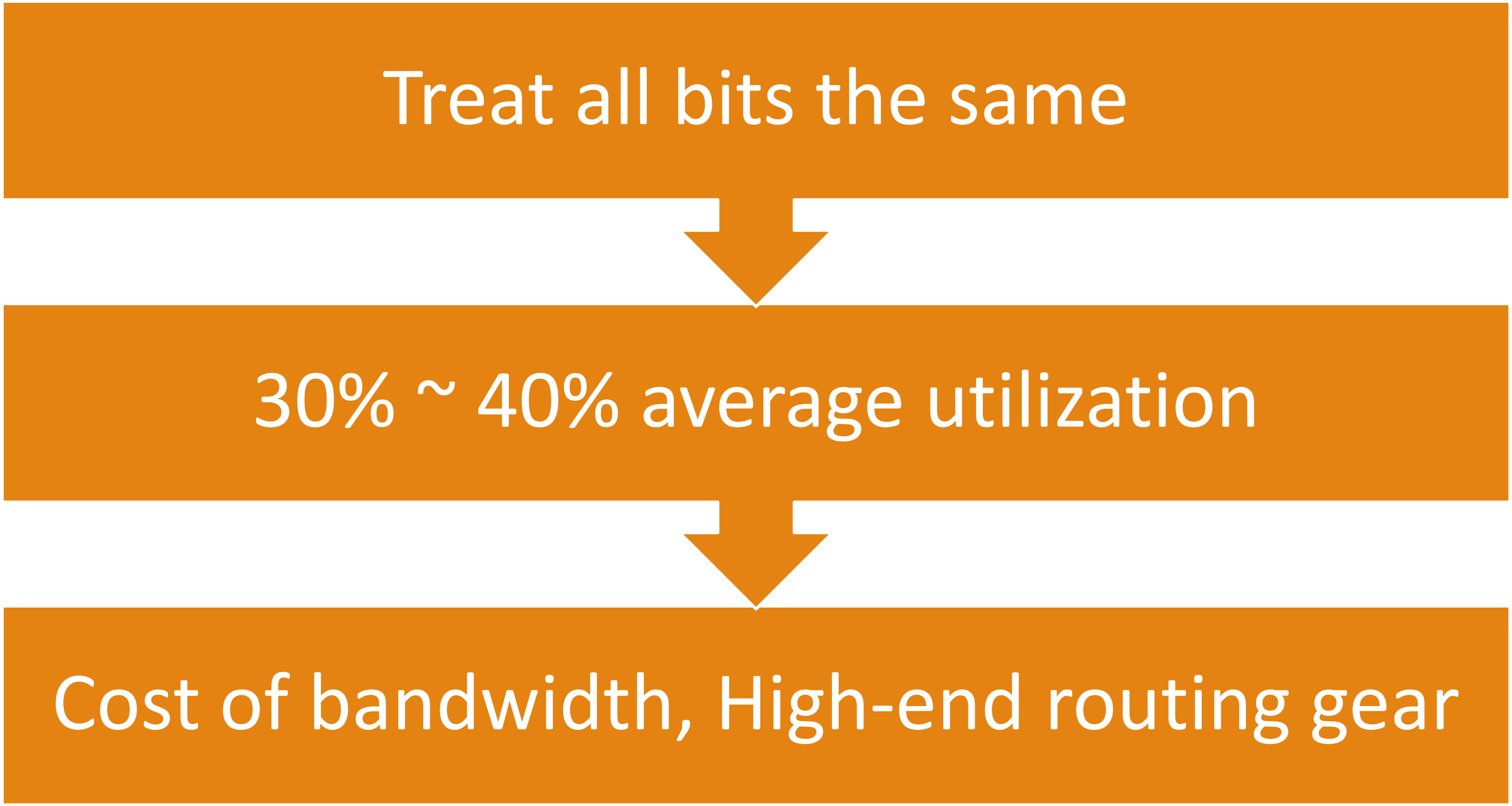
B4: Experience with a Globally-Deployed Software Defined WAN

TO APPEAR IN SIGCOMM'13

Google’s Software Defined WAN

Traditional WAN Routing



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)

Centralized Traffic Engineering (TE)

↓

Drive links to near 100% utilization

Fast, global convergence for failures

SDN Architecture

Switch hardware

- Forwards traffic.
- Does NOT run complex control software.

OpenFlow controllers (OFC)

- Maintain network state based on network control application directive and switch events.
- Instruct switches to set forwarding entries.

Central application (logical)

- Central control of the entire network.

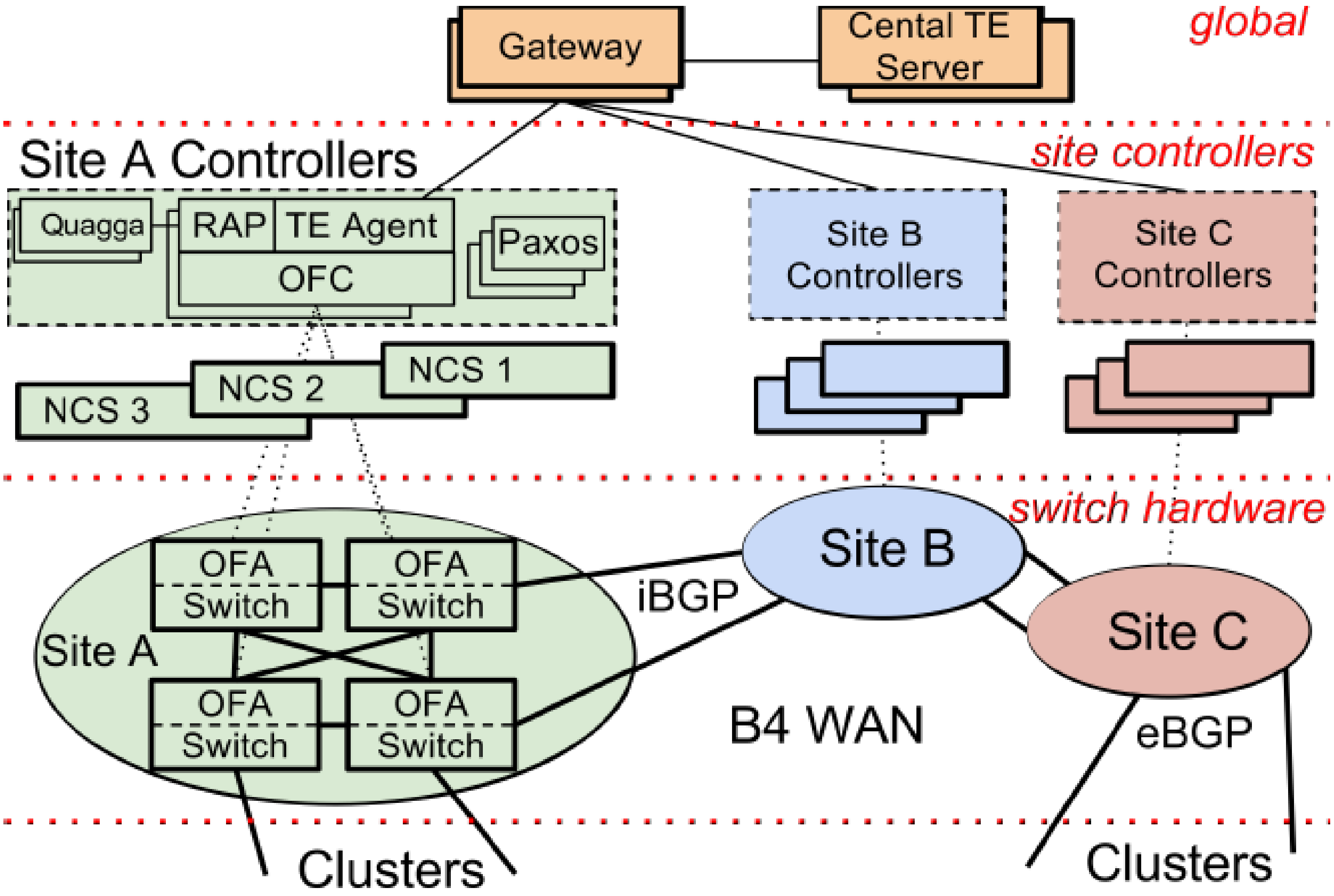
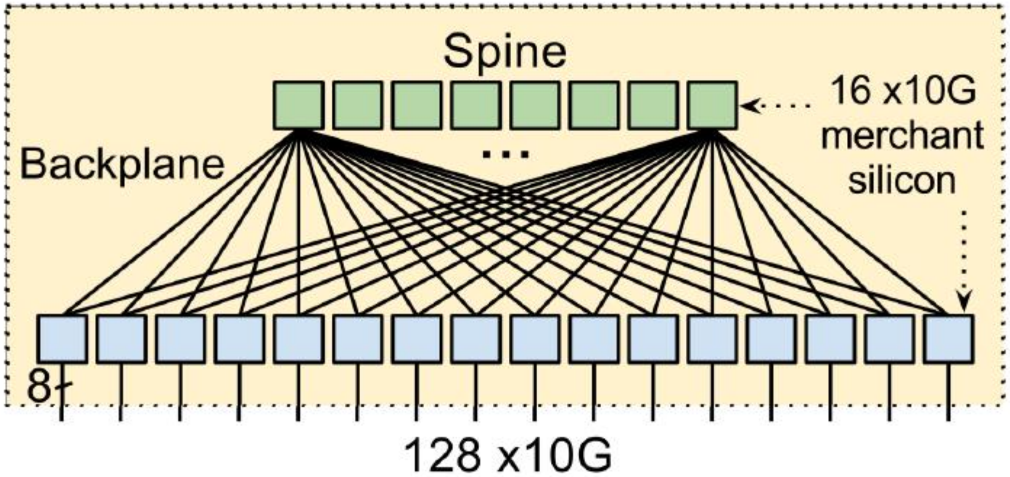
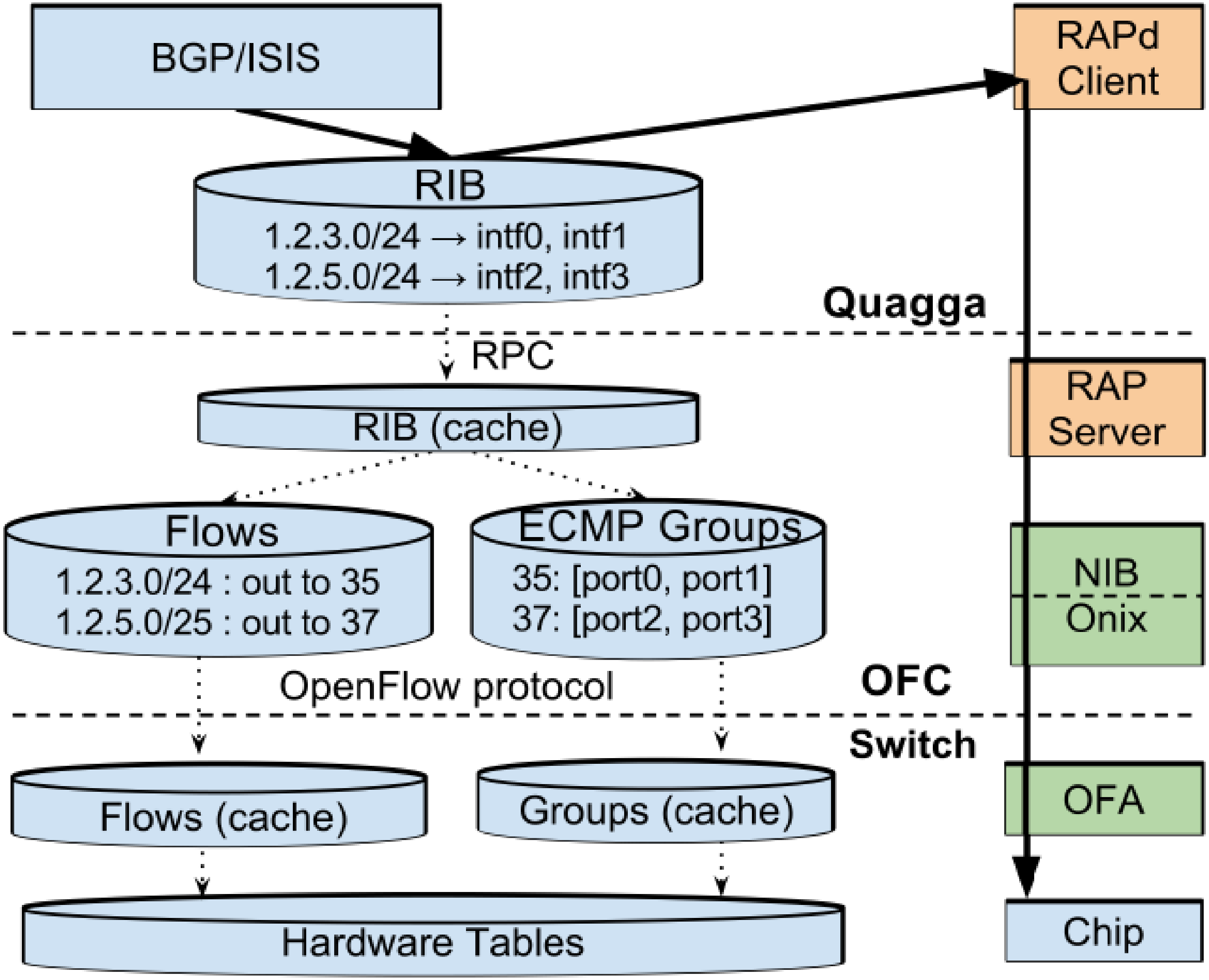


Figure 2: B4 architecture overview.

Switch Design





BGP/ISIS

RAPd Client

RIB

1.2.3.0/24 → intf0, intf1

1.2.5.0/24 → intf2, intf3

Quagga

RPC

RAP Server

RIB (cache)

Flows

1.2.3.0/24 : out to 35

1.2.5.0/25 : out to 37

ECMP Groups

35: [port0, port1]

37: [port2, port3]

NIB

Onix

OpenFlow protocol

OFC

Switch

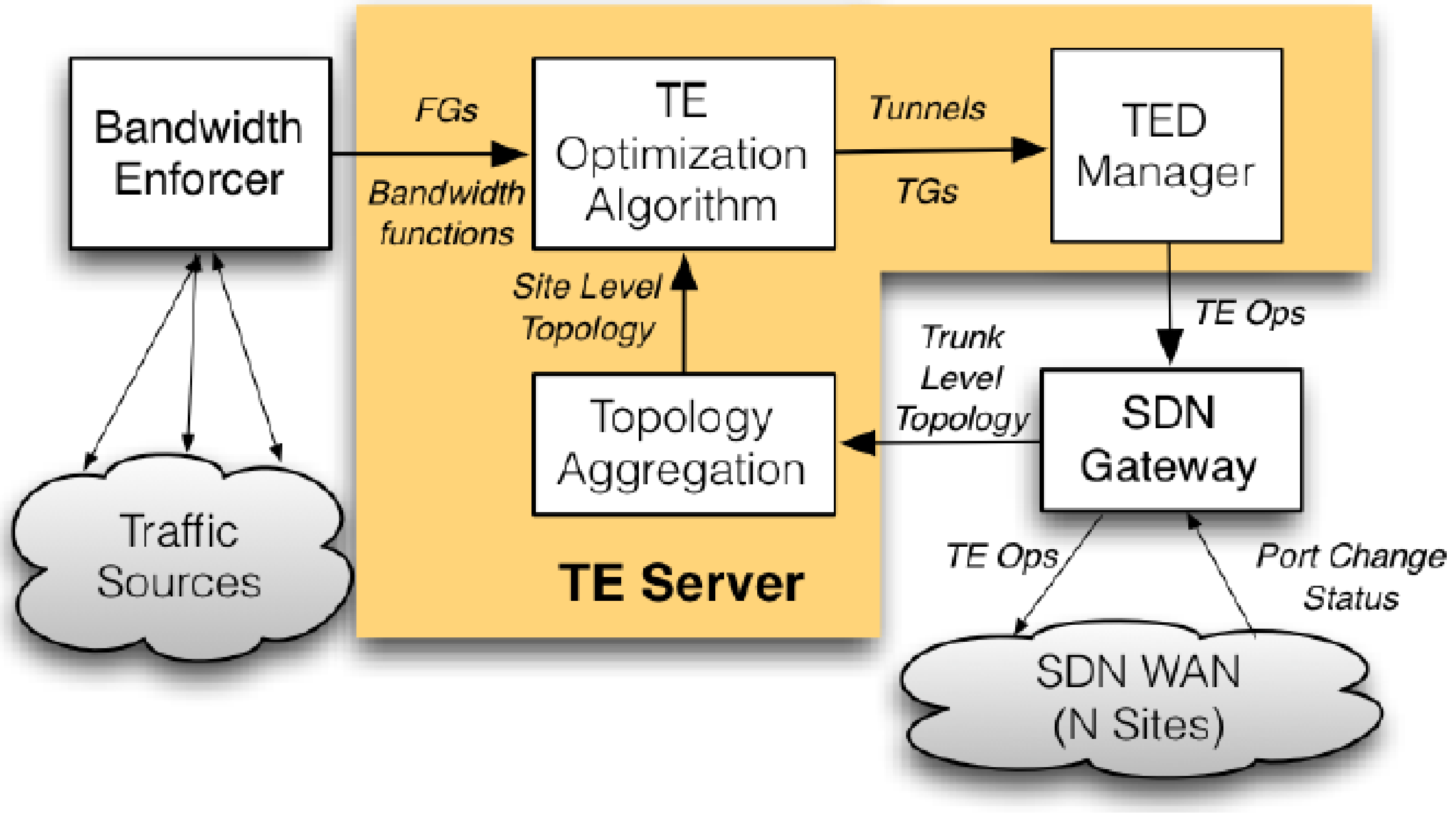
Flows (cache)

Groups (cache)

OFA

Hardware Tables

Chip



Applications are aggregated to ***Flow Group***: {source site, dest site, QoS}

Bandwidth function: linear to application weight, becomes flat at required bandwidth. (discuss later)

TE Optimization Algorithm

Target: Achieve **max-min** fairness.

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

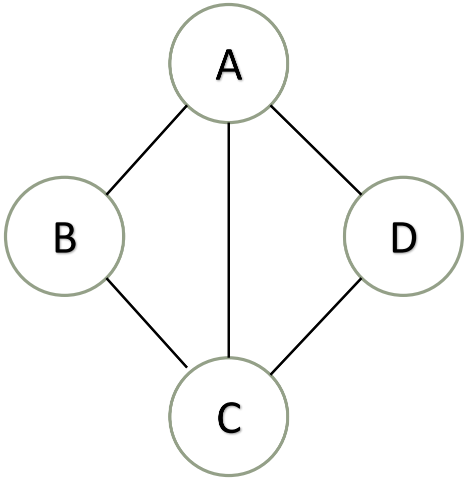
Tunnel Group Generation allocates bandwidth to FGs using *bandwidth functions* to prioritize at bottleneck links.

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.



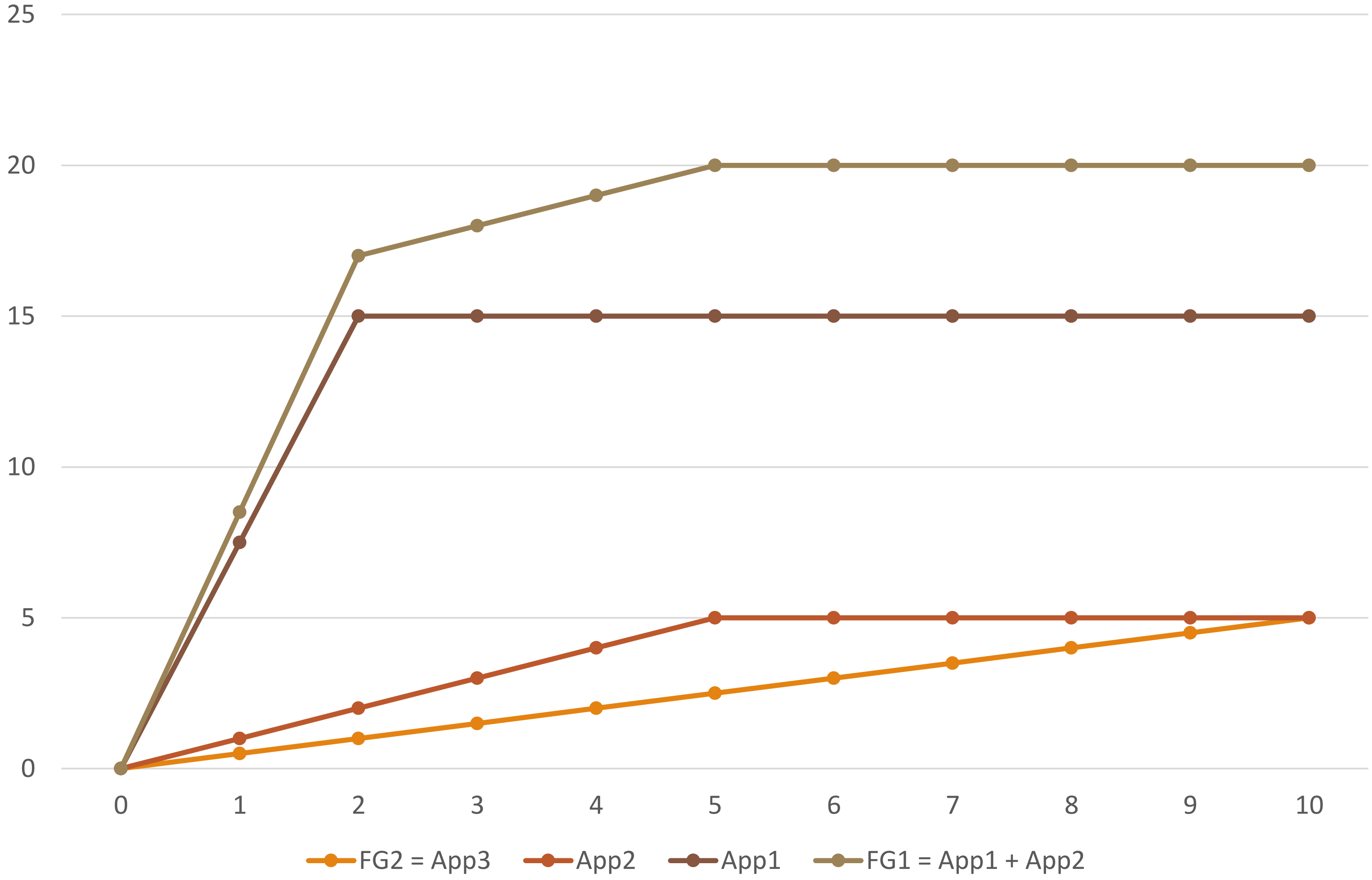
FG[1]: A → B

- 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

Composition of FG level bandwidth functions

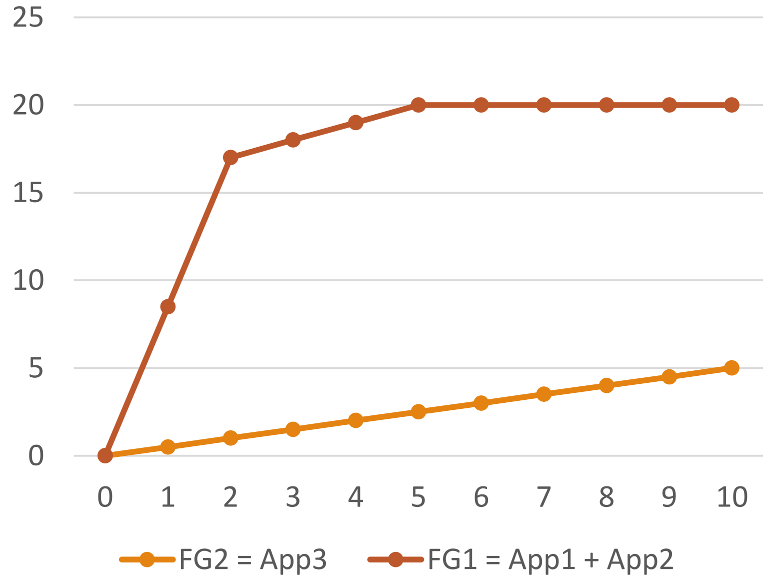


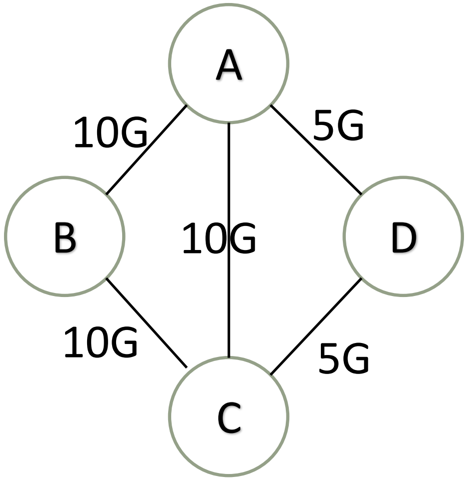
Tunnel Group Generation

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, goto 2.

	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

Tunnel Group Quantization

Determining the optimal split: integer programming problem.

Greedy Approach:

1. Down quantize (round) each split.
2. Add a remaining quanta to a non-frozen tunnel that makes the solution max-min fair (with minimum *fair share*).
3. If there are still remaining quantas, goto 2.

Tunnel Group Quantization

Example split:

- FG2: 0.3:0.0:0.7
- FG1: 0.5:0.4:0.1

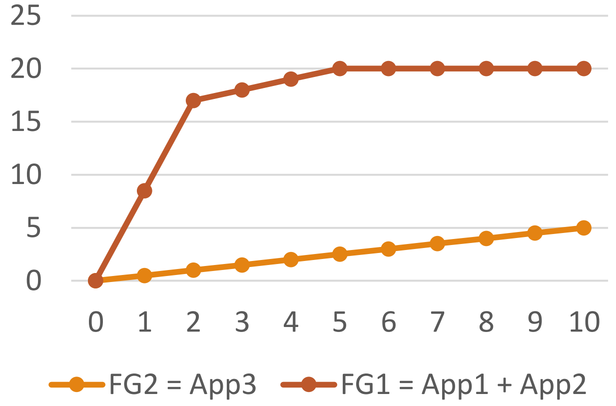
Assume quanta is 0.5.

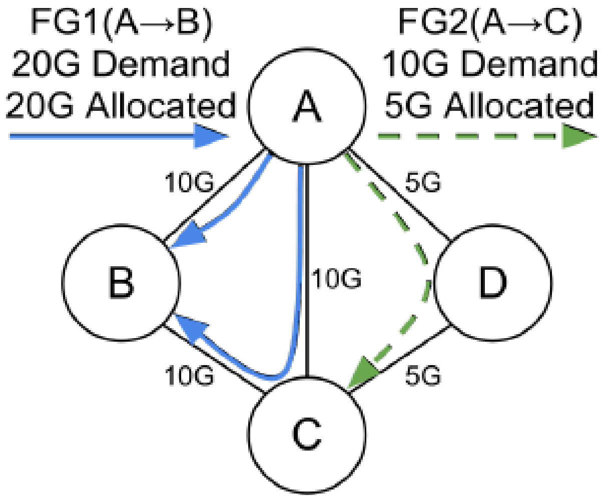
FG2 (A→C):

- A→C, A→B→C, A→D→C
- Down quantize: 0.0:0.0:0.5
- Add remaining: 0.0:0.0:1.0

FG1 (A→B):

- A→B, A→C→B, A→D→C→B
- Down quantize: 0.5:0.0:0.0
- Add remaining: 0.5:0.5:0.0





Impact of Quantization

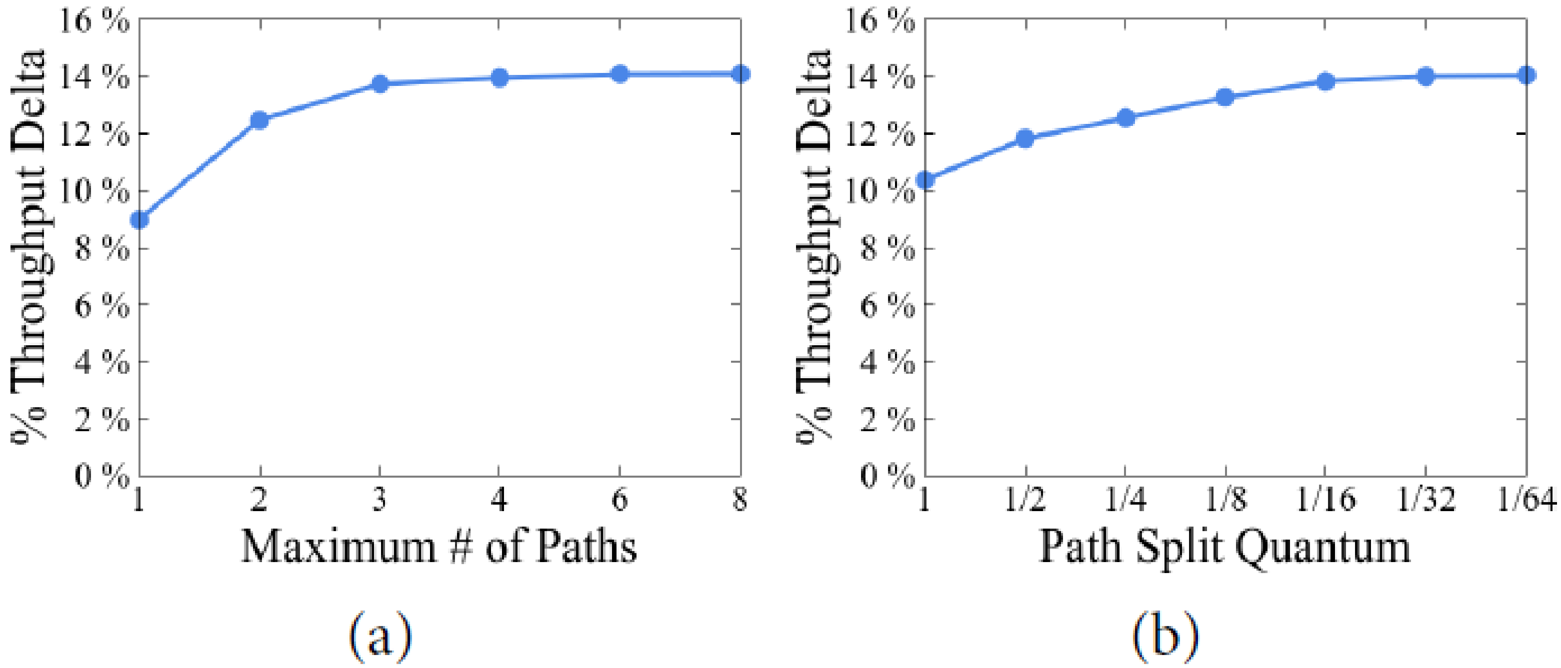


Figure 13: TE global throughput improvement relative to shortest-path routing.

TE as overlay

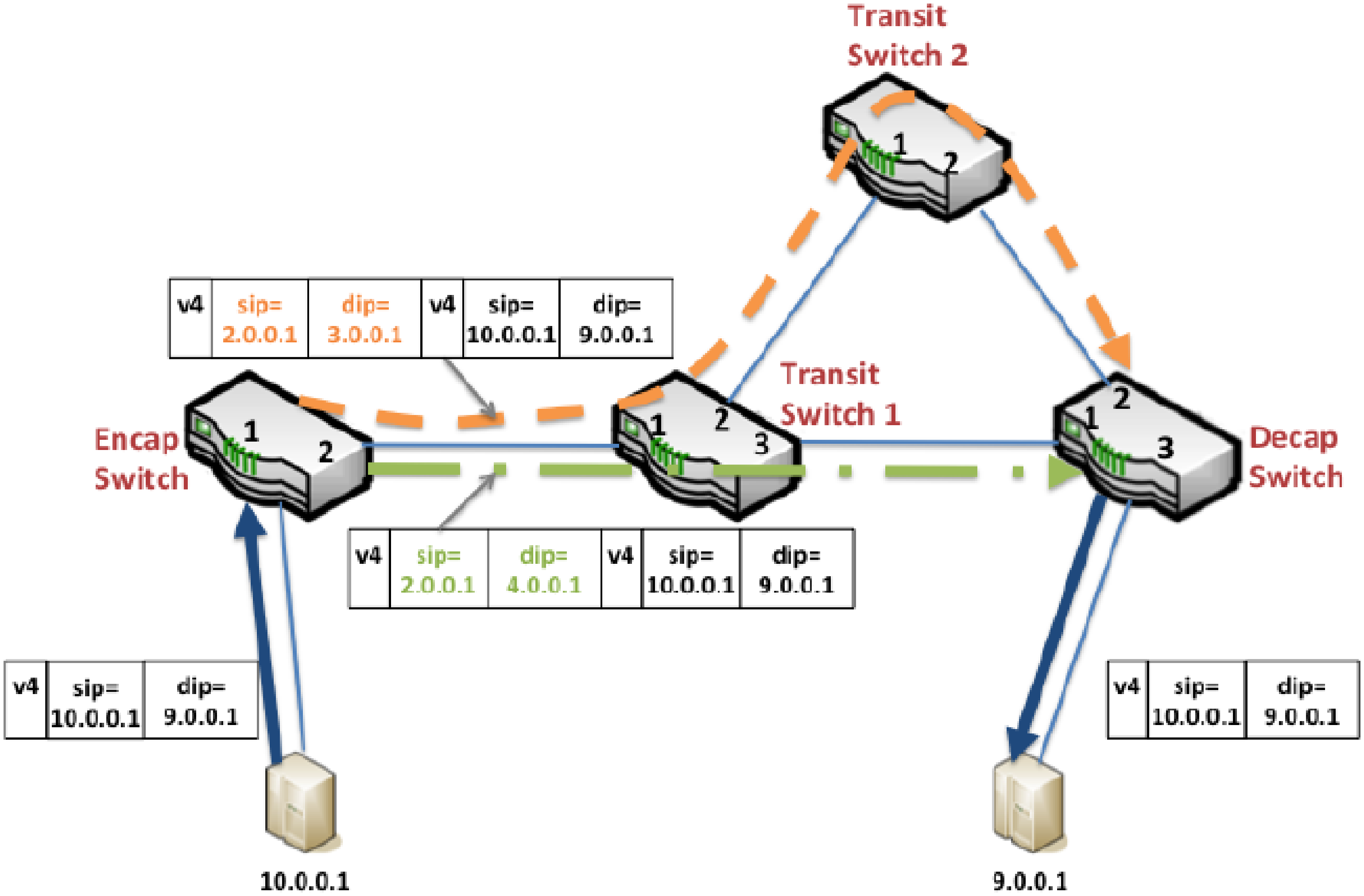
~~Integrated, centralized service combining routing and traffic engineering?~~

Traffic Engineering (central)

Standard Routing (ISIS) (per switch)

Use **prioritized** switch forwarding table entries

“**Big red button**”: disable TE service and fall back to shortest-path forwarding at any time



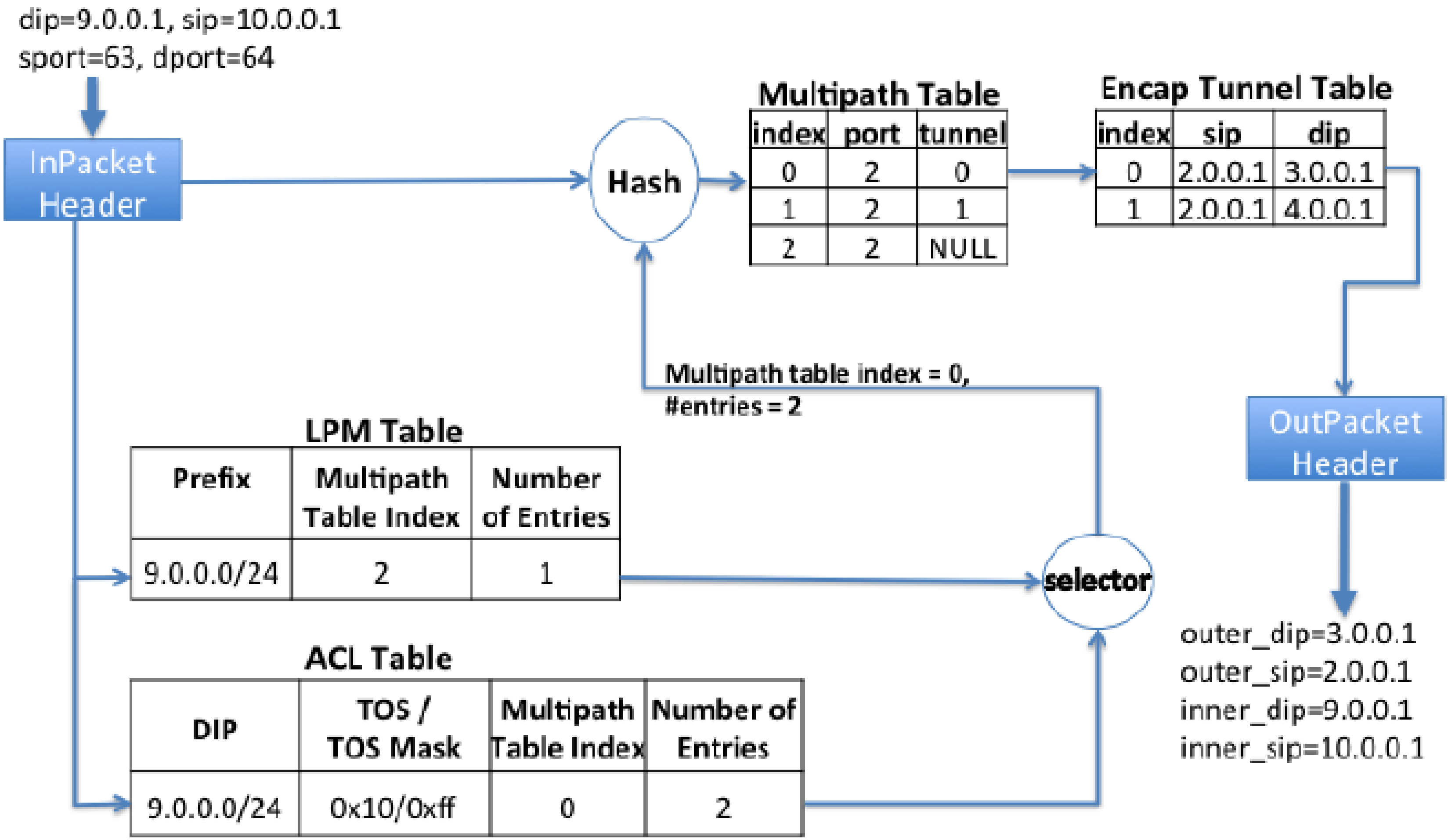
v4	sip= 2.0.0.1	dip= 3.0.0.1	v4	sip= 10.0.0.1	dip= 9.0.0.1

v4	sip= 2.0.0.1	dip= 4.0.0.1	v4	sip= 10.0.0.1	dip= 9.0.0.1

v4	sip= 10.0.0.1	dip= 9.0.0.1

v4	sip= 10.0.0.1	dip= 9.0.0.1

Figure 8: Multipath WAN Forwarding Example.



Multipath Table

index	port	tunnel
0	2	0
1	2	1
2	2	NULL

Encap Tunnel Table

index	sip	dip
0	2.0.0.1	3.0.0.1
1	2.0.0.1	4.0.0.1

LPM Table

Prefix	Multipath Table Index	Number of Entries
9.0.0.0/24	2	1

ACL Table

DIP	TOS / TOS Mask	Multipath Table Index	Number of Entries
9.0.0.0/24	0x10/0xff	0	2

Figure 9: Layering traffic engineering on top of shortest path forwarding in an encap switch.

Ops Dependency

In order to avoid packet drops, not all ops can be issued simultaneously.

Rules:

- Configure a *tunnel* at all affected sites before sending TG and FG.
- A *tunnel* cannot be deleted until all referencing entries are removed.

Enforce dependencies (in case of network delays / reordering):

- OFC maintains the highest session sequence number.
- OFC rejects ops with smaller sequence number.
- TE Server retries any rejected ops after a timeout.

Deployment

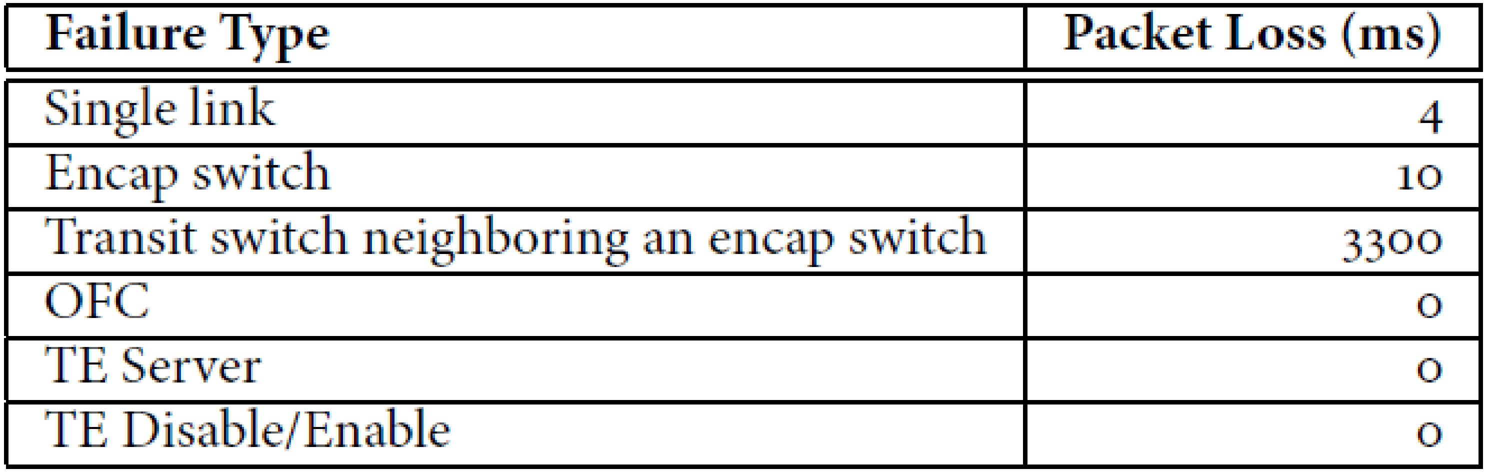
Statistics

- 13 topology changes per minute
- Trimming maintenance updates: 0.2 changes per minute
- Edge add/delete events 7 changes per day (TE algorithm rums on aggregated topology view)

Takeaways:

- Topology aggregation significantly reduces path churn and system load.
- Even with topology aggregation, edge removals happen multiple times a day.
- WAN links are susceptible to frequent port flaps and benefit from dynamic centralized management.

Impact of Failures

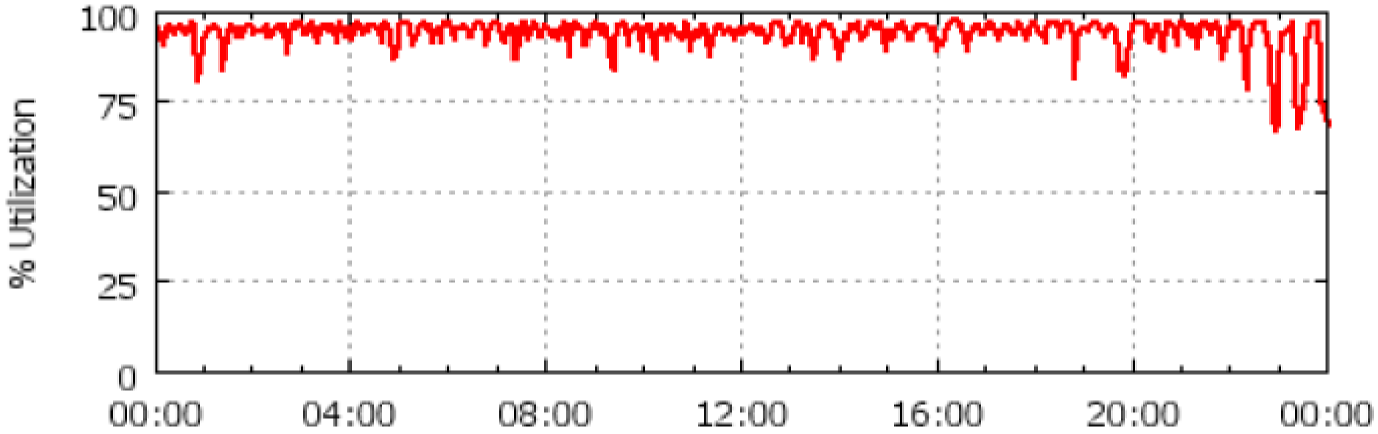


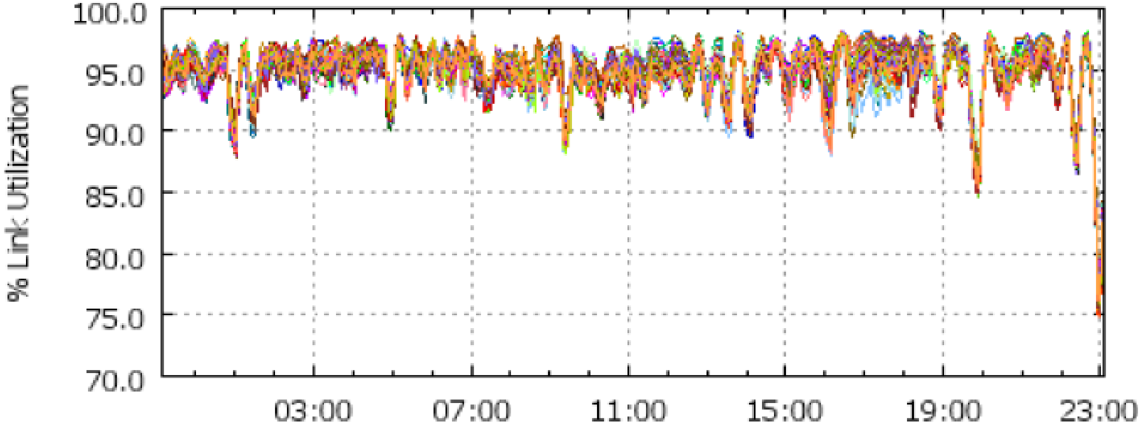
Failure Type	**Packet Loss (ms)**
Single link	4
Encap switch	10
Transit switch neighboring an encap switch	3300
OFC	0
TE Server	0
TE Disable/Enable	0

Table 5: Traffic loss time on failures.

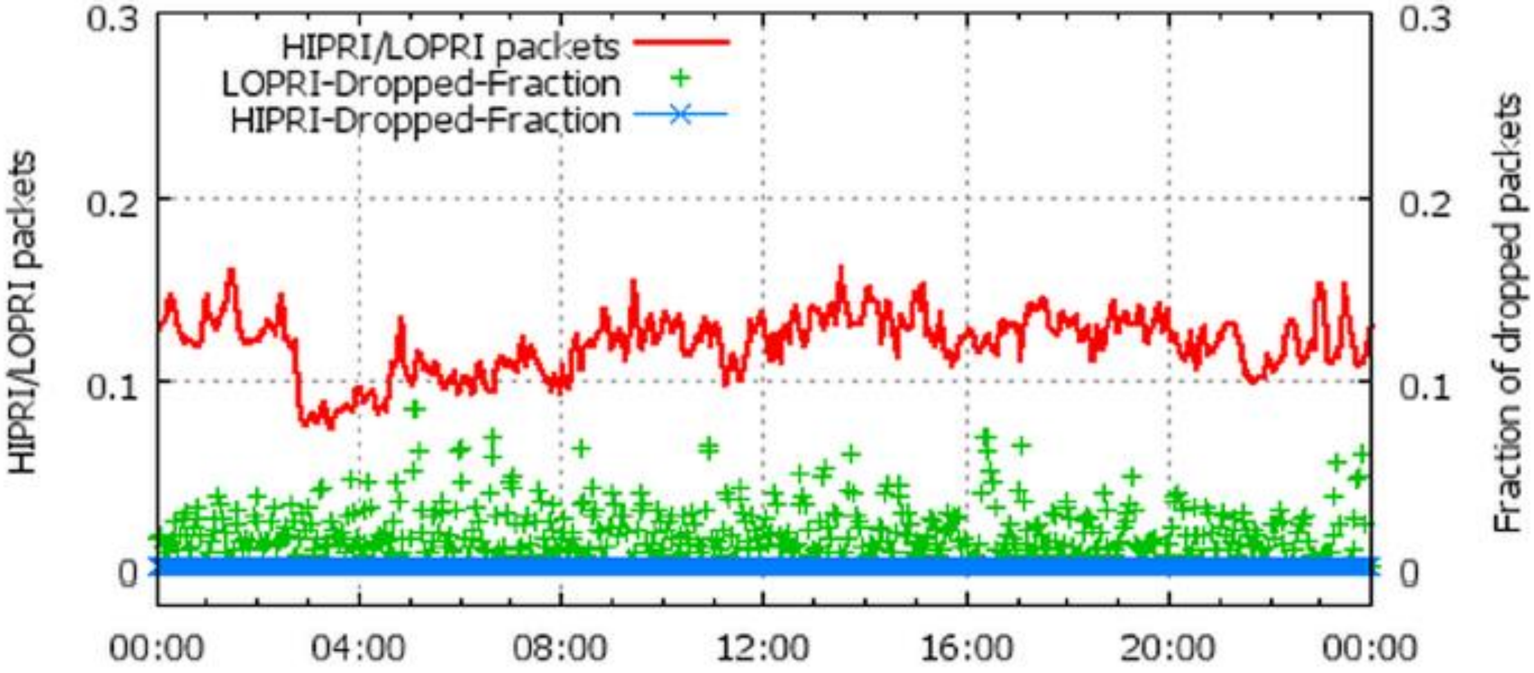
Centralized TE is not a cure-all.

Link Utilization





Packet Drops



Future Work

Overheads in hardware programming.

- Each multipath table operation is typically slow (~100ms), forming the principal bottleneck in reliability.

Scalability and latency of the packet I/O path between OFC and OFA.

- OpenFlow might support two communication channels to separate high-priority operations from throughput-oriented operations.

Thanks

Q&A