Traffic-Engineering in Software Defined Networks Using Label Switching

在軟體定義網路下利用標籤交換之流量工程系統

Chen-Nien Mao

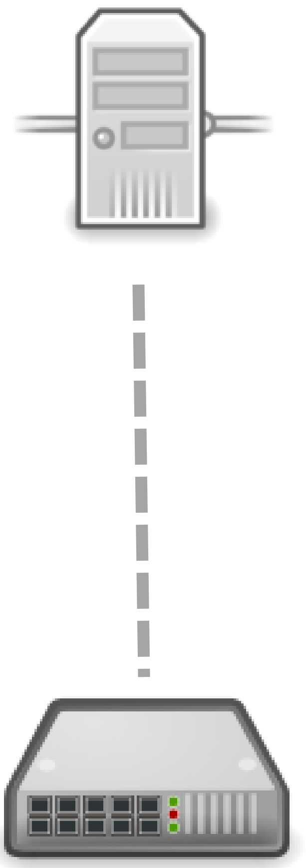
Advisor: Cheng-Hsin Hsu

Networking and Multimedia System Lab

CS Dept. , National Tsing Hua University

Outline

- Introduction
- Label Switching
- System Architecture
- Routing Mechanism
  - Tunnel Table Construction
  - Path Table Construction
  - Dynamic Modules
- Evaluation
- Conclusion

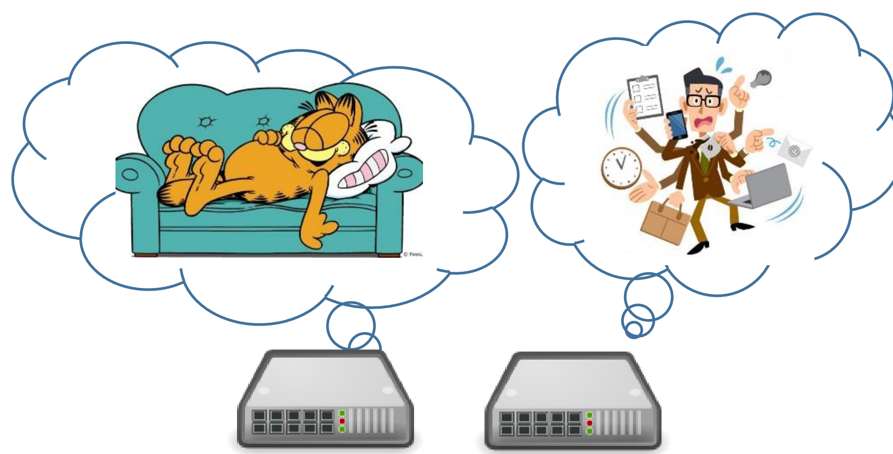


Introduction

Idle Resource in Networks

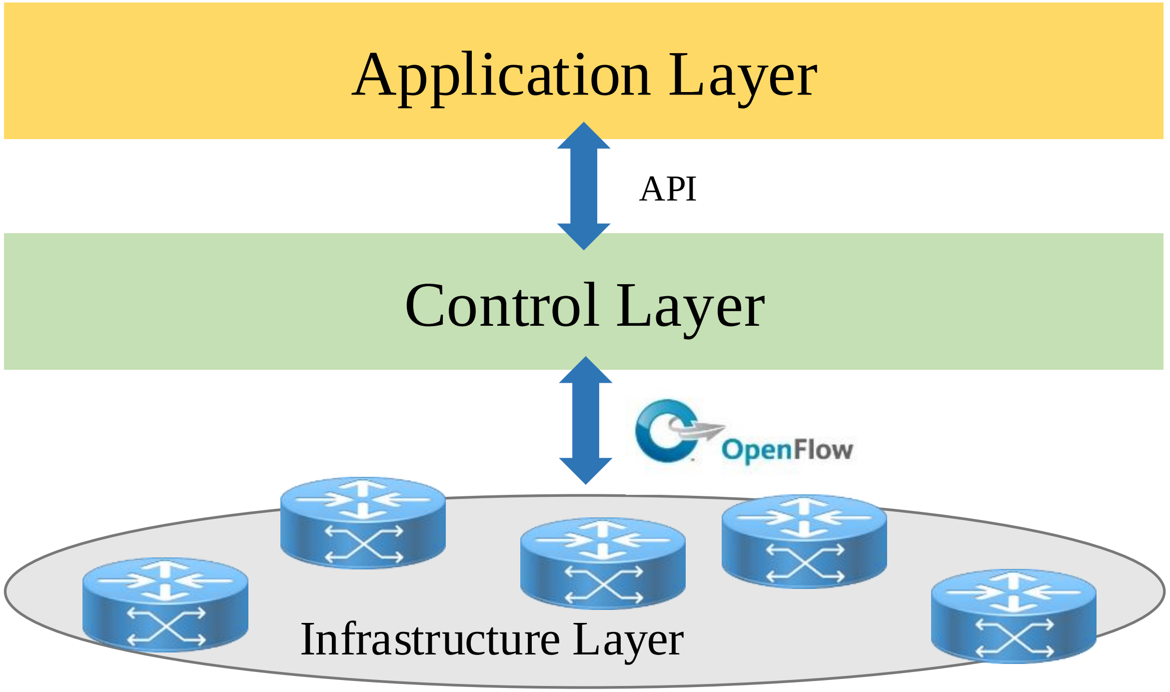
- Explosive traffic flow comes from different services has brought many challenges on
  - Quality of Service (QoS)
  - Quality of Experience (QoE)

- Traditional core networks perform Interior Gateway Protocol (IGP)
  - Shortest path routing
  - **Idling resources**



Software-Defined Networks (SDN)

- Decoupling the **control plan** & **data plan**
- The routing rules are decided by a central controller



Is SDN sufficient ?

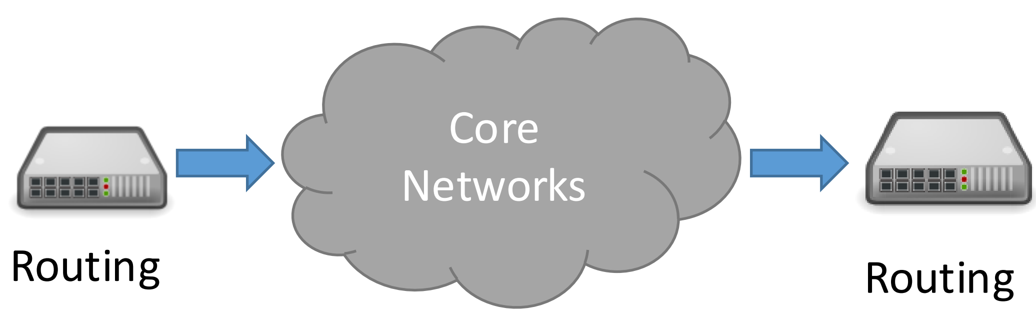
- Controller can change the routing behaviors
  => Are the problems solved by SDN?

Challenges:

- **Initialization** time in large networks
  - Core networks are vulnerable to delay
- **Scalability**
  - Decoupling the data flow and control flow brings the scalability issues
- **Flexibility** & **Efficiency**
  - The cost of changing routing behavior is high and complex

Contribution

- Propose label routing algorithms to solved the traffic engineering problems in SDNs
  - Minimize the initialization delay
  - Perform load balancing
  - Perform fast recovering
- Develop a flexible network architecture
  - Virtualize physical links with virtual paths
  - Simplify routing mechanisms inside the core networks



Label Switching

Example to illustrate label switching

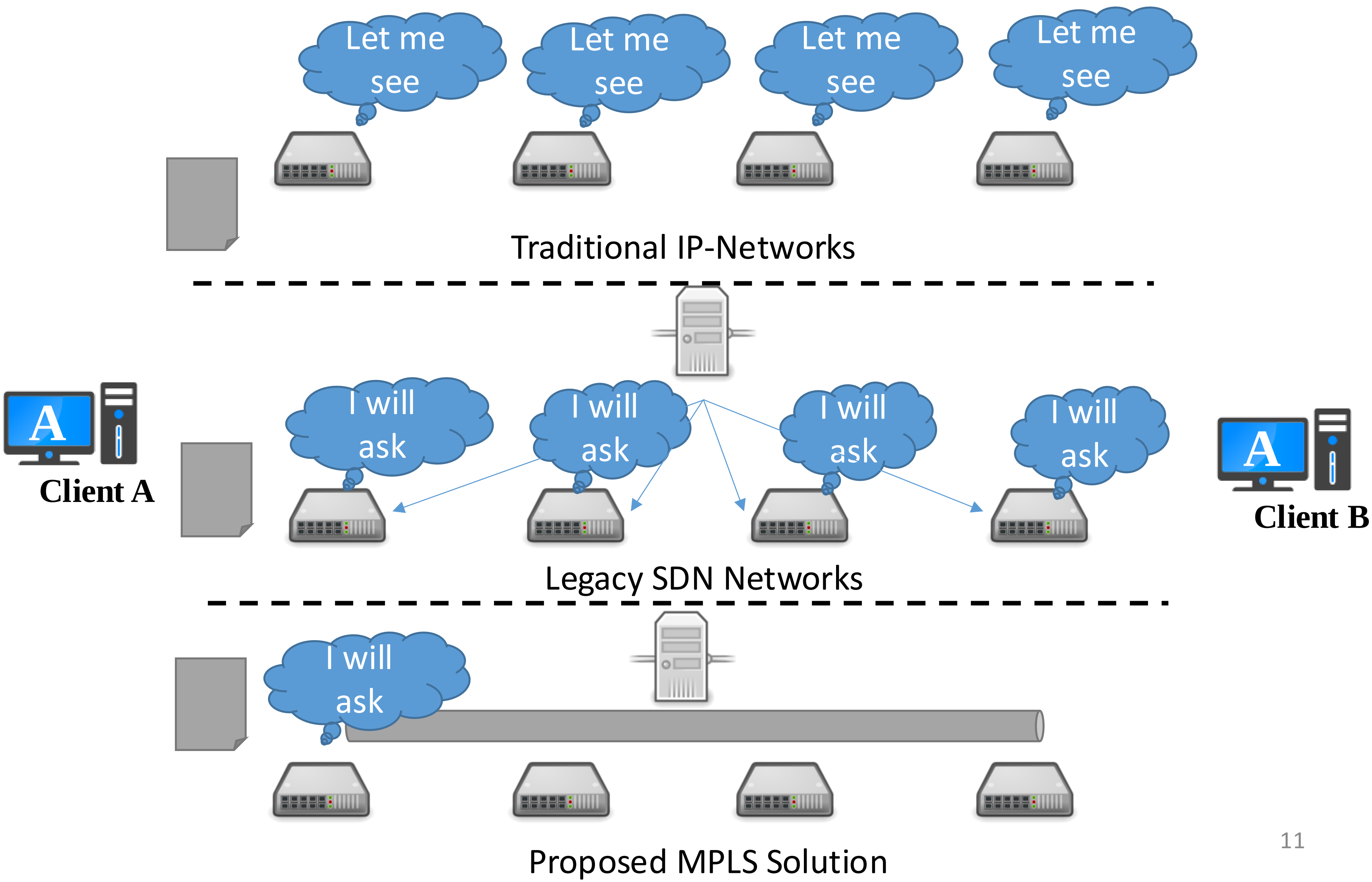
- Traveler wants to visit Taipei 101 from Taipei Main Station

- Passerby A : “I am not sure. Maybe you can go to next stop first”
- Passerby B : “I am not sure. Maybe you can go to next stop first” …..

Example to illustrate label switching

- If there is a smart guy who knows the best route.
- Smart guy : “Take the bus line 22, and you will arrive 101 without traffic gam”

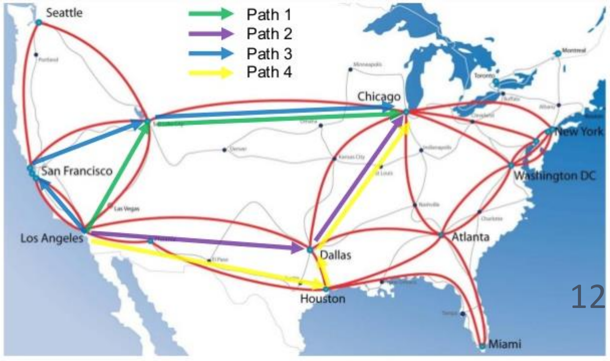
Difference between traditional approaches



Multiprotocol Label Switching

- Forwards the packets according to the label without looking up the network address
- MPLS Label
  - Label Distribution Protocol
  - Stackable, providing higher extensibility,
  - Fixed length, allowing more efficient matching
  - Supported in OpenFlow protocol

- MPLS shows its strength on Traffic Engineering in legacy IP network
  - Resource Reservation Protocol
  - Reserve bandwidth for QoS



MPLS in SDN

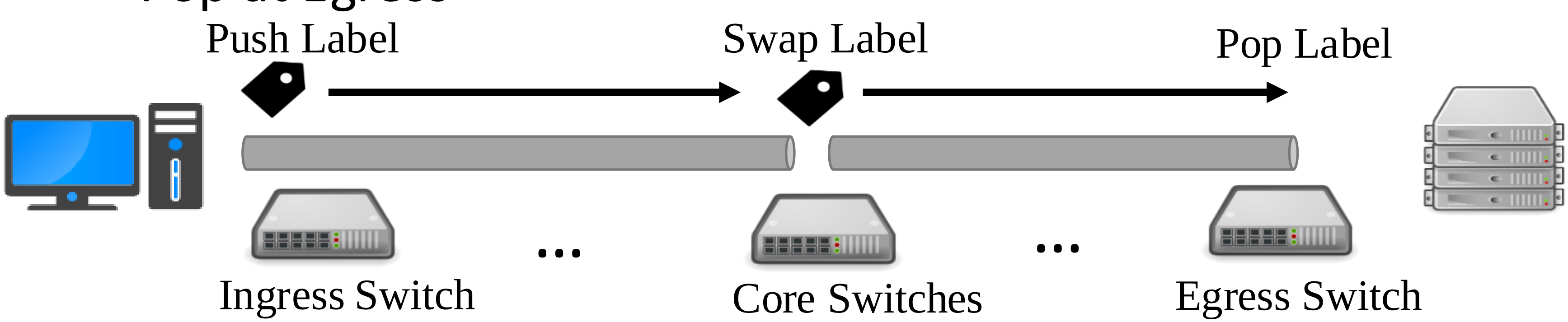
- Difficulties of performing MPLS in traditional network
  - Scope of the whole system
  - Hierarchy of MPLS system
  - Path attributes of label switching paths

- Some issues can be solved in SDNs
  - Global view of the system => Optimized the Path selection
  - Ability to coordinate each switches => Assign Label & handle hierarchy

We can build a our system without lower level protocols !

Label Switching

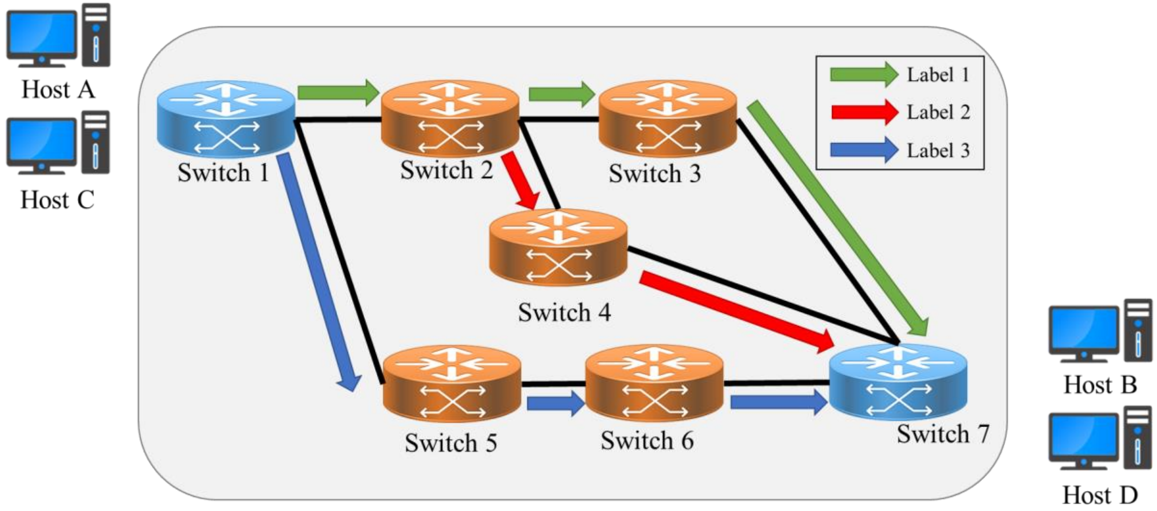
- MPLS labels
  - Extract MPLS protocols
  - Represent a virtual tunnel (multiple physical links)
- Routing actions in proposed system:
  - Push at Ingress
  - Swap at Medium
  - Pop at Egress



Benefits of using Label Switching

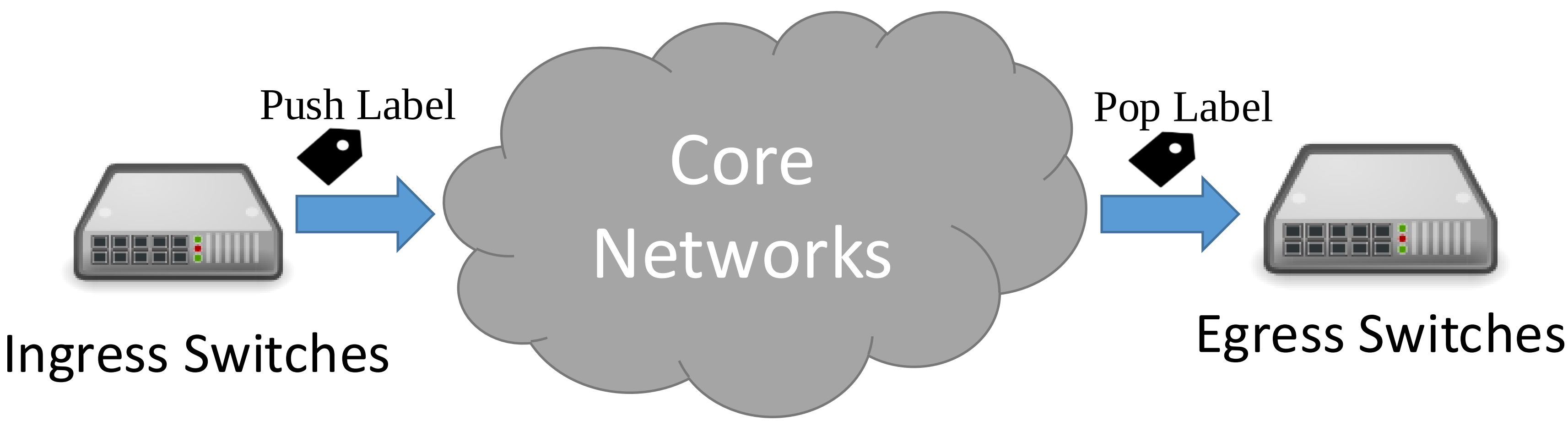
- Perform Traffic engineering easily
  - Labels => routing decision
  - Changing routing behaviors by changing labels

- Balance traffic by "switching label"

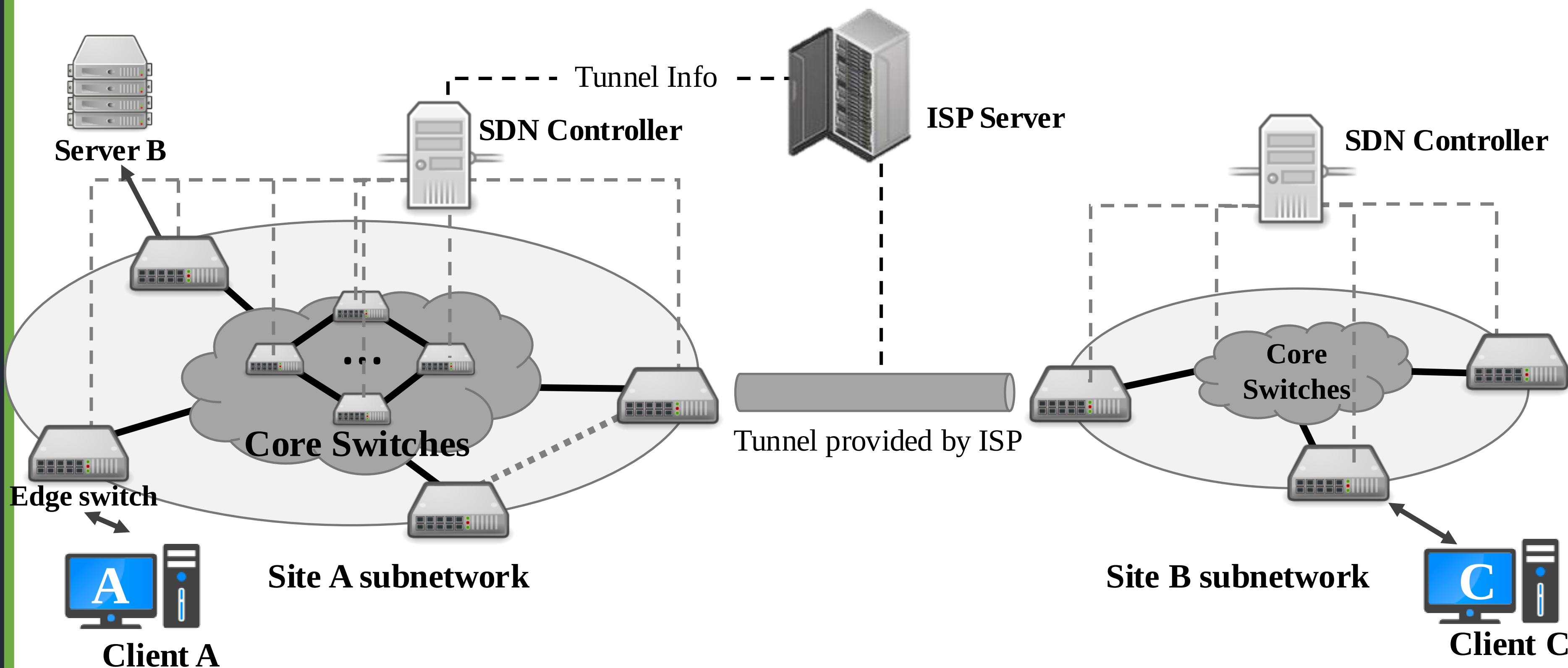


Routing inside the core networks

- Simplify routing mechanisms inside core networks
  - Complex routing decisions are made in edge switches
  - Switches in core network only focus on packet-forwarding



Use case



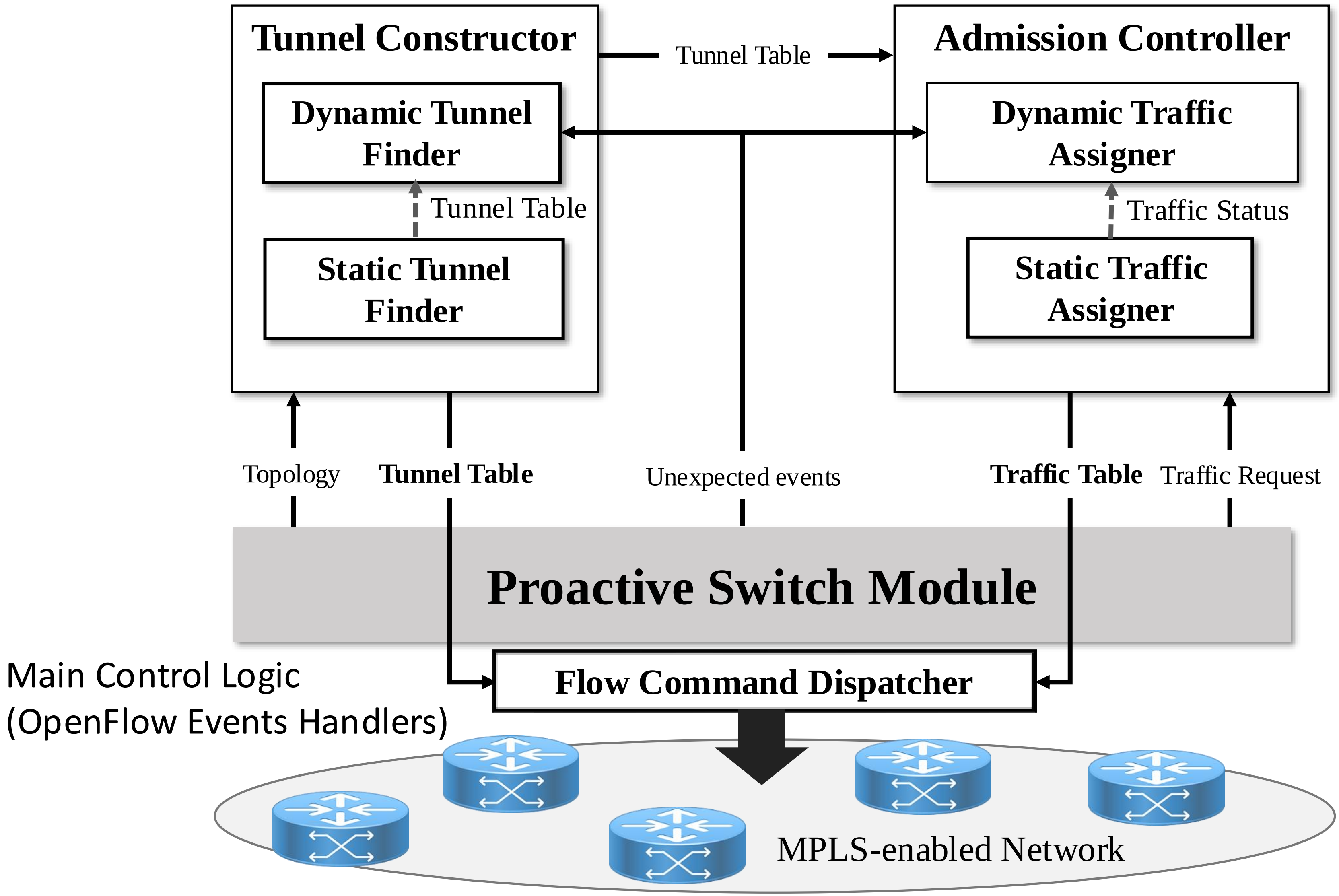
Tunnel Info
ISP Server
Server B
SDN Controller
SDN Controller
Core Switches
Core
Switches
Edge switch
Tunnel provided by ISP
Site A subnetwork
Site B subnetwork
A
C
Client A
Client C

System Architecture

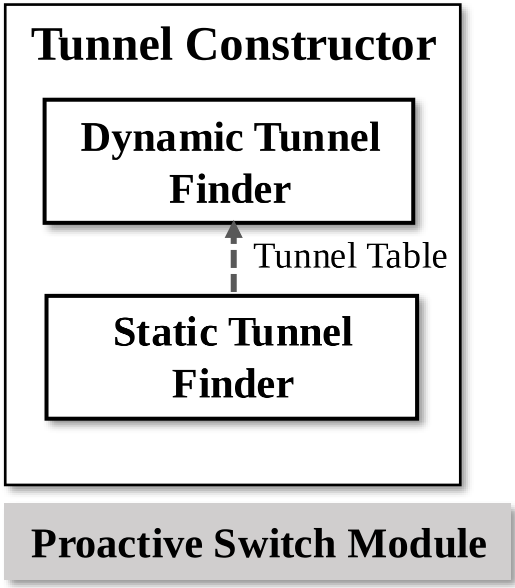
Problem statement

- Initialization Delay
  - Use pre-build tunnels
  - Avoid congestion and packet loss
- Load Balancing
  - Offload the traffic to idling links
- Error Resilience
  - Fast-rerouting
  - Dynamic traffic assigner

Controller Components

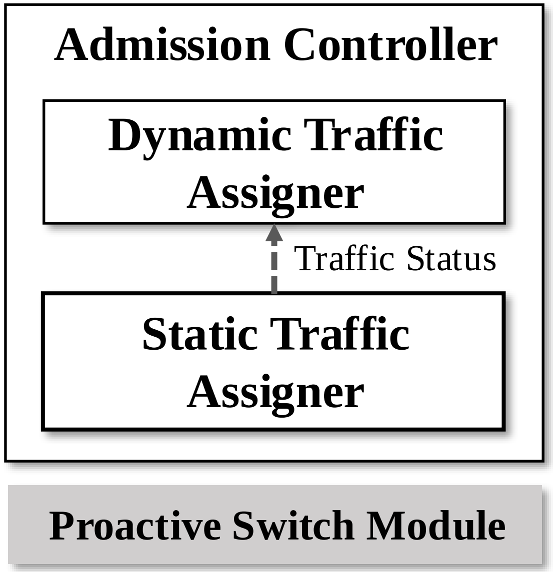


Tunnel Constructor



- Construct Tunnels inside SDN domain
- Static Tunnel Finder (**STF**)
  - Find tunnels among every two nodes
  - Pre-built tunnels in networks
    (System setup or topology change)
- Dynamic Tunnel Finder (**DTF**)
  - Take link usage into consideration
  - Recover tunnels
  - Connect new edge switches to the network

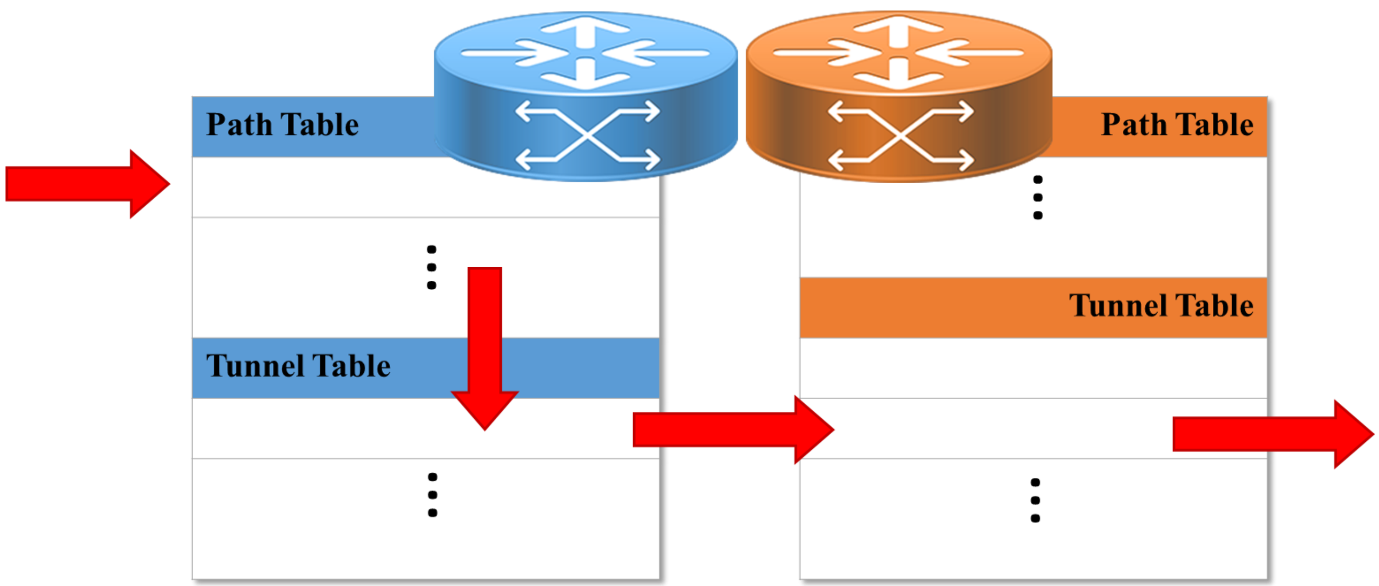
Admission Controller



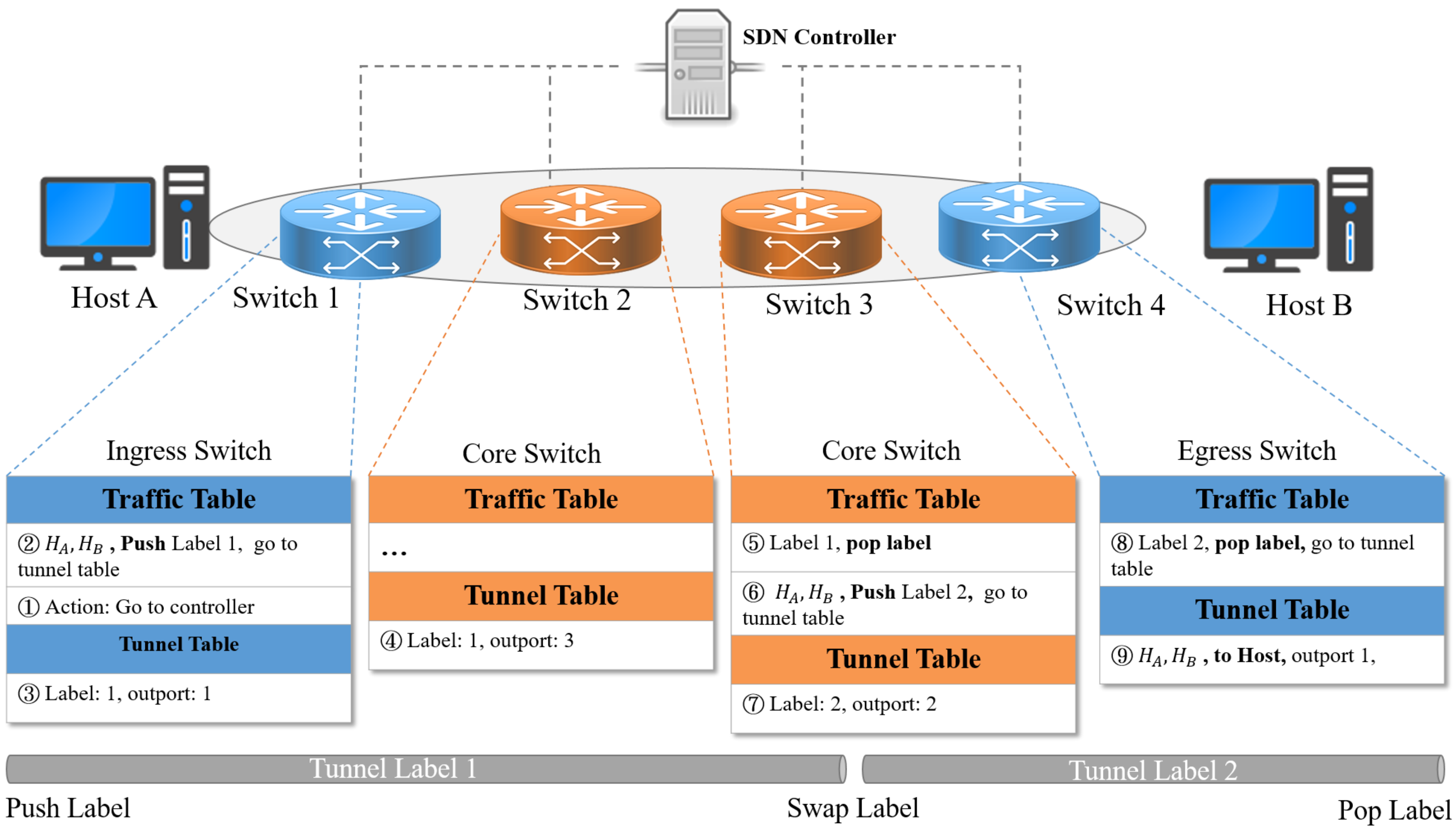
- Allocate traffic into the system
- Dynamic Path Assigner (**DPA**)
  - Real-time traffic assigner (Label Tagger)
  - Handle new traffic request
  - Handle unexpected traffic re-route
- Static Path Assigner (**SPA**)
  - Load Balancer
  - Consider Link utilization & Perform load balance
  - Avoid congestion

Decoupled Flow Tables

- Tunnel table (Lower Table)
  - Store pre-built tunnels information (label info.)
- Path table (Upper Table)
  - Store the bindings between labels (tunnels) and the traffic flows

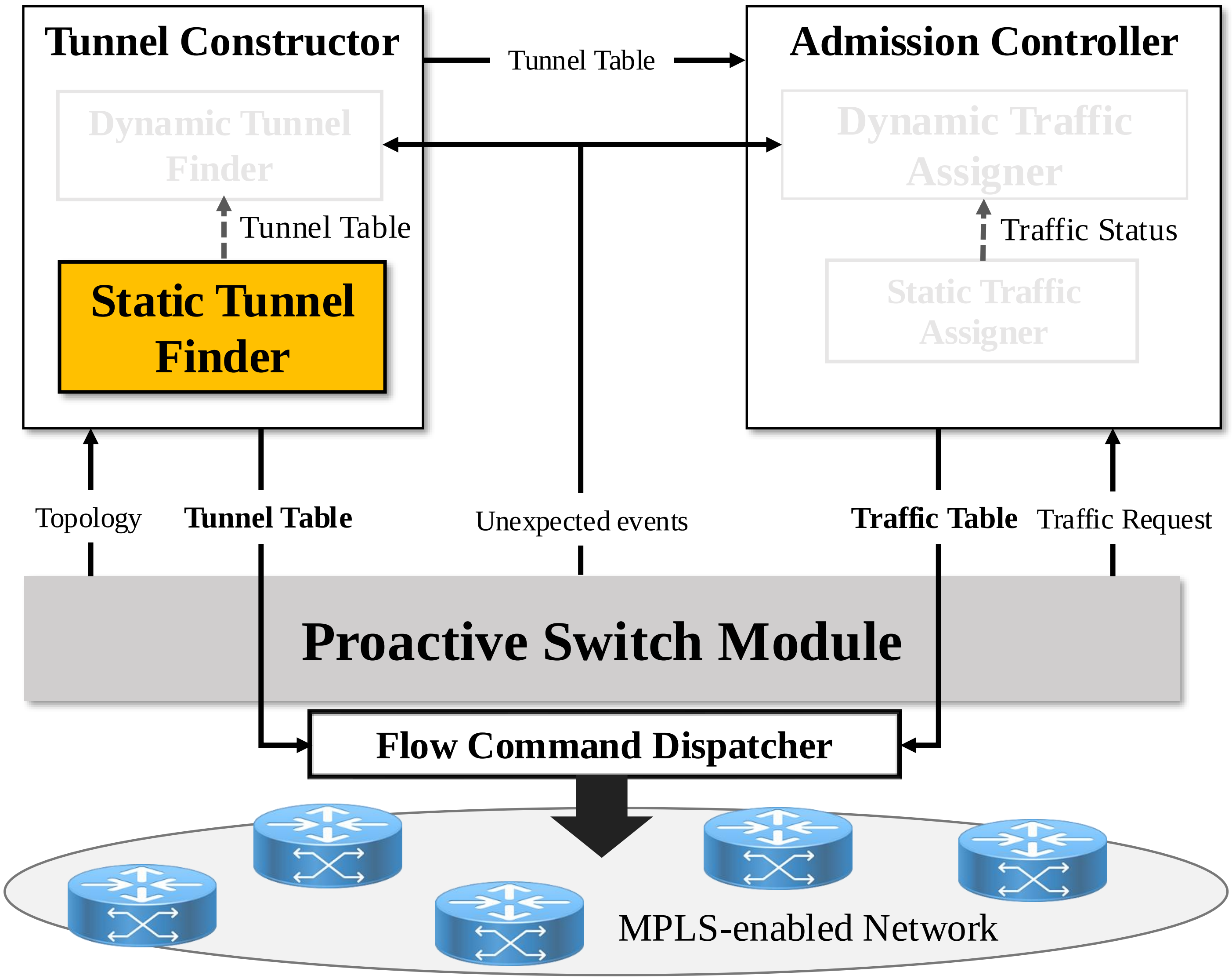


Packet Forwarding



Routing Mechanism

Controller Components



Tunnel Table Problem Formulation

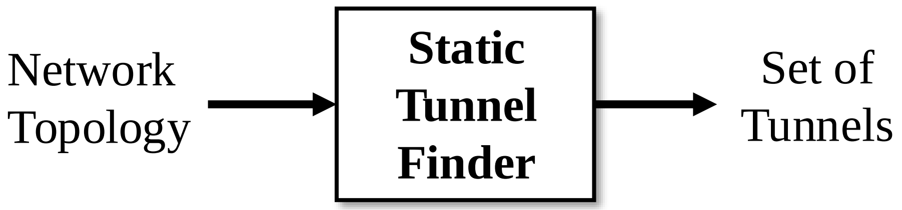
- Goal: to find mutually disjoint tunnels between each switch pairs
  - Maximize the available bandwidth among each switch pairs
  - Reliable and Flexible

Constraint:

Length of tunnels

- $stretch\ facotr = \frac{\sum_{l=1}^{L} m_{p,l}}{shortest\ path\ hop}$
- $k = stretch\ facotr * shortest\ path\ hop$

Path Table



Maximize bandwidth of each paths

$$\text{maximize} \sum_{p=1}^{P} B(p) \qquad (6.1a)$$

Path capacity is less than or equal to the minimum link share

$$\text{s.t.} B(p) m_{p,l} \leq \frac{c(l)}{(\sum_{p'=1}^{P} y_{p'} m_{p',l})}, \forall l \in L, \forall p \in P; \qquad (6.1b)$$

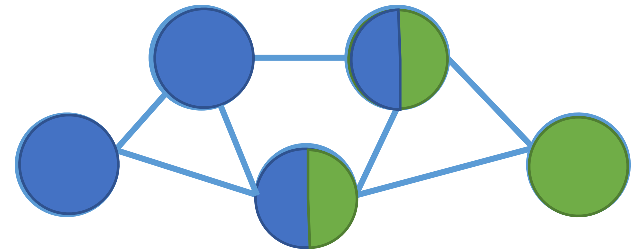
$$B(p) \leq BottleNeck_p, \forall p \in P; \qquad (6.1c)$$

Minimal link capacity among all the links

$$\sum_{l=1}^{L} B(p) m_{t,l} \leq k, \forall p \in P \qquad (6.1d)$$

Length of selected tunnel will never exceed k

Heuristic Algorithm for Tunnels finding



```
function DisjointFinder(Src, Dst, Adj_matrix)
    Prefix_tunnels //List of sub-tunnel list from source node
    Suffix_tunnels //List of sub-tunnel list to destination node
    length //current target length
    disjoint_ans //Final ans which is the optimal set
    while length ≤ k do
        Break if neither Prefix_tunnels nor Suffix_tunnels are not able to increase
        //skip the checking until shortest path reach
        if length ≥ SP_hop then //Check interaction
            for each sub_tunnel_A in Prefix_tunnels do
                for each sub_tunnel_B in Suffix_tunnels do
                    if sub_tunnel_A[-1] == sub_tunnel_B[0] then
                        disjoint_ans.append(sub_tunnel_A + sub_tunnel_B)
                        Adj_matrix(sub_tunnel_A[−1], n) = 0, ∀n ∈ N
                        Adj_matrix(n, sub_tunnel_B[0]) = 0, ∀n ∈ N
        //Need to prepare Prefix_tunnels for next round
        if ceil((length + 1)/2) > (length + 1)/2 then
            for each sub_tunnel in Prefix_tunnels do
                if Adj_matrix(sub_tunnel[−1], n), ∀n ∈ N then
                    sub_tunnel.append(n)
        //Need to prepare Suffix_tunnels for next round
        if floor((length + 1)/2) == (length + 1)/2 then
            for each sub_tunnel in Suffix_tunnels do
                if Adj_matrix(n, sub_tunnel[−1]), ∀n ∈ N then
                    sub_tunnel.prepend(n)
    return(disjoint_ans)
```

Check Intersection

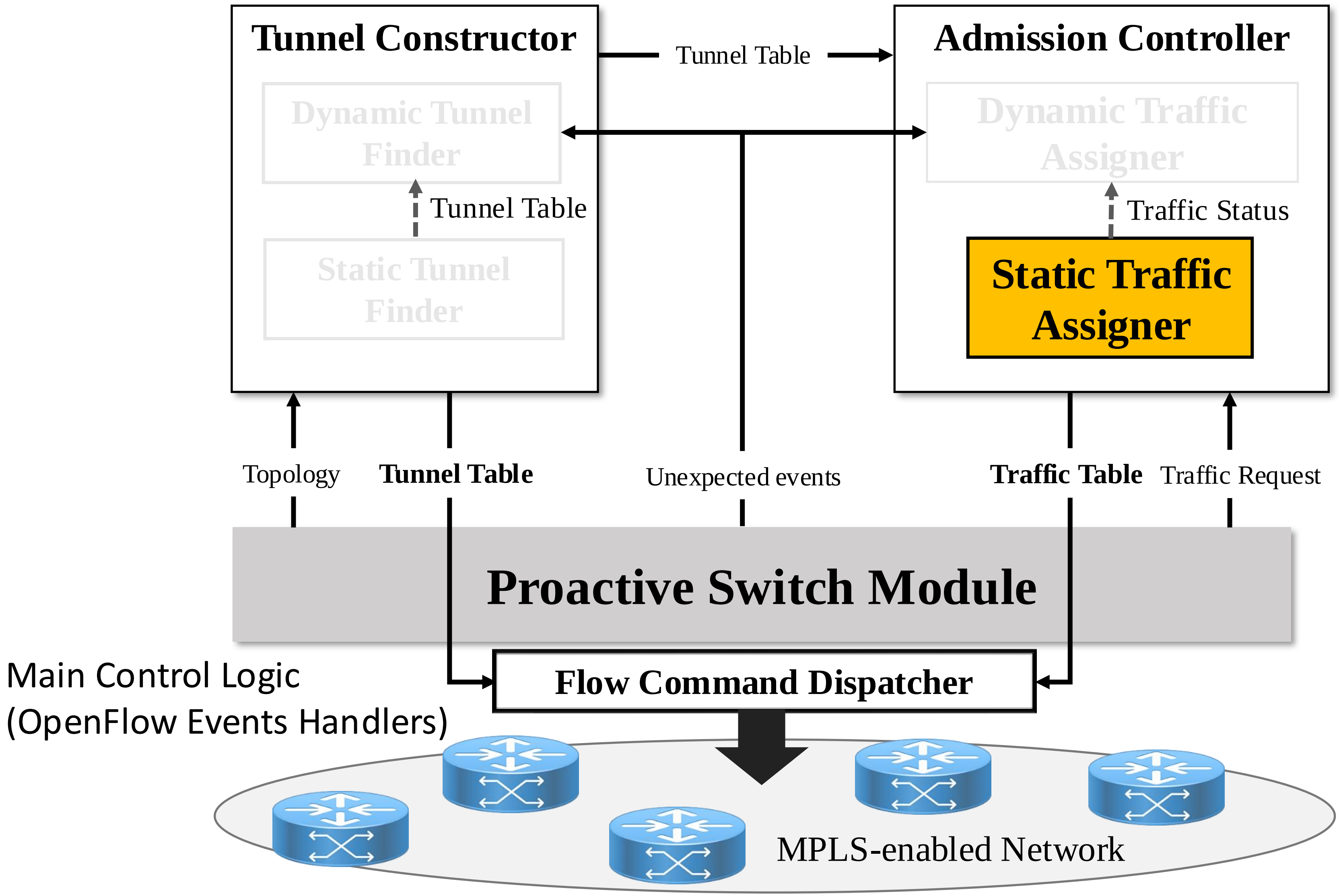
Update Adjacency Matrix

Move start indicator forward

Move end indicator backward

D. Torrieri. “Algorithms for finding an optimal set of short disjoint paths in a communication network”. IEEE Transactions on Communications, 1992.

Controller Components



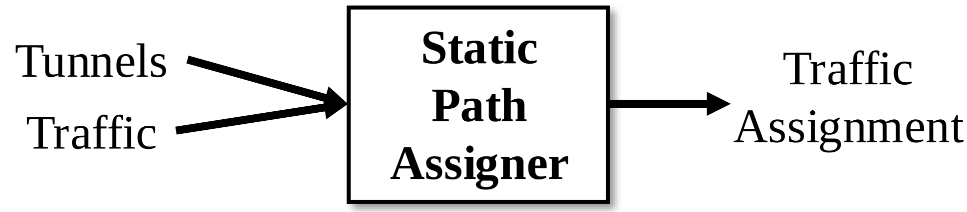
Path Assigner

- Goal: Minimize the links utilization
  - Balance the traffic load inside the system
  - We assume that all the traffic can be handled by current tunnels (Admission Control)

- Find a suitable path for each traffic flows

  $x_{t,f}$, whether tunnel is assigned to traffic flow

Load balancer



Minimize the maximal tunnel utilization

$$\text{minimize} \max_{1 \le l \le L} \sum_{t=1}^{T} \sum_{f=1}^{F} \boxed{x_{t,f}} m_{t,l} b_f / c(l) \tag{7.1a}$$

Traffic flows always reach their destination

$$\text{s.t.} \sum_{n \in S'(\theta_f)} \sum_{t \in T} w_{\theta_f,n,t} x_{t,f} - \sum_{n \in S'(\theta_f)} \sum_{t \in T} w_{n,\theta_f,t} x_{t,f} = 1, \forall f \in F; \tag{7.1b}$$

Traffic flows will not leave from the middle nodes

$$\sum_{n \in S'(e)} \sum_{t \in T} w_{e,n,t} x_{t,f} - \sum_{n \in S'(e)} \sum_{t \in T} w_{n,e,t} x_{t,f} = 0, \quad \forall e \in S'(\theta_f, \eta_f), \forall f \in F; \tag{7.1c}$$

$$\sum_{n \in S'(\eta_f)} \sum_{t \in T} w_{\eta_f,n,t} x_{t,f} - \sum_{n \in S'(\eta_f)} \sum_{t \in T} w_{n,\eta_f,t} x_{t,f} = -1, \forall f \in F; \tag{7.1d}$$

Traffic flows never leave its destination

$$\sum_{t=1}^{T} \sum_{f=1}^{F} x_{t,f} m_{t,l} b_f \le c(l) \; \forall l \in L; \tag{7.1e}$$

Traffic over a link does not exceed its capacity

$$x_{t,f} \in \{0,1\}, \; 1 \le t \le T, 1 \le f \le F. \tag{7.1f}$$

$$m_{t,l} \in \{0,1\}, \; 1 \le l \le L, 1 \le t \le T. \tag{7.1g}$$

$$w_{i,j,t} \in \{0,1\}, \; 1 \le i,j \le S, 1 \le t \le T. \tag{7.1h}$$

How to find best utilization?

- Path Finding:
  - Find a shortest path to carry the traffic (set of tunnels)
  - Constrained BFS

If the tunnel can accept traffic, and its total utilization doesn't exceed α

$$\sum_{t=1}^{T}\sum_{f=1}^{F} x_{t,f} m_{t,l} b_f \leq c(l)\alpha, \forall l \in L$$

$$0 \leq \alpha \leq 1$$

- A flow can only flow through tunnel when:

$$C_l + b_f \leq c(l)\alpha$$

- How to find best α elgently?
  - Adopt Binary Search

Heuristic Algorithm for Paths finding

Algorithm 2 Static Traffic Assigner (SPA).

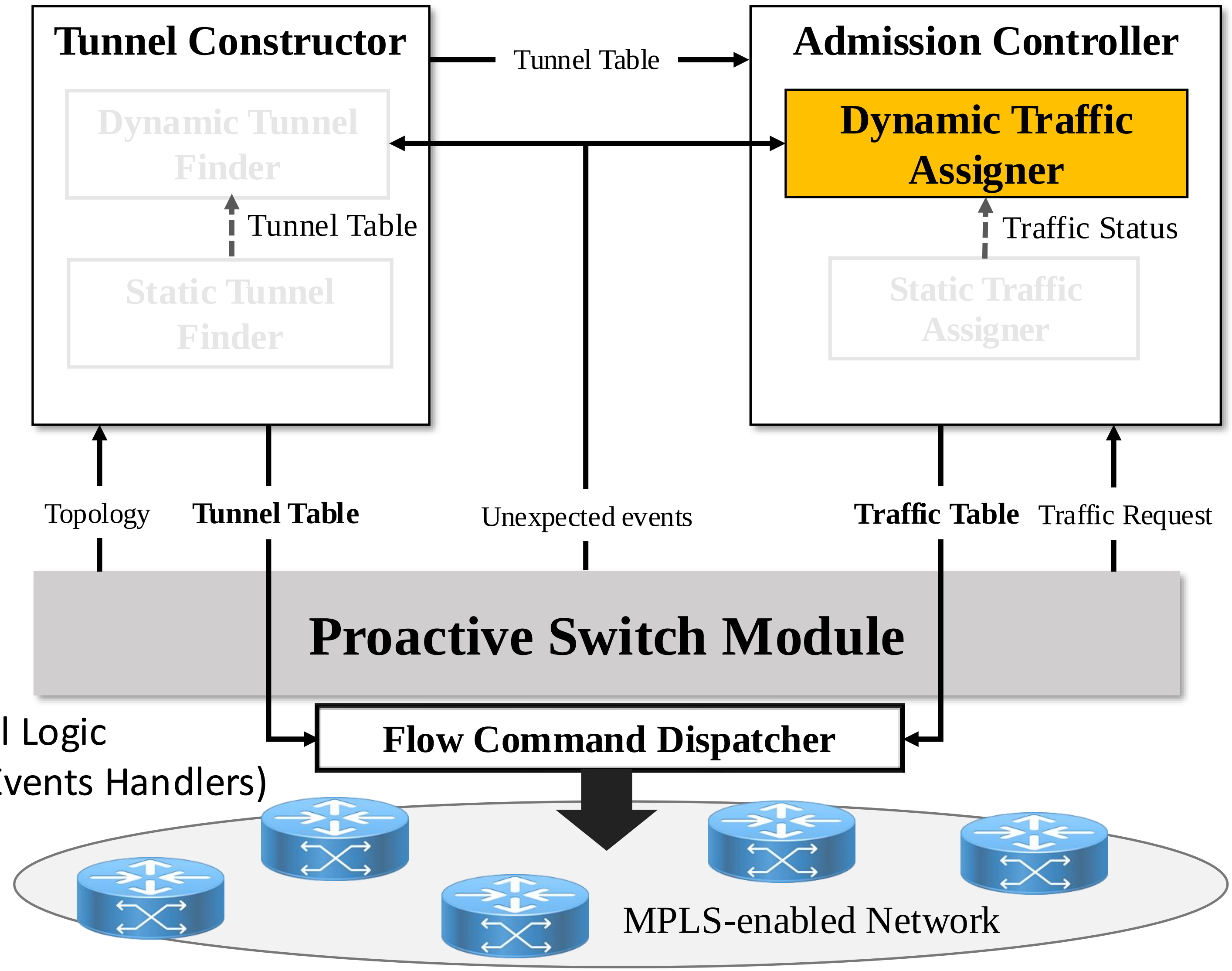
$Upper = 1.0$, $Lower = 0$ //The upper/lower bounds
Final_Assignment $= \phi$ //The final answer
Sort traffic flow **F** by bandwidth b_f in desc. order
$\alpha = \phi$ // Utilization
while $Upper - Lower < threshold$ **do**
  $\alpha' = (Upper + Lower)/2$
  CurrentAssignment $= \phi$
  for each flow f in **F do**
    $path =$ **ConstrainedBFS**$(\theta_f, \eta_f, b_f, \alpha')$
    if path $= \phi$ **then**
      $\mathbf{A}' \leftarrow \phi$
      Break
    else
      update available link bandwidth
      CurrentAssignment.$append$(**path**)
  if CurrentAssignment $= \phi$ **then**
    $Lower = \alpha'$
  else
    $Upper = \alpha'$
    $\alpha = \alpha'$
    $\mathbf{A} \leftarrow \mathbf{A}'$
if α is not defined, return no answer

Binary search

Assign traffic into system

Adjust α
$$\begin{cases} \alpha = \dfrac{\alpha + lower}{2}, success \\ \alpha = \dfrac{upper + \alpha}{2}, failed \end{cases}$$

Controller Components



Main Control Logic
(OpenFlow Events Handlers)

Dynamic Path Assigner

- Determine the routing path in real time
  - New traffic request
  - Unexpected Events -> traffic flows need to be re-allocated
- Decide whether the network can handle traffic
  - Quick response to minimize delay
  - Leave the utilization optimizing to SPA

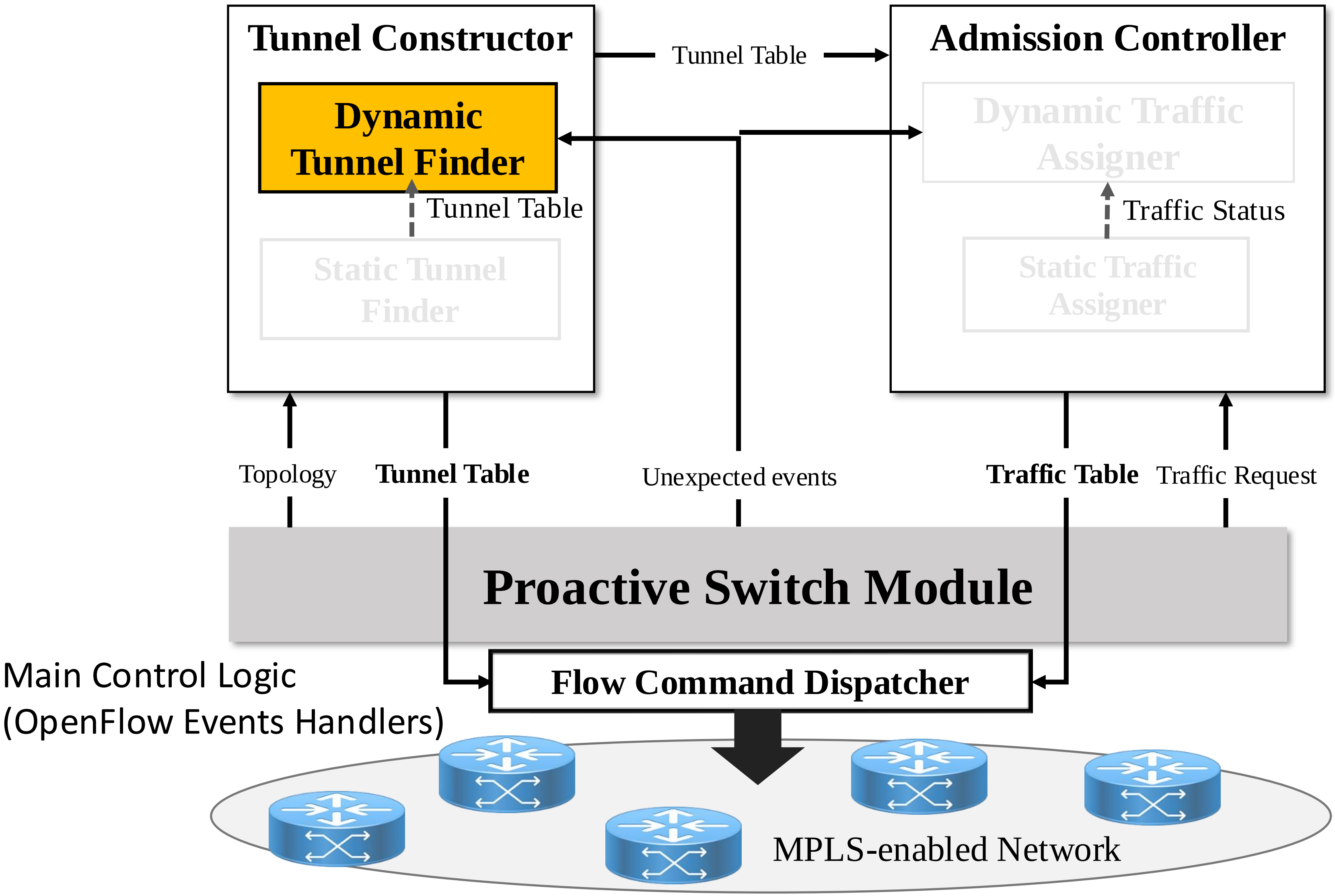
Find a path in real-time

- Goal: find sufficient path to fit the traffic
  - Using the same BFS module in Static Path Assigner

Algorithm 4 Dynamic Path Assigner (DPA)

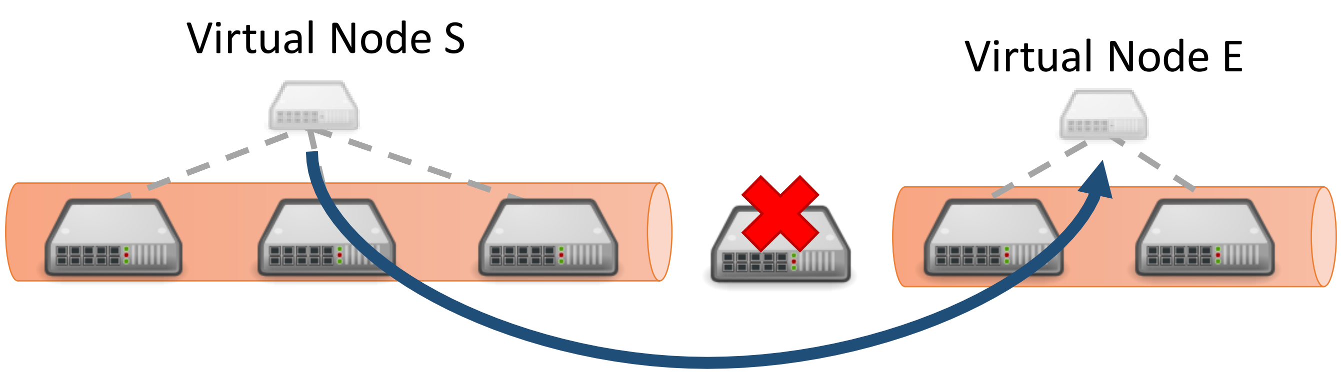
F //set of flows need to be assign
Sort traffic flow **F** by bandwidth b_f in desc. order
while F is not empty **do**
    $\mathbf{A} \leftarrow \phi$ //The final answer
    $\mathbf{A} = ConstrainedBFS(\theta_f, \eta_f, b_f)$
    if $\mathbf{A} = \phi$ **then**
        //Needs to reallocate traffic
        $TriggerSPA()$;
    else
        // Update utilization
        $UpdateFlow()$; //OpenFlow commands

Controller Components



Switch Dynamic

- Goal: to find a tunnel with lowest utilization
  - Consider link utilization => govern more traffic
- Connect New edge switches to the network
  - Find lowest utilization tunnels to all the nodes
- Recover failed tunnels
  - Create virtual nodes => connect two tunnels with overlapping points



Algorithm 3 Dynamic Tunnel Finder (DTF)

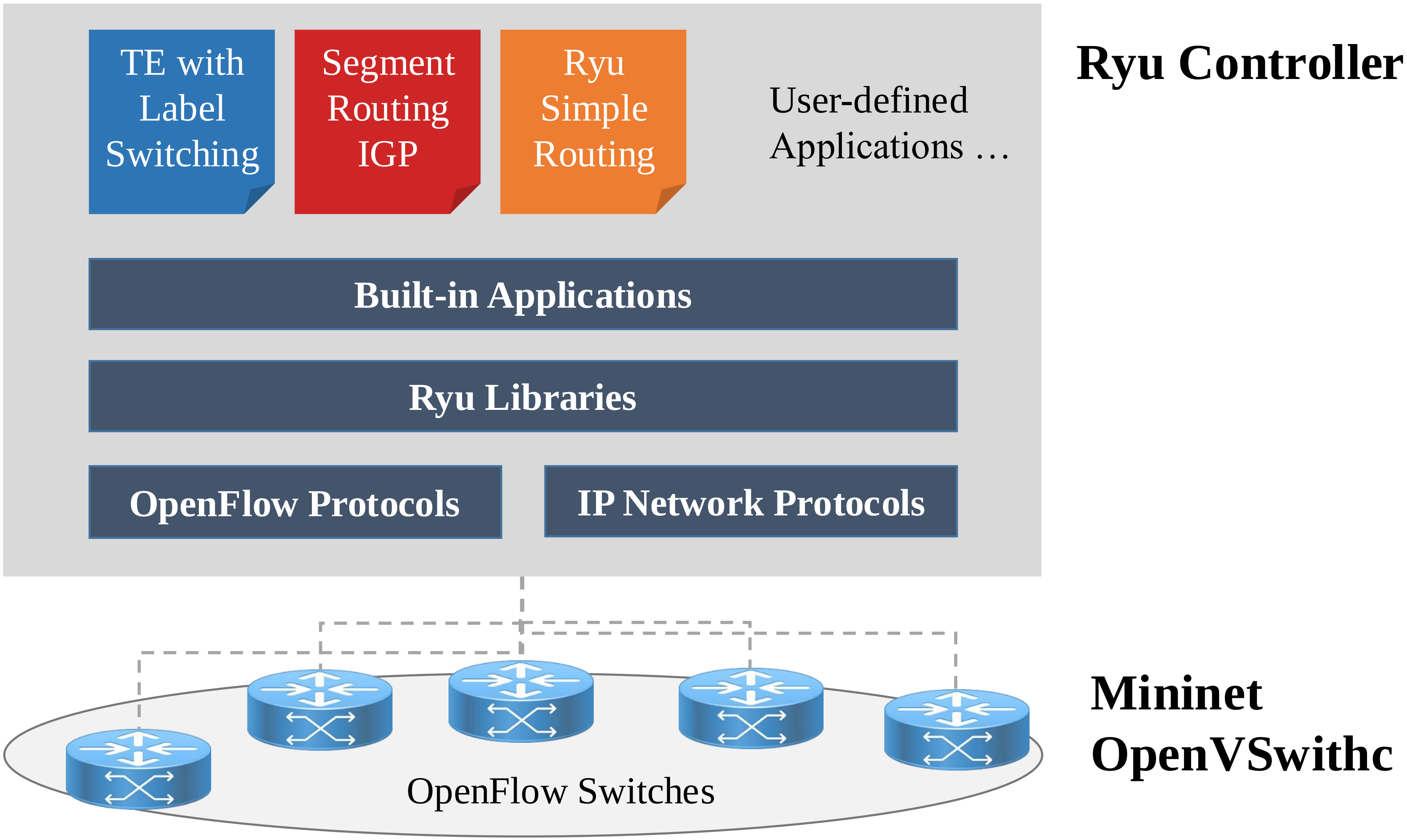
M //the map of the given topology
S //set of start nodes
E //set of end nodes
U //utilization of each links as edge weight
function DisjointFinder(S,E,U)
    s //A node which connect to all the nodes in S
    e //A node which connect to all the nodes in E
    for each v in M **do**
        $\mathbf{Util}[\mathbf{v}] \leftarrow \infty$
        $\mathbf{Prev}[\mathbf{v}] \leftarrow \phi$
        $\mathbf{queue}(\mathbf{v})$
    $\mathbf{Util}[\mathbf{s}] = 0$
    while $queue \neq \phi$ **do**
        $n \leftarrow$ node with lowest utilization
        $\mathbf{dequeue}(\mathbf{n})$
        for each neighbor v of n **do**
            if lower utilization is found **then**
                $\mathbf{Util}[\mathbf{v}] = \mathbf{Util}[\mathbf{v}] + \mathbf{utilization}(\mathbf{n}, \mathbf{v})$
                $\mathbf{Prev}[\mathbf{v}] = n$
        if n is e **then**
            $path = \mathbf{convertToPath}(\mathbf{Prev})$
            $\mathbf{return}\ path$
            break

Dijkistra to find path

- Weight <- Current link util.

Evaluation

Implementation



Application in RYU

- Traffic Engineering with Label Switching (TEL)
  - Optimize traffic by proposed algorithms
  - Proactively install disjoint paths in system
- Segment Routing with IGP (SRI)
  - Cisco Pathman[1]
  - Shortest Path Routing
  - Proactively install paths in system
  - Choose path with lowest link usage
- Ryu Simple Routing (RSR)
  - Spanning Tree Protocols

[1] OpenDaylight Pathman SR App from cisco. https://github.com/CiscoDevNet/pathman-sr.

Experiment setup

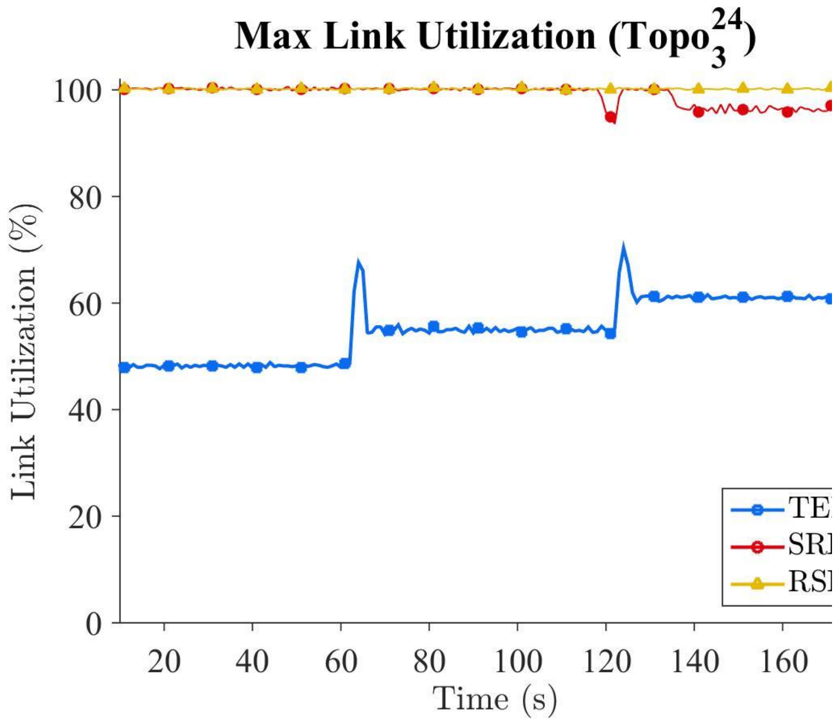
- Mininet network emulator
  - OpenVSwich
  - Real traffic flows generated by iPerf
- 4 different sizes, each size has 5 different topologies
- Conduct experiments for 10 times
  - UDP packets

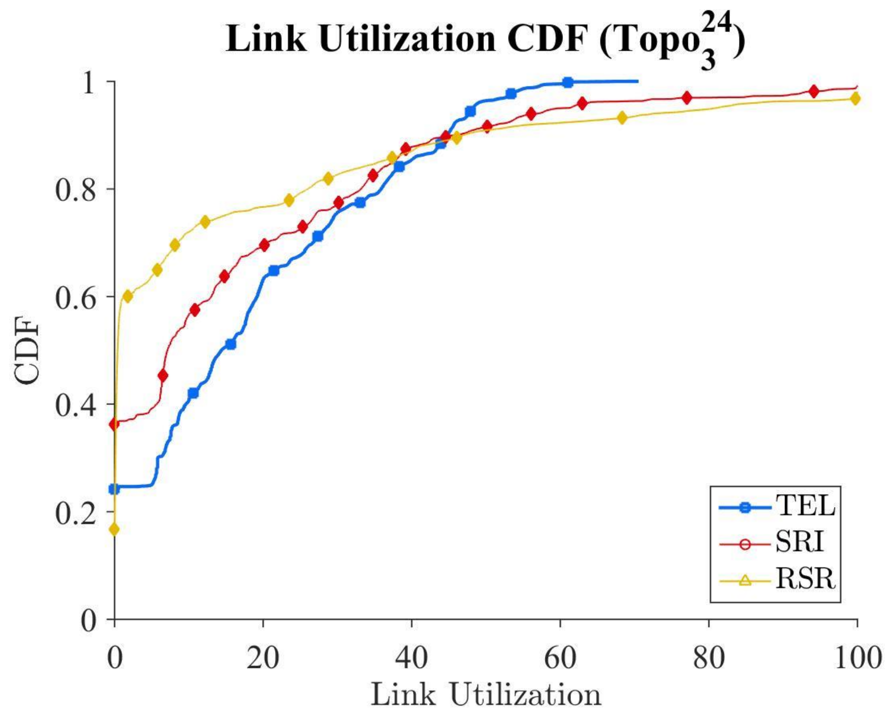
Table 10.2: Bandwidth Range of each topologies

Topo	Min. Bandwidth	Max. Bandwidth	Avg. Request/sec	Bandwidth (Mbps/#req.)
$Topo_i^8$	10 Mbps	100 Mbps	22.33	53.56
$Topo_i^{16}$	10 Mbps	100 Mbps	27.33	103.49
$Topo_i^{24}$	50 Mbps	100 Mbps	50.33	73.70
$Topo_i^{32}$	10 Mbps	100 Mbps	67.00	64.77

TEL optimizes the max. link utilization

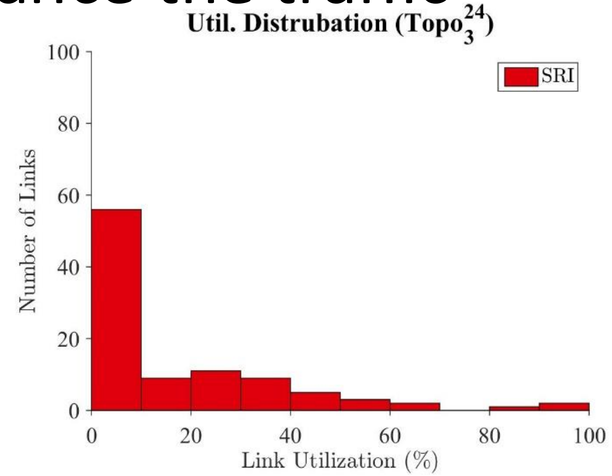
- TEL optimize the traffic
  - TEL always achieve the lowest max. link utilization
  - 90% of links have less than 50% link usage

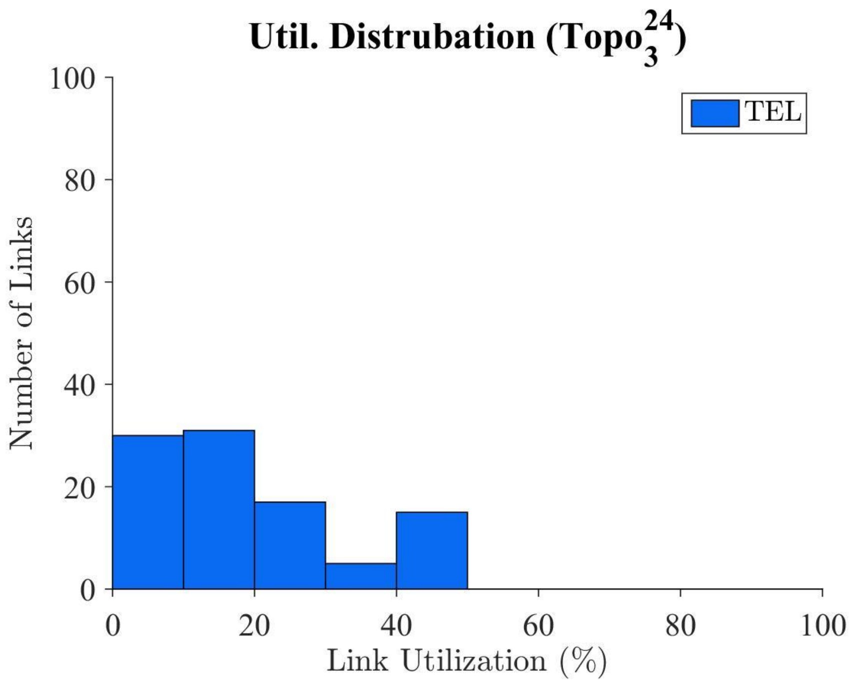


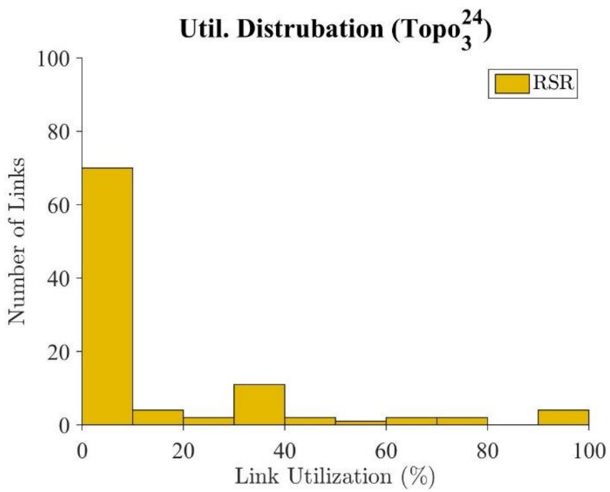


Utilize Idle resources

- TEL uses idle resources to balance the traffic
  - More links are used
  - Minimize utilization



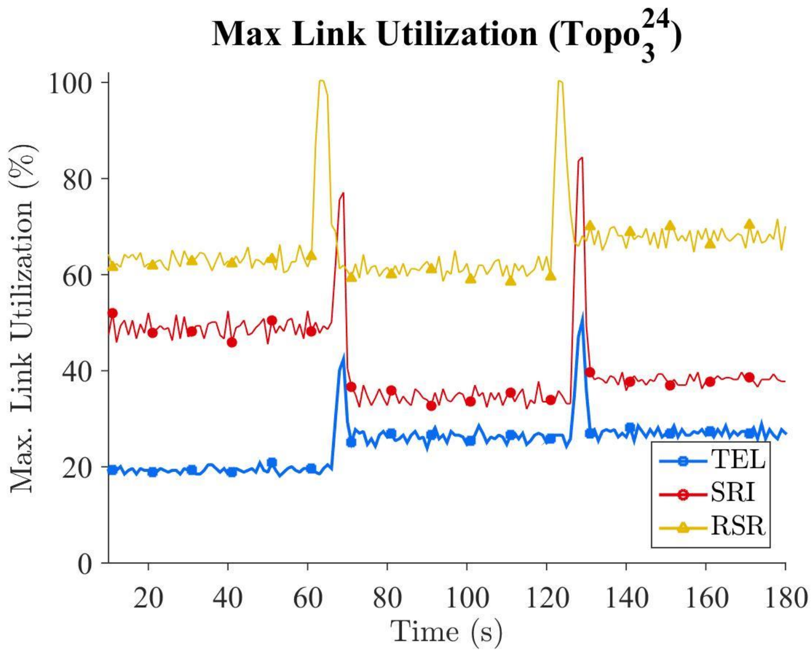




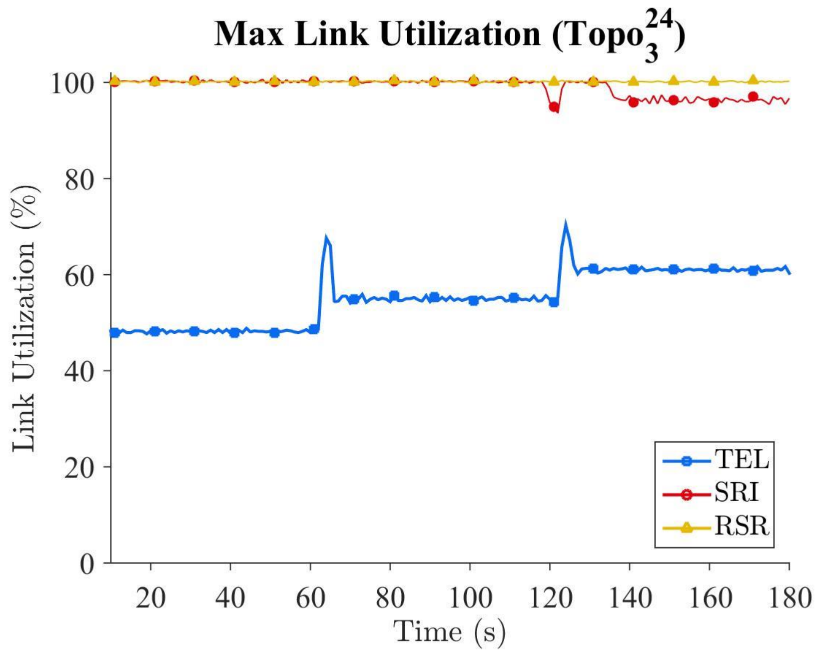
Heavy and Light traffic

- TEL is able to minimize the traffic in both heavy traffic & light traffic

Light traffic : 28.96 Mbps

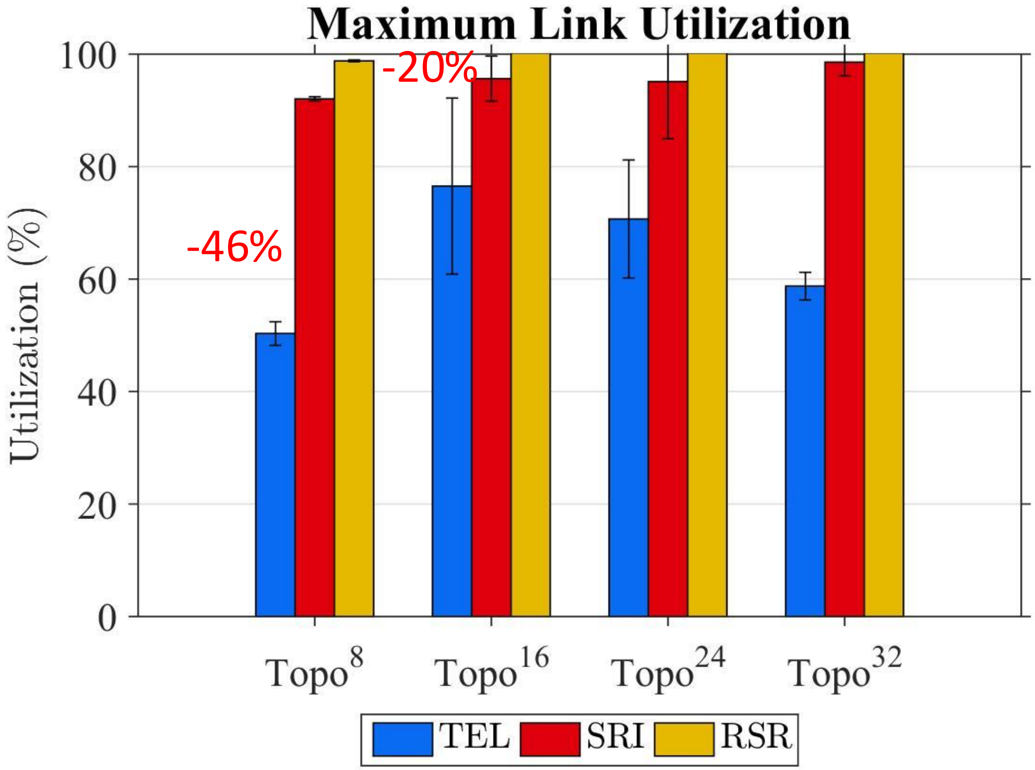


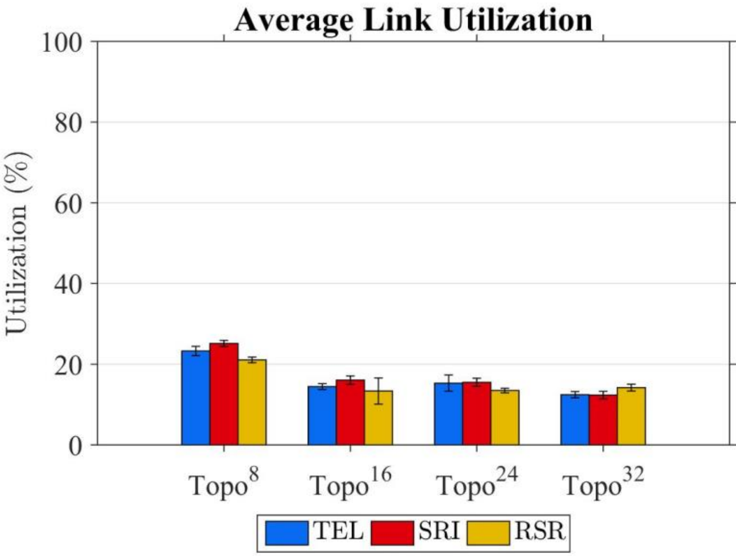
Heavy traffic : 73.70 Mbps



TEL balances the traffic in different topologies

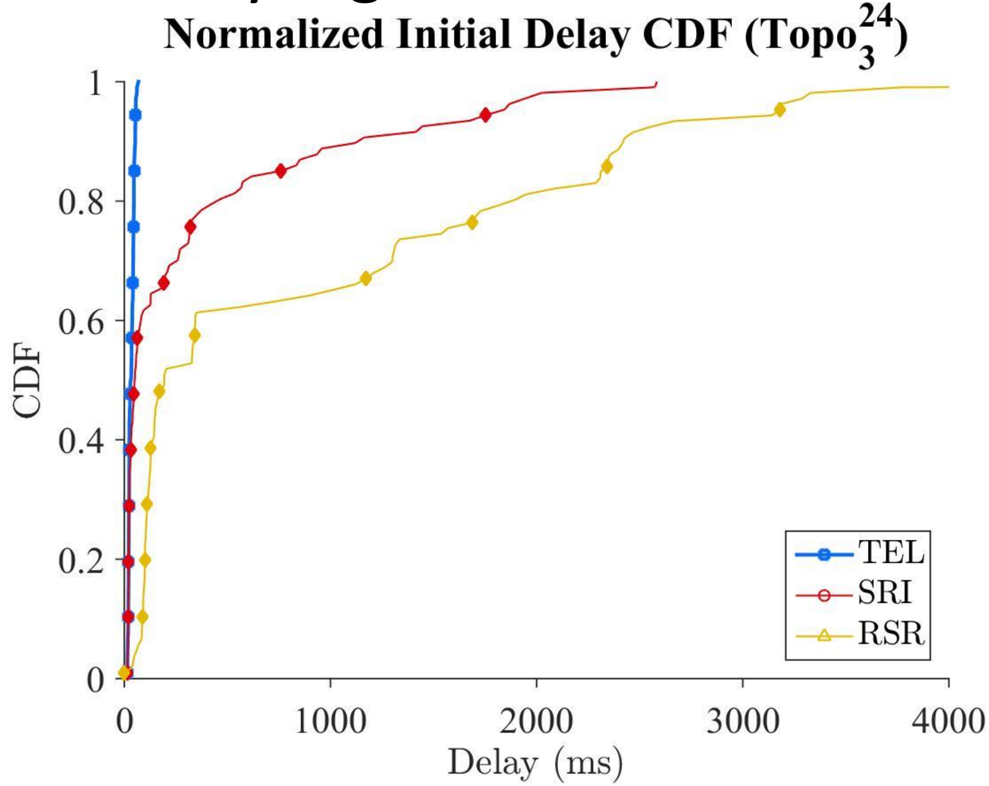
- TEL reduces maximal link utilization over
  - SRI between **20% ~ 46%**
  - RSR between **30% ~ 50%**





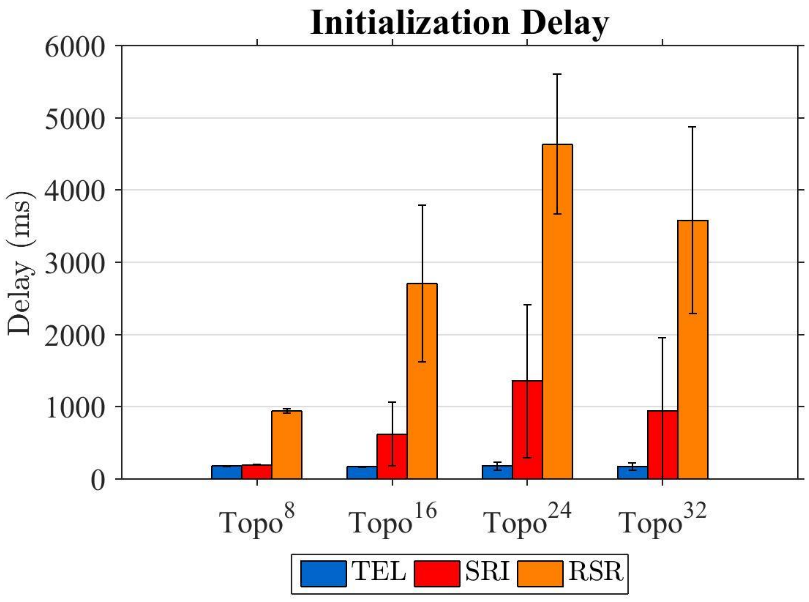
TEL achieves the lowest Init. delay

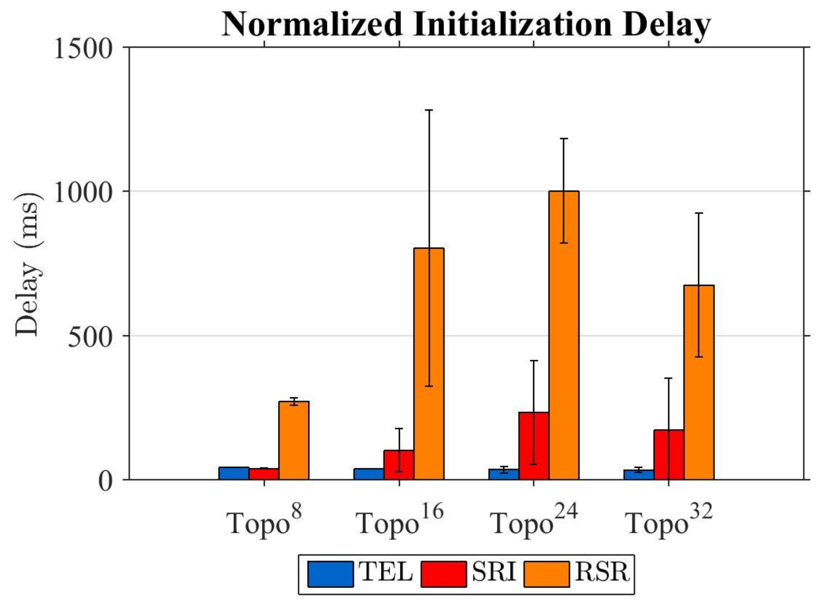
- TEL achieves lowest initialization delay
- Both SRI and RSR suffer from congestion, and the delay is relatively high



Congestion increases the delay

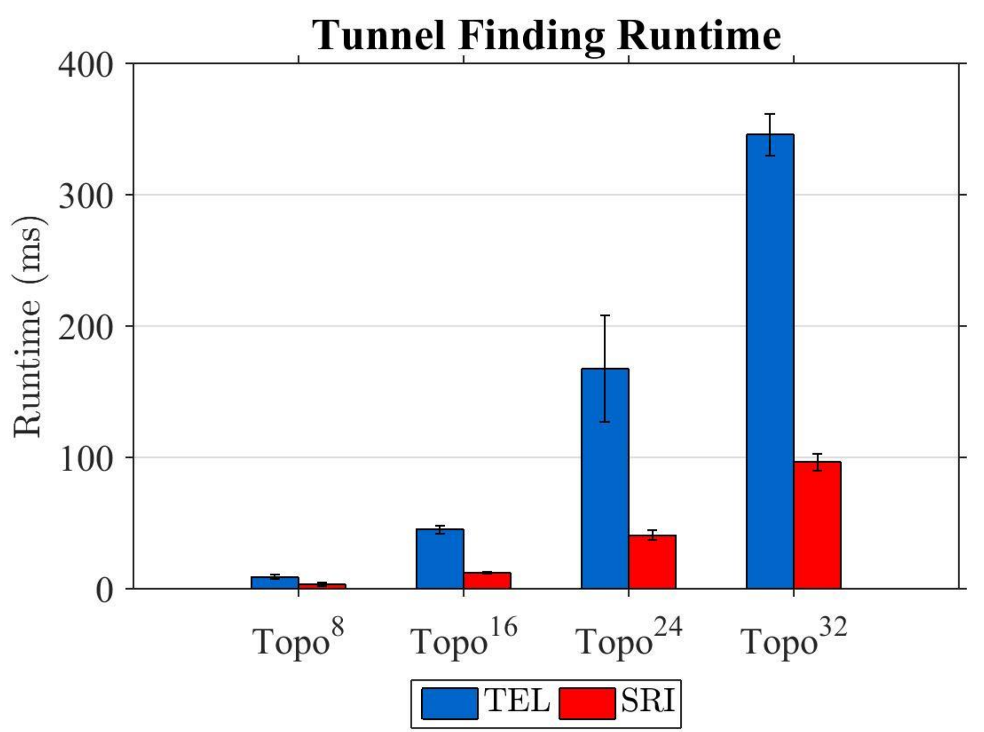
- Initialization delay is reduced by **39.02%** and **93.22%** compared to the SRI and RSR
  - Delay increases along the size
  - Heavy congestion in $Topo^{24}$ (9% of links suffer from congestion) 3-5% others





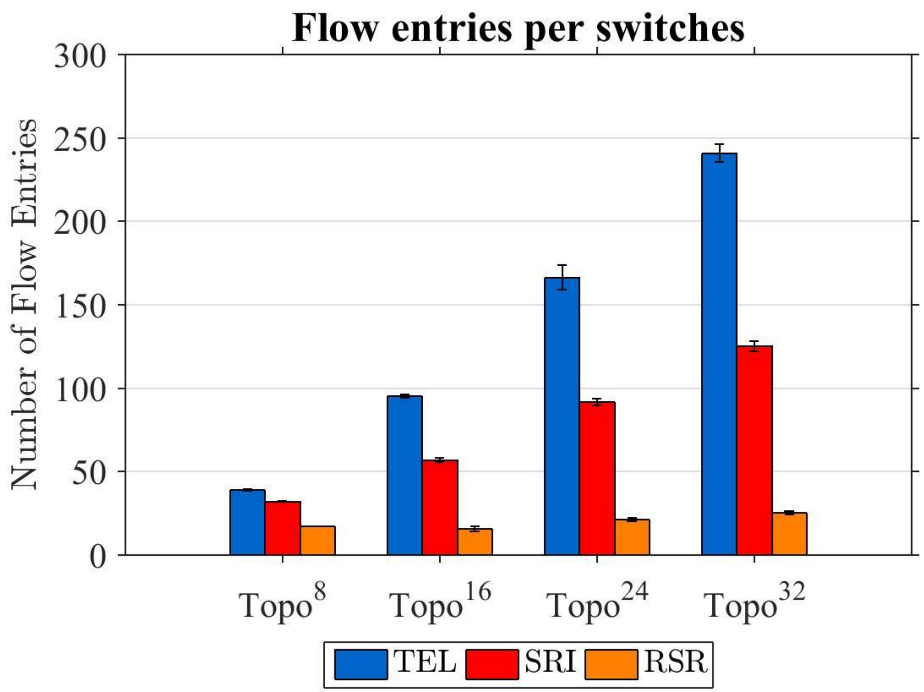
TEL takes more time on pre-built tunnels

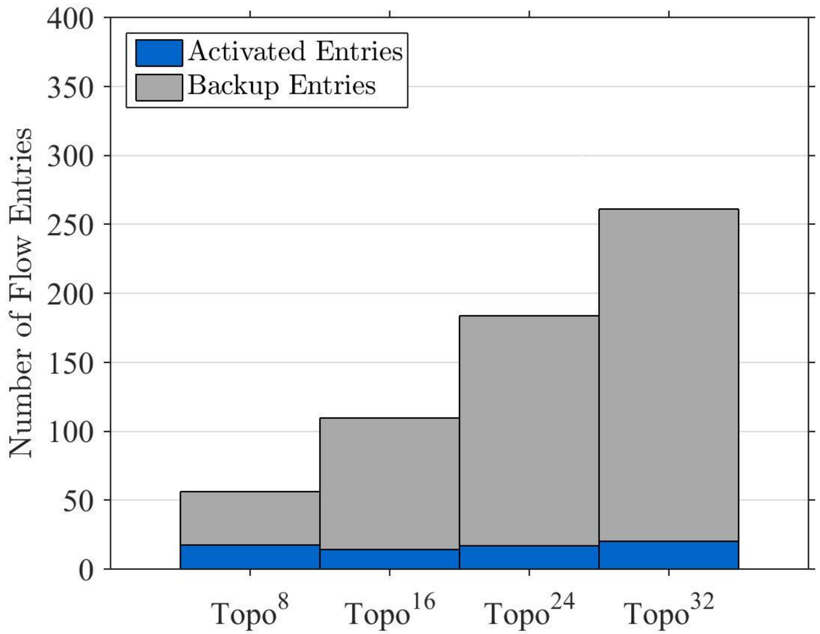
- We don’t need to compute the tunnels constantly
  - The state changes only if the topology changes



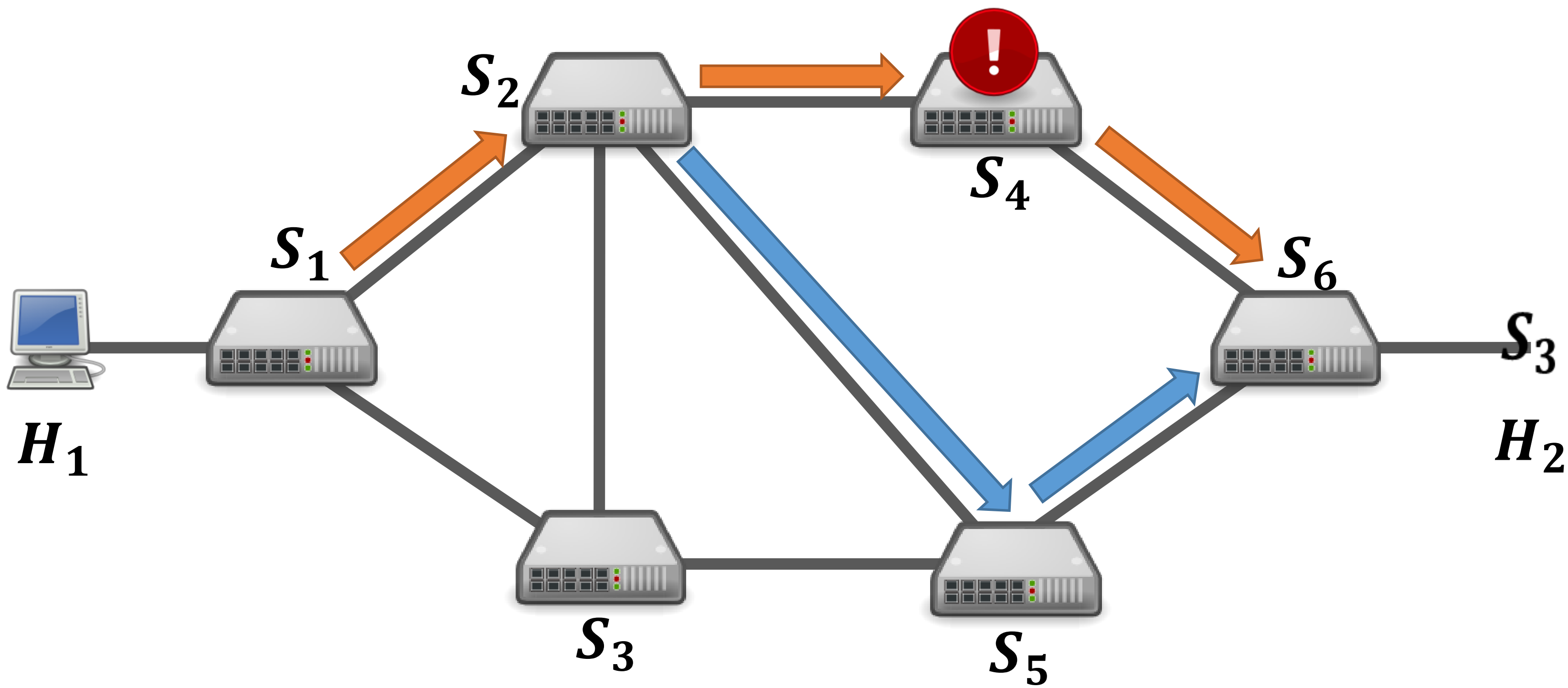
TEL consumes more flow entries

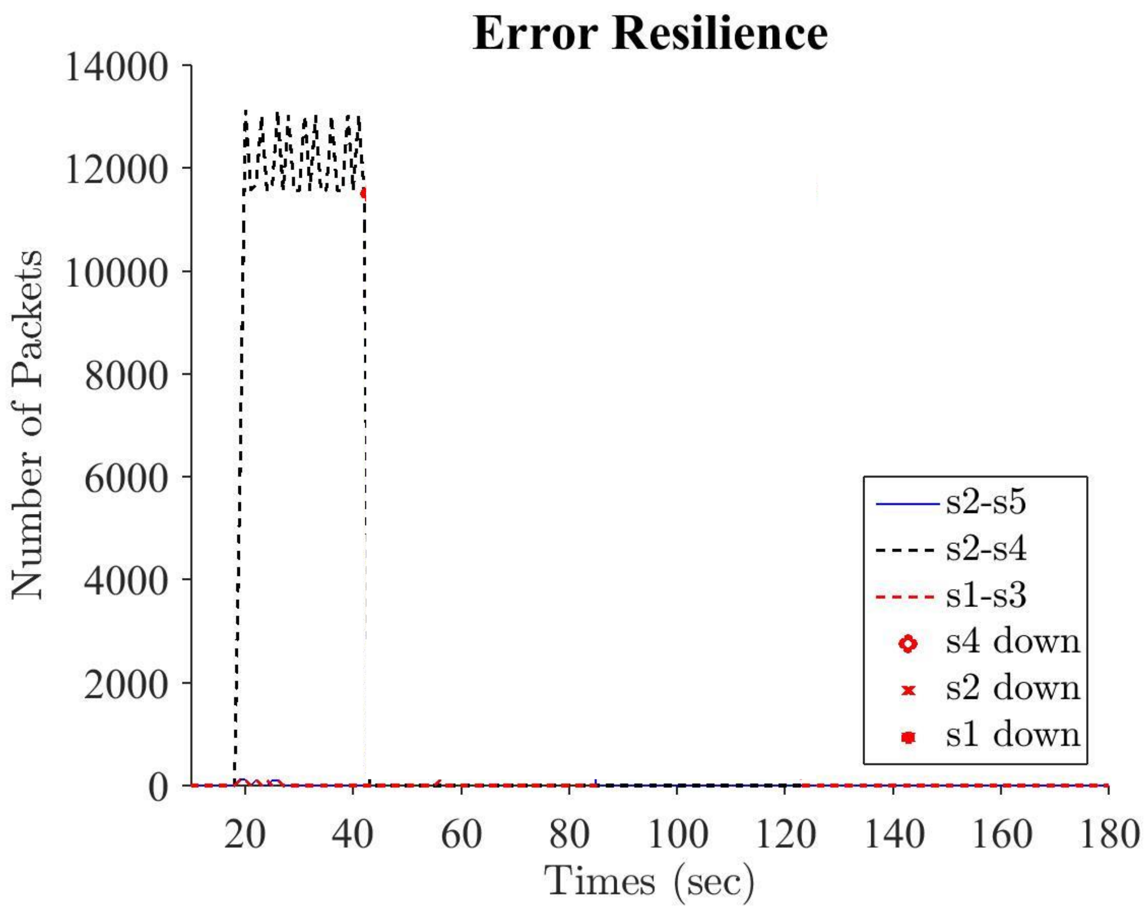
- TEL consume more flow entries
  - Reductant flow rules => quick response, error resilience
  - Tunnels can be further optimized (Future works)





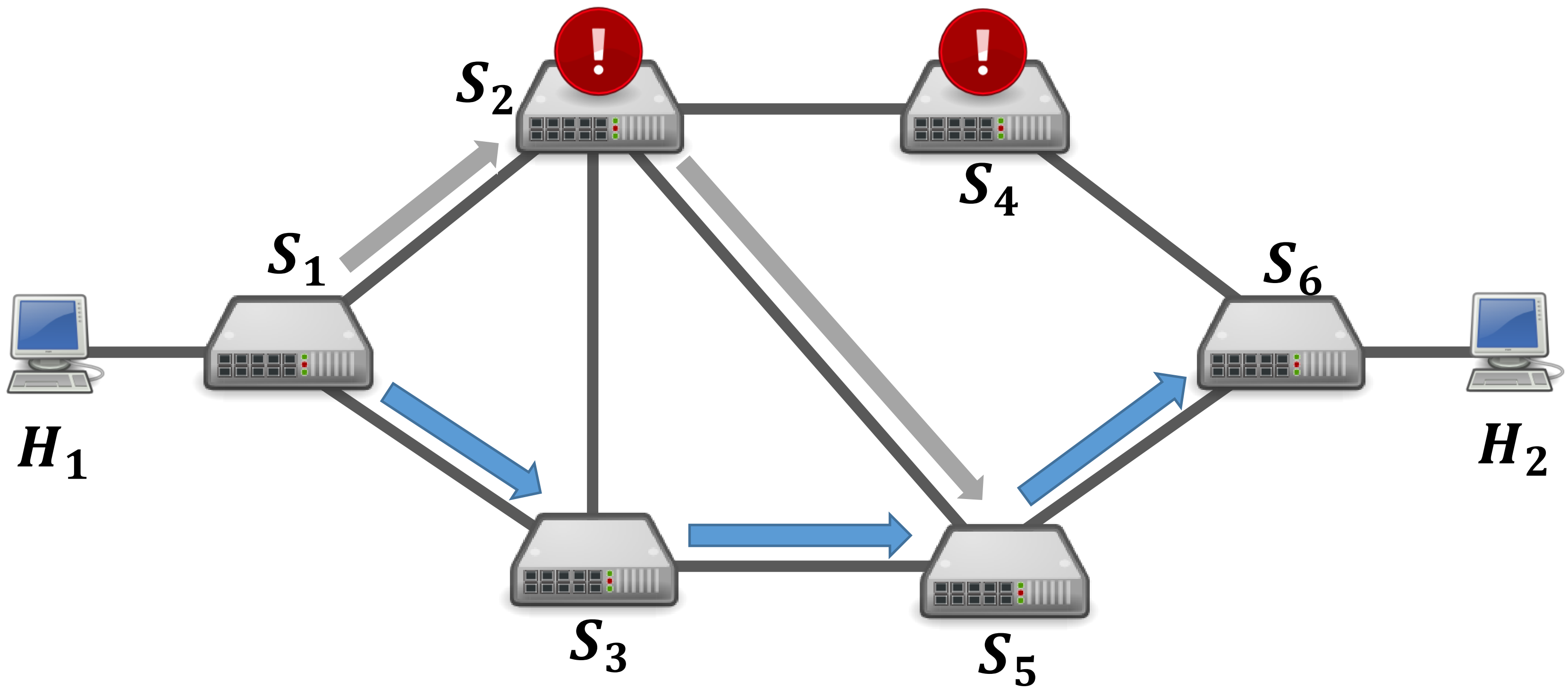
Recover path with backup

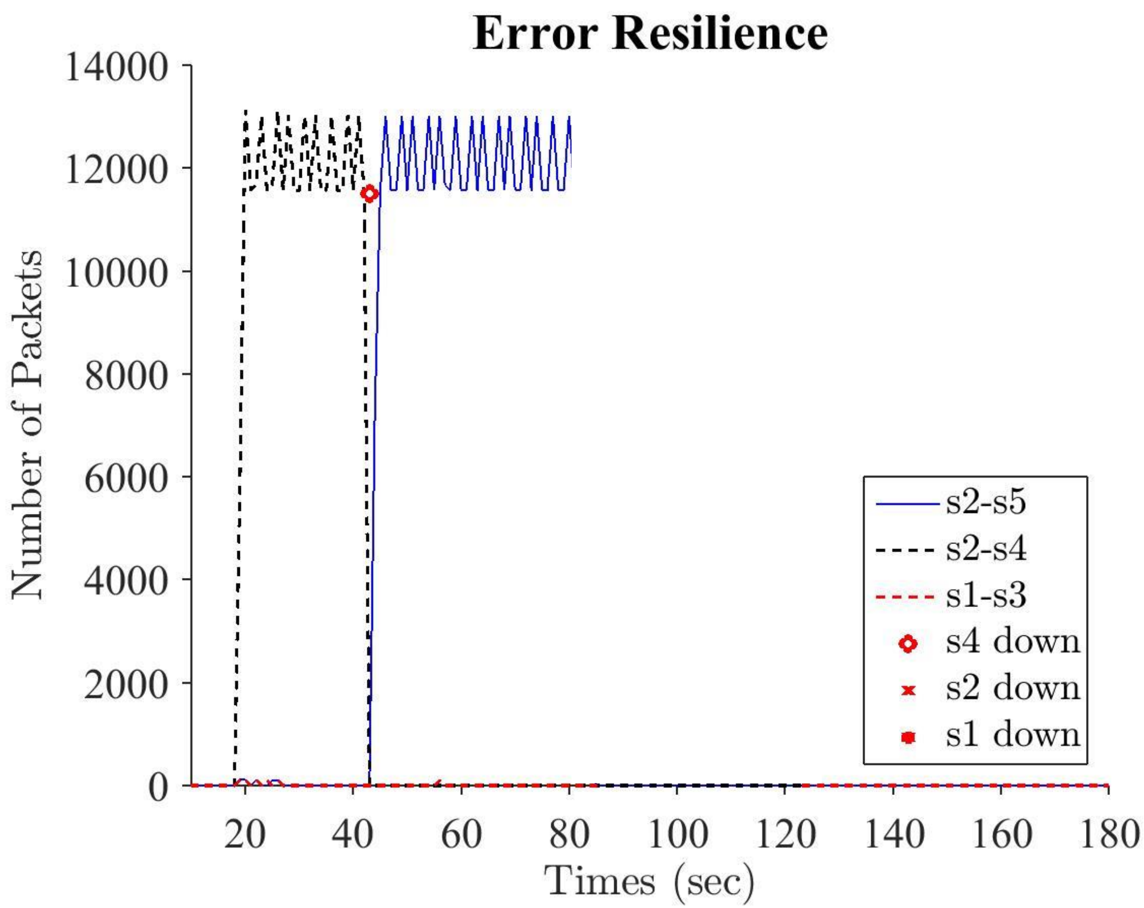




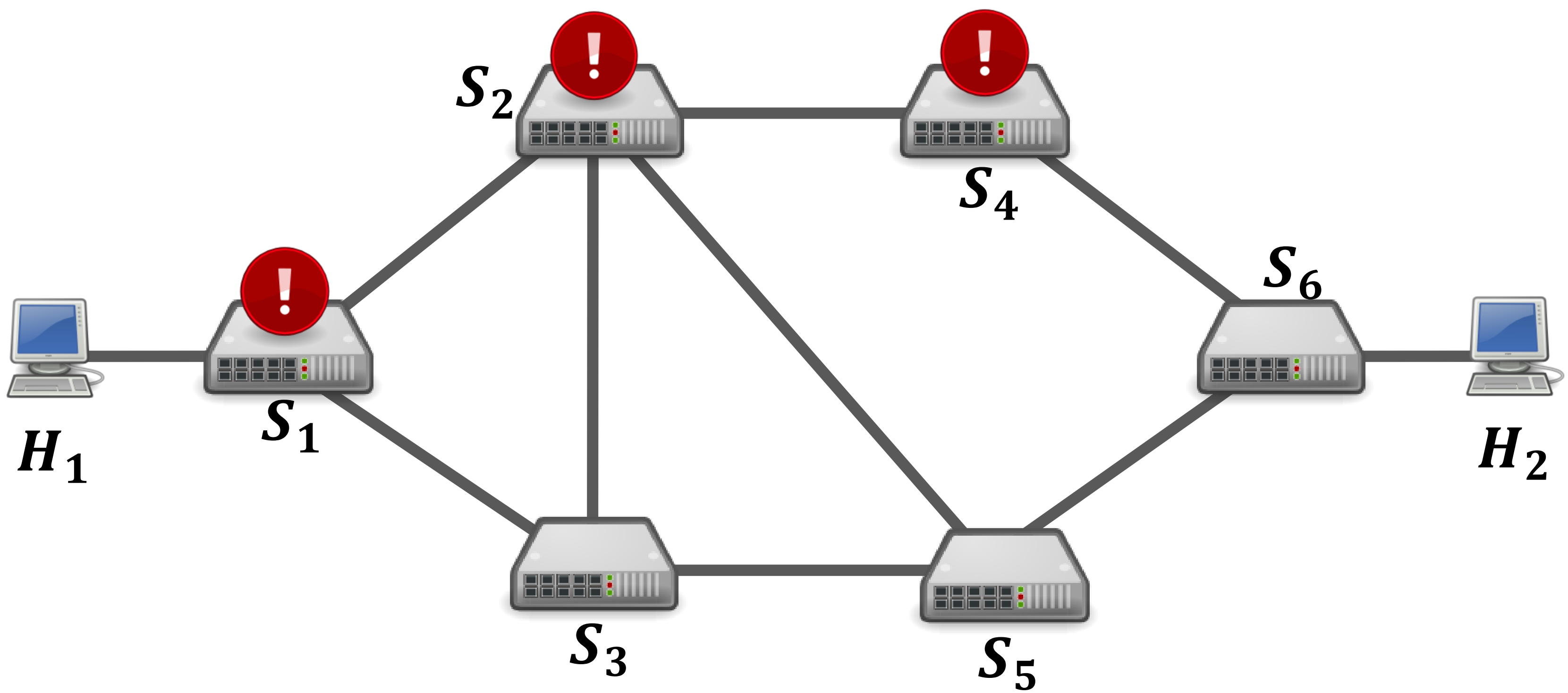
Error Resilience
14000
12000
10000
8000
6000
4000
2000
0
Number of Packets
20
40
60
80
100
120
140
160
180
Times (sec)
s2-s5
s2-s4
s1-s3
s4 down
s2 down
s1 down

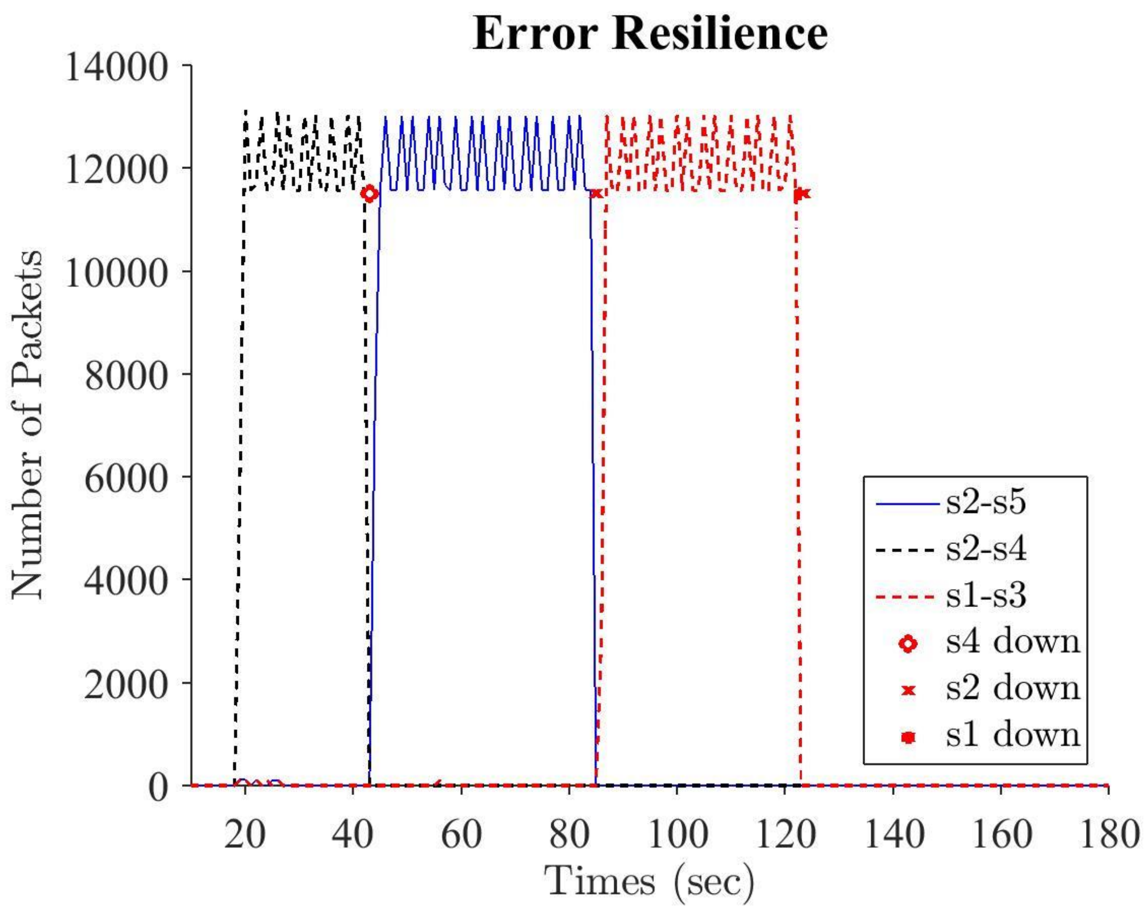
Recover using DPA




Error Resilience
Number of Packets
Times (sec)
14000
12000
10000
8000
6000
4000
2000
0
20
40
60
80
100
120
140
160
180
s2-s5
s2-s4
s1-s3
s4 down
s2 down
s1 down

Wait for edge reconnecting




Error Resilience
14000
12000
10000
8000
6000
4000
2000
0
Number of Packets
20
40
60
80
100
120
140
160
180
Times (sec)
s2-s5
s2-s4
s1-s3
s4 down
s2 down
s1 down

Conclusion

Conclusion

- Purposed a label switching for solving Traffic Engineering in SDNs
  - Flexibility, Load balancing and Error resilience
- Purposed algorithms for the label switching
  - 2 Tunnel finder algorithm
  - 1 Load balancer, and 1 admission controller
- Emulation results shows that **Purposed TEL:**
  - Reduce the max. link utilization in different condition (Using idle resources)
  - Minimize initialization delay
  - The dynamic algorithm provide error resilience

Thanks for listening

Q & A