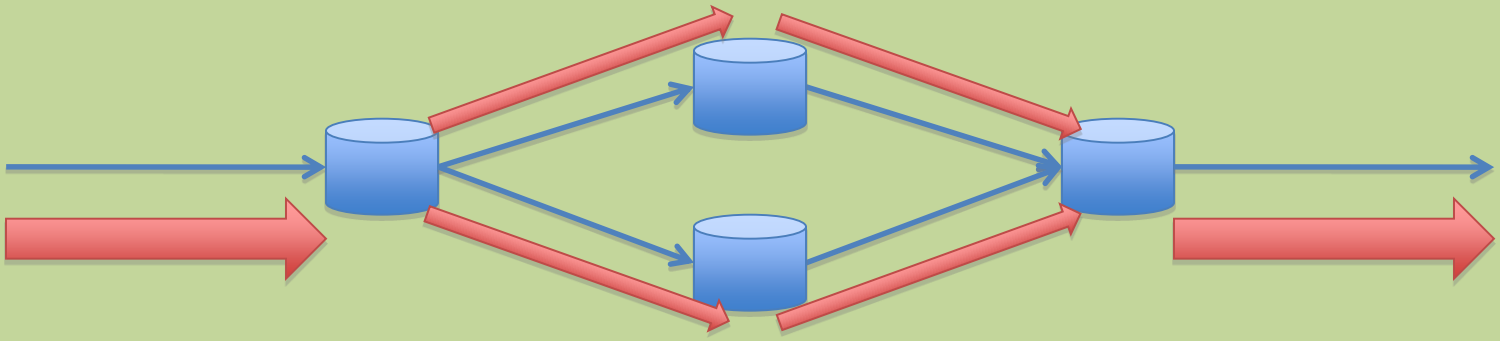
- When the paper was published, B4 had been in deployment for 3 years and carried more traffic than their public facing WAN

Design and architecture



- **Routing**
  - They support, simultaneously, standard routing protocols and centralized Traffic Engineering (TE)
  - Open-source routing source software Quagga running BGP/ISIS, with Routing Application Proxy (RAP) "translating" RIB entries into Onix NIB entries
  - TE as a routing overlay
    - This design choice was crucial, giving Google a "red button" to switch from TE to shortest path routing in case of problems
- **Dependability**
  - Replication of (modified) Onix controllers for fault-tolerance, with Paxos electing primary
  - Replication of central TE services, again using Paxos
- **Scalability**
  - Topology abstraction introduced
  - Each site is represented as a single node with a single edge to other sites
    - ECMP hashing is essential to balance load across constituent links

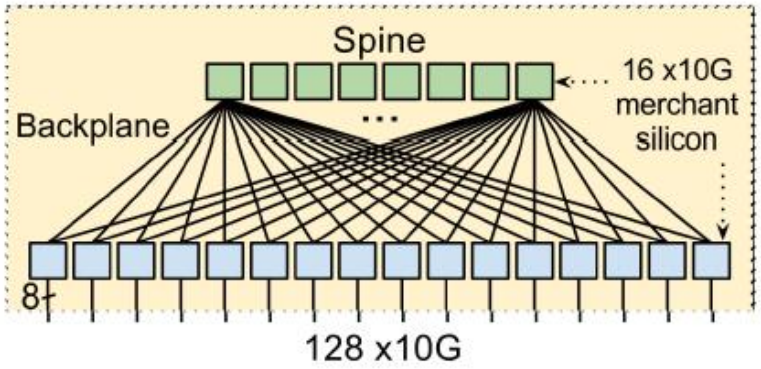
ECMP



- Multipath routing strategy that splits traffic over multiple paths for load balancing
- Can we just round-robin packets? Not really.
  - Reordering (may lead to triple duplicate ACKs in TCP)
  - Different RTTs per path (for TCP RTO)
  - Different MTUs per path
- Usually: path-selection via hashing
  - # buckets = # outgoing links
  - Hash network information (source/dest. IP address) to select outgoing link preserves flow affinity

Switch design

- They built their own Openflow switches
  - Opted for fewer but larger switches
  - 128-port 10GE built from 24 cheap, commodity, 16-port 10GE switches



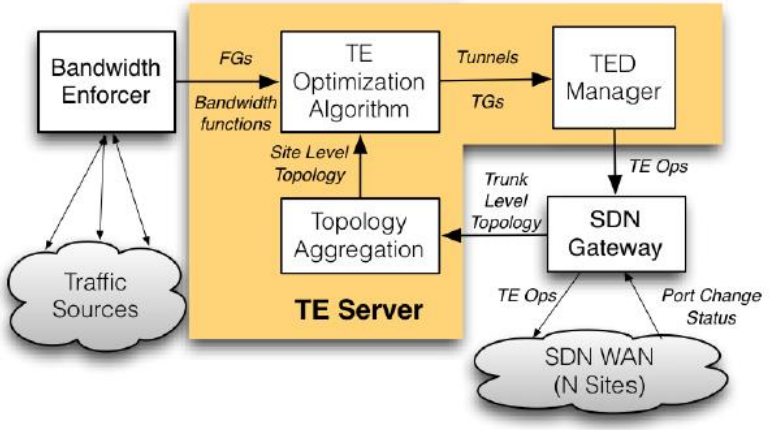
- Why commodity cheap switches, not high-end expensive ones?
  - No need for deep buffers
    - Careful endpoint management (e.g., rate limiting) enabled by SDN is enough to guarantee small drop rates
  - Large forwarding tables not needed
    - They have a small set of data centers
  - Hardware support for high availability not needed
    - Most faults are software issues, and as software functionality is moved from the switch, typical distributed systems techniques for fault-tolerance can be used

Traffic Engineering: main ideas

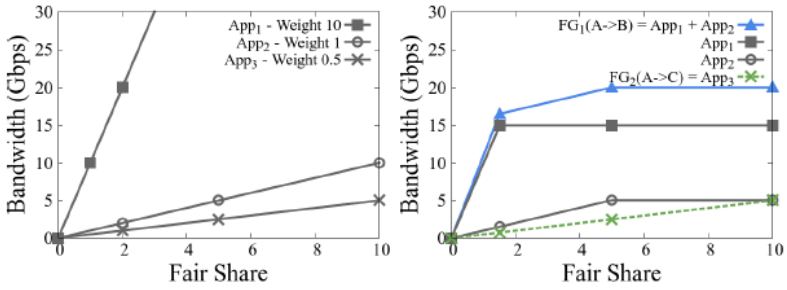
- Leverage control at network edge to adjudicate among competing demands during resource constraint
- Use multipath forwarding/tunneling to leverage available capacity according to application priority
- Dynamically reallocate bandwidth in the face of link/switch failures or shifting application demands

Centralized TE architecture

- Topology aggregation
  - Trunks are aggregated to compute site-to-site edges
  - Scalability: reduces input size simplifying TE optimization



- Bandwidth enforcer (BE)
  - Scalability: applications are aggregated in flow groups (FGs)
  - To capture relative priority, one bandwidth function is associated with every application
  - The FGs bandwidth function is the linear additive composition of per-app functions
  - BE enforces bandwidth limits



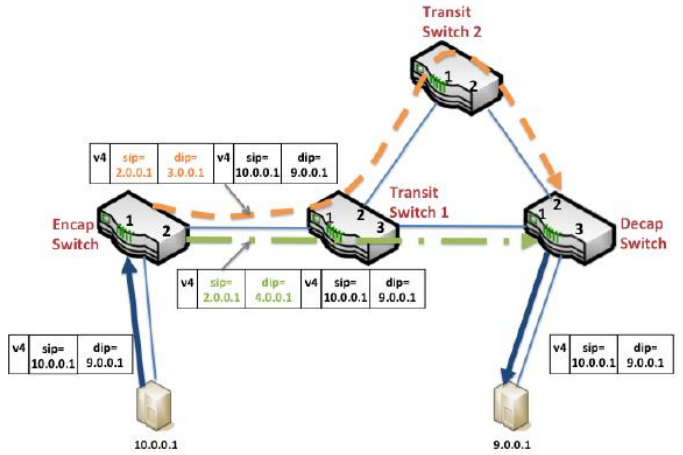
(a) Per-application. (b) FG-level composition.

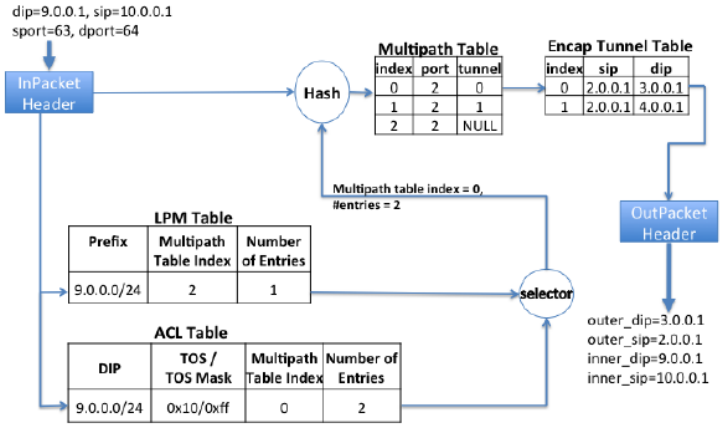
TE Optimization Algorithm

- The Linear Programming optimal solution for allocating a fair share among Flow Groups is expensive
  - (Linear programming: mathematical formulation for the problem of maximizing or minimizing a linear function subject to linear constraints)
  - They propose a heuristic that achieves similar fairness, 99% bandwidth utilization, and is 25x faster
  - Two components: Tunnel Group Generation and Tunnel Group Quantization
- Tunnel Group Generation
  - Allocates bandwidth to FGs based on demand and priority (bandwidth functions)
    - It starts filling flow groups on their favourite tunnels (according to their fair shares) until the tunnel becomes a bottleneck
    - Next moves to next favourite tunnels until each FG is satisfied
- Tunnel Group Quantization
  - Adjusts the splits to the granularity supported by the hardware
    - e.g. if the split between 3 tunnels is 0.5:0.4:0.1 and we can only use multiples of 0.5 it may become 0.5:0.5:0.0

TE protocol and OpenFlow

- How are tunnel groups and flow groups mapped to OpenFlow switches?
- B4 switches operate in three roles
  - Encapsulating switch initiates tunnels and splits traffic between them
    - hashing the packet header
  - Transit switches forward packets based on the outer header
  - Decapsulating switch terminates tunnels and then forwards packets using regular routes
- Composing routing and TE
  - They leverage on multiple forwarding tables
    - Routing/BGP populates LPM (lower priority)
    - TE uses ACL table (higher priority)





Selected result #1

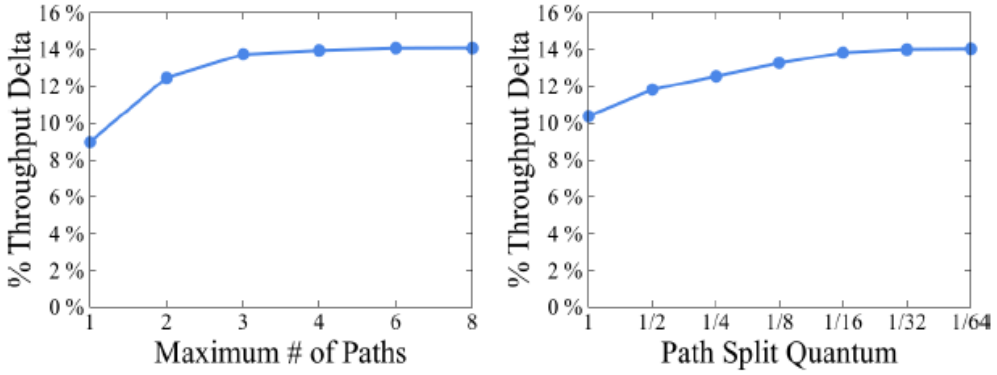
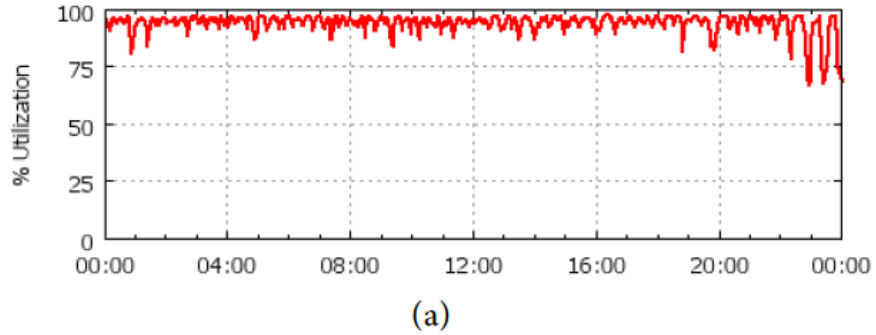


Figure 13: TE global throughput improvement relative to shortest-path routing.

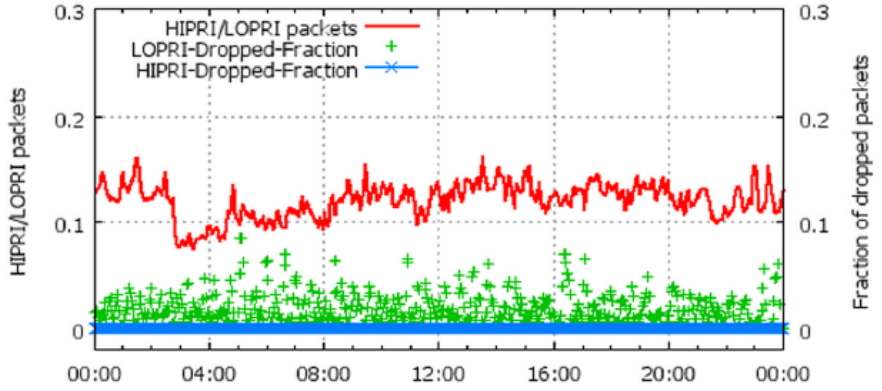
- **14% average throughput increase over shortest path routing**
  - **But the impact (and the benefit) is more pronounced during periods of failure or high demand**

Selected result #2

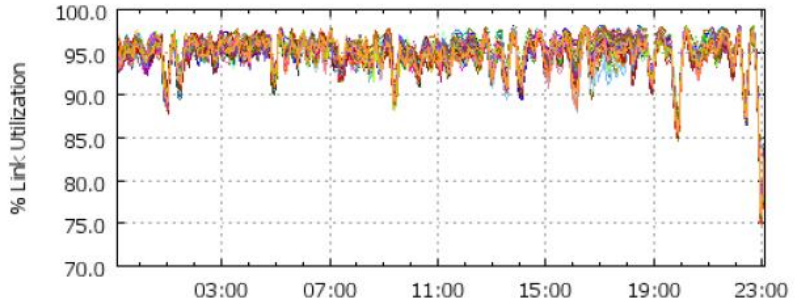
- Many B4 links run at near 100% utilization without affecting high priority traffic
  - And all links average 70% over long time periods
  - This is a 2-3x efficiency improvement relative to standard practice
- ECMP hashing is key, and is very effective



(a)



(b)



What you said

An important remark of this paper is how the authors dealt with an outage…We can see here that even though a lot of care was put into separating stacks, software and hardware, human error can still put the network in very tight constraints and even cause downtime.

Guilherme, Francisco

Lecture plan

An introduction to Software-Defined Networking

[Onix]

The first production-level SDN controller

[B4]

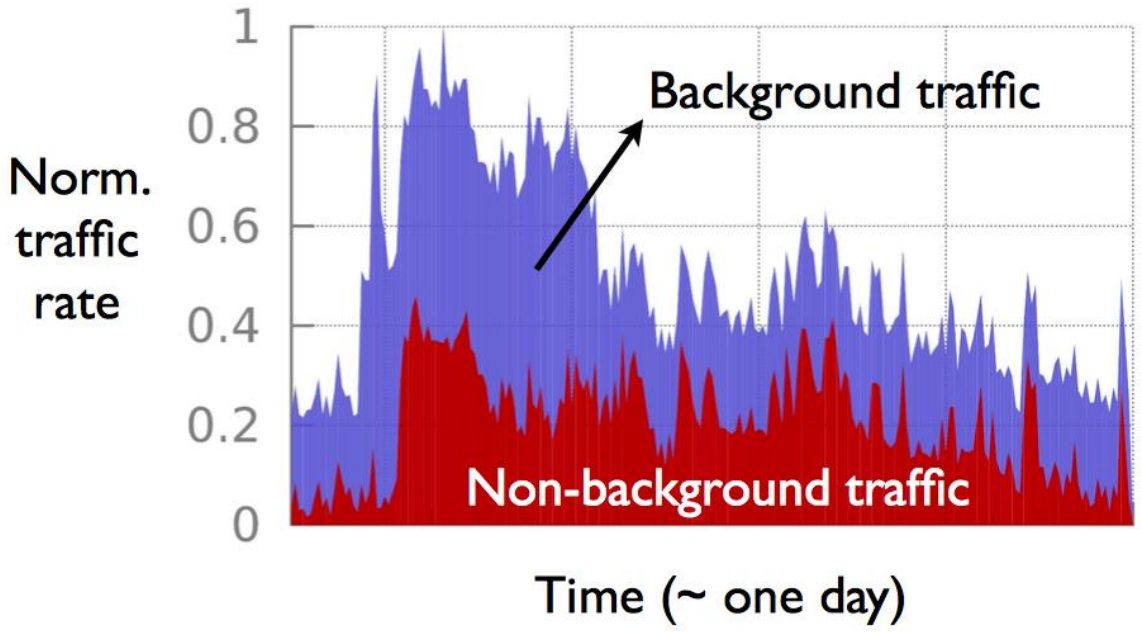
Google's WAN SDN

[SWAN]

Microsoft's WAN SDN

Context and motivation

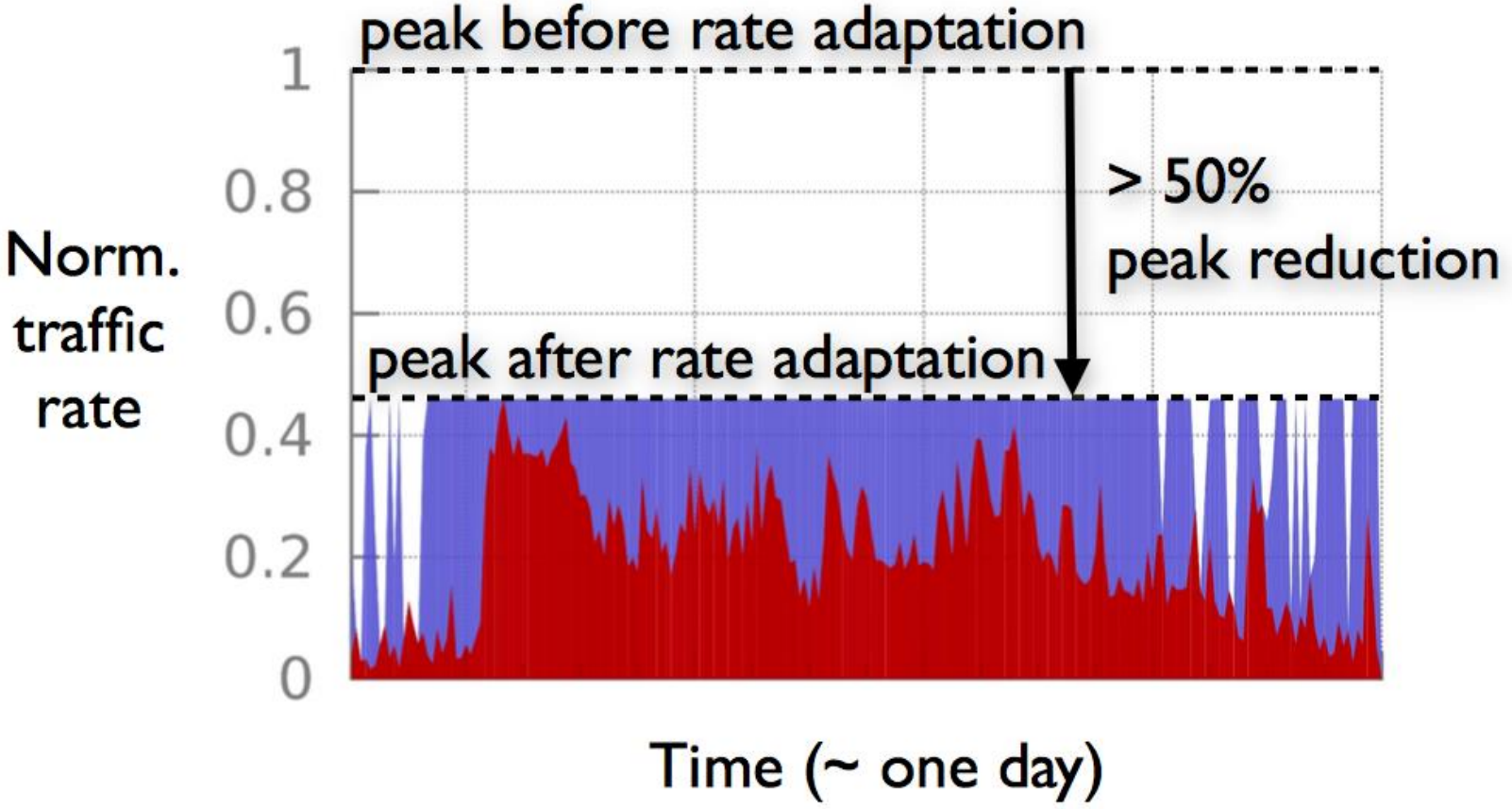
- **Basically the same as B4**
  - **Poor efficiency of highly expensive links**
    - **average utilization over time of busy links is only 30-50%**
  - **Poor sharing**
    - **Traditional techniques do not achieve global optimal (e.g. MPLS TE greedily selects shortest path fulfilling capacity constraints)**



Proposed solution

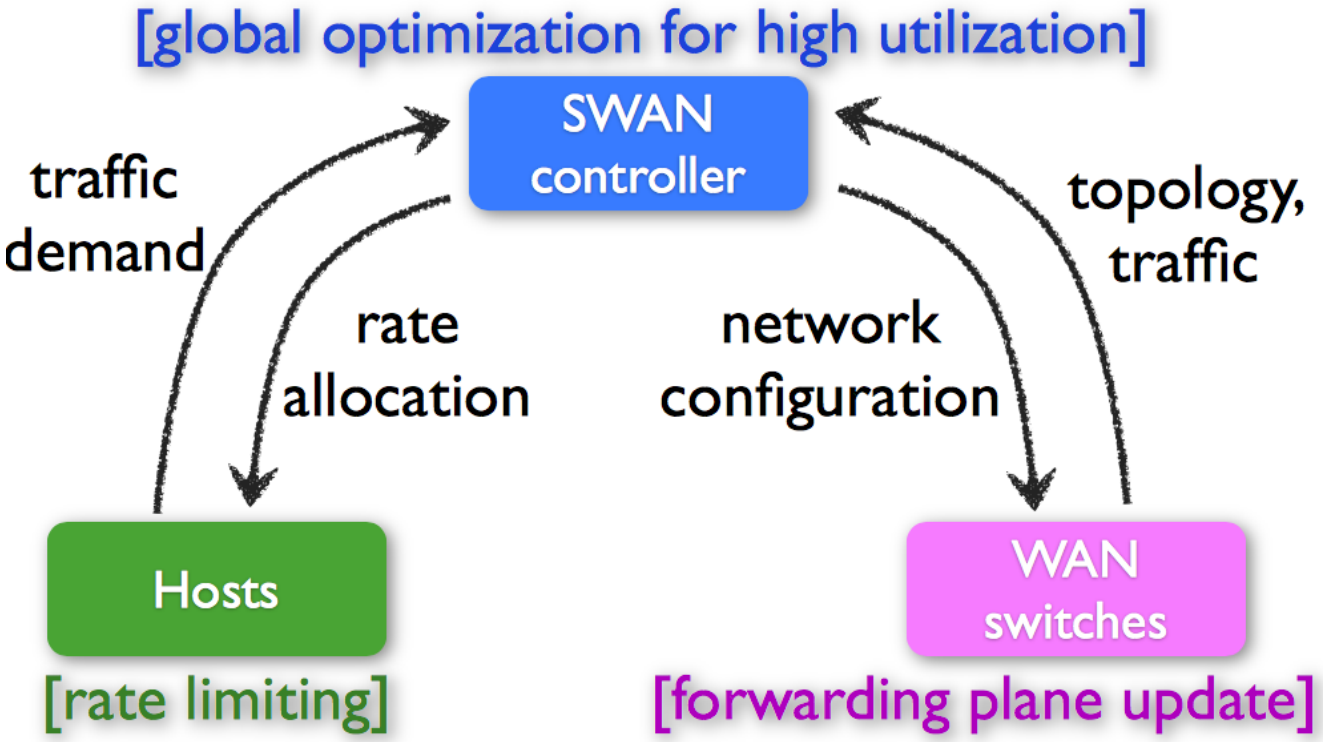
- **Also similar to B4**

- **General goal:**



Proposed solution

- Also similar to B4
  - Logically-centralised control, global coordination
    - SDN-based approach
  - Rate limiting at the edge
    - They control all servers



Challenges

- Scalable allocation computation
- Congestion-free data plane update
  - Not considered in B4
- Working with limited switch memory
  - Not considered in B4

Challenges

- **Scalable allocation computation**
  - How to compute allocation in a time-efficient manner?
- **Main ideas**
  - Path-constrained, multi-commodity flow problem
    - allocate higher-priority traffic first
    - ensure weighted max-min fairness within a class
  - Solving at the granularity of {DC pairs, priority class}-tuple
    - split the allocation fairly among service flows
- **Problem: computing max-min fairness is hard**
  - They propose an approximation

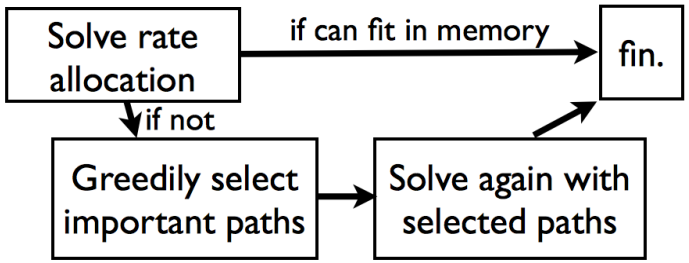
Challenges

- **Congestion-free data plane update**
  - **How to update forwarding plane without causing transient congestion?**

- **Main ideas**
  - **Leave a small amount of scratch capacity on each link**
    - **They demonstrate that with a certain scratch capacity there exists a congestion-free update in a fixed number of steps**
  - **They proposed an LP-based solution to find it**

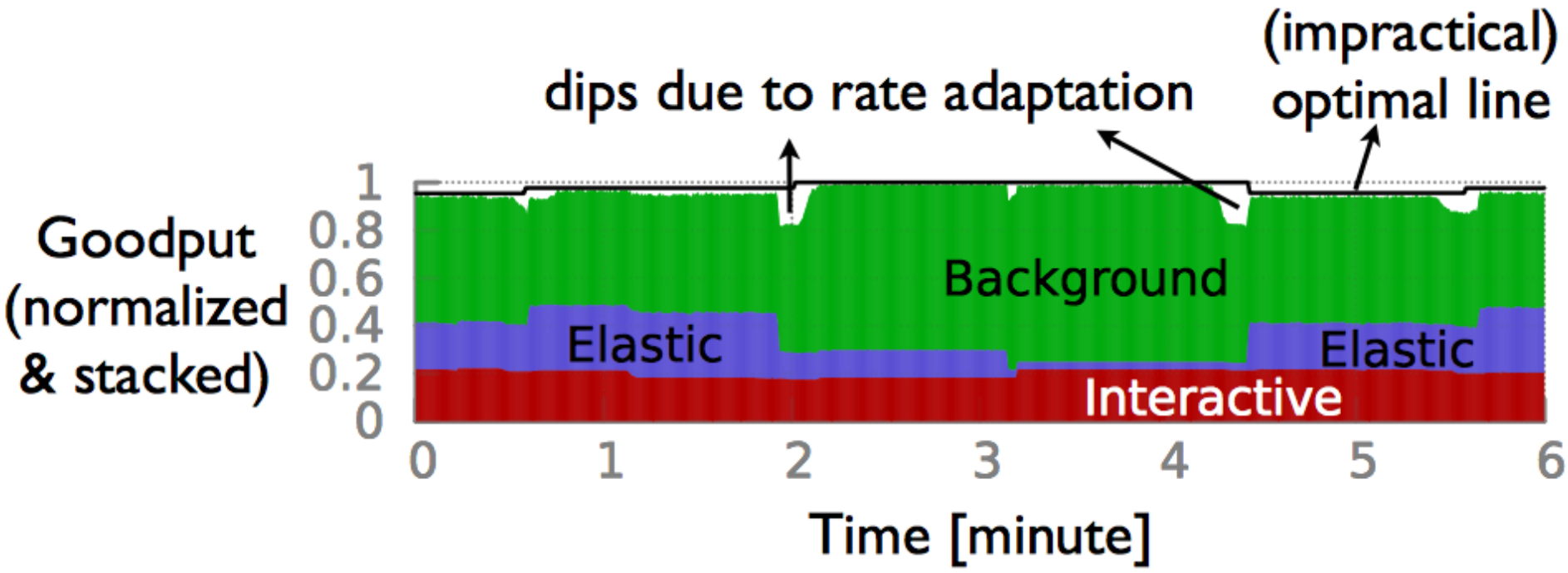
Challenges

- Working with limited switch memory
- Why does switch memory matter?
  - Back of the envelope calculation
    - 50 sites = 2,500 pairs
    - 3 priority classes
    - static k-shortest path routing
    - More than 20k rules needed to fully use network capacity
  - Commodity switches (at the time and similar today)
    - 1-4k rules
    - Next generation: 16k rules
- Solution
  - They propose a heuristic for dynamic path set adaptation
    - Main intuition: working path set << total needed paths



    - 10x fewer rules than static k-shortest path routing
  - Rule update
    - multi-stage rule update
    - with 10% memory slack, typically 2 stages needed

Results

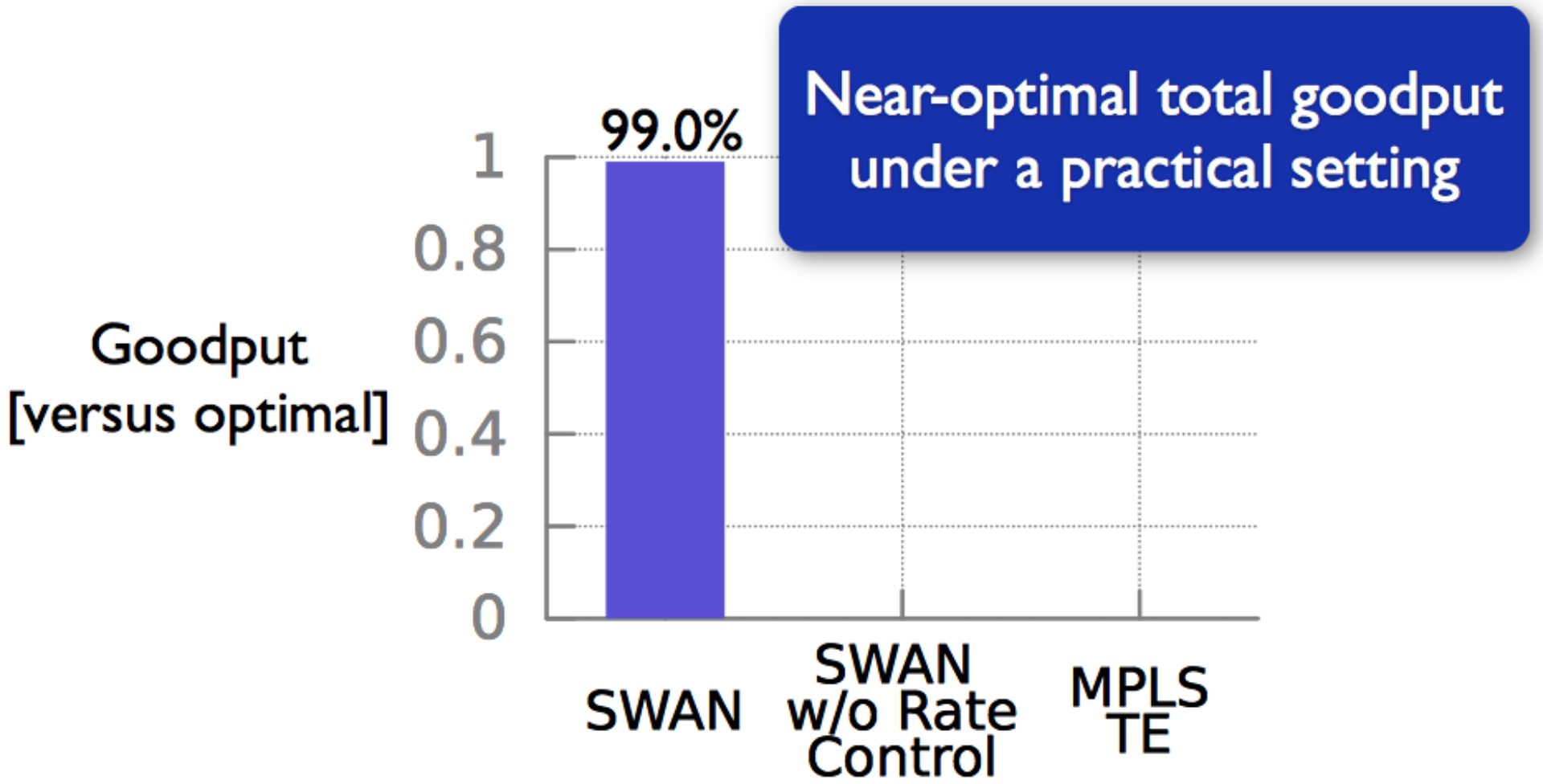


Traffic: (∀DC-pair) 125 TCP flows per class

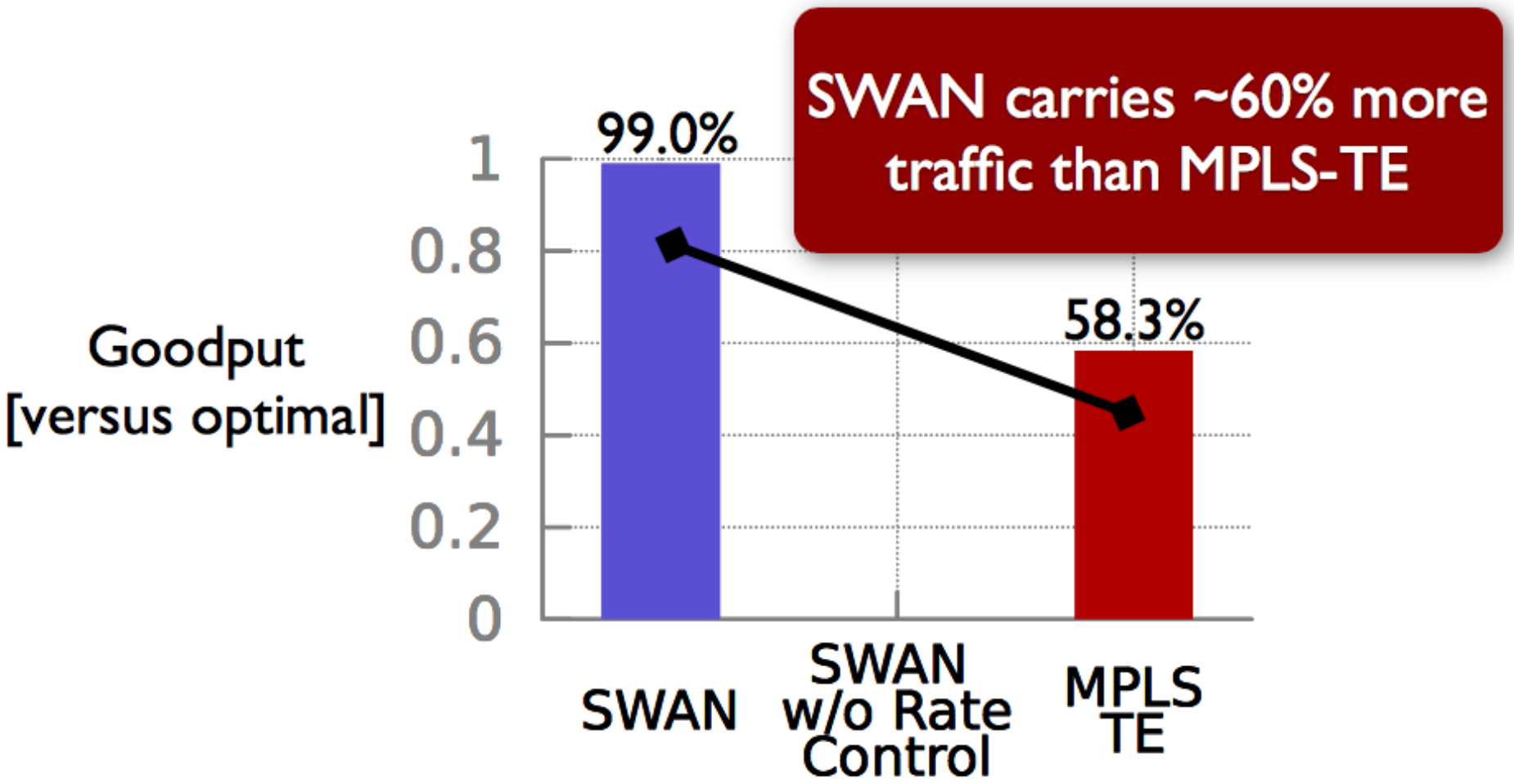
High utilization
SWAN's goodput:
98% of an optimal method

Flexible sharing
Interactive protected;
background rate-adapted

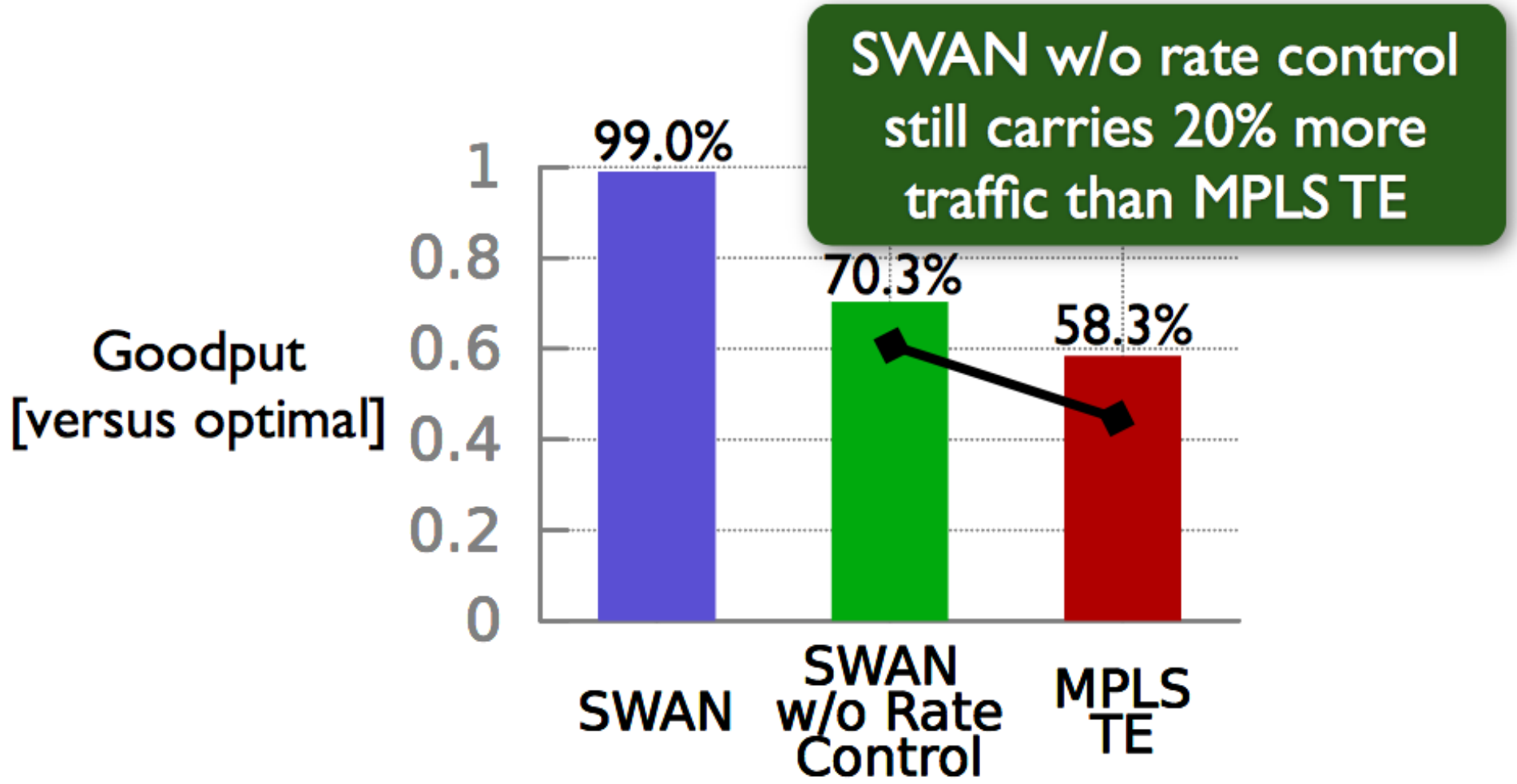
Results


Near-optimal total goodput under a practical setting
99.0%
Goodput [versus optimal]
1
0.8
0.6
0.4
0.2
0
SWAN
SWAN w/o Rate Control
MPLS TE

Results


SWAN carries ~60% more traffic than MPLS-TE
99.0%
58.3%
Goodput [versus optimal]
1
0.8
0.6
0.4
0.2
0
SWAN
SWAN w/o Rate Control
MPLS TE

Results



- This is interesting, as centralised control alone is already helpful
  - So could also be used by ISPs for instance

Lecture plan

Next lecture: Congestion control

Problem

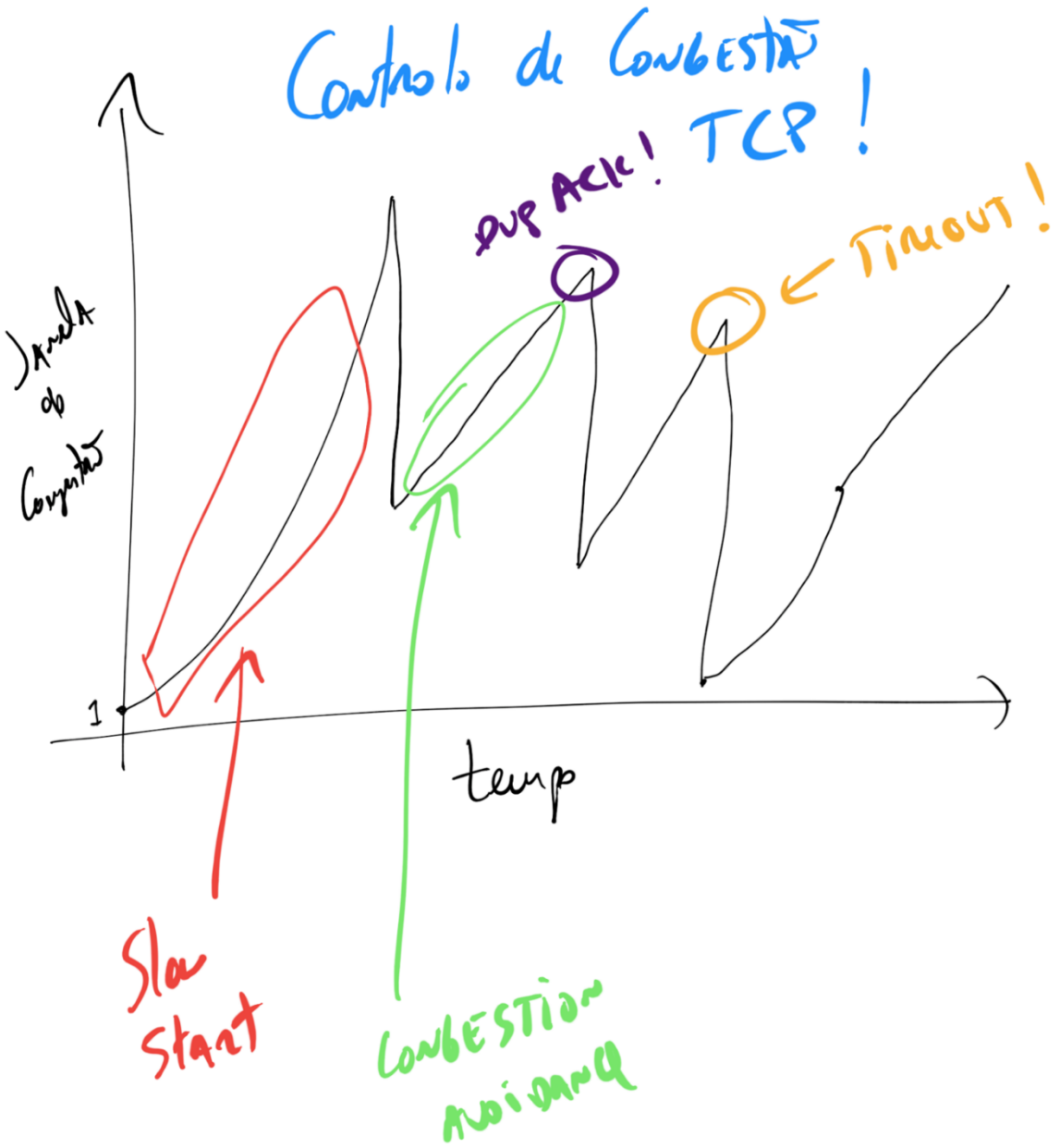
- How to effectively and fairly allocate resources among a collection of competing users?

Congestion collapse

- Just around 8 years after the TCP/IP protocol stack had become operational, the Internet was suffering from congestion collapse
  - hosts would send their packets into the Internet as fast as the advertised window would allow, congestion would occur at some router (causing packets to be dropped), and the hosts would time out and retransmit their packets, resulting in even more congestion

- TCP congestion control was introduced into the Internet in the late 80s by Van Jacobson

TCP congestion control



Next lecture: congestion control

- **Mandatory (1 out of 3)**
  - A. Langley et al., The QUIC Transport Protocol: Design and Internet-Scale Deployment, SIGCOMM 2017

  Google's encrypted, multiplexed, and low-latency transport protocol
  - K. Wonstein et al., Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks, NSDI 2013

  End-to-end transport protocol for interactive applications that desire high throughput and low delay, targeting cellular networks
  - M. Alizadeh et al., Data center tcp (dctcp), SIGCOMM 2010

  A version of TCP targeting data centers

- **[Optional]**
  - V. Jacobson and M. J. Karels, "Congestion Avoidance and Control", SIGCOMM, 1988

  The congestion control problem is introduced, and the original solution is proposed
  - A. Afanasyev et al., "Host-to-Host Congestion Control for TCP", IEEE Communications Surveys & Tutorials, 2010

  A comprehensive survey on TCP congestion control

References

- [ONIX]
  - Teemu Koponen et al., "Onix: A Distributed Control Platform for Large-scale Production Networks," OSDI 2010
- [B4]
  - Sushant Jain et al., "B4: Experience with a Globally-Deployed Software Defined WAN," SIGCOMM 2013
- [SWAN]
  - C.-Y. Hong et al., "Achieving High Utilization with Software-Driven WAN," SIGCOMM 2013