CS6204
Advanced Topics in Networking

Assoc Prof. Chan Mun Choon

School of Computing

National University of Singapore

Aug 14, 2015

Lecturer

Chan Mun Choon

Office: COM2, #04-17

Email: chanmc@comp.nus.edu.sg

URL: http://www.comp.nus.edu.sg/~chanmc

Course Description

This graduate **(PhD)** level course covers a broad range of the latest developments in computer networking and telecommunications in terms of new techniques and technologies, trends, open research issues and some related new principles and approaches in networking.

Course Pre/co-requisites

CS5229 Advanced Computer Networks

Or

Permission from Lecturer

Topics Covered

Software Defined Network (SDN) – 6 weeks

- Topology design, resource allocation, policy verification/checking, transport protocols (TCP)

Wireless Communication – 6 weeks

- Backscatter
- Cellular network scheduling and power saving
- Mobile Sensing
- Wireless Sensor Network Protocol
- Delay/Disruption Tolerant Network Protocols

Evaluation

Weekly paper reviews (week 3 onwards): 35%

1 Paper presentation (week 3 onwards): 25%

Term paper

- Abstract: 5%
- Full paper: 35%

- No final exam

Term Paper

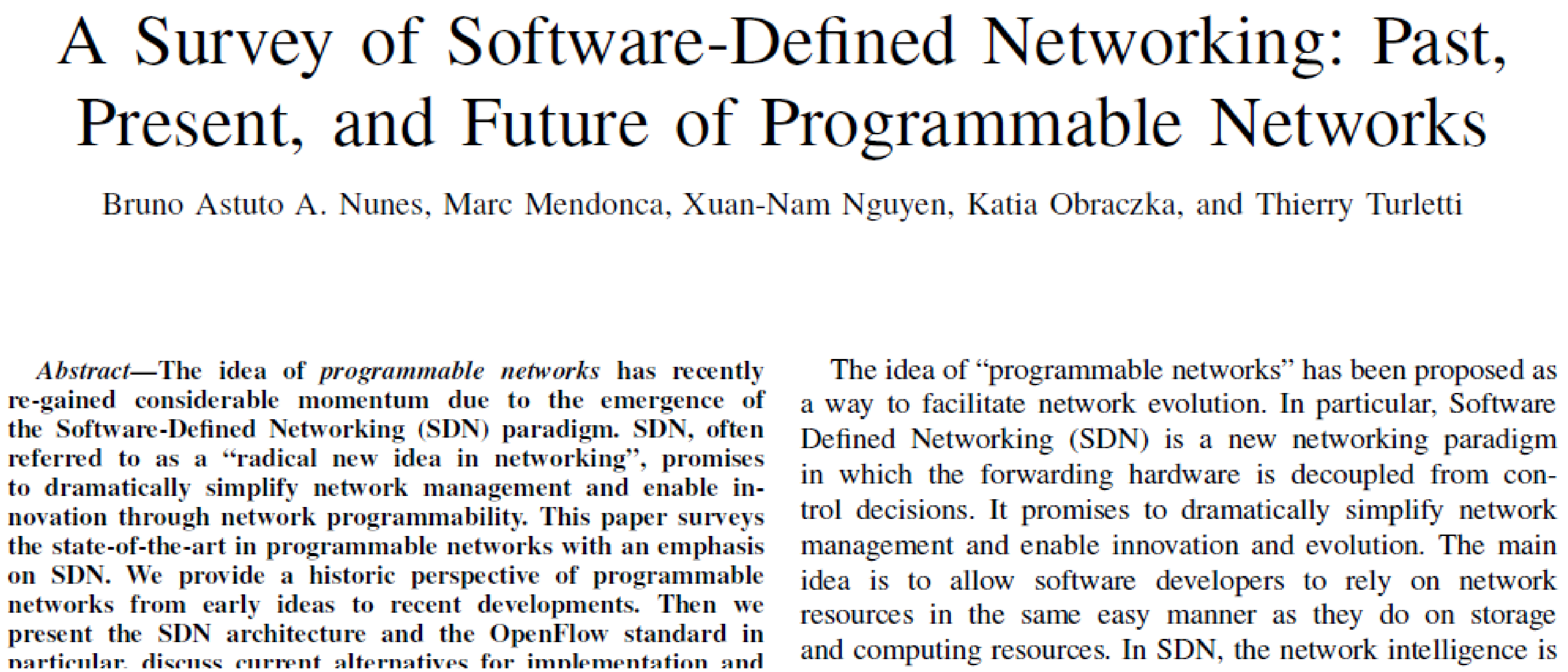
- Individual work
- Student pick his/her topic
- Two categories
  - Survey paper
  - Solve a "small" problem (preferred)

- Abstract: 1 – 2 pages (due week 9)
- Full paper: 4-8 pages (due week 13)
  - *ACM* Proceedings *Format, two column, 10 point font*

General Information

IVLE: CS6204

- General information
- Reading list, lecture notes
- Discussion forum



A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks

Bruno Astuto A. Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti

***Abstract*—The idea of *programmable networks* has recently re-gained considerable momentum due to the emergence of the Software-Defined Networking (SDN) paradigm. SDN, often referred to as a "radical new idea in networking", promises to dramatically simplify network management and enable innovation through network programmability. This paper surveys the state-of-the-art in programmable networks with an emphasis on SDN. We provide a historic perspective of programmable networks from early ideas to recent developments. Then we present the SDN architecture and the OpenFlow standard in particular, discuss current alternatives for implementation and**

The idea of "programmable networks" has been proposed as a way to facilitate network evolution. In particular, Software Defined Networking (SDN) is a new networking paradigm in which the forwarding hardware is decoupled from control decisions. It promises to dramatically simplify network management and enable innovation and evolution. The main idea is to allow software developers to rely on network resources in the same easy manner as they do on storage and computing resources. In SDN, the network intelligence is

IEEE Communications Survey and Tutorials, Vol. 16, Issue 3, August 2014.

Why SDN?

Additional Reference:

- **Scott Shenker, et.al. The Future of Networking, and the Past of Protocols (***opennetsummit.org/archives/oct11/shenker-tue.pdf)*

An Example Transition: Programming

Machine languages: no abstractions

- Mastering complexity was crucial

Higher-level languages: OS and other abstractions

- File system, virtual memory, abstract data types, ...

Modern languages: even more abstractions

- Object orientation, garbage collection,…

Abstraction is the key to extracting simplicity

Key to Internet Success: Layers

Applications

…built on…

Reliable (or unreliable) transport

…built on…

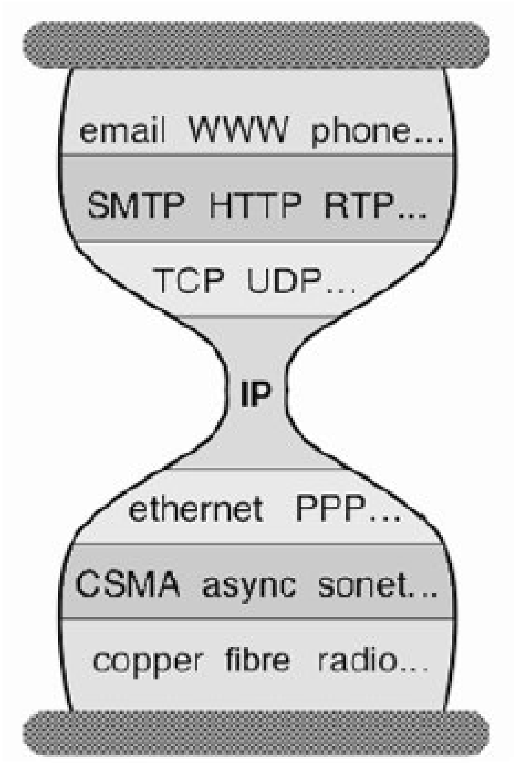
Best-effort global packet delivery

…built on…

Best-effort local packet delivery

…built on…

Physical transfer of bits



Problems

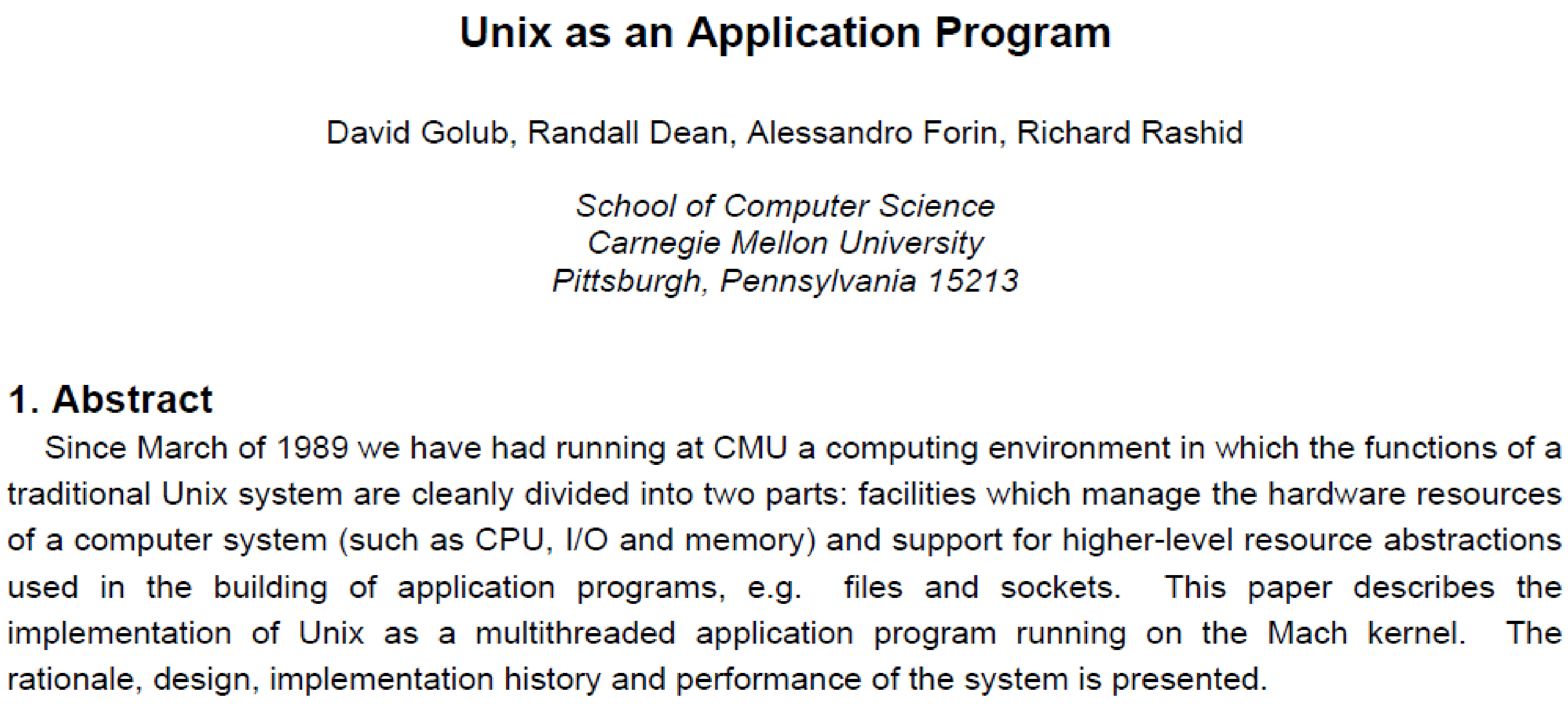
Networking:

- Teach big bag of protocols

- Notoriously difficult to manage

- Evolves very slowly

How about the other fields in “systems” such OS, DB, computer architecture….?


Unix as an Application Program

David Golub, Randall Dean, Alessandro Forin, Richard Rashid

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

1. Abstract

Since March of 1989 we have had running at CMU a computing environment in which the functions of a traditional Unix system are cleanly divided into two parts: facilities which manage the hardware resources of a computer system (such as CPU, I/O and memory) and support for higher-level resource abstractions used in the building of application programs, e.g. files and sockets. This paper describes the implementation of Unix as a multithreaded application program running on the Mach kernel. The rationale, design, implementation history and performance of the system is presented.

Golub, David B., et al. "UNIX as an Application Program." *UsENIX summer*. 1990.

Desirable Features

- Idea:
  - Operating system: decouple policy and mechanism
  - Network: decouple control plane and data plane
- Programmable Network
  - Simplify network management
  - enable evolution and innovation
- Commoditizing network hardware
- Eliminate (special purposes) middleboxes
- Support “third-party” apps

Brief History

Open Signaling (1990s)

- Make ATM, Internet and mobile networks more open, extensible, and programmable
- Separation between hardware and control software
- Access to the network hardware via open programmable network interfaces.

- Focuses on connection-oriented network services in the early days
  - General Switch Management Protocol (GSMP), IETE Working Group
- Example of such interfaces?

Brief History (2)

Active Networks (1990s)

- user programmable switches, with inband data transfer and out-of-band management channels
- Capsules or program fragments that can be carried in user messages and executed by the routers

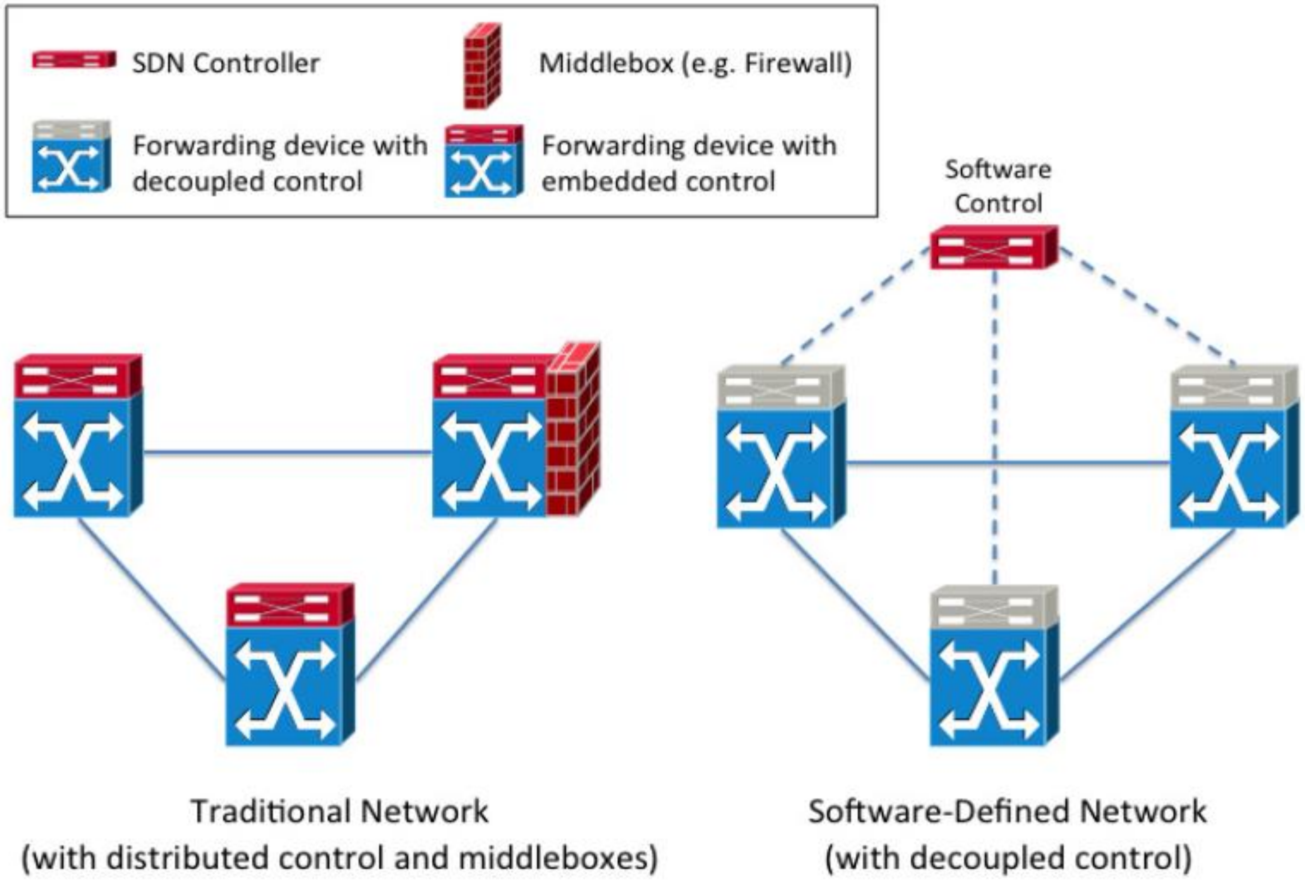
What are the research issues?

Brief History (3)

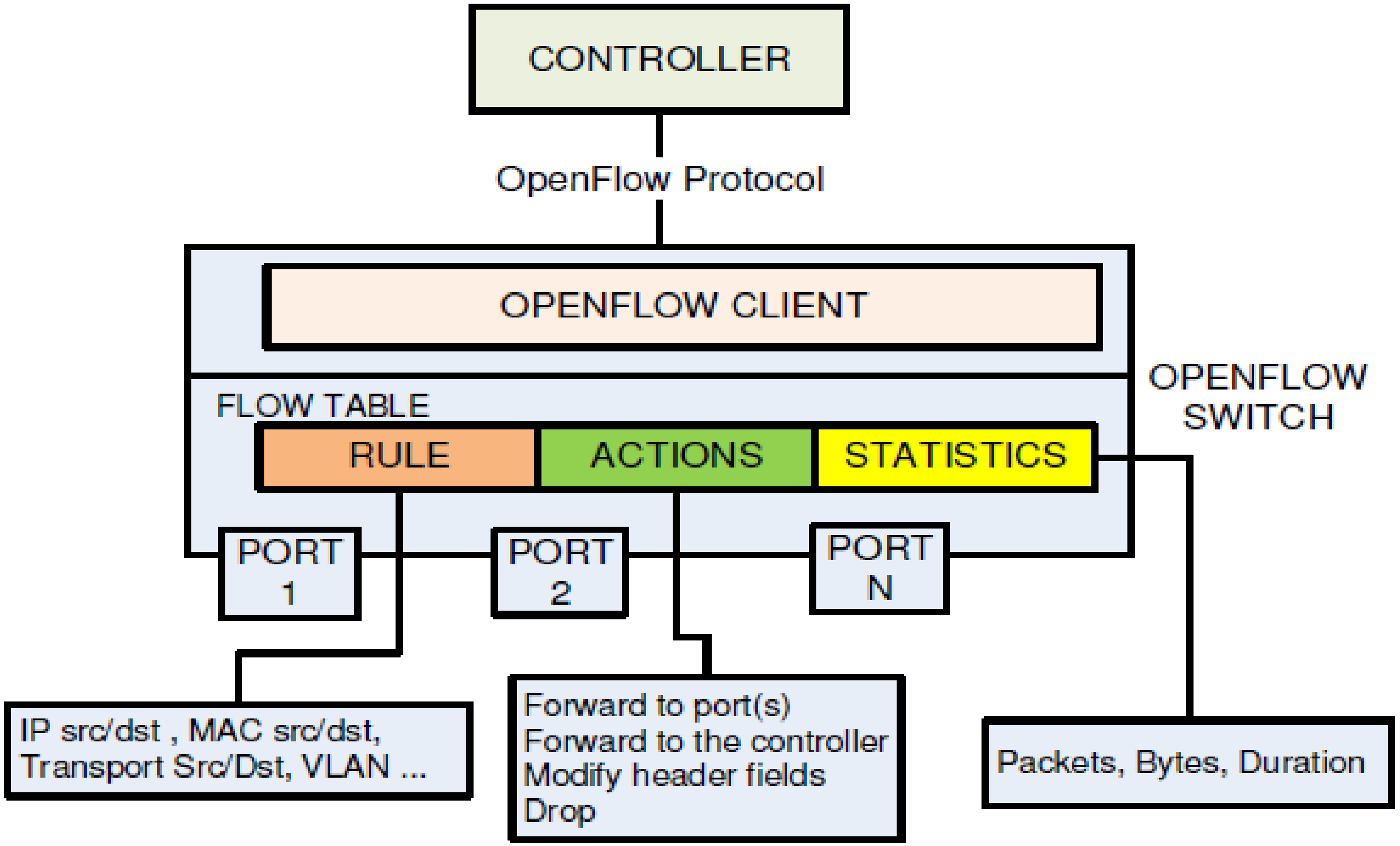
DCAN (1990s)

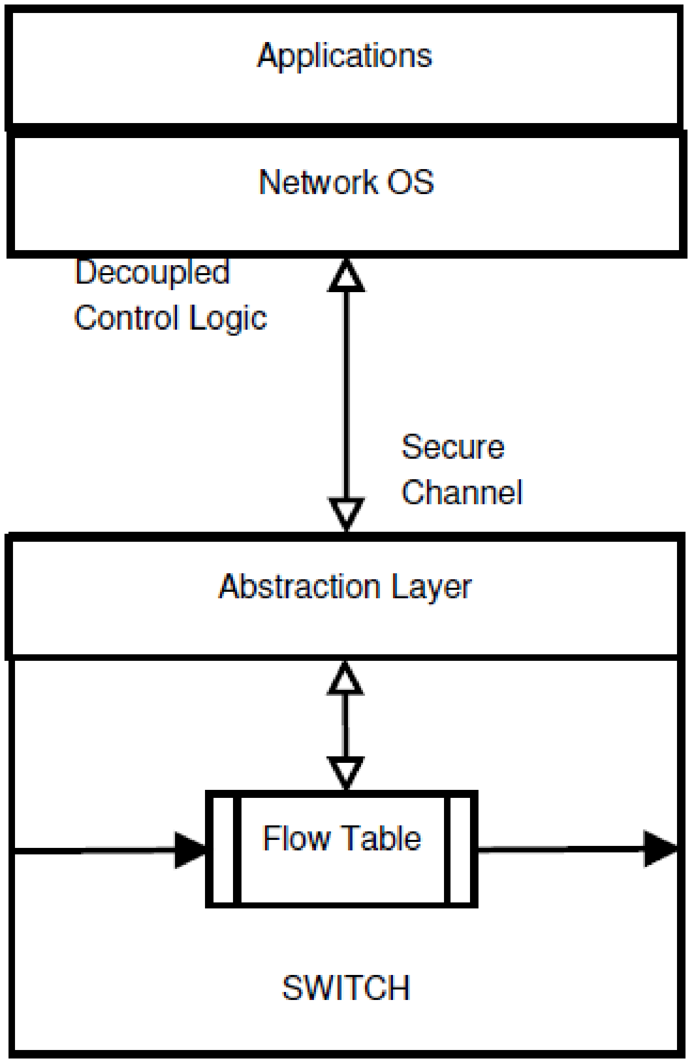
- Control and management functions of many devices should be decoupled from the devices and delegate to external entities dedicated to that purpose
- Multiple heterogeneous control architectures are allowed to run simultaneously over a single ATM switch

Traditional vs SD Network


SDN Controller
Middlebox (e.g. Firewall)
Forwarding device with decoupled control
Forwarding device with embedded control
Software Control
Traditional Network (with distributed control and middleboxes)
Software-Defined Network (with decoupled control)

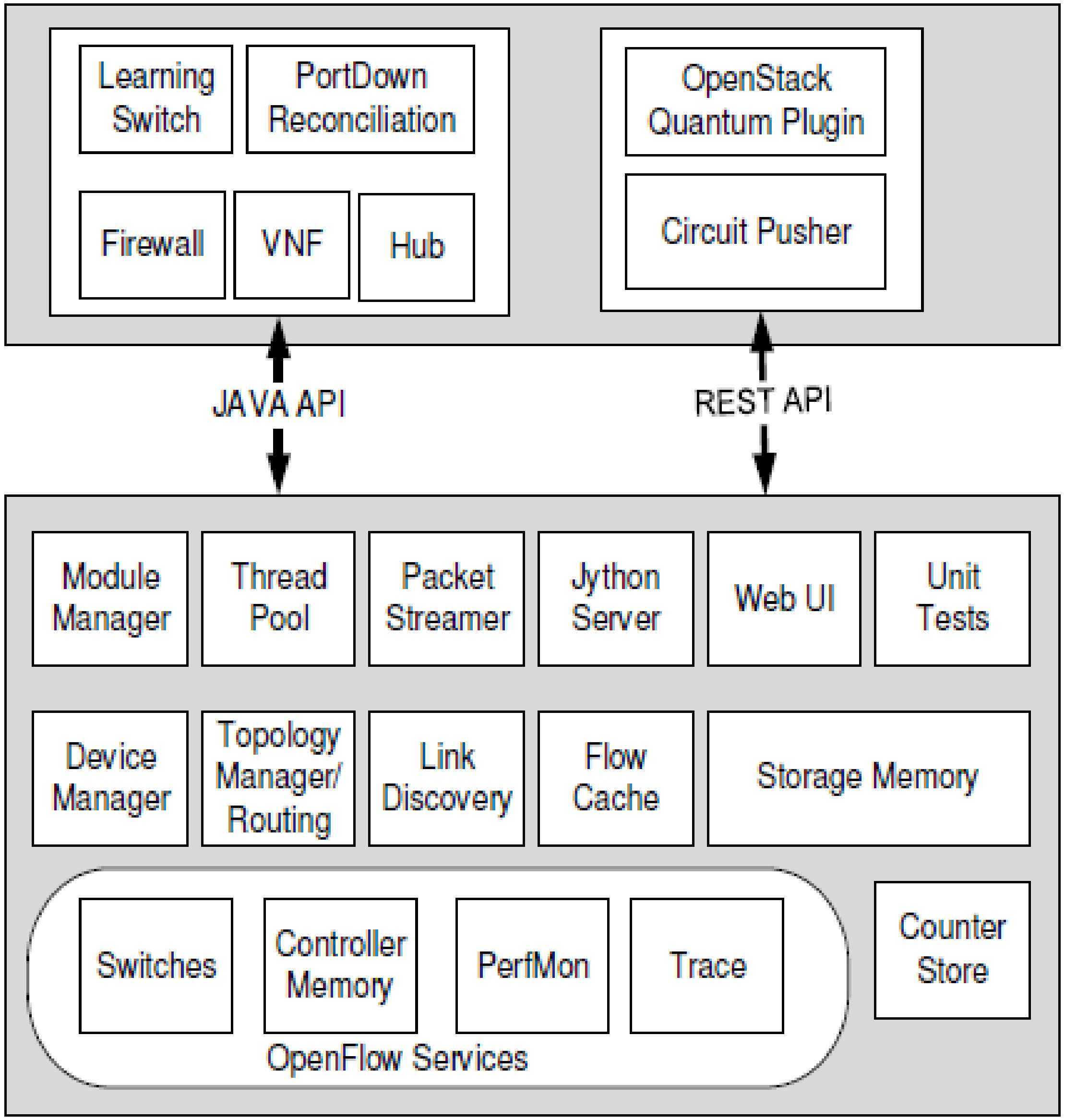
OpenFlow Architecture





- Supports multiple tables and pipeline processing
- Hybrid switches: switches that have both OpenFlow and non-OpenFlow ports

Example of a Controller: Floodlight



Some Design Issues

Ternary Content Addressable Memory (TCAM) has been used to support rule lookup and is expensive and power-hungry

- The number rules that can be supported is a bottleneck to the scalability

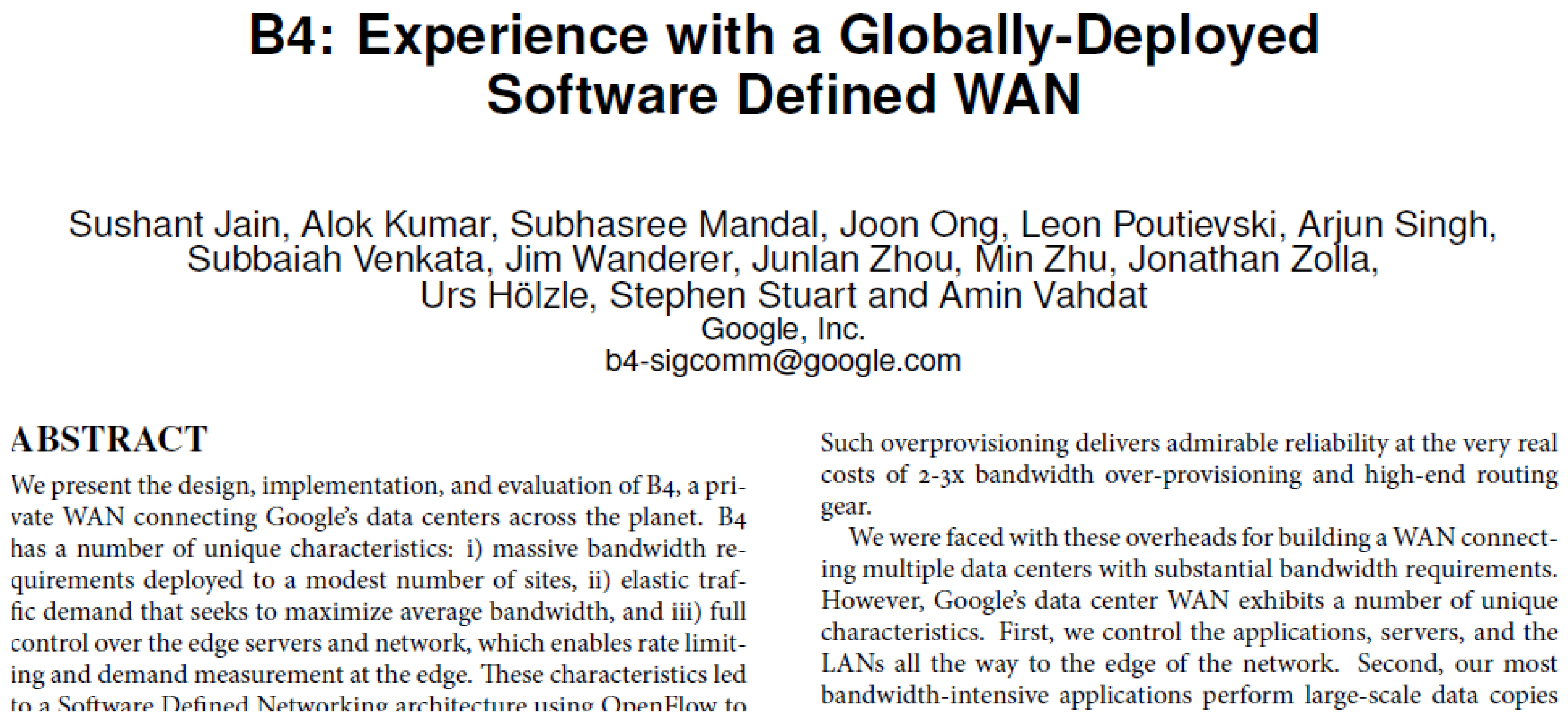
The use of a single controller impose limitations on the scalability in terms of network size and number of request processed

- Earlier controller running on desktop: 11,000 request/sec, response time 1.5ms
- Recent controller running on 8-core processor: 1.6M request/sec, response time 2ms

Placement of controller and bandwidth of control links

SDN Applications

- Enterprise Networks: many middleboxes functions have been implemented using SDN including NAT, firewall and load balancers.
- Data Centers: reduce energy consumption, load balancing
- Wireless Access Networks
- Optical Networks



B4: Experience with a Globally-Deployed Software Defined WAN

Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jonathan Zolla, Urs Hölzle, Stephen Stuart and Amin Vahdat
Google, Inc.
b4-sigcomm@google.com

ABSTRACT

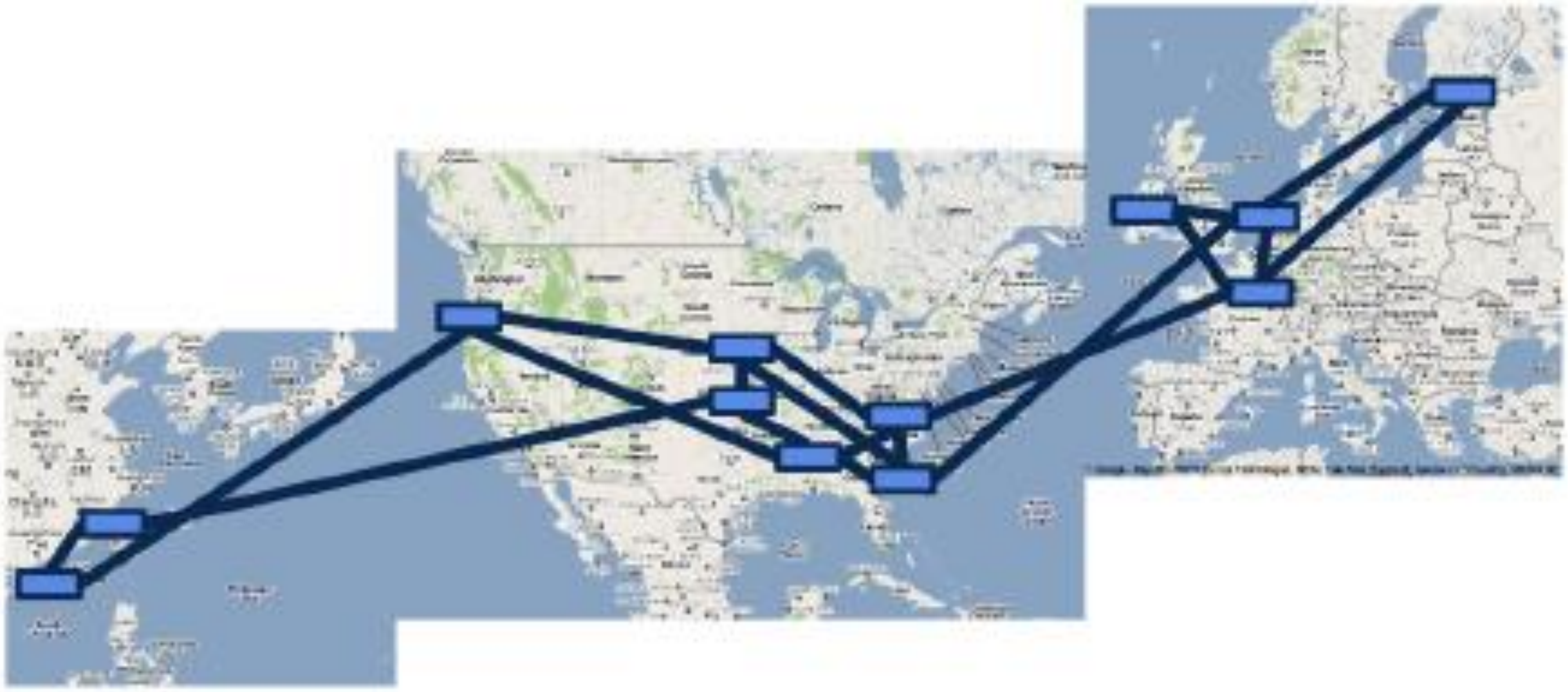
We present the design, implementation, and evaluation of B4, a private WAN connecting Google's data centers across the planet. B4 has a number of unique characteristics: i) massive bandwidth requirements deployed to a modest number of sites, ii) elastic traffic demand that seeks to maximize average bandwidth, and iii) full control over the edge servers and network, which enables rate limiting and demand measurement at the edge. These characteristics led to a Software Defined Networking architecture using OpenFlow to

Such overprovisioning delivers admirable reliability at the very real costs of 2-3x bandwidth over-provisioning and high-end routing gear.

We were faced with these overheads for building a WAN connecting multiple data centers with substantial bandwidth requirements. However, Google's data center WAN exhibits a number of unique characteristics. First, we control the applications, servers, and the LANs all the way to the edge of the network. Second, our most bandwidth-intensive applications perform large-scale data copies

ACM SIGCOMM 2013

Google’s Deployment

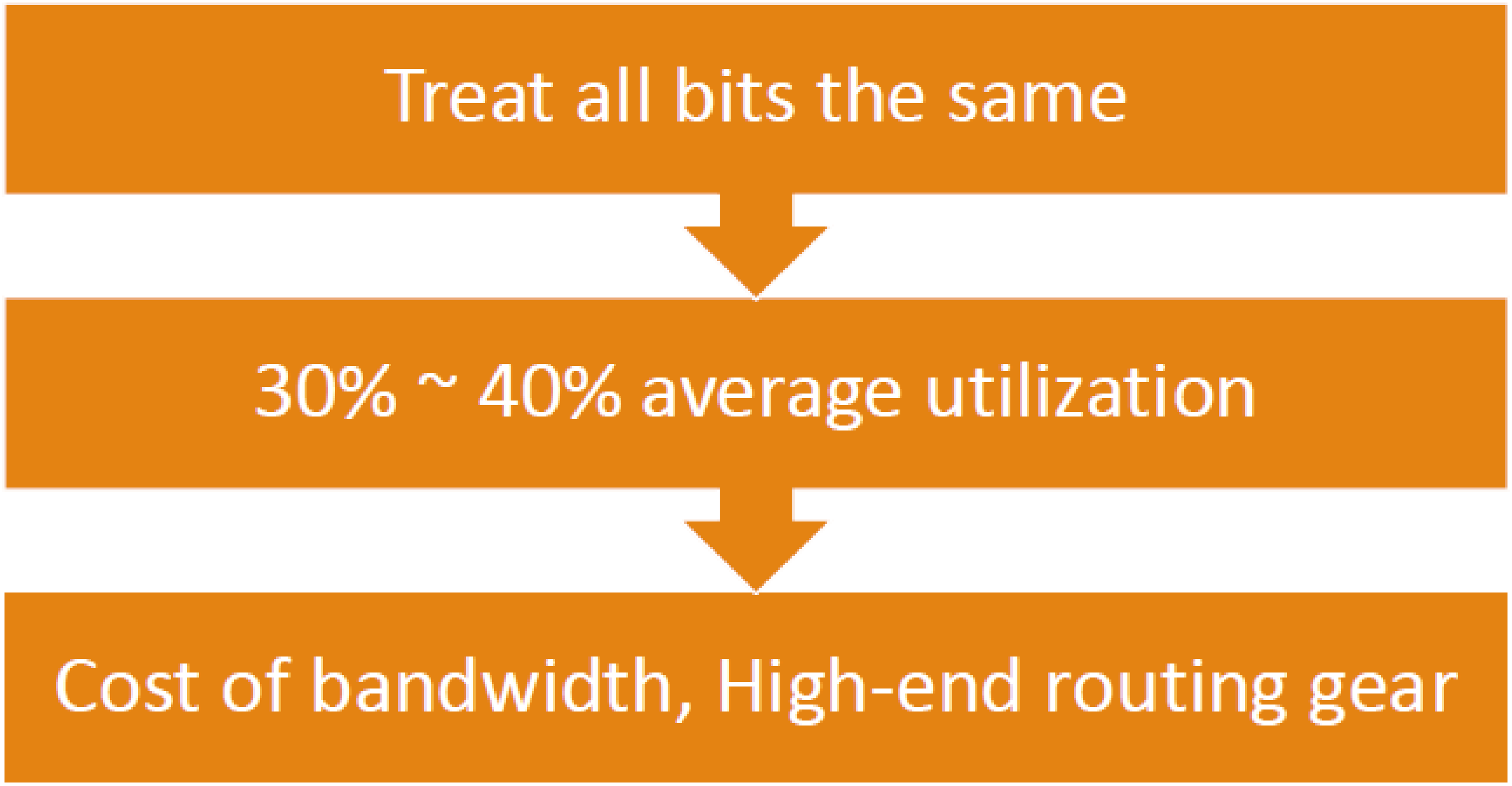


Google’s Data Center

- Massive data transfer
- Elastic traffic demand
- Full control over Edge servers and networks

- B4 has been deployed for more than 3 years. Among the first and largest SDN deployment,
- Carries more traffic than Google’s public facing WAN

Traditional Approach


Treat all bits the same
30% ~ 40% average utilization
Cost of bandwidth, High-end routing gear

Traffic Priority

- **User data copies** to remote data centers for availability/durability (lowest volume, most latency intensive, highest priority)
- **Remote storage access** for computation over distributed data sources
- **Large-scale data push** synchronizing state across multiple data centers (highest volume, least latency intensive, lowest priority)

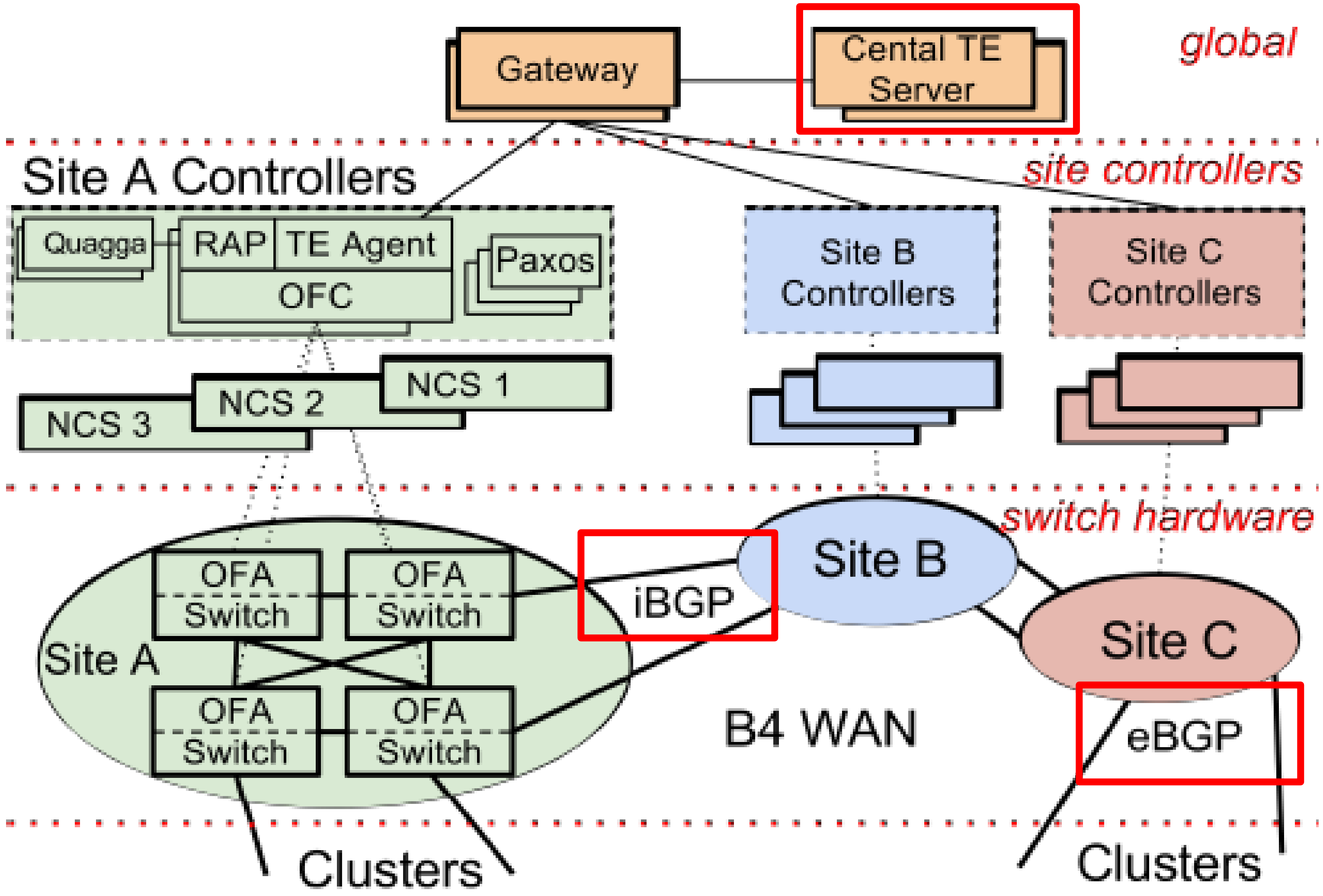


Figure 2: B4 architecture overview.

Traffic Engineering States

- Network Topology
- Flow Group (FG): aggregates flows into groups based on {source, dest, QoS} tuple
- Tunnel: a site level path, e.g. sequence of sites {A->B->C}
- Tunnel Groups (TG): maps FGs to a set of tunnels with weights

Traffic Engineering Algorithm

Target: Achieve max-min fairness.

Tunnel Selection selects the tunnels to be considered for each FG.

Tunnel Group Generation allocates bandwidth to FGs using bandwidth functions to prioritize at bottleneck links.

Tunnel Group Quantization changes split ratios in each FG to match the granularity supported by switch hardware tables.

Tunnel Selection

Find the *k* shortest tunnels in the topology graph.

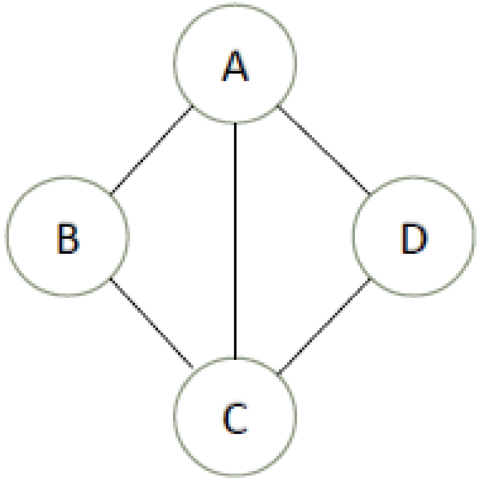
Example: Assume *k* = 3.

FG[1]: A → B

- T[1][1] = A → B
- T[1][2] = A → C → B
- T[1][3] = A → D → C → B

FG[2]: A → C

- T[2][1] = A → C
- T[2][2] = A → B → C
- T[2][3] = A → D → C



Use of Equal-cost multi-path routing (**ECMP**)

Tunnel Group Generation

Allocate bandwidth to FGs based on demand and priority.

1.Initialize each FG with their most preferred tunnels.

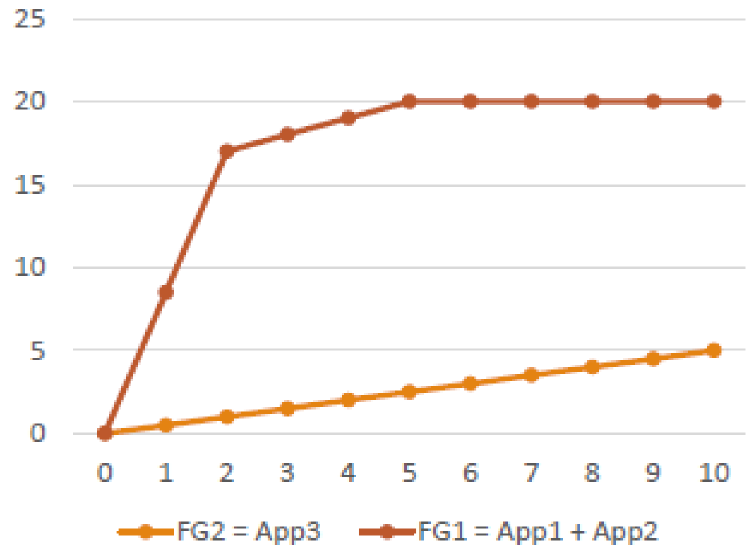
2.Allocate bandwidth by giving equal fair share to each preferred tunnel.

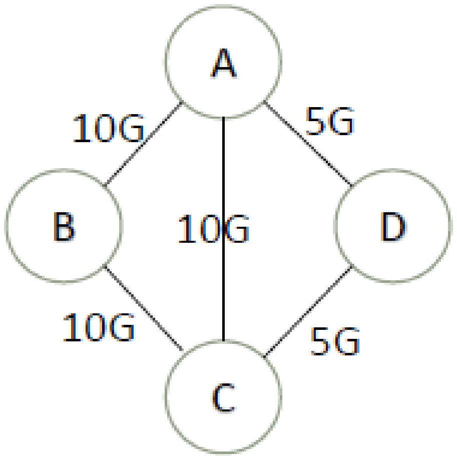
3.Freeze tunnels containing any bottlenecked link.

4.If every tunnel is frozen, or every FG is fully satisfied, end.

5.Select the most preferred non-frozen tunnel for each non-satisfied FG, goto2.

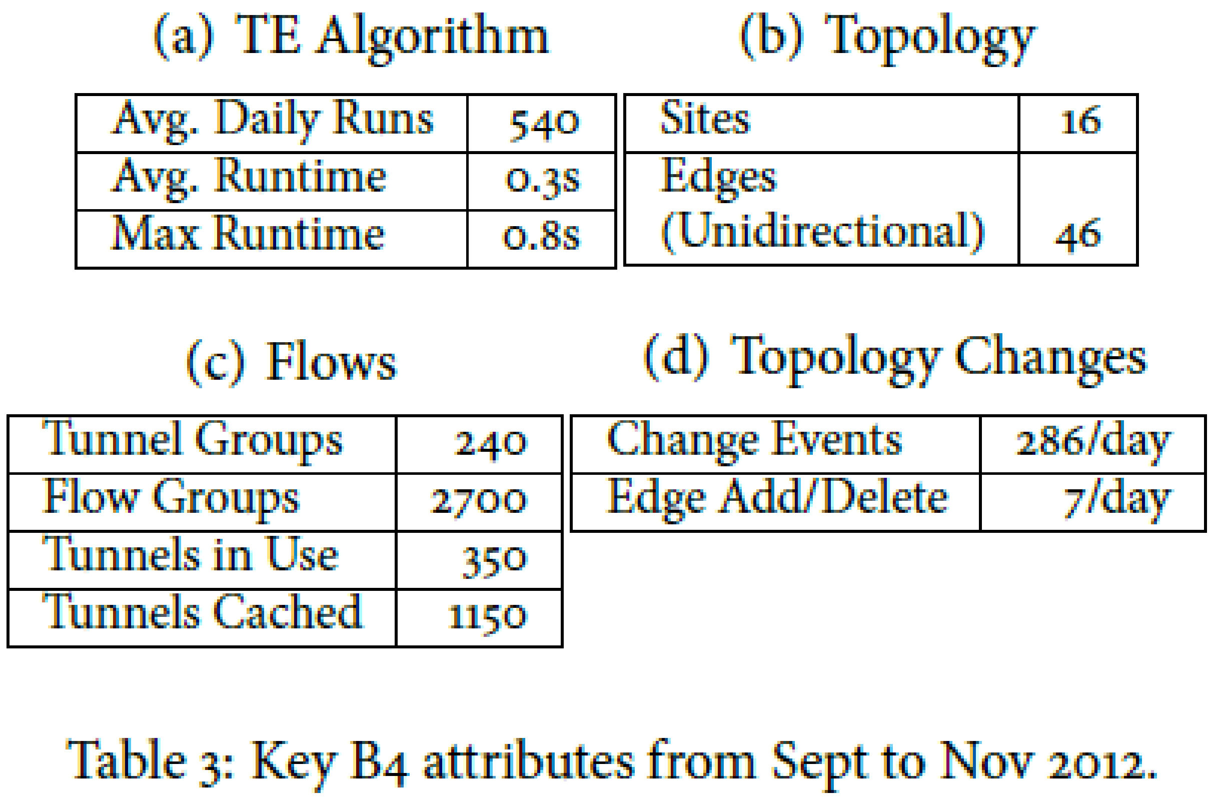
	FG1 prefer	FG2 prefer	Fair share	FG1 get/need	FG2 get/need	Bottle neck links	Freeze tunnels
1	A→B	A→C	0.9	10 / 20	0.45 / inf	A→B	A→B, A→B→C
2	A→C	A→C	3.33	8.33 / 10	1.21 / inf	A→C	A→C→B, A→C
3	A→D→C→B	A→D→C	1.67	1.67 / 1.67	3.34 / inf	all	all





Result:
FG1 (20/20):
A→B: 10
A→C→B: 8.33
A→D→C→B: 1.67
FG2 (5/inf):
A→C: 1.67
A→B→C: 0
A→D→C: 3.34

Evaluation



(a) TE Algorithm

Avg. Daily Runs	540
Avg. Runtime	0.3s
Max Runtime	0.8s

(b) Topology

Sites	16
Edges (Unidirectional)	46

(c) Flows

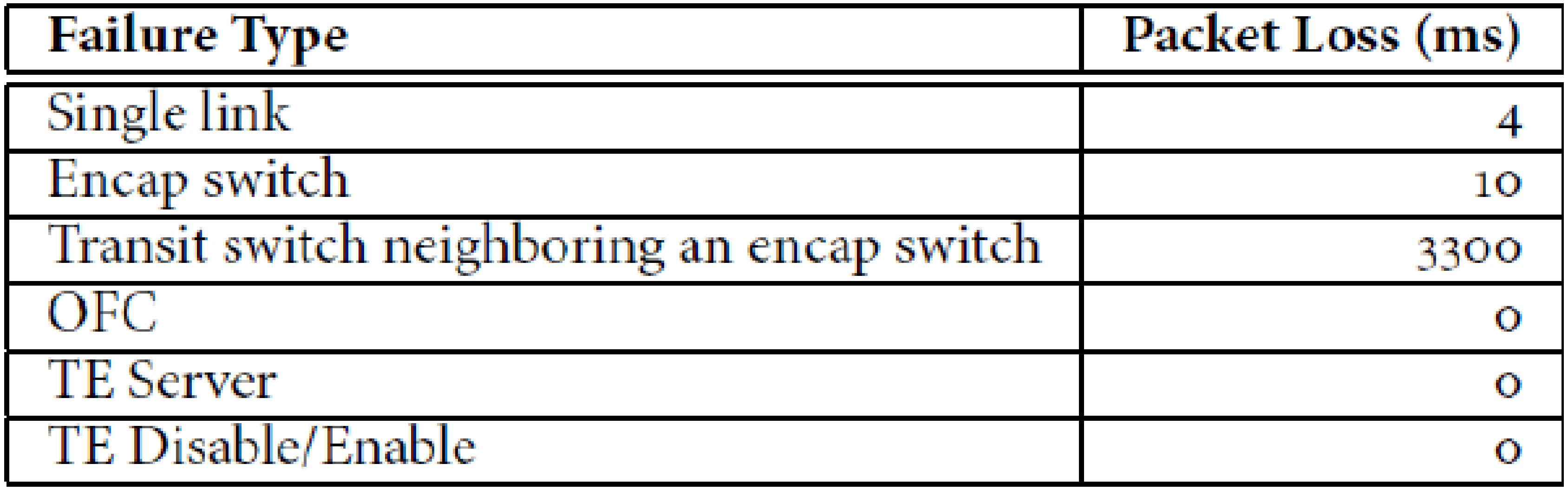
Tunnel Groups	240
Flow Groups	2700
Tunnels in Use	350
Tunnels Cached	1150

(d) Topology Changes

Change Events	286/day
Edge Add/Delete	7/day

Table 3: Key B4 attributes from Sept to Nov 2012.

Impact of Failure



Failure Type	Packet Loss (ms)
Single link	4
Encap switch	10
Transit switch neighboring an encap switch	3300
OFC	0
TE Server	0
TE Disable/Enable	0

Table 5: Traffic loss time on failures.

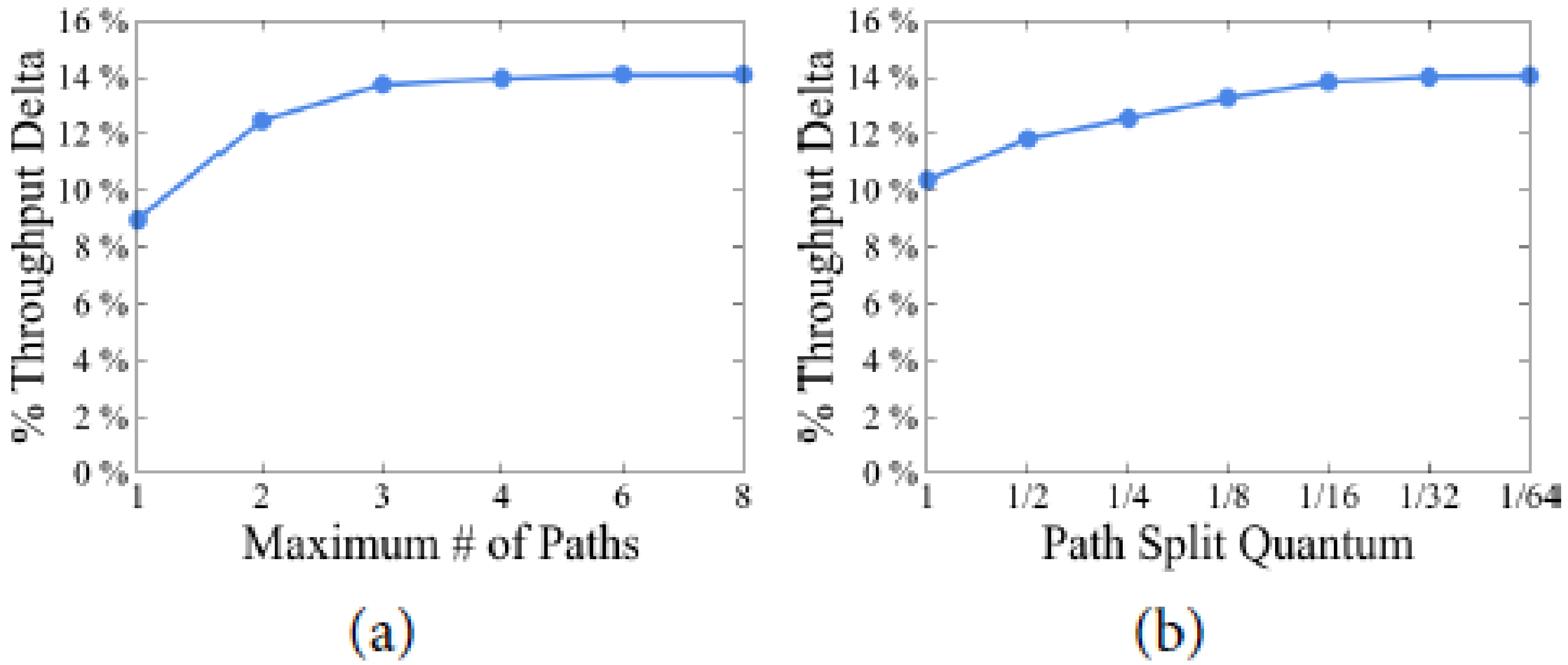
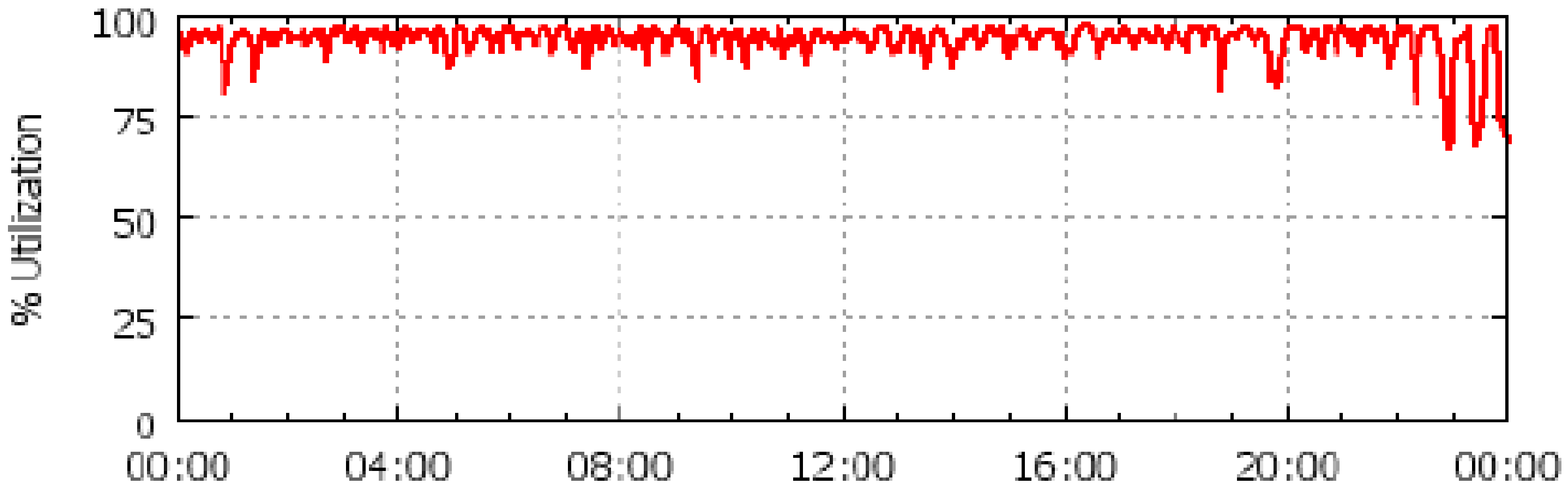
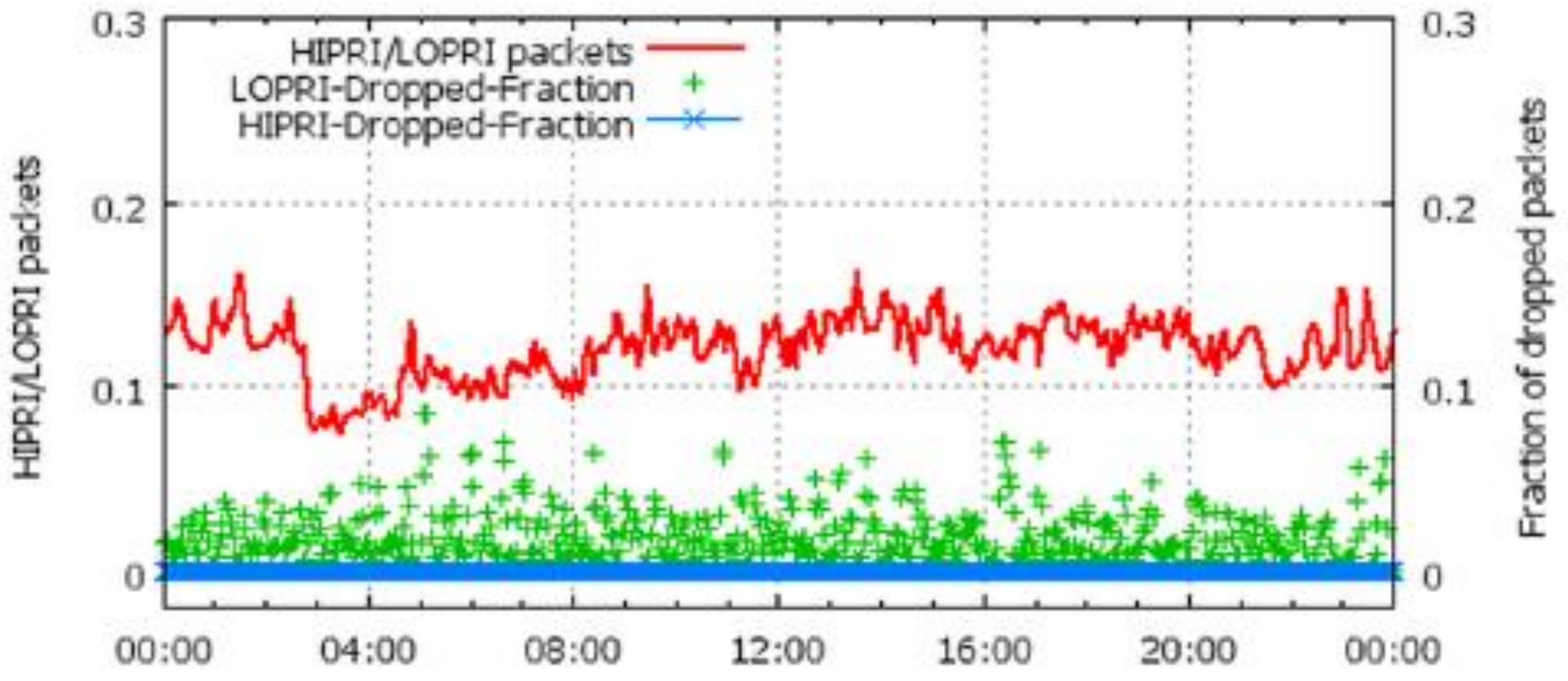


Figure 13: TE global throughput improvement relative to shortest-path routing.

Link Utilization



Packet Loss


HIPRI/LOPRI packets
LOPRI-Dropped-Fraction
HIPRI-Dropped-Fraction
HIPRI/LOPRI packets
Fraction of dropped packets
0
0.1
0.2
0.3
00:00
04:00
08:00
12:00
16:00
20:00
00:00

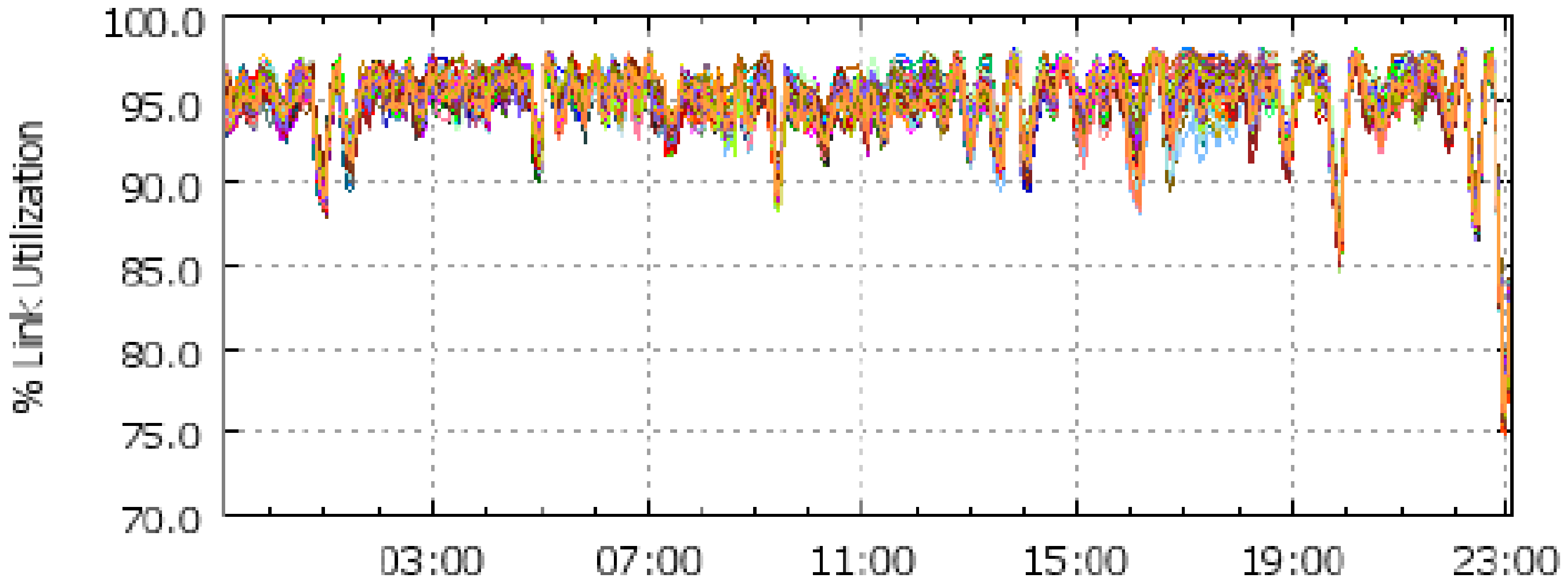


Figure 15: Per-link utilization in a trunk, demonstrating the effectiveness of hashing.