DVAD42 – Load Balancing

P4 programmable Load Balancing: HULA and MP-HULA

Today's Webinar agenda

- **HULA**
  - Landscape
  - Load balancing granularity (RECAP)
  - Background
  - Introduction
  - Probes
  - Best-path identification
  - HULA – P4 Exercise
- **MP-HULA**
  - Introduction
  - Challenges
  - HULA Problems for Multipath protocols
  - Design & Implementation

HULA

HULA: Scalable Load Balancing Using Programmable Data Planes

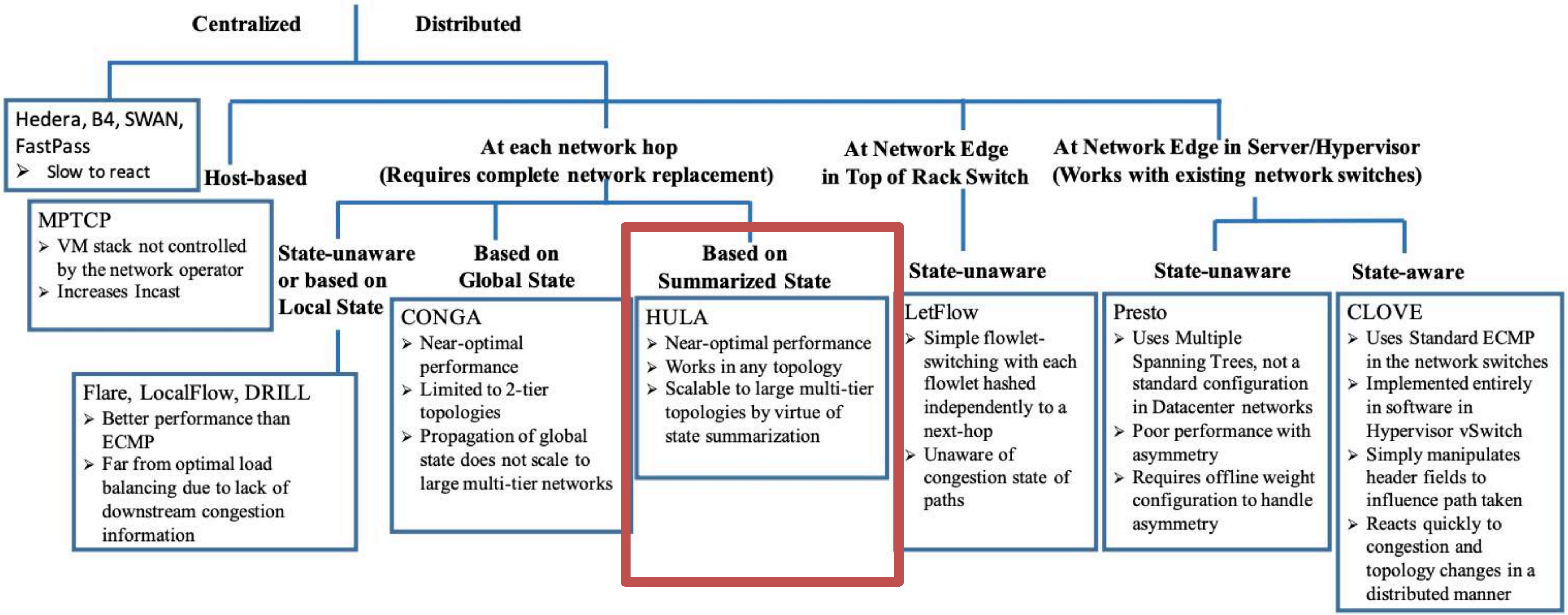
Naga Katta[*], Mukesh Hira[†], Changhoon Kim[‡], Anirudh Sivaraman[+], Jennifer Rexford[*]
[*]Princeton University, [†]VMware, [‡]Barefoot Networks, [+]MIT CSAIL
{nkatta, jrex}@cs.princeton.edu, mhira@vmware.com, chang@barefootnetworks.com, anirudh@csail.mit.edu

HULA - Scalable, Adaptable, Programmable

LB Scheme	Congestion aware	Application agnostic	Dataplane timescale	Scalable	Programmable dataplanes
ECMP (Switch)					
SWAN, B4 (Controller)					
MPTCP (EndHost)					
CONGA (Switch)					
HULA (Switch)					

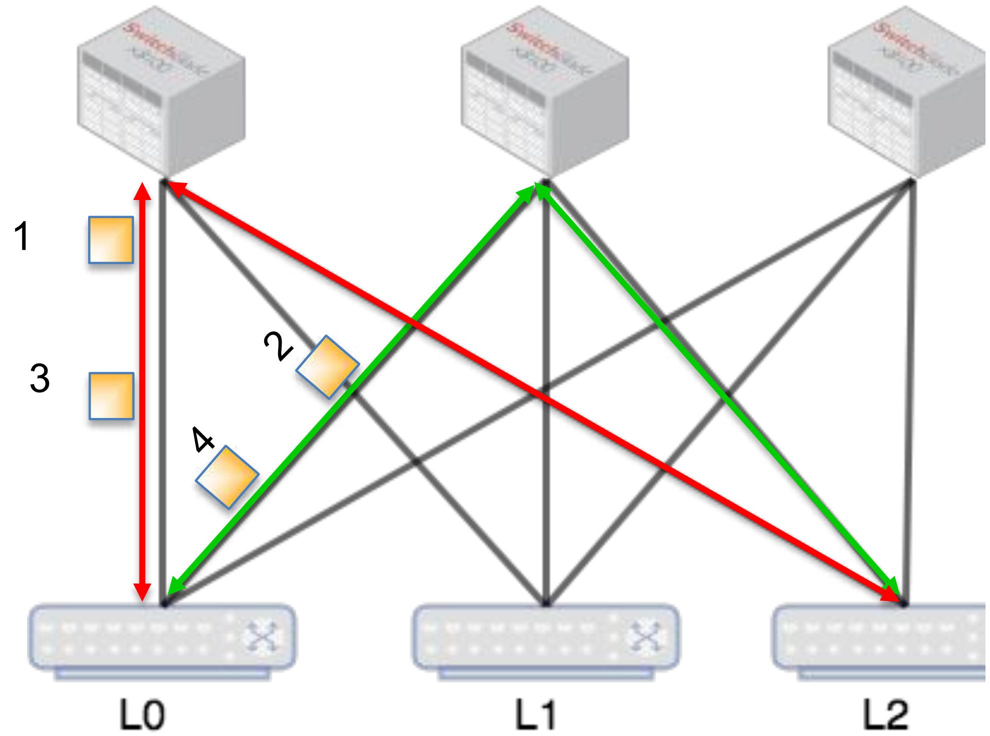
HULA - Summary

- **Scalable to large topologies (in contrast to Conga which works only for leaf/spine)**
  - HULA distributes congestion state
- **Adaptive to network congestion**
- **Proactive path probing**
- **Reliable when failures occur**
- **Programmable in P4**


Centralized
Distributed
Hedera, B4, SWAN, FastPass
➢ Slow to react
Host-based
MPTCP
➢ VM stack not controlled by the network operator
➢ Increases Incast
At each network hop
(Requires complete network replacement)
State-unaware or based on Local State
Flare, LocalFlow, DRILL
➢ Better performance than ECMP
➢ Far from optimal load balancing due to lack of downstream congestion information
Based on Global State
CONGA
➢ Near-optimal performance
➢ Limited to 2-tier topologies
➢ Propagation of global state does not scale to large multi-tier networks
Based on Summarized State
HULA
➢ Near-optimal performance
➢ Works in any topology
➢ Scalable to large multi-tier topologies by virtue of state summarization
At Network Edge in Top of Rack Switch
State-unaware
LetFlow
➢ Simple flowlet-switching with each flowlet hashed independently to a next-hop
➢ Unaware of congestion state of paths
At Network Edge in Server/Hypervisor
(Works with existing network switches)
State-unaware
Presto
➢ Uses Multiple Spanning Trees, not a standard configuration in Datacenter networks
➢ Poor performance with asymmetry
➢ Requires offline weight configuration to handle asymmetry
State-aware
CLOVE
➢ Uses Standard ECMP in the network switches
➢ Implemented entirely in software in Hypervisor vSwitch
➢ Simply manipulates header fields to influence path taken
➢ Reacts quickly to congestion and topology changes in a distributed manner

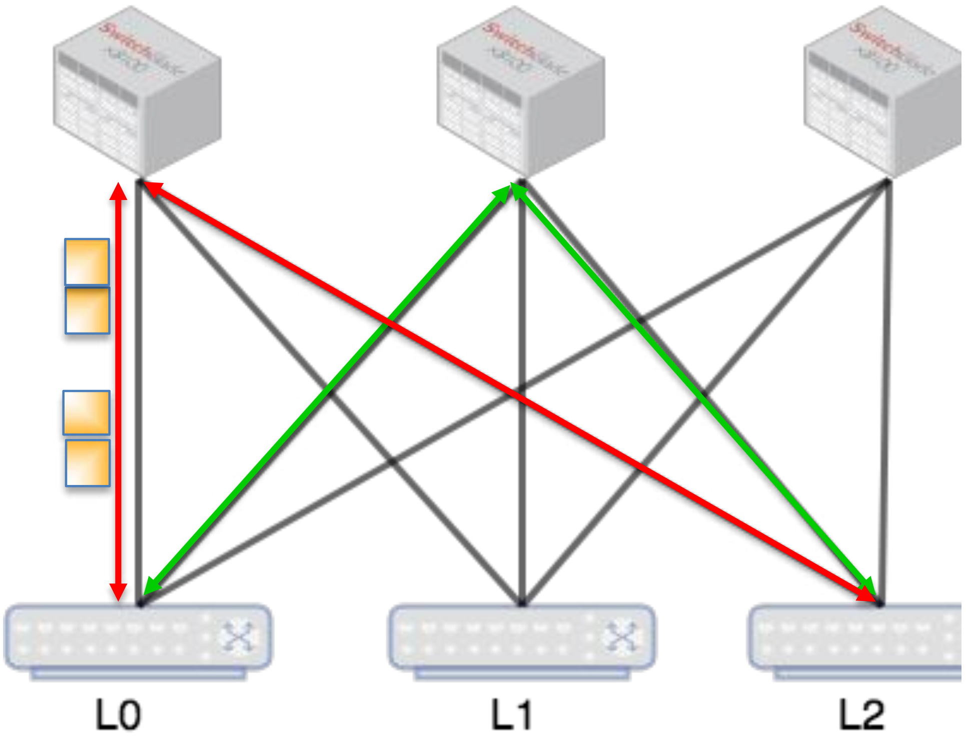
Load balancing granularity (RECAP)

- **Load-balancing granularity: by packet, flow or flowlet.**
  - Need to avoid reordering (may lead to TCP timeouts)
- **Packet-based load-balancing**
  - achieves highest granularity
  - But may lead to reordering



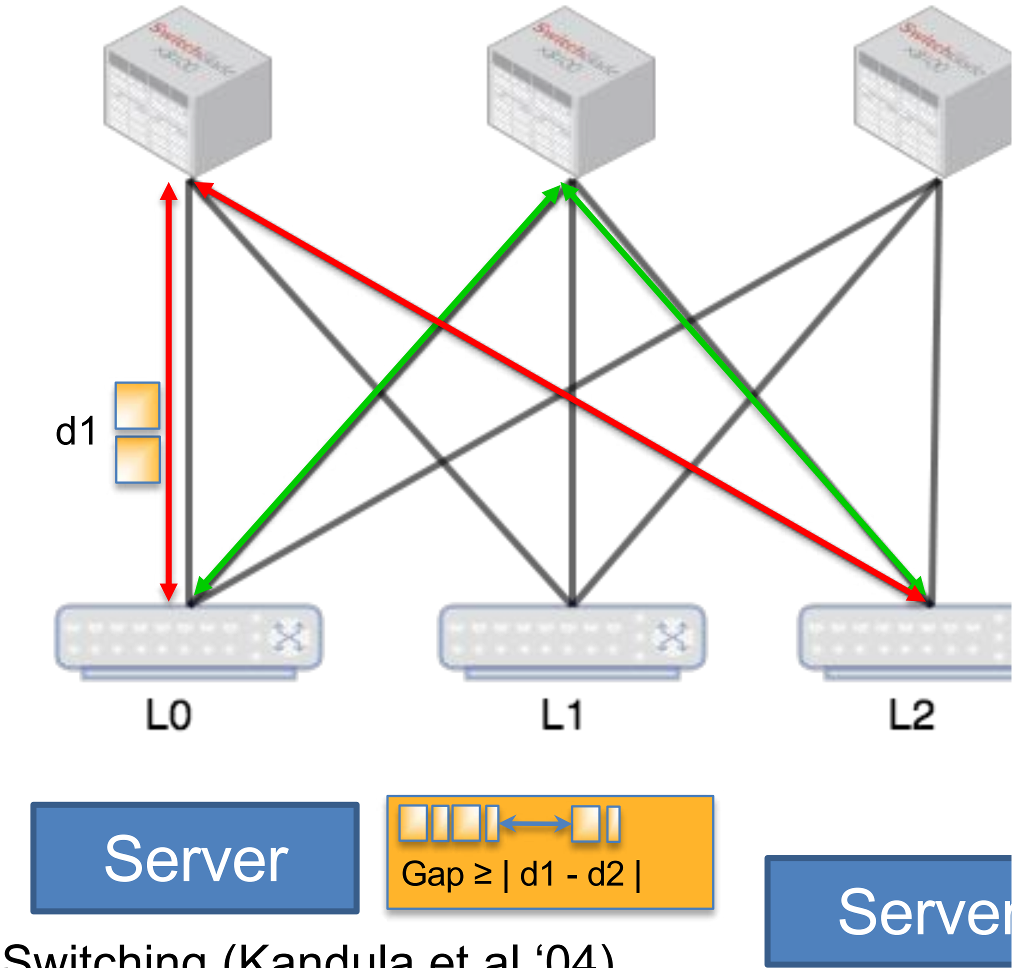
Load balancing granularity (RECAP)

- **Load-balancing granularity: by packet, flow or flowlet.**
  - Need to avoid reordering (may lead to TCP timeouts)
- **Flow-based load-balancing**
  - achieves lowest granularity
  - Avoids reordering completely
  - Flow collisions may lead to congested or low utilized links



Load balancing granularity (RECAP)

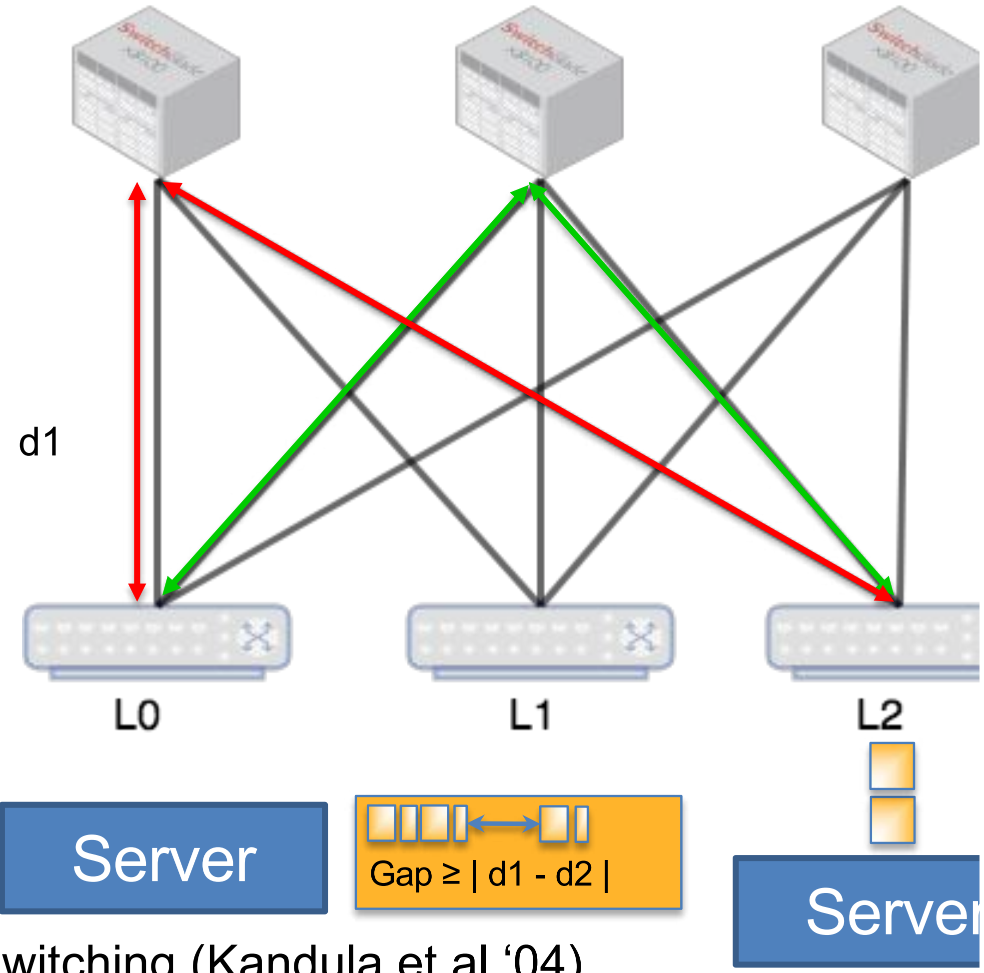
- **Load-balancing granularity: by packet, flow or flowlet.**
  - Need to avoid reordering (may lead to TCP timeouts)
- **Flowlet based load-balancing**
  - Strikes a balance between granularity while still being able to utilize all paths properly
  - Works only for TCP variants that create packet bursts
  - Exploits TCPs burstiness



*Flowlet Switching (Kandula et al ‘04)

Load balancing granularity (RECAP)

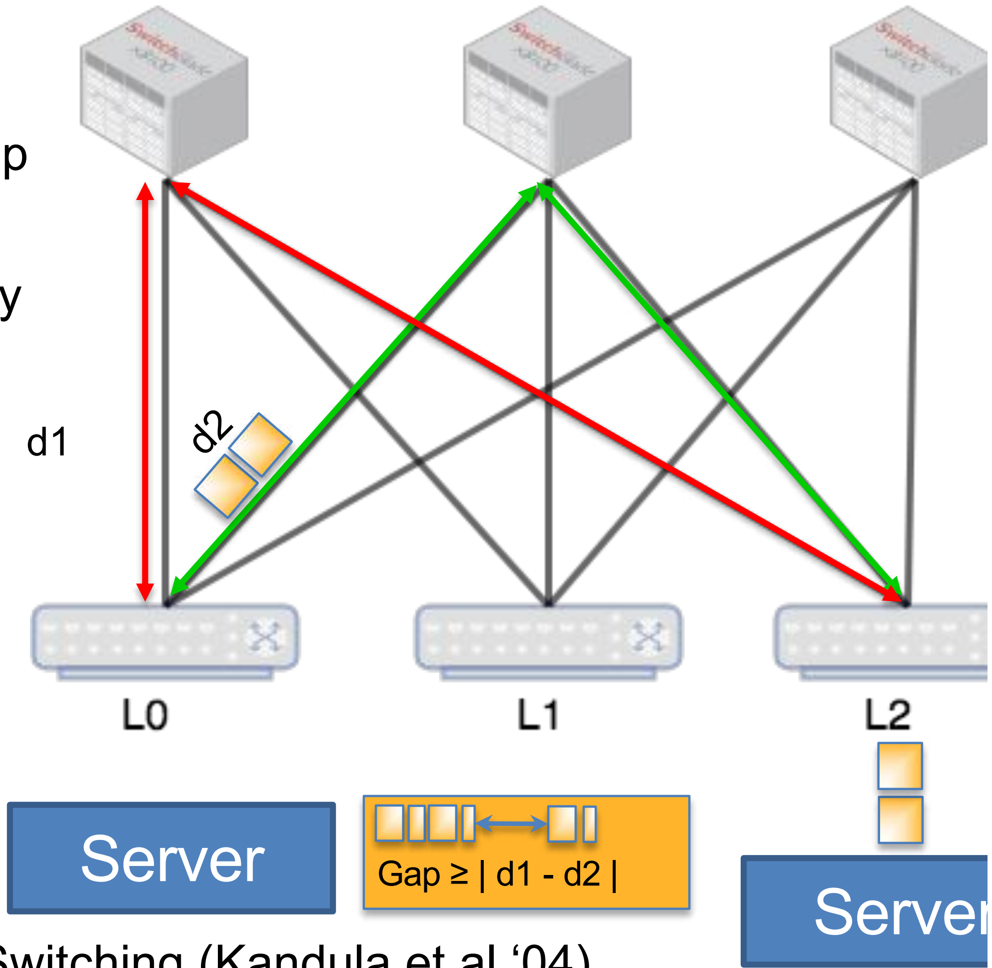
- **Load-balancing granularity: by packet, flow or flowlet.**
  - Need to avoid reordering (may lead to TCP timeouts)
- **Flowlet based load-balancing**
  - Strikes a balance between granularity while still being able to utilize all paths properly
  - Works only for TCP variants that create packet bursts
  - Exploits TCPs burstiness



*Flowlet Switching (Kandula et al '04)

Load balancing granularity (RECAP)

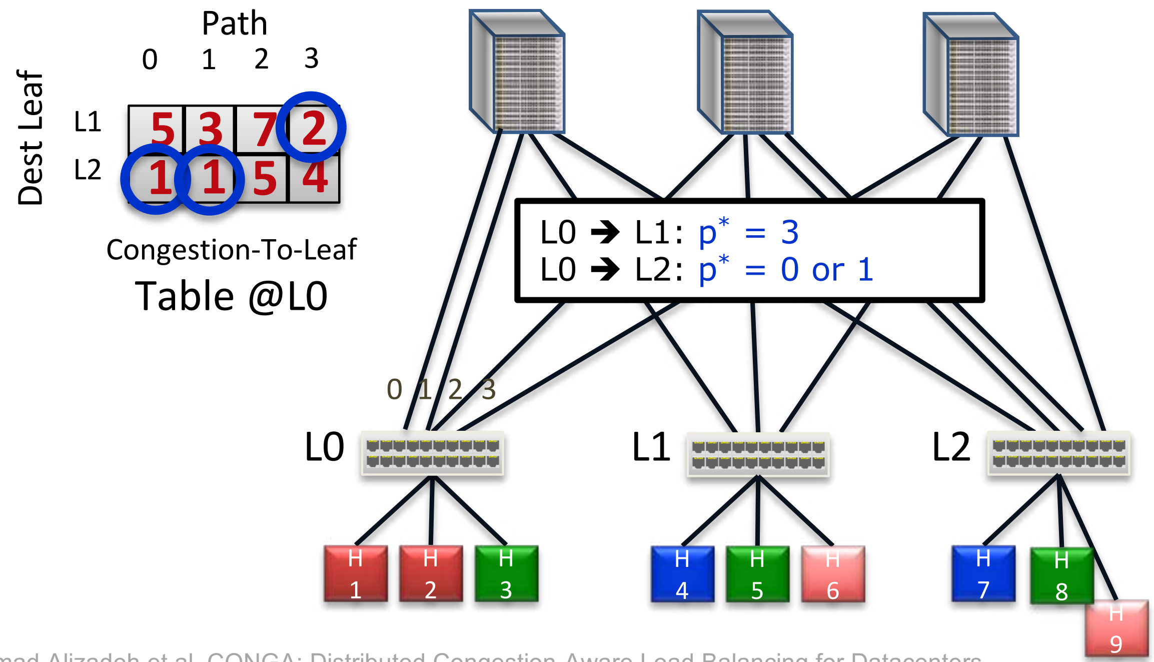
- **Flowlet summary**
  - Flowlets are burst of packets.
  - Large TCP flows can be splitted into many small flowlets, given enough inter-packet gap is detected
  - A new flowlet can be switched independently on a new path, given the inter-packet gap is large enough to avoid re-ordering (typical setting: maximum delay difference between any possible path).
    - In general, flowlet load balancing will not cause TCP reordering.
    - Requires proper setting of flowlet gap
    - However, some TCP variants create less bursts (e.g. when using pacing)



*Flowlet Switching (Kandula et al '04)

CONGA: Design: LB Decisions

- Track path-wise congestion metrics (3 bits) between each pair of leaf switches
- Send each flowlet on least congested path



Scalability to large topologies?

Source: Mohammad Alizadeh et al. CONGA: Distributed Congestion-Aware Load Balancing for Datacenters

HULA - Background

- **Main idea: route new flowlets along least-congested paths (as in Conga) for larger topologies (e.g. fat-tree)**

- **Main Questions to solve:**
- **How to infer path congestion?**
  - Periodic probes carry path utilization
  - Distance-vector like propagation
- **How to find and keep track of least congested path?**
  - Each switch chooses best downstream path
  - Maintains only best next hop
  - Scales to large topologies
- **How to implement on programmable switches?**
  - Programmable at line rate in P4

HULA - Background

- **Hop-by-hop Utilization-aware Load-balancing Architecture (HULA)**

- **Distance-vector like propagation**
  - Periodic probes carry path utilization
- **Each switch chooses best downstream path**
  - Maintains only best next hop
  - Scales to large topologies
- **Programmable at line rate**
  - Written in P4.

HULA - Background

- **Hop-by-hop Utilization-aware Load-balancing Architecture (HULA)**

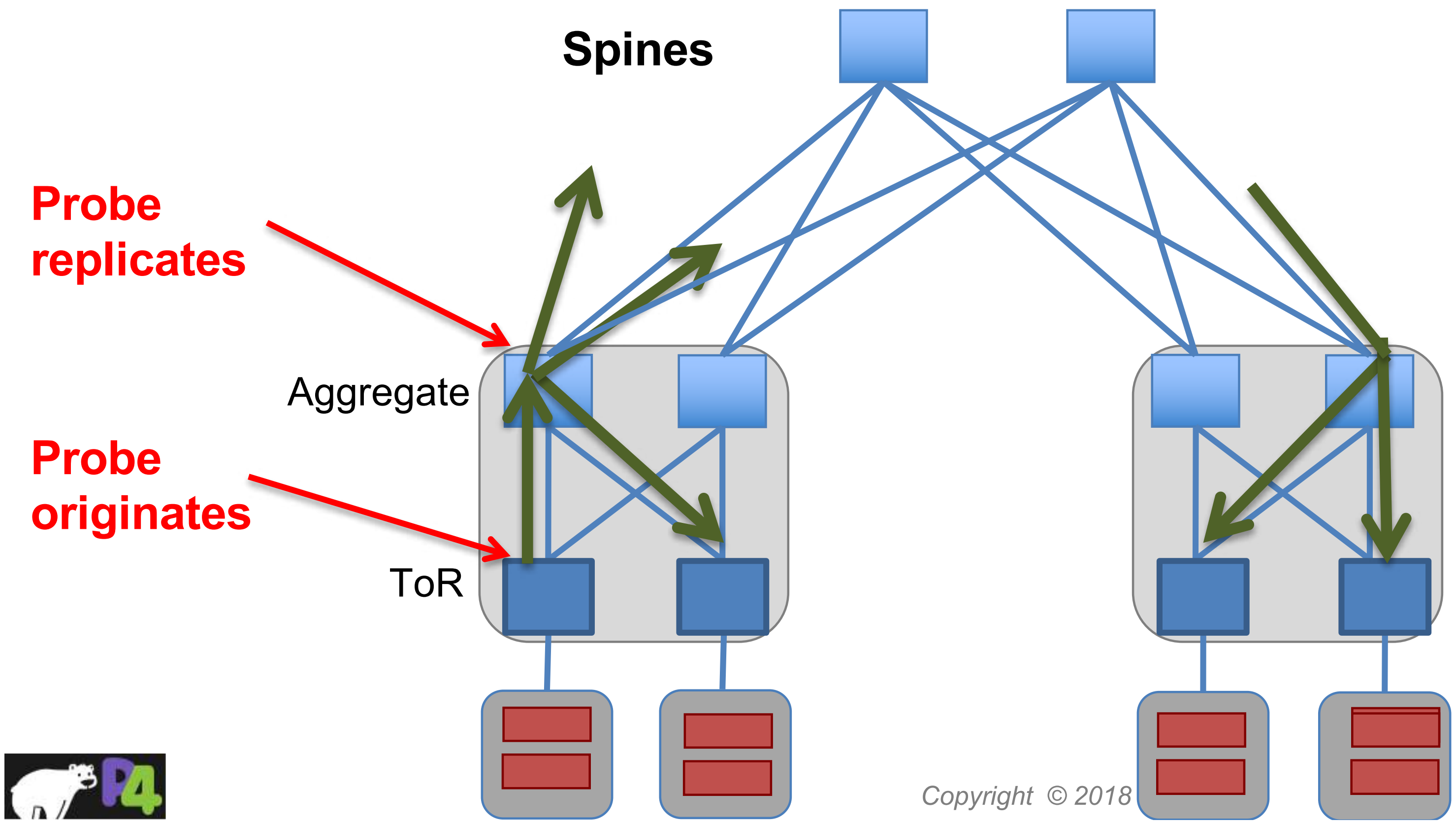
- **Distance-vector like propagation**
    - Periodic probes carry path utilization
- **Each switch chooses best downstream path**
    - Maintains only best next hop
    - Scales to large topologies
- **Programmable at line rate**
    - Written in P4.

HULA - Probes carry path utilization

- **HULA probes:**
  - Proactively disseminate network utilization information to all switches
  - Proactively update the network switches with the best path to any given leaf ToR.
- **Flows are split into flowlets**
  - This minimizes receive-side packet-reordering when a HULA switch sends different flowlets on different paths

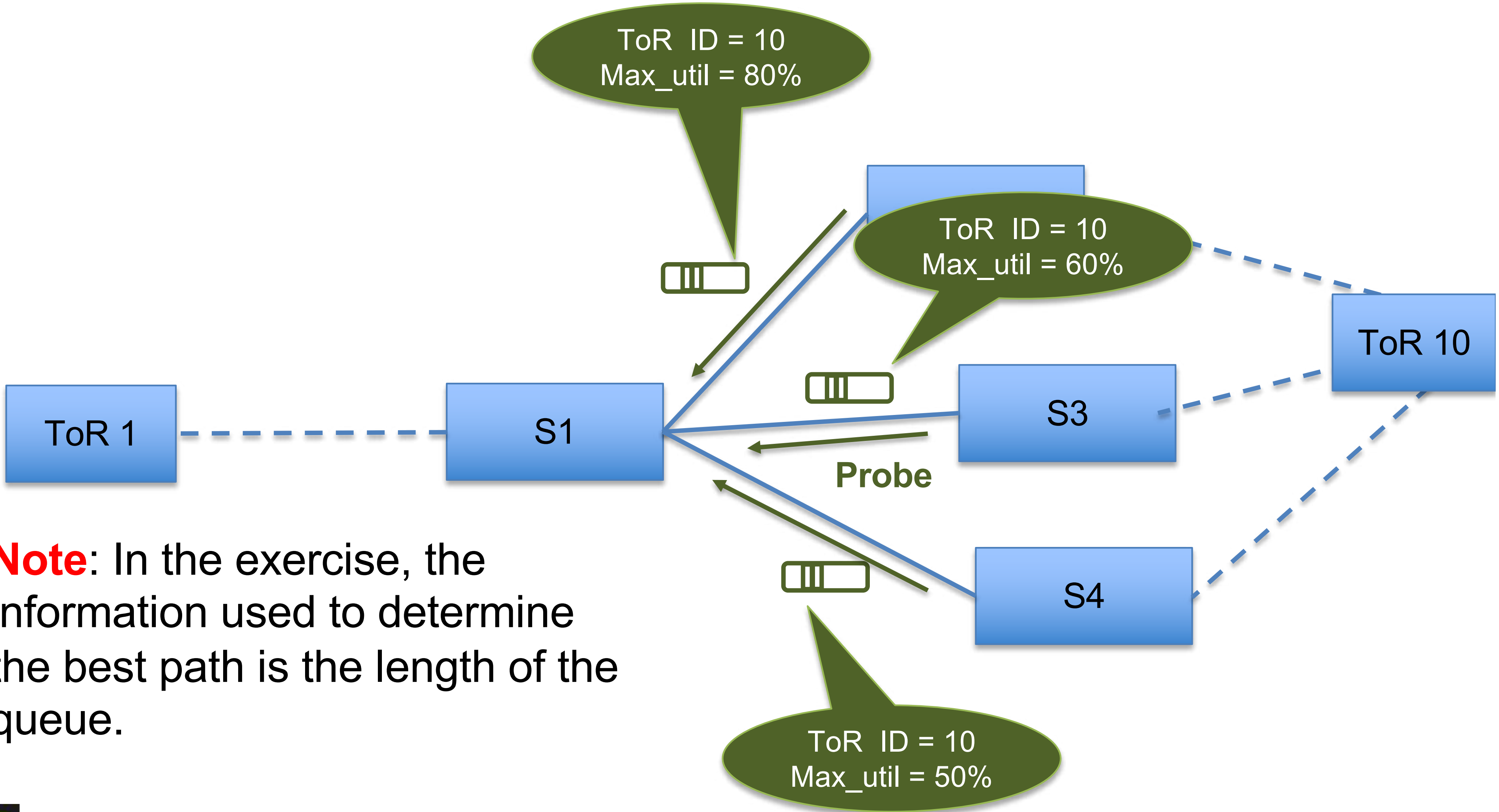
HULA - Probes carry path utilization

- The probes originate at the ToRs and are replicated on multiple paths as they travel the network.
- Once a probe reaches another ToR, it ends its journey.



P4 primitives
New header format
Programmable Parsing
RW packet metadata

HULA - Probes carry path utilization



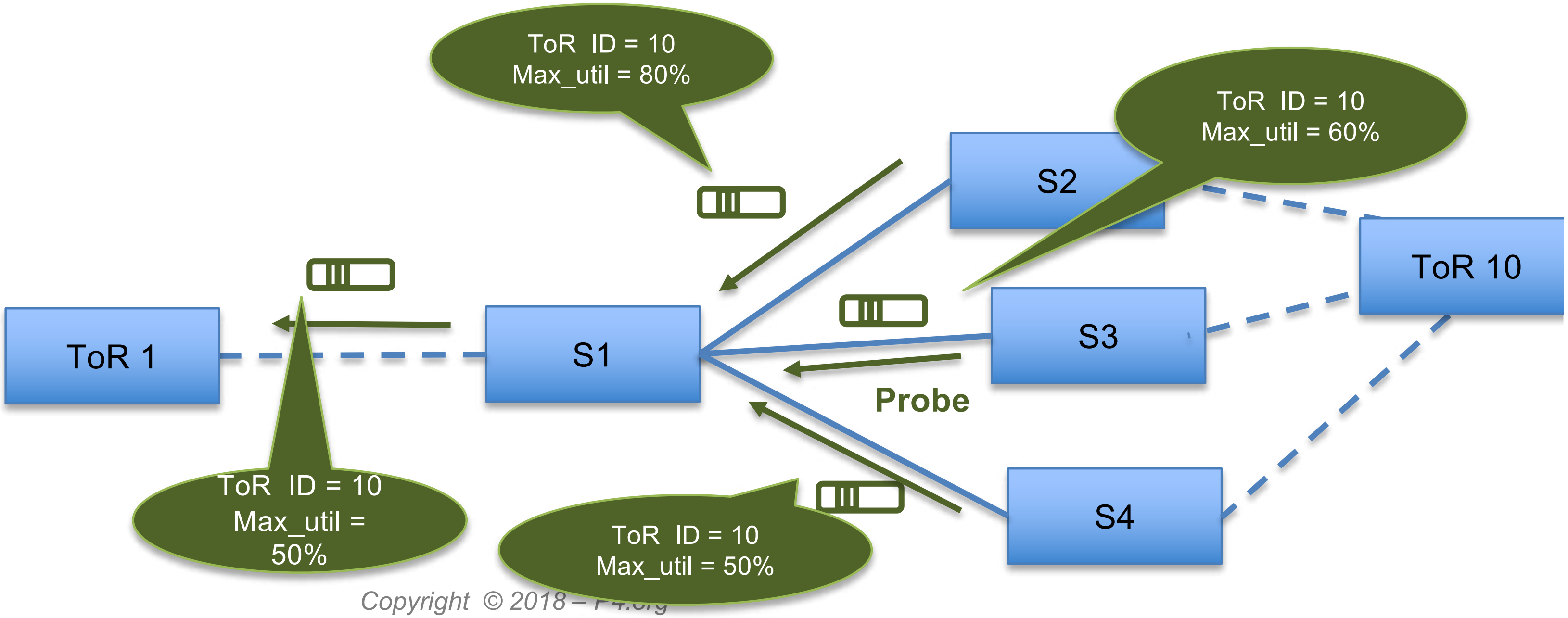
Note: In the exercise, the information used to determine the best path is the length of the queue.

HULA - Best downstream path identification

1. **The switch takes the minimum from among the probe given utilization and stores it in the local routing table.**
2. **The switch S1 then sends its view of the best path to the upstream switches (e.g. S1 to ToR1), which processes incoming probes and repeats this process.**
3. **Each switch only needs to keep track of the best next hop towards a destination.**

Dst	Best hop	Path util
ToR 10	S4	50%
ToR 1	S2	10%
…	…	

Best hop table

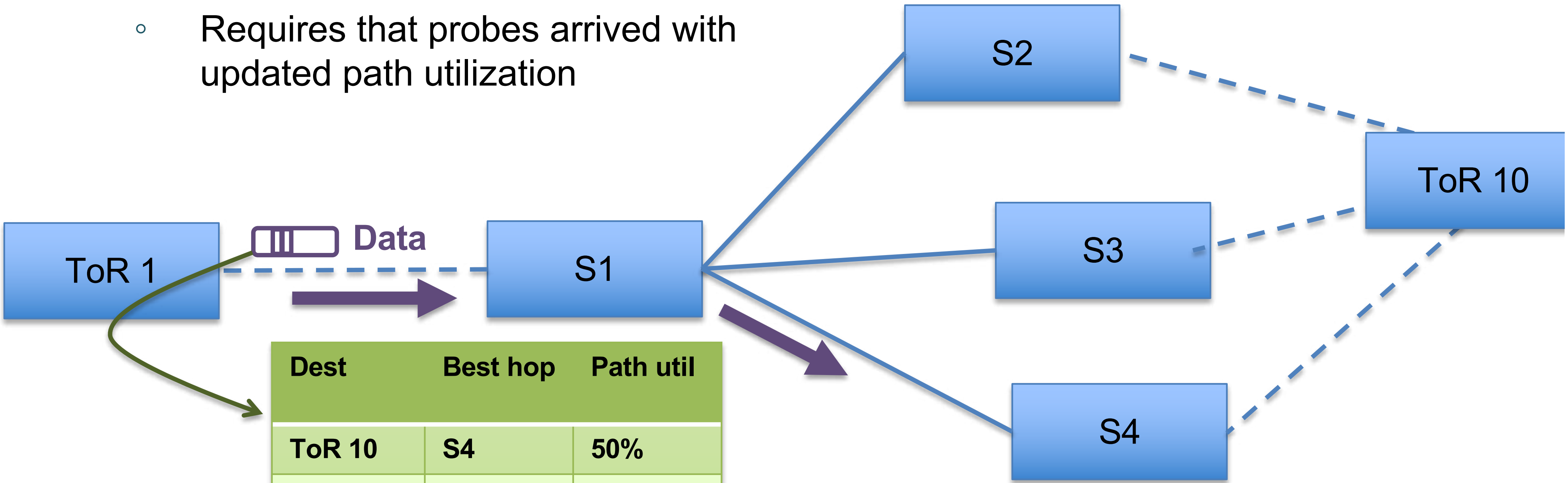


HULA - Background

- **Hop-by-hop Utilization-aware Load-balancing Architecture (HULA)**
- **Distance-vector like propagation**
  - Periodic probes carry path utilization
- **Each switch chooses best downstream path**
  - Maintains only best next hop
  - Scales to large topologies
- **Programmable at line rate**
  - Written in P4.

HULA - Switches load balance flowlets

- **The switches route data packets in the opposite direction.**
  - Each switch independently chooses the best next hop to the destination.
- **Once flowlet gap expires, new best path is selected**
  - Can be old one or new better one
  - Requires that probes arrived with updated path utilization



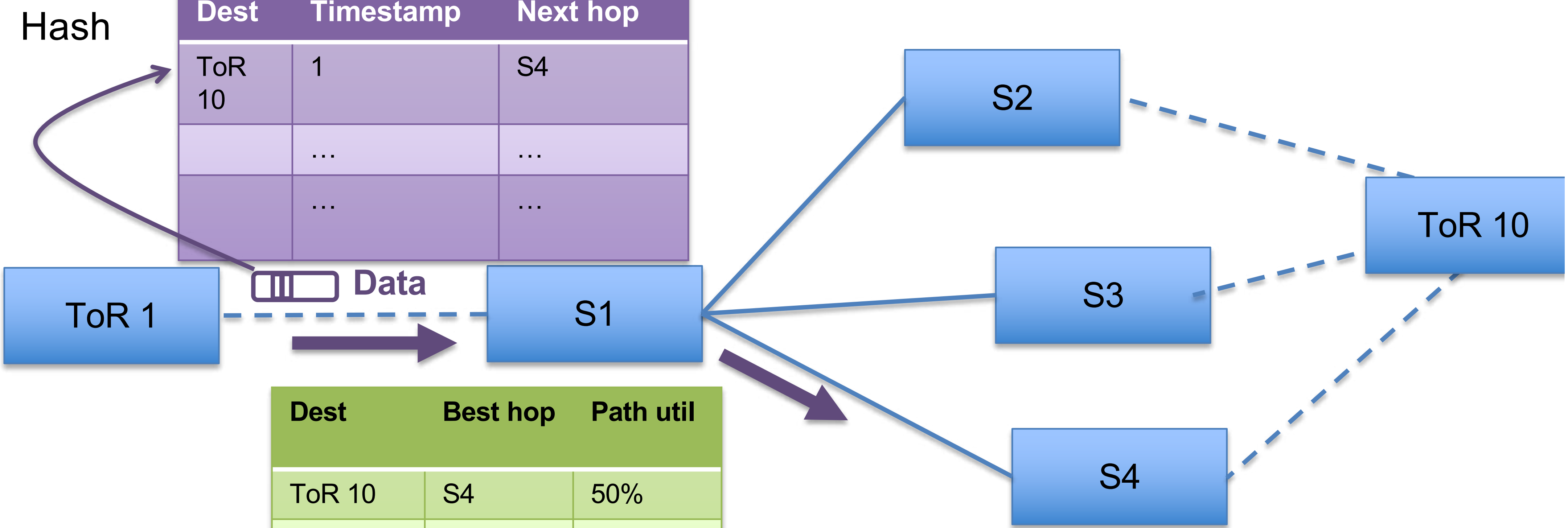
Dest	Best hop	Path util
ToR 10	**S4**	**50%**
ToR 1	S2	10%
…	…	

Best hop table

HULA - Switches load balance flowlets

Flowlet table

Dest	Timestamp	Next hop
ToR 10	1	S4
	…	…
	…	…



Dest	Best hop	Path util
ToR 10	S4	50%
ToR 1	S2	10%
…	…	

Best hop table

HULA - Background

- **Hop-by-hop Utilization-aware Load-balancing Architecture (HULA)**

- **Distance-vector like propagation**
  - Periodic probes carry path utilization
- **Each switch chooses best downstream path**
  - Maintains only best next hop
  - Scales to large topologies
- **Programmable at line rate**
  - Written in P4.

HULA - Programmable at line rate

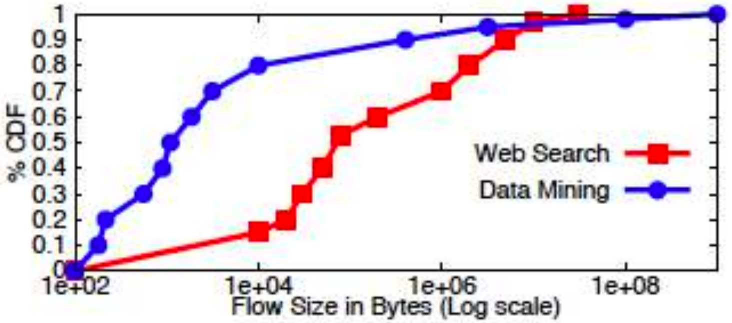
- **HULA requires both stateless and stateful operations to program HULA's logic**

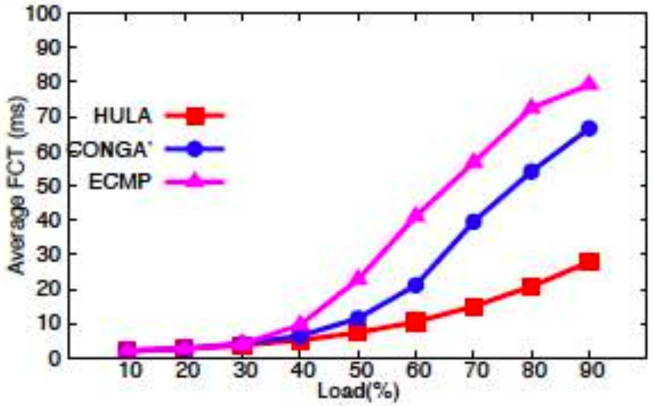
- **Processing a packet in a HULA switch involves switch state updates at line rate in the packet processing pipeline.**

- **HULA maintains a current best hop and replace it in place when a better probe update is received**
  - using register read/write

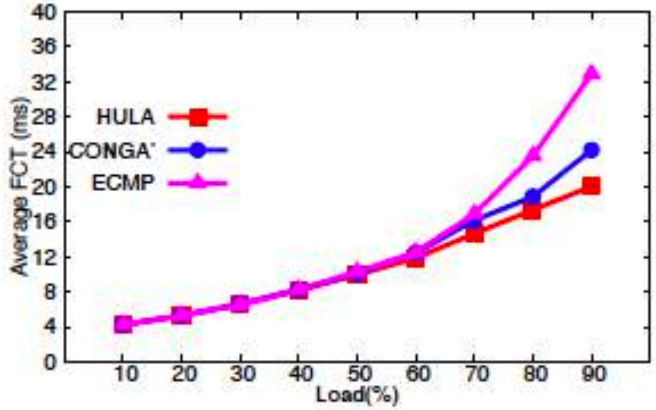
HULA - Evaluation

- **Different Datacenter Workload traces**

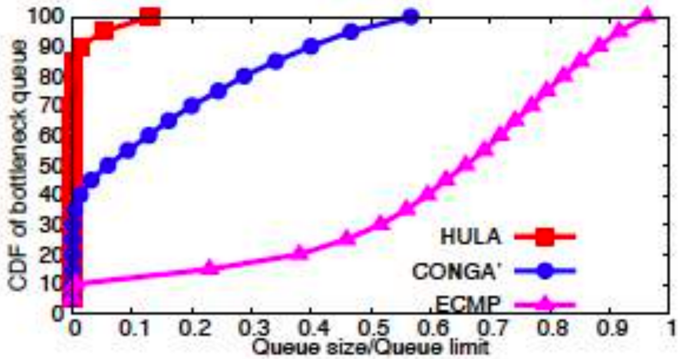




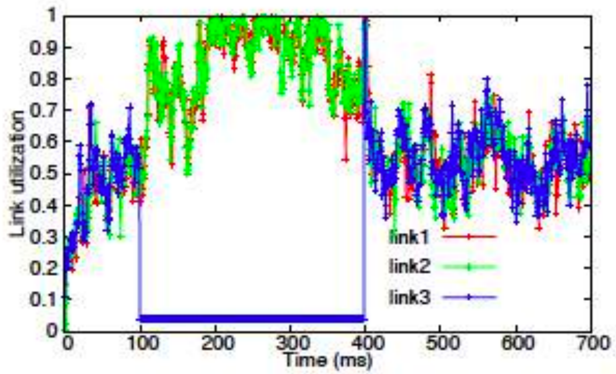
(b) Web-search overall avg FCT



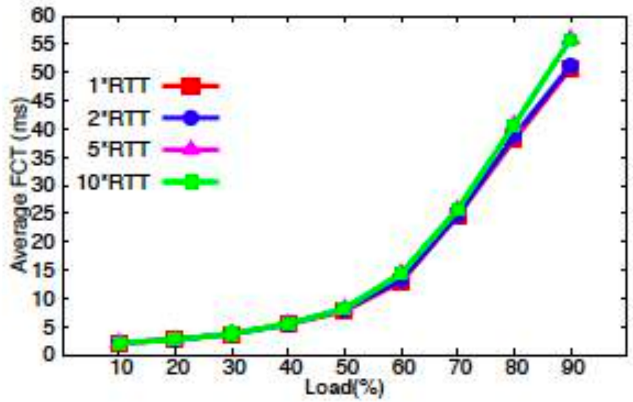
(c) Data-mining overall avg FCT



(c) Queue length at bottleneck link (S2->A3) in the link failure scenario



(a) Link utilization on failures with Web-search workload

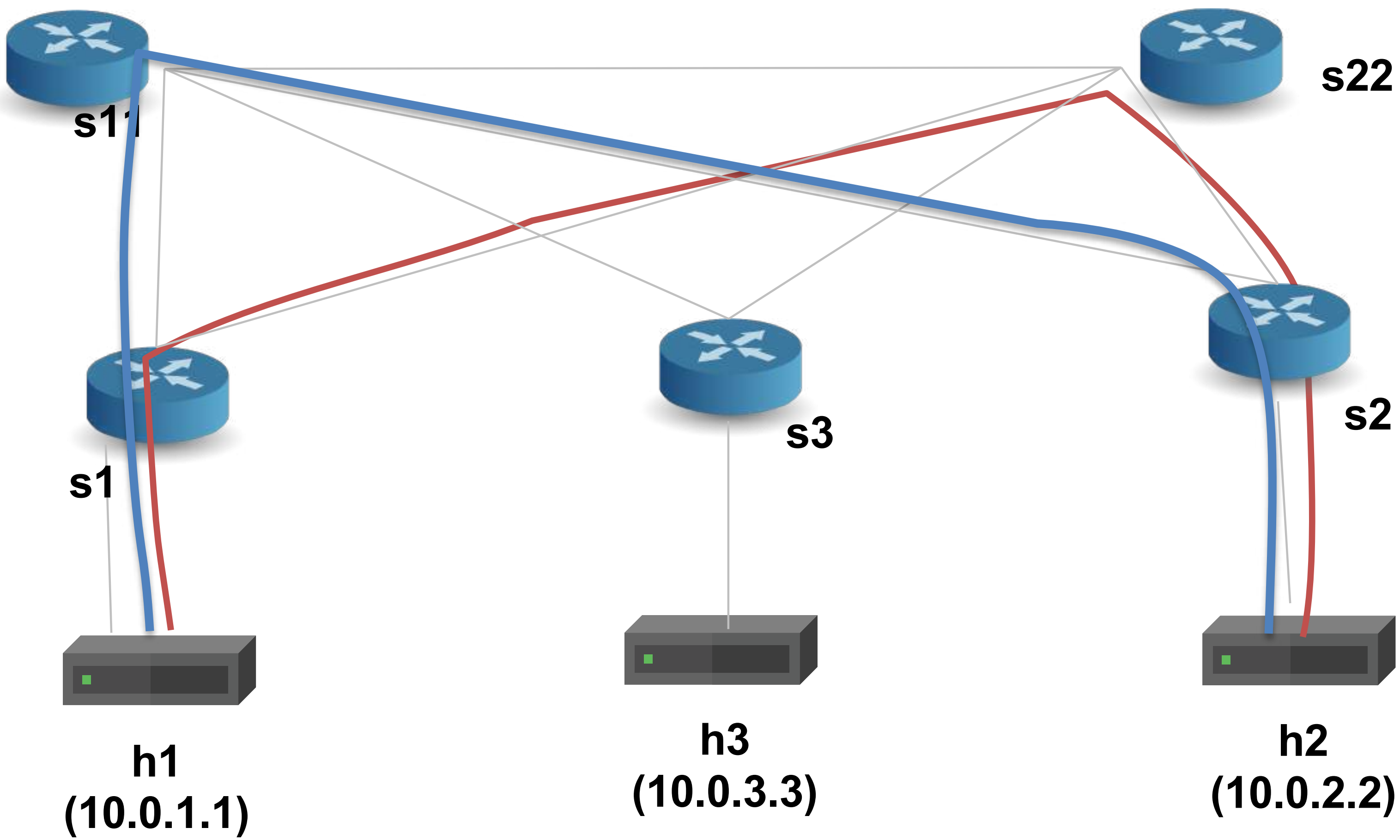


(c) Effect of decreasing probe frequency

HULA - Exercises

Goal: implement a simple variant

HULA: Topology



HULA - Headers

```
header hula_t {
/* 0 is forward path, 1 is the backward path */
      bit<1> dir;
/* max qdepth seen so far in the forward path */
      qdepth_t qdepth;
/* digest of the source routing list to uniquely
identify each path */
      digest_t digest;
}
```

- `hula_t` –Header for the HULA probe packet.
- `dir (1bit)` – To indicate the direction of the probe packet
- `Qdepth (15bit)` – maximum queue length seen so far (will be updated)
- `Digest (32bit)` – This field is set by the ToR to identify the path

To share the best path information with the source ToRs so that the sources can use that information for new flows, the destination ToRs notify source ToRs of the current best path by returning the HULA probe back to the source ToR (reverse path) only if the current best path changes.

This python script makes each ToR switch generate one HULA probe for each other ToR and through each separate forward path

generatehula.py →

Probes can be generated from Control Plane (e.g. Switch CPU). In the example, they include a digest of the source routing list to uniquely identify each path and a source routing list that uniquely identifies the forwarding behavior.

HULA – Agg & ToR: Tables and actions

```
#define TOR_NUM 32

* index is set based on dstAddr */
table hula_bwd {
    key = {
       hdr.ipv4.dstAddr: lpm;
}   actions = {
       hula_set_nhop;
}
    size = TOR_NUM;
}

action hula_set_nhop(bit<32> index) {
      dstindex_nhop_reg.write(index,
(bit<16>)standard_metadata.ingress_port); }

register<bit<16>>(TOR_NUM)
dstindex_nhop_reg;

/* At each hop saves the next hop for each flow */
register<bit<16>>(65536) flow_port_reg;
```

- **`hula_bwd`** –Update the next hop to destination ToR for reverse_path using the *hula_set_nhop* action by updating the register *dstindex_nhop_reg*.
- **`hula_set_nhop`** – We store the next hop to reach each destination ToR
- **`dstindex_nhop_reg`** – At each hop, saves the next hop to reach each destination ToR
- **`flow_port_reg`** – At each hop, saves the next hop for each flow

Example: `table_add hula_bwd hula_set_nhop 10.0.1.0/24 => 0`

`10.0.1.0/24` → hdr.ethernet.dstAddr   `0` → index

```
table_add hula_bwd hula_set_nhop 10.0.2.0/24 => 1
table_add hula_bwd hula_set_nhop 10.0.3.0/24 => 2
```

HULA – ToR: Tables and actions

```
table hula_src {
     Key = {
         hdr_ipv4.srcAddr: exact;
     }
     actions = {
             srcRoute_nhop;
             drop;
     }
     default_action = srcRoute_nhop;
      size = 2;
}

action srcRoute_nhop() {
        standard_metadata.egress_spec =
           (bit<9>)hdr.srcRoutes[0].port;
        hdr.srcRoutes.pop_front(1);
}

action drop() {
       mark_to_drop();
}

/* At destination ToR, saves the queue depth of
the best path from * each source ToR */
register<qdepth_t>(TOR_NUM) srcindex_qdepth_reg;
```

- **`hula_src`** – Checks the source IP address of a HULA packet in reverse path. If this switch is the source, this is the end of reverse path, thus drop the packet. Otherwise use srcRoute_nhop action to continue source routing in the reverse path.
- **`srcRoute_nhop`** – to perform source routing.
- **`srcindex_qdepth_reg`**: At destination ToR saves queue length of the best path from each Source ToR
- Example:

```
table_add hula_src drop 10.0.1.0 =>
register_write srcindex_qdepth_reg 0 256
```

Removes the first element of the stack. Returns the number of elements removed. The second element of the stack becomes the first element, and so on...

HULA – Agg & ToR: Tables and actions

```
#define TOR_NUM 32

table hula_nhop {
    key = {
       hdr.ipv4.dstAddr: lpm;
}   actions = {
       hula_get_nhop;
       drop;
}
    size = TOR_NUM;
}
action hula_get_nhop(bit<32> index) {
  bit<16> tmp;
  dstindex_nhop_reg.read(tmp, index);
  standard_metadata.egress_spec =
     (bit<9>)tmp;
}
action drop() {
  mark_to_drop(std_metadata);
}
```

- **hula_nhop** – table for data packets, reads destination IP/24 to get an index. It uses the index to read dstindex_nhop_reg register and get best next hop to the destination ToR.
- **hula_get_nhop** – It uses the index to read dstindex_nhop_reg register and get best next hop to the destination ToR for data packets.
- **drop** - Drops the packet

Example: `table_add hula_nhop hula_get_nhop 10.0.1.0/24 => 0`

hdr.ethernet.dstAddr index

```
table_add hula_nhop hula_get_nhop 10.0.2.0/24 => 1
table_add hula_nhop hula_get_nhop 10.0.3.0/24 => 2
```

HULA – ToR: Tables and actions

```
#define TOR_NUM 32
struct metadata {
/* At destination ToR, this is the index of
register that saves qdepth for the best path
from each source ToR */
              bit<32> index;
}
* index is set based on dstAddr */
table hula_fwd {
    key = {
       hdr.ipv4.dstAddr: exact;
       hdr.ipv4.srcAddr: exact;
}   actions = {
       hula_dst;
       srcRoute_nhop;
}
    default_action = srcRoute_nhop;
    size = TOR_NUM + 1;
}
action hula_dst(bit<32> index) {
     meta.index = index;
}
action drop() {
     mark_to_drop(std_metadata);
}
```

- **hula_fwd** –looks at the destination IP of a HULA packet. If it is the destination ToR, it runs hula_dst action. Otherwise perform source routing.
- **hula_dst** – Set meta.index field based on source IP (source ToR). The index is used later to find queue depth and digest of current best path from that source ToR. Otherwise, this table just runs srcRoute_nhop to perform source routing.
- **drop** – Drops the packet

HULA – ToR: Tables and actions

```
table dmac {
     key = {
         standard_metadata.egress_spec : exact;
     }
     actions = {
             set_dmac;
             nop;
     }
     default_action = nop;
      size = 16;
}

action set_dmac(macAddr_t dstAddr){
         hdr.ethernet.srcAddr =
             hdr.ethernet.dstAddr;
         hdr.ethernet.dstAddr = dstAddr;
}

action nop() {
}
```

- **`dmac`** – Updates ethernet destination address based on next hop.
- **`set_dmac`** – Sets the destination macAddr

Example:

```
table_add dmac set_dmac 1 =>
00:00:00:00:01:01
```

Output port

Dst macAddr

HULA – ToR: Tables and actions (Logic)

```
control MyIngress (inout headers hdr,
                   inout metadata meta,
                   inout standard_metadata_t standard_metadata)
{

. . .

apply {
    if (hdr.hula.isValid()){
       if (hdr.hula.dir == 0){
         switch(hula_fwd.apply().action_run){
               /* if hula_dst action ran, this is the
destination ToR */
              hula_dst: {
                 /*Compare and update the queue size and best
path*/

       }else { /* hdr.hula.dir == 1 */
          /* update routing table in reverse path */
          hula_bwd.apply();
          /* drop if source ToR */
          hula_src.apply();
       }
    } else if (hdr.ipv4.isValid()) {
       1. Get the hash of the flow
       2. Look into the hula table
       3. Check if it is a new flowlet
          3.1 Check hula path for new flowlets
          3.2 Use old port for old flowlet
       4. Set the right dmac
    }else {
       drop();
    }
```

Check if it’s a hula probe packet

Check the direction of the hula probe

Check if it is a ipv4 packet

We drop packets that are neither hula nor ipv4

HULA – Your Homework for next 2 weeks

- **Skeleton code is available**
- **Hula probe processing already implemented**
- **Your job**
  - If it is a data packet compute the hash of flow
  - TODO read nexthop port from flow_port_reg into a temporary variable, say port.
  - TODO If no entry found (port==0), read next hop by applying hula_nhop table. Then save the value into flow_port_reg for later packets.
  - TODO if it is found, save port into standard_metadata.egress_spec to finish routing.
  - apply dmac table to update ethernet.dstAddr. This is necessary for the links that send packets to hosts. Otherwise their NIC will drop packets.
- **TODO:** An egress control that for HULA packets that are in forward path (hdr.hula.dir==0)
  compares standard_metadata.deq_qdepth to hdr.hula.qdepth in order to save the maximum in hdr.hula.qdepth

MP-HULA

MP-HULA: Multipath Transport Aware Load Balancing Using Programmable Data Planes

Cristian Hernandez Benet[1], Andreas J. Kassler[1] , Theophilus A. Benson[2], Gergely Pongracz[3]
Karlstad University[1], Brown University[2], Networking Research – Ericsson[3]

Motivation

- **Multiple Paths**
- **Large Bisection Bandwidth**
  - But: at most 25% of core links are highly utilized → effective load balancing required
- **Volatile, Unpredicted Traffic patterns**
- **Multipath Transport Protocols (e.g. MPTCP)**
  - Applications enhance their performance using several paths (e.g. SIRI)
- **Symmetric/Assymetric topologies with different number of layers**

ECMP CONGA HULA DRILL CLOVE

GRANULARITY

CONGESTION-AWARE

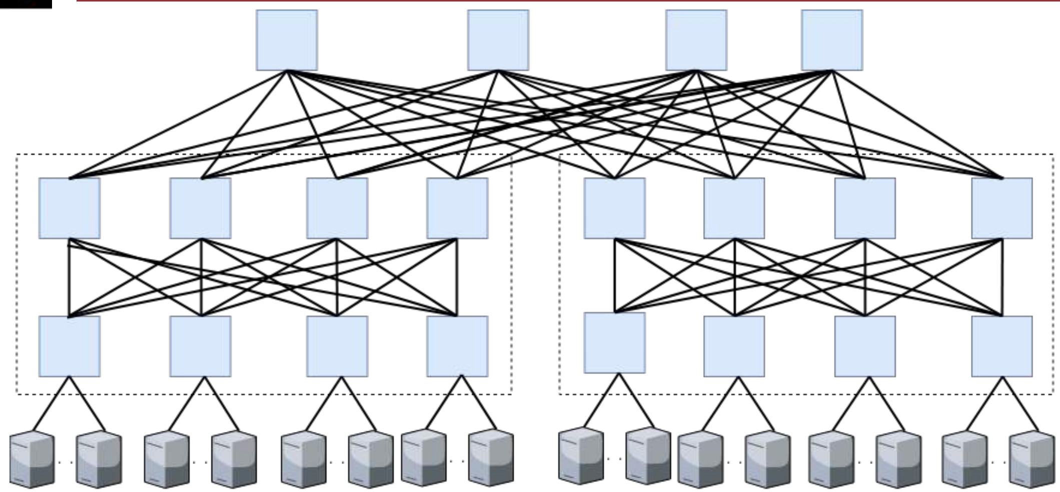
CUSTOM-ASIC

PROGRAMMABLE

SCALABLE

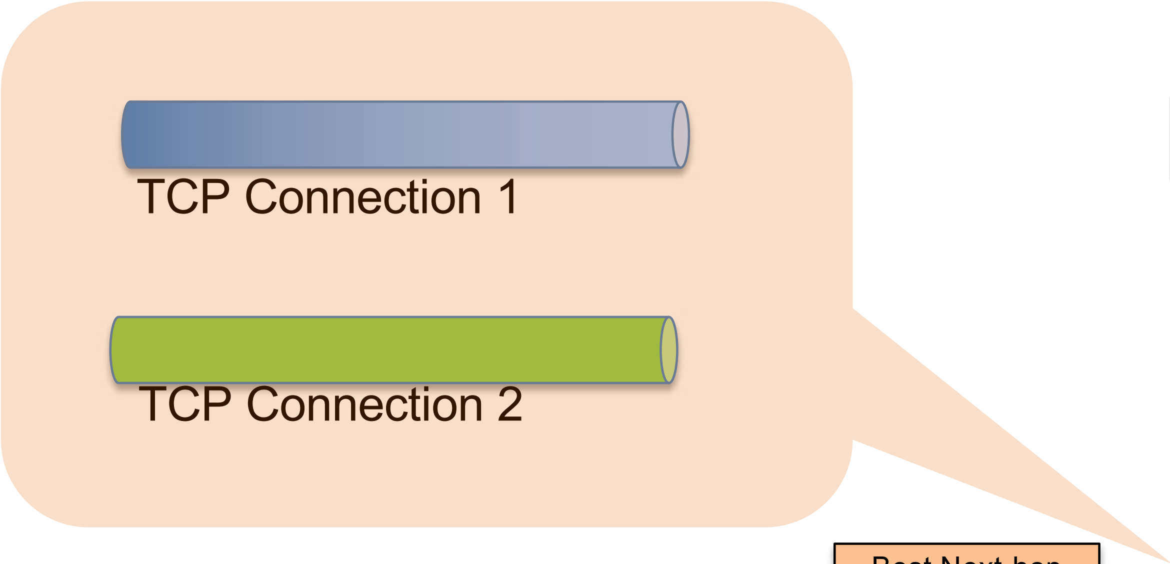
MULTIPATH-TRANSPORT-AWARE

Not Multipath Transport Aware



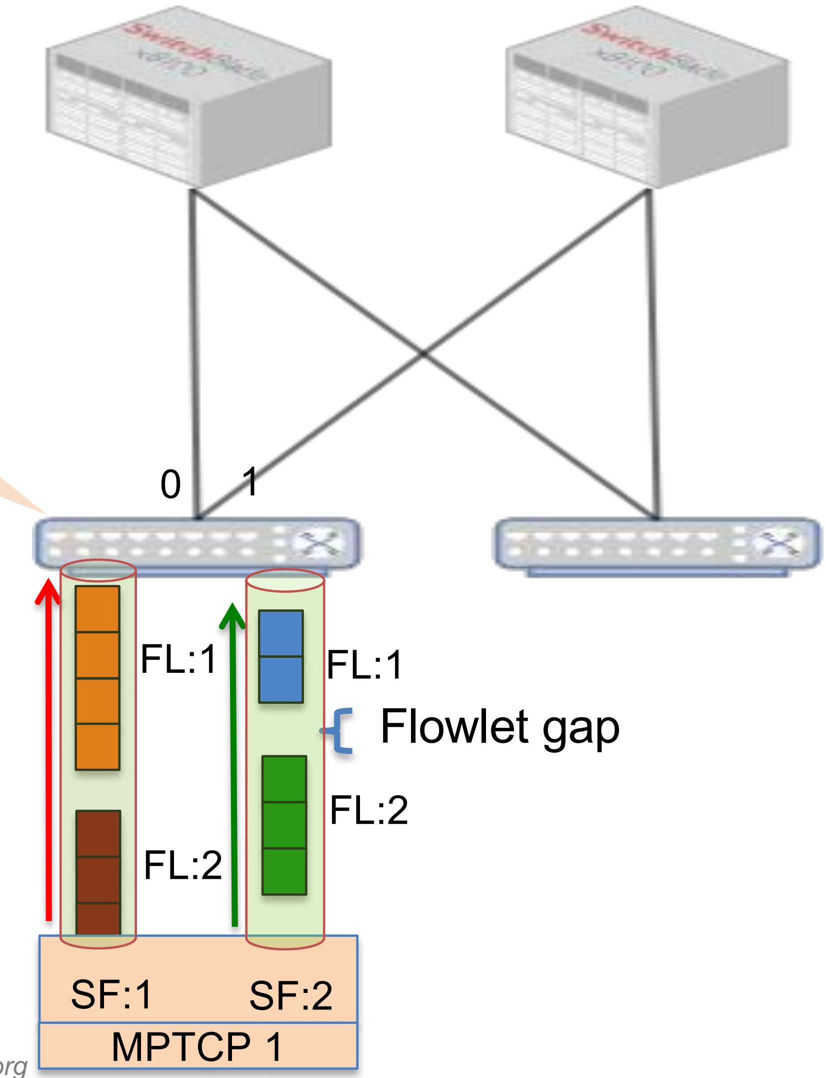
E.g. SCTP, MPTCP, QUIC

MP-HULA – Problem statement



The switch does not have contextual information about MPTCP

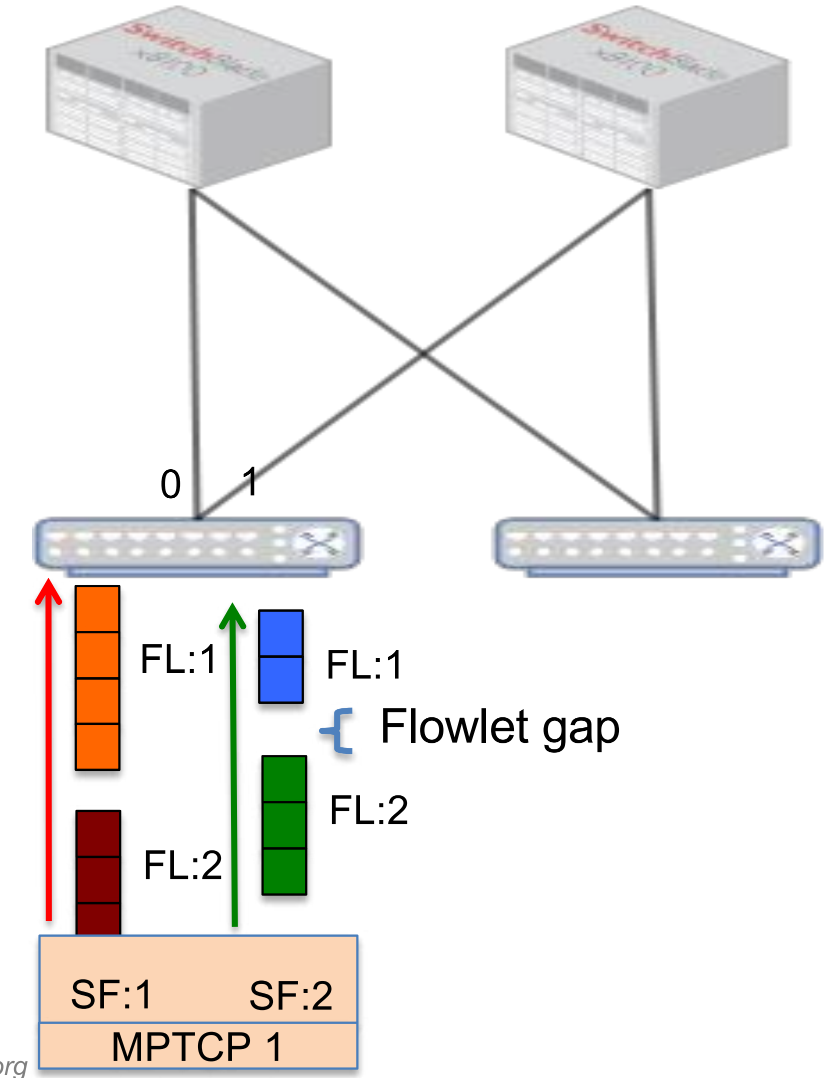
Best Next-hop
0



MP-HULA – Problem statement

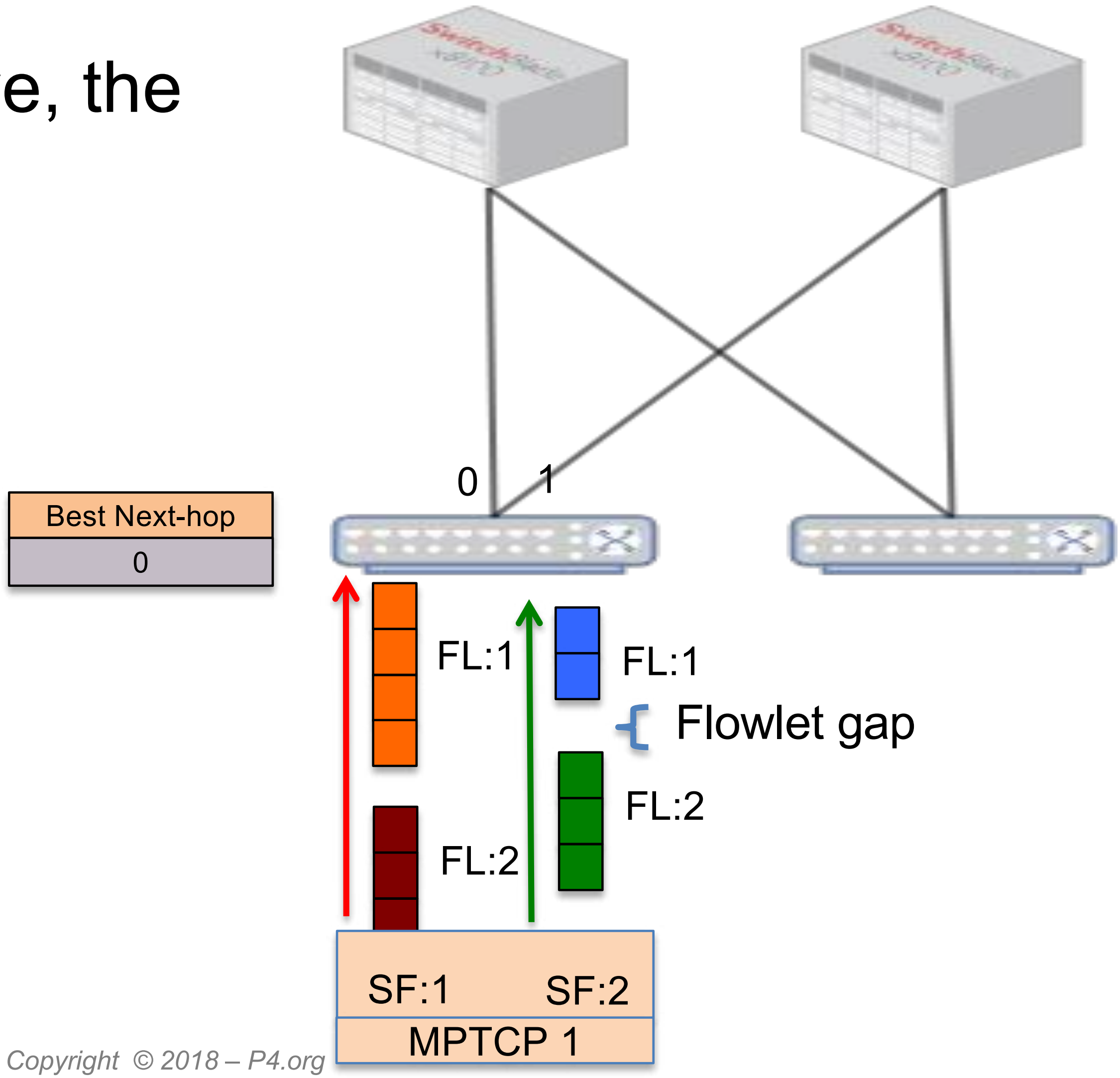
- **Most of the Load balancing schemes are not Multipath Transport Aware**
  - Sub-flows might be routed over the same path→ **bandwidth aggregation might be reduced**
  - Redundancy and persistence might be reduced if all sub-flows end-up in a failed link

Best Next-hop
0



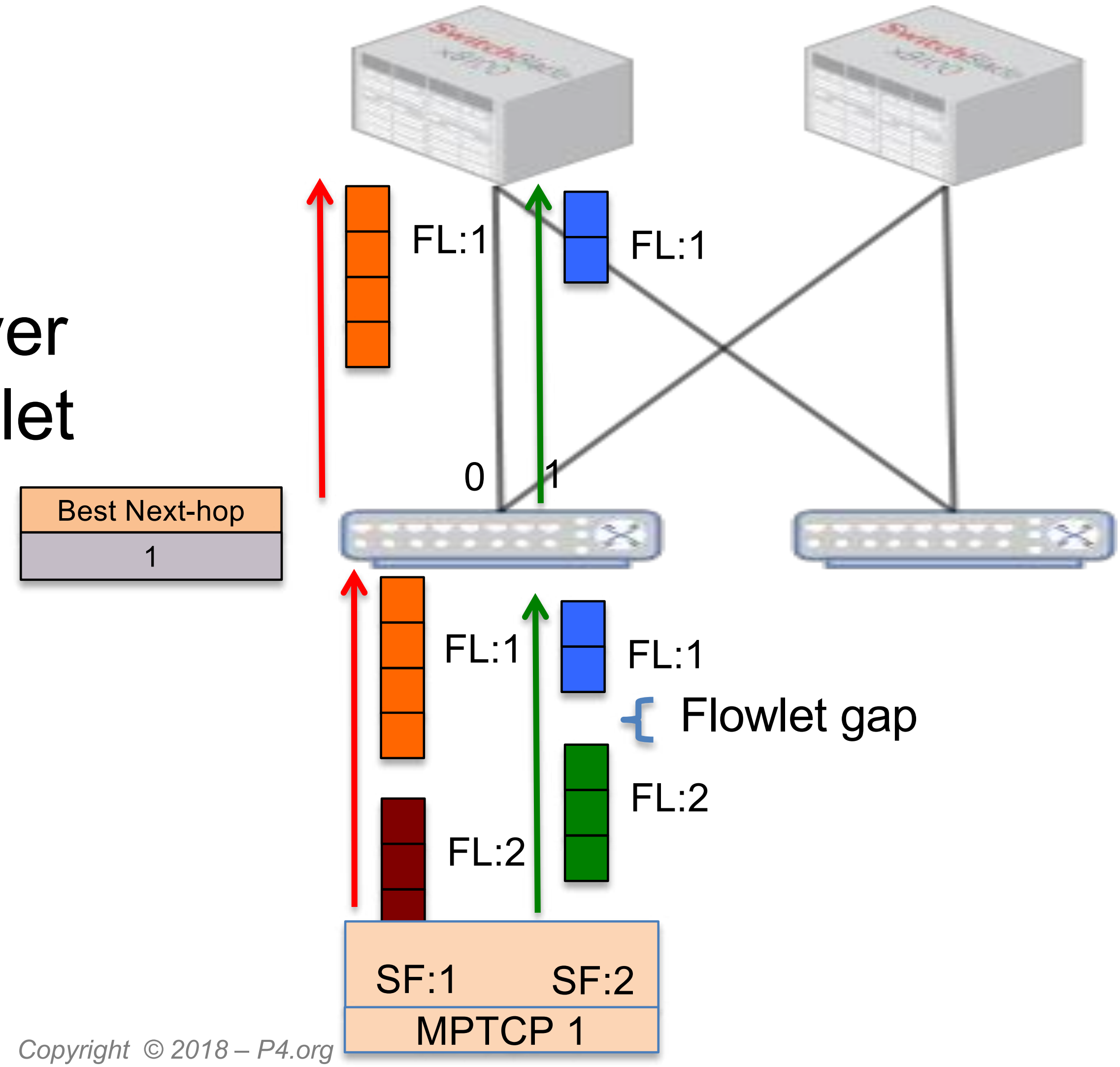
MP-HULA – Problem statement

➢ When both flowlets arrive, the best next-hop is port 0



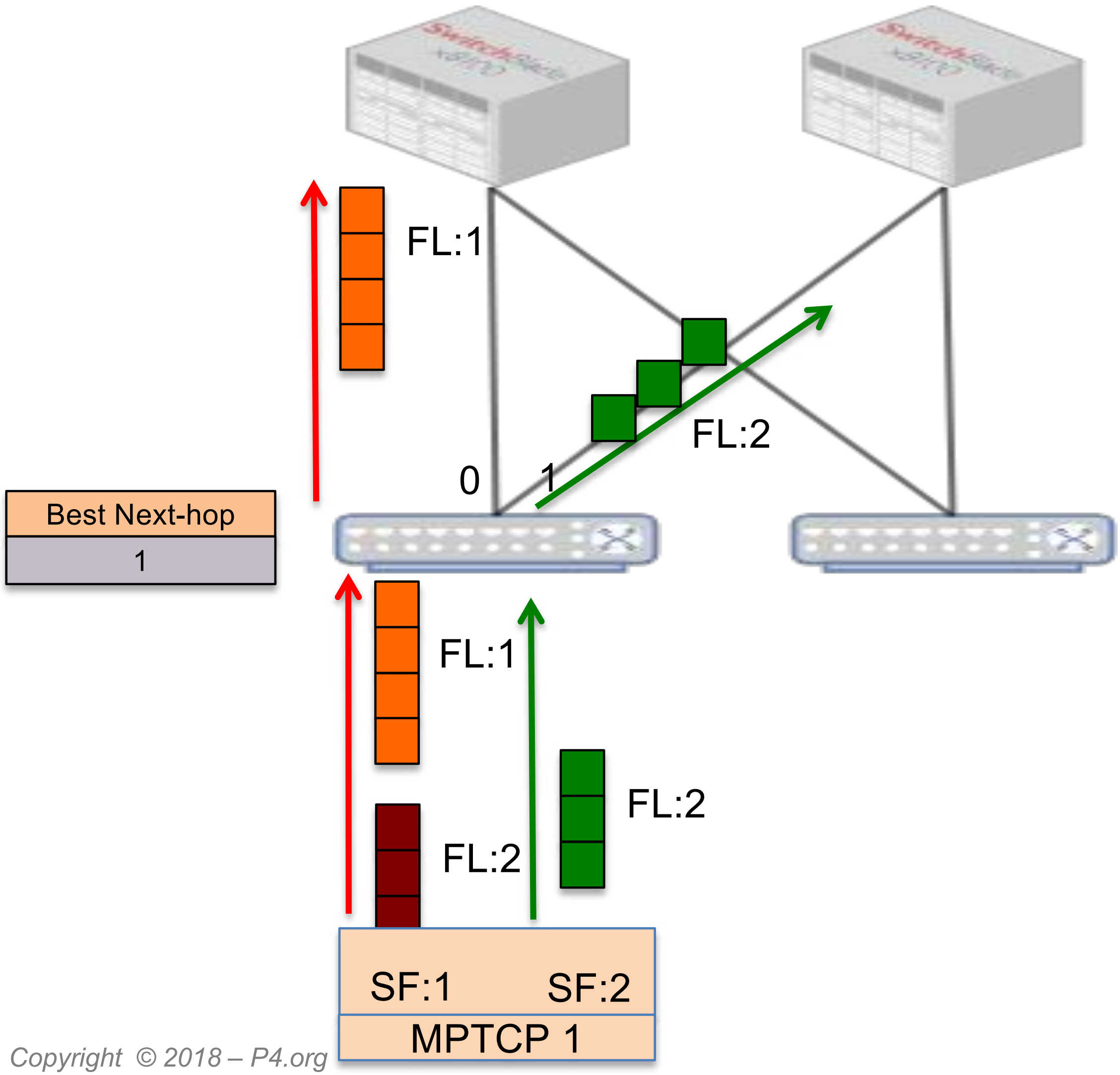
MP-HULA – Problem statement

- Both flowlets are sent over port 0. Best Next-hop is updated but flowlets are still sent over the same hop until flowlet expires



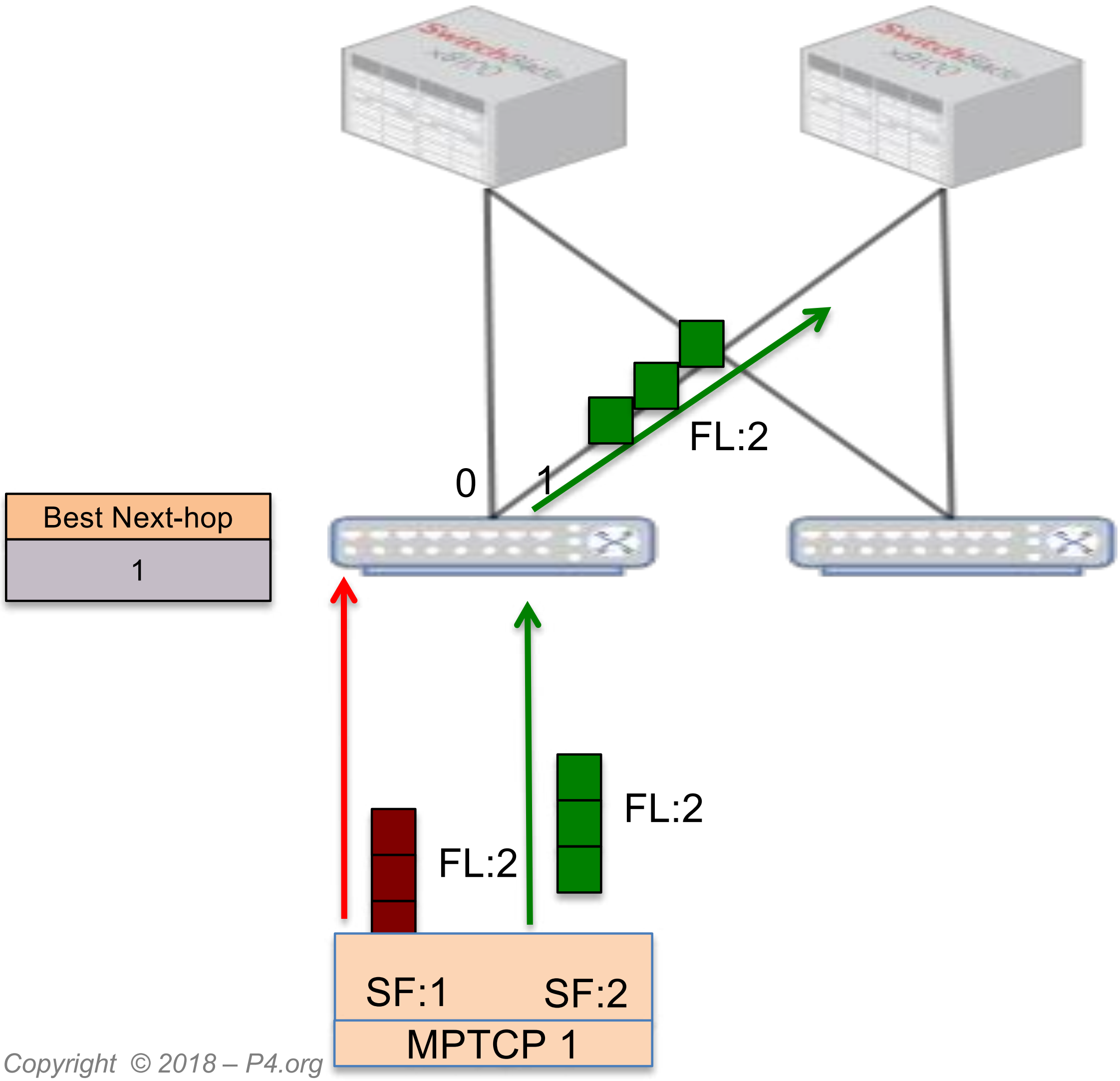
MP-HULA – Problem statement

- When the flowlet expires, the new flowlet is sent over the current best next-hop (port 1)



MP-HULA – Problem statement

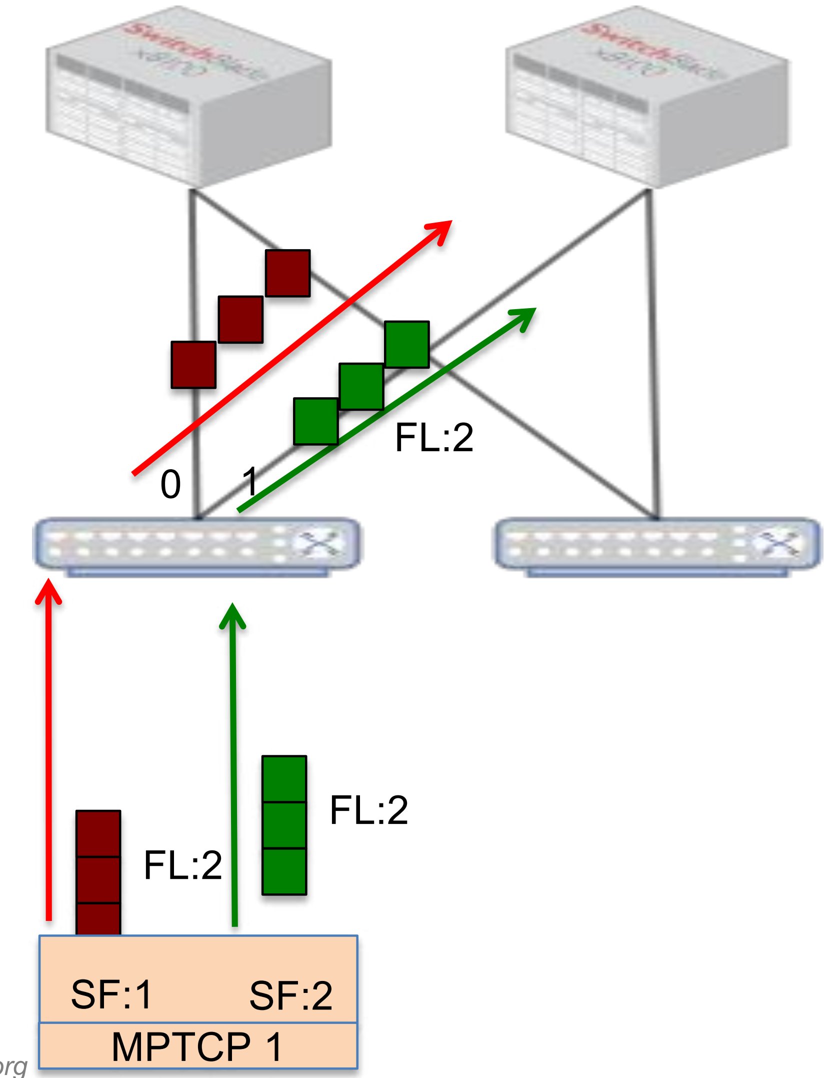
- When the flowlet expires, the new flowlet is sent over the current best next-hop (port 1)



MP-HULA – Problem statement

- Best Next-hop is port 1, so we send flowlet 2 over port 1

Best Next-hop
1

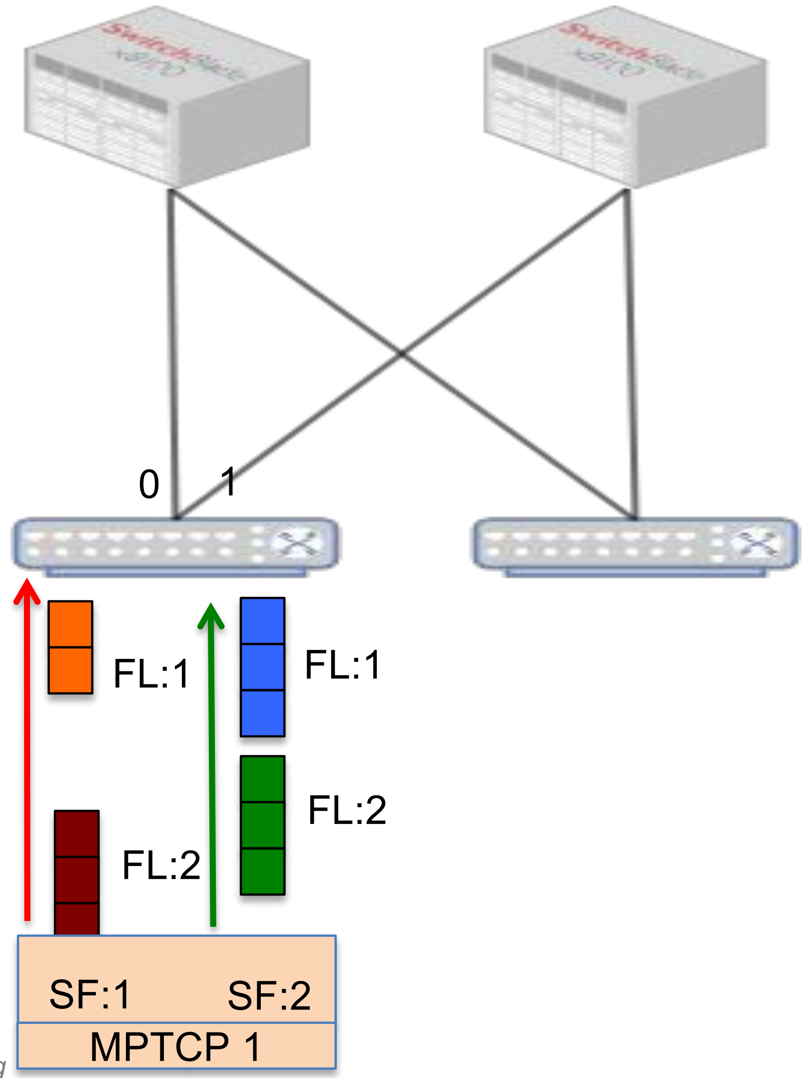


MP-HULA – Problem statement

What do we want to achieve instead?

- Bandwidth aggregation
- Redundancy & Persistence

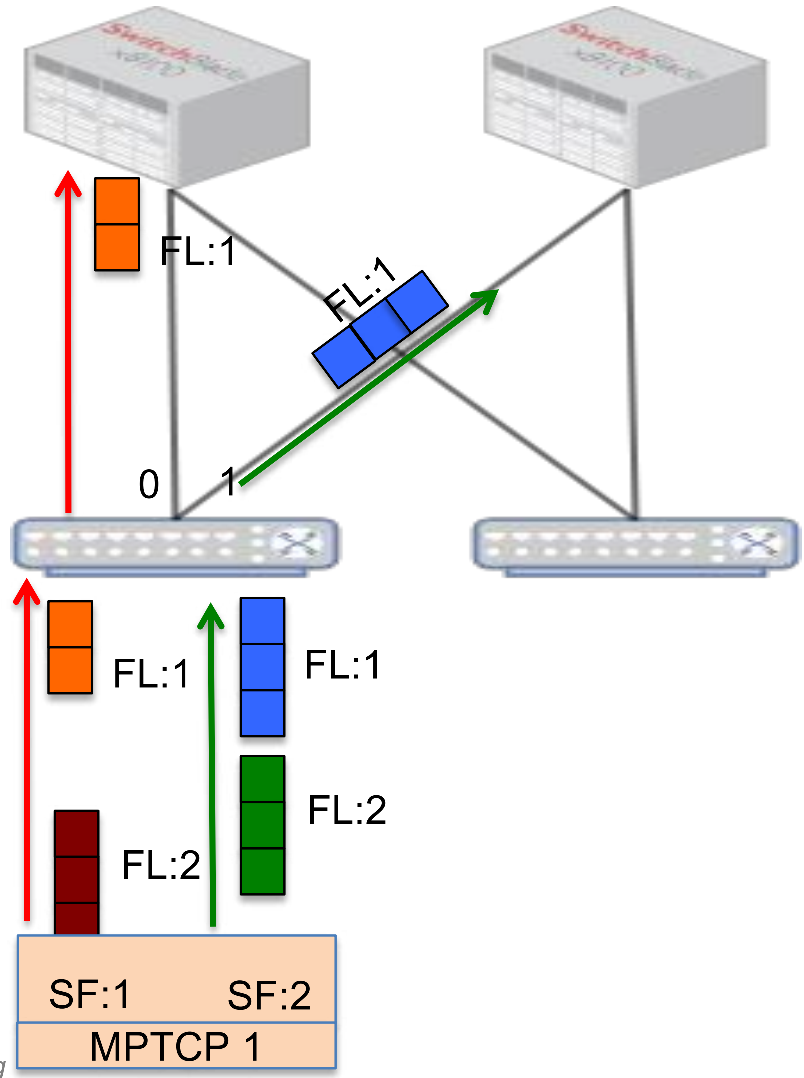
Best Next-hop
0



MP-HULA – Problem statement

What do we want to achieve instead?

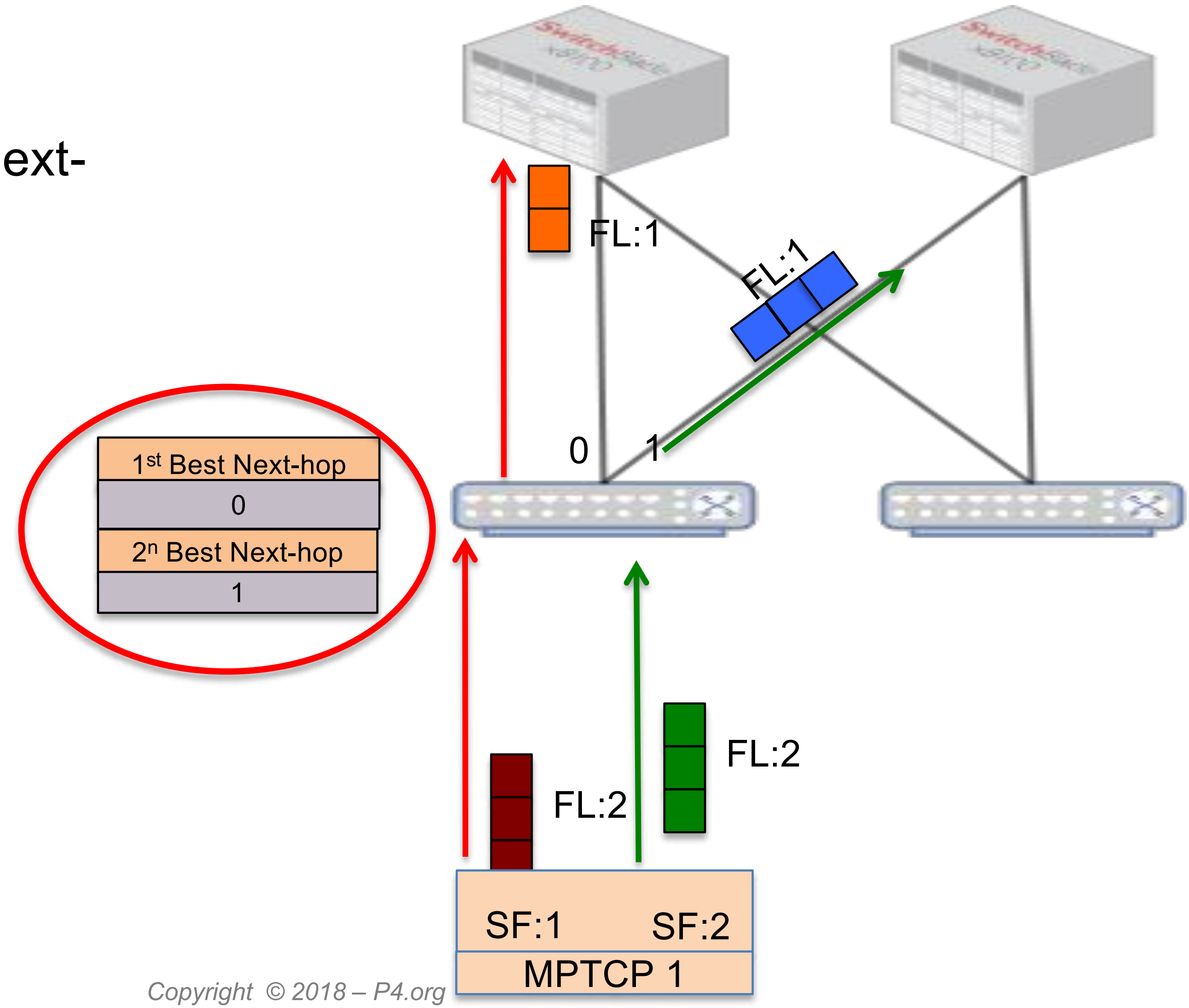
1st Best Next-hop
0
2n Best Next-hop
1



MP-HULA – Problem statement

How can we do it?

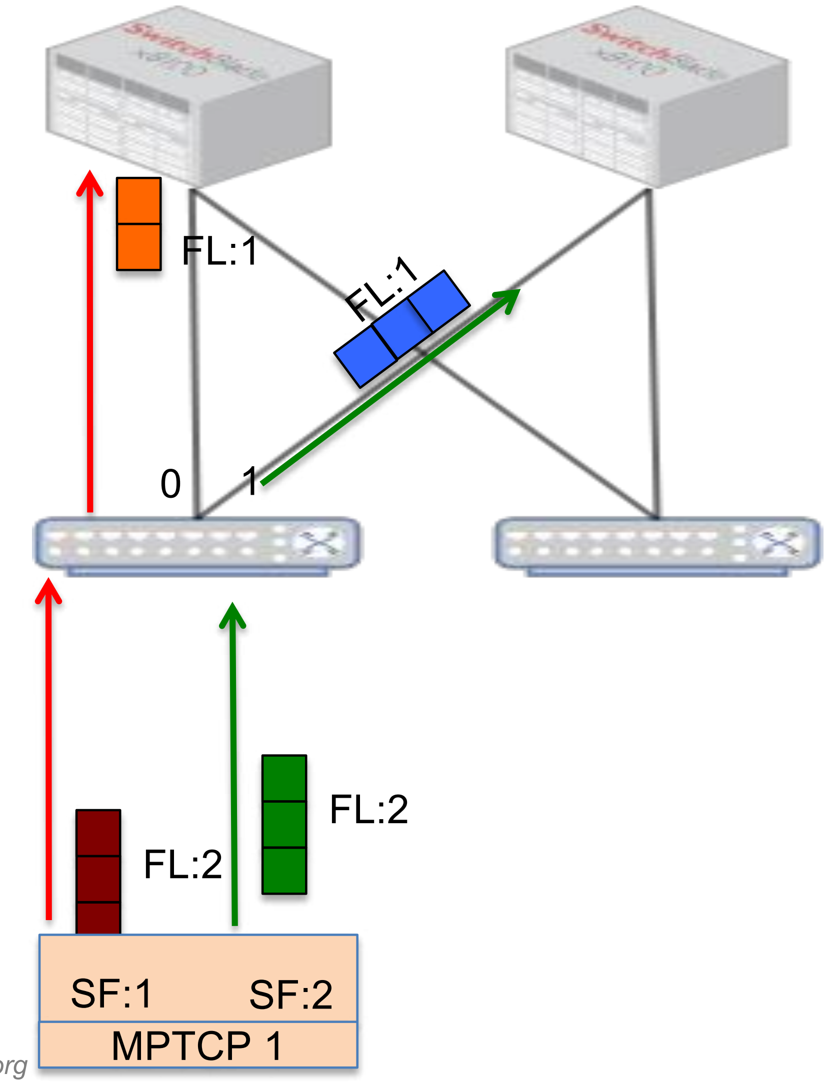
1) Tracking not only the best next-hop but k-best hops



MP-HULA – Problem statement

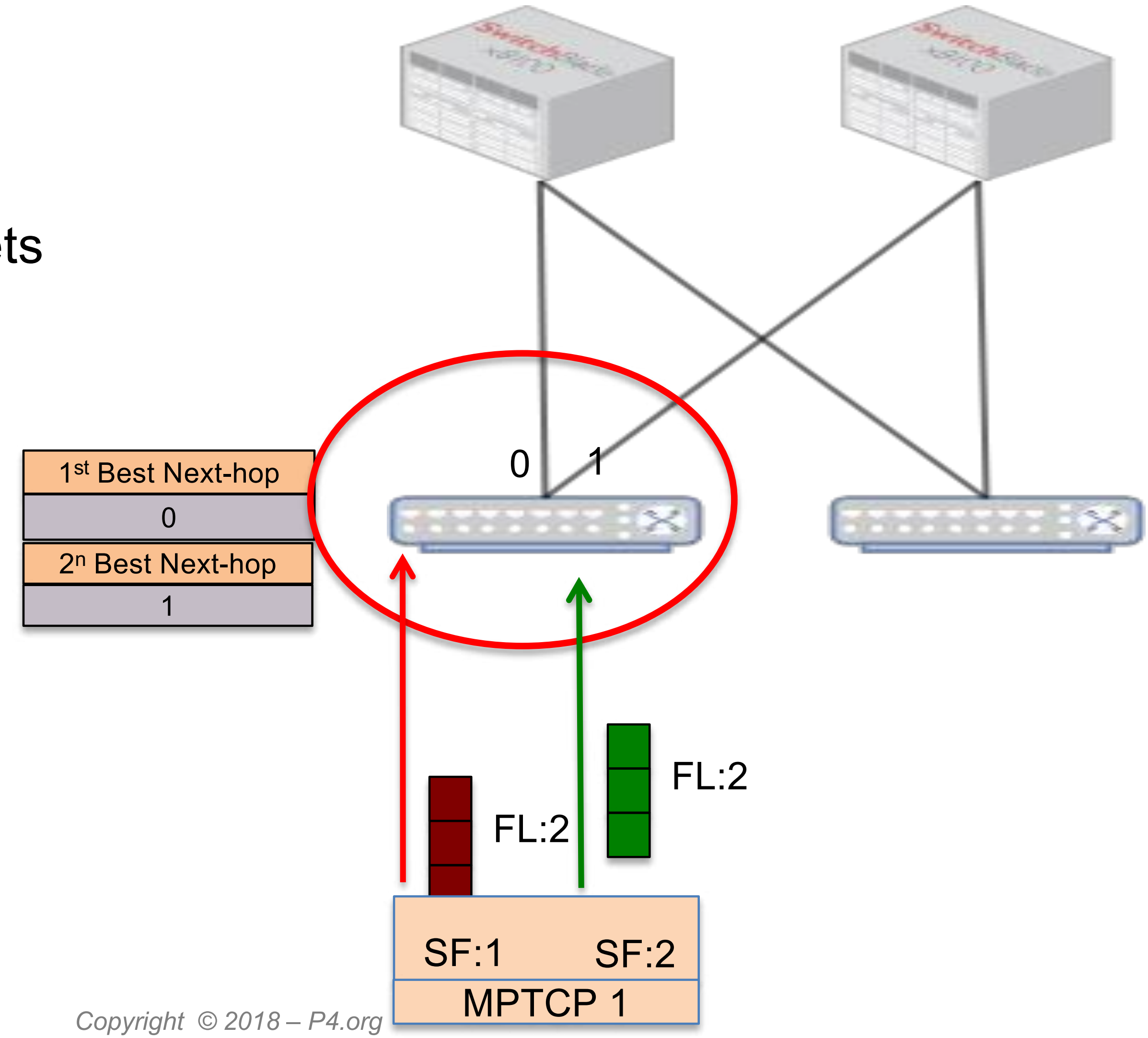
2) Identifying the MPTCP session and sub-flows to send their flowlets over different ports

1st Best Next-hop
0
2n Best Next-hop
1



MP-HULA – Problem statement

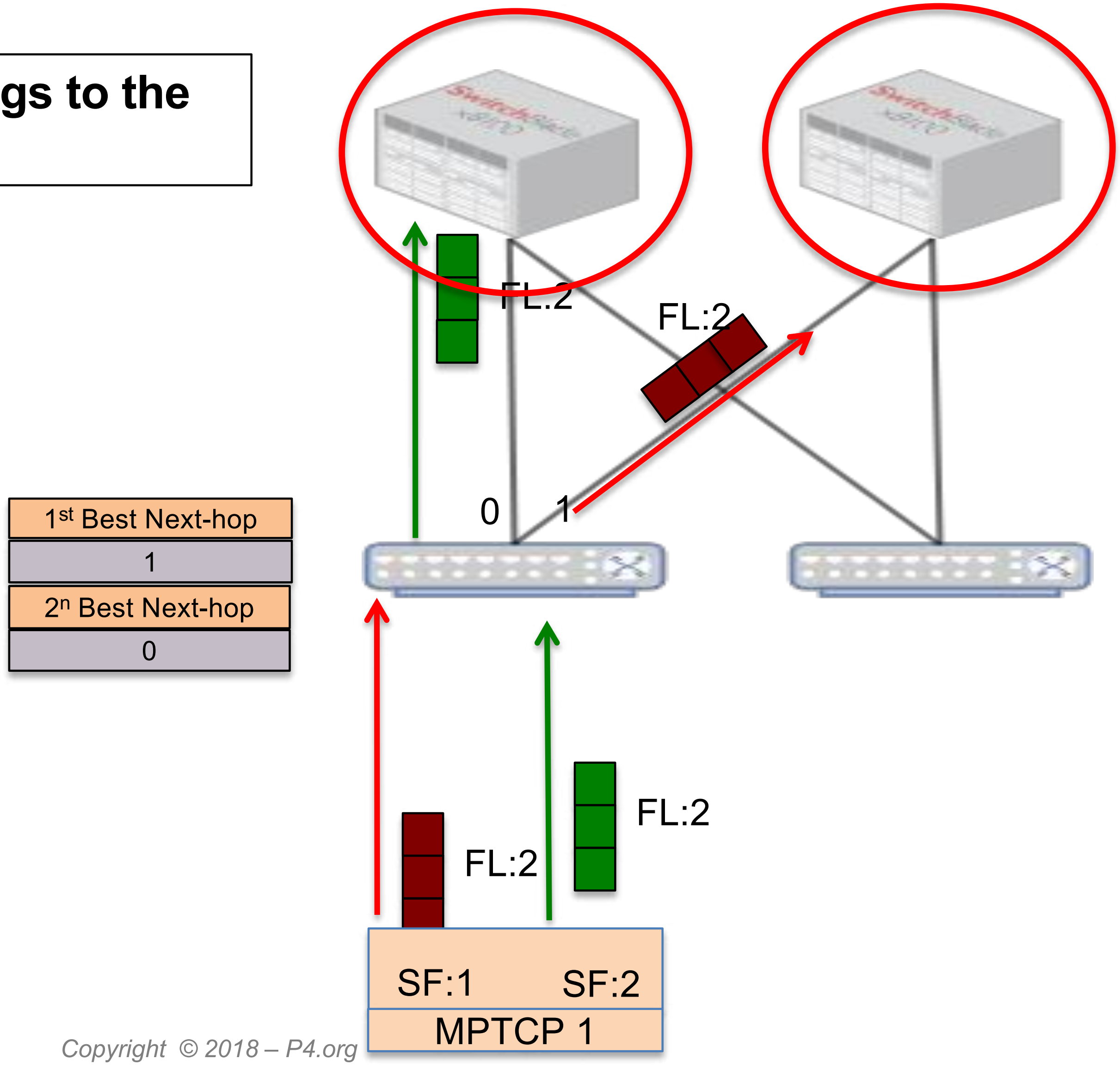
2) Identifying the MPTCP session and sub-flows to send their flowlets over different ports



MP-HULA – Problem statement

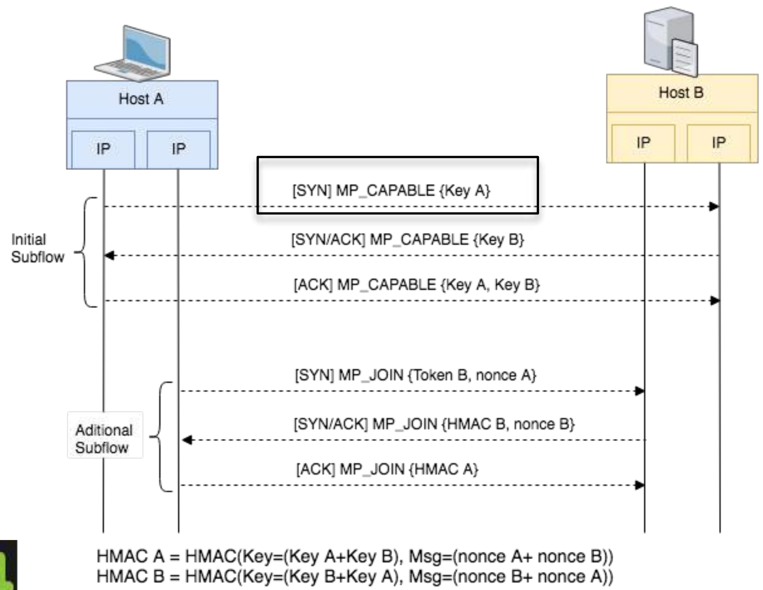
Not aware that this flowlet belongs to the same MPTCP connection

3) Mark sub-flows belonging to a specific MPTCP session

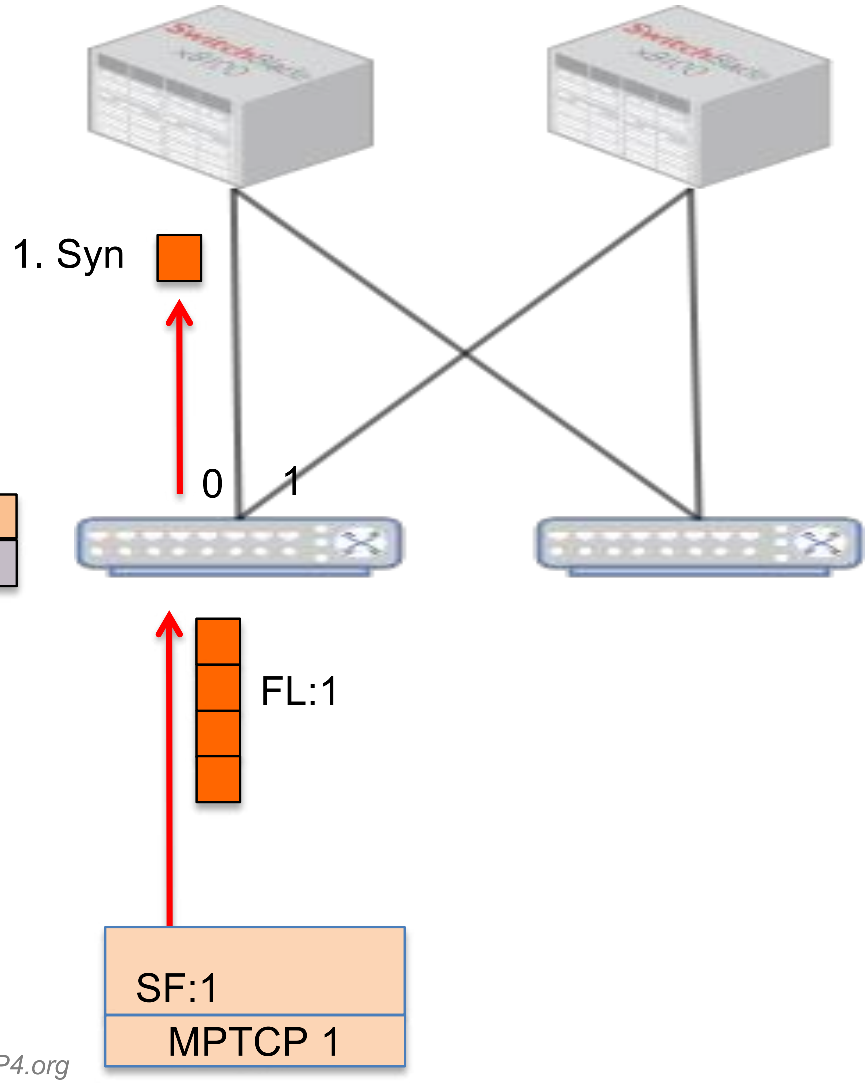


MP-HULA – MPTCP Identification Problem

- MPTCP spreads application data over multiple sub-flows
- MPTCP in general improves fairness, throughput and robustness
- Beneficial for long flows (elephant flows)

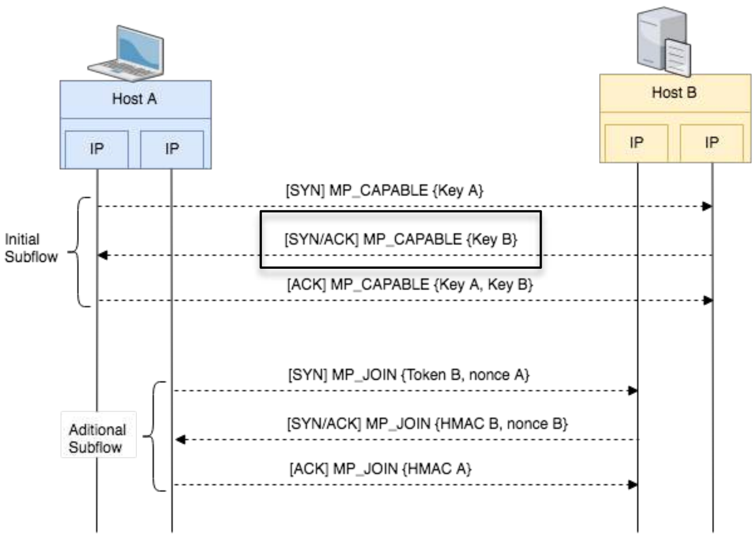


Best Next-hop
0



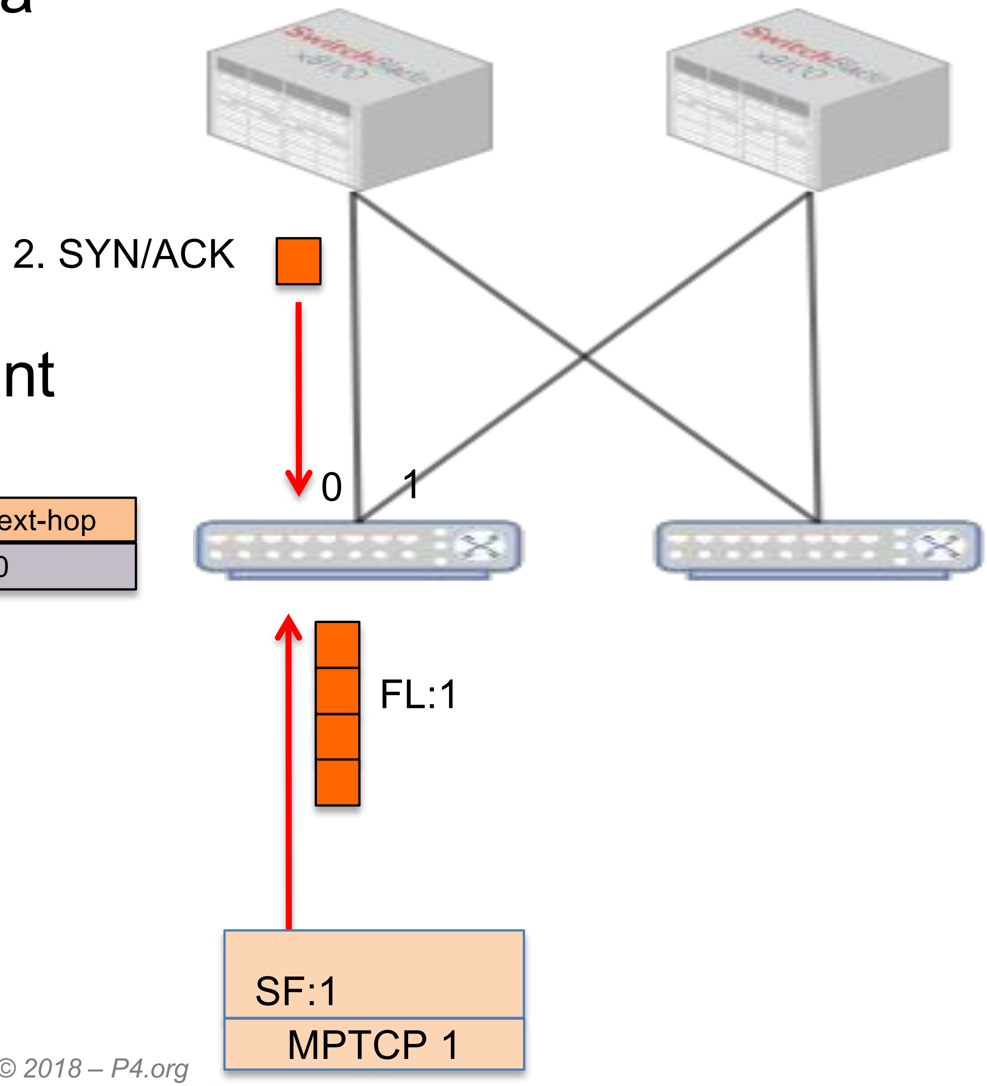
MP-HULA – MPTCP Identification Problem

- MPTCP spreads application data over multiple sub-flows
- MPTCP in general improves fairness, throughput and robustness
- Beneficial for long flows (elephant flows)



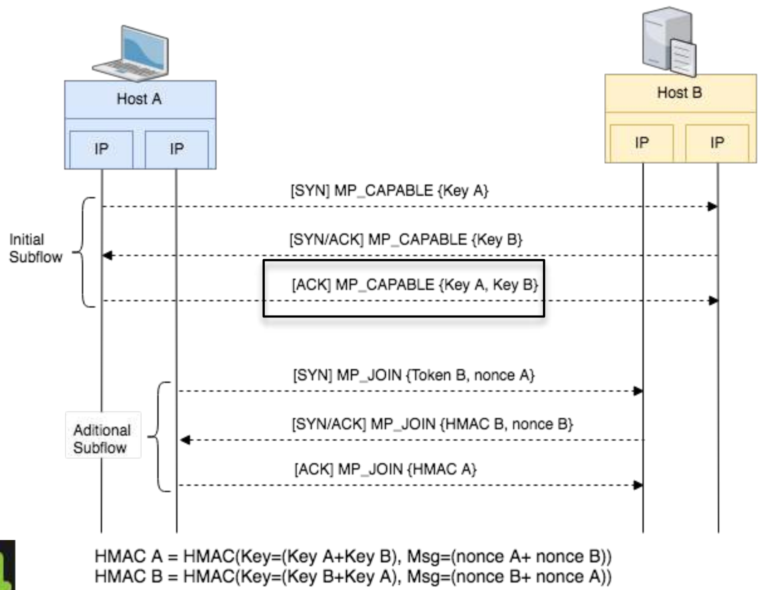
HMAC A = HMAC(Key=(Key A+Key B), Msg=(nonce A+ nonce B))
HMAC B = HMAC(Key=(Key B+Key A), Msg=(nonce B+ nonce A))

Best Next-hop
0

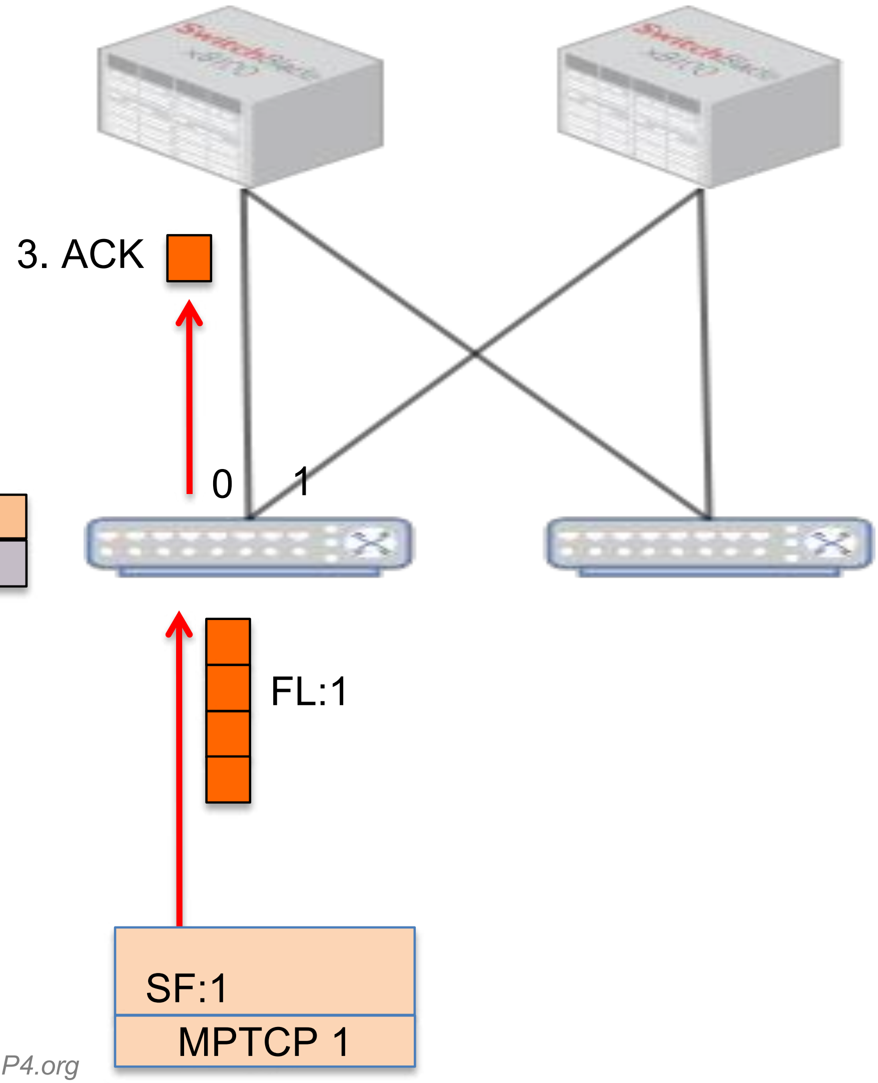


MP-HULA – MPTCP Identification Problem

MPTCP sender/receiver generates token A and B from {Key A} and {Key B} for authentication

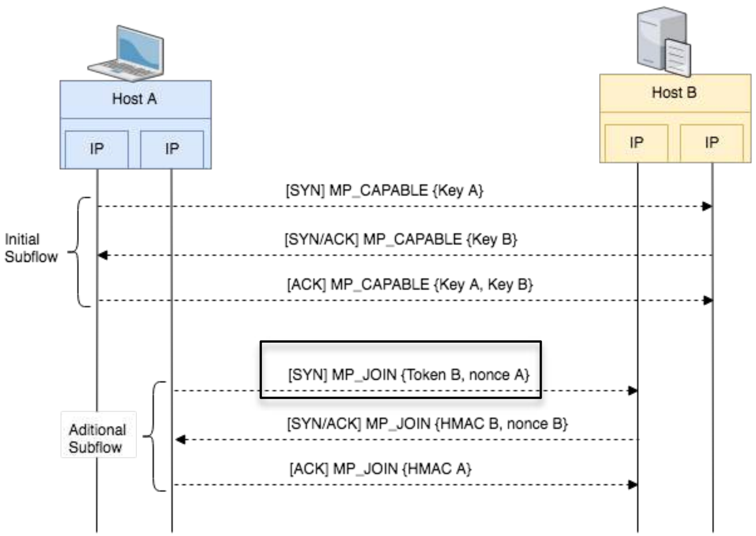


Best Next-hop
0



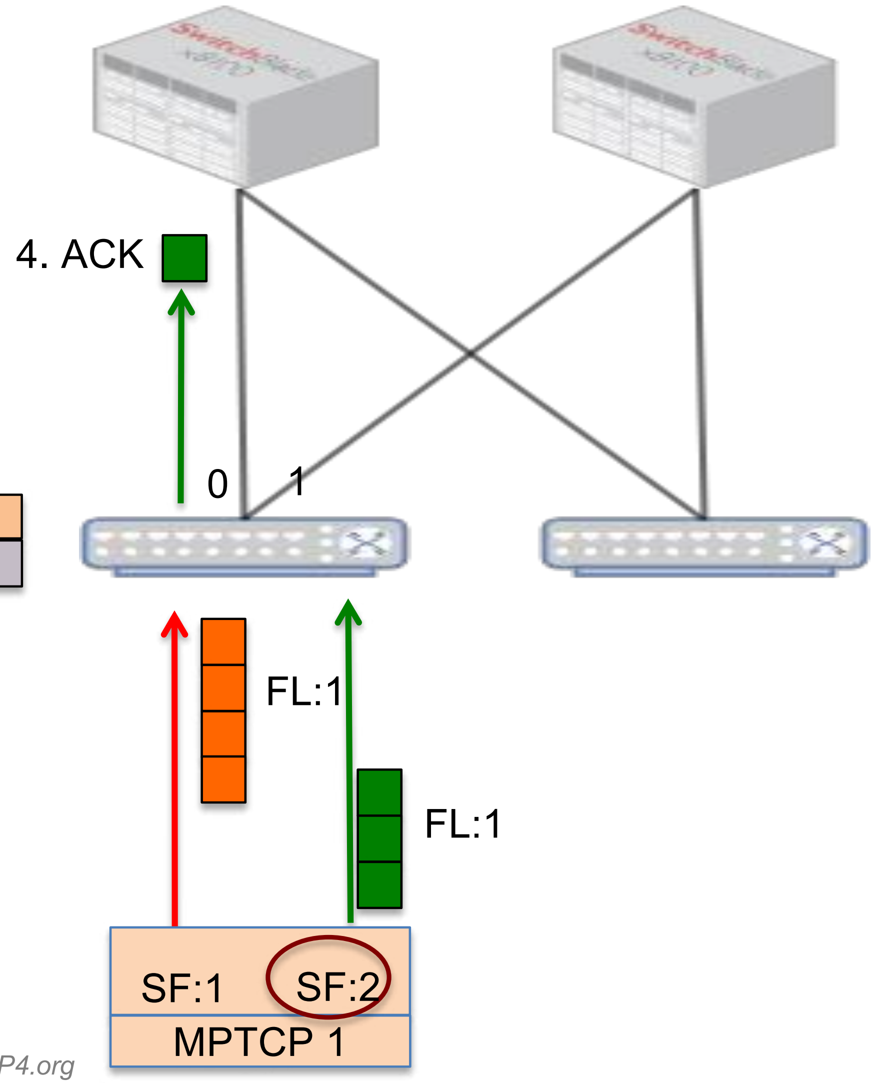
MP-HULA – MPTCP Identification Problem

Sender MPTCP A sends the generated Token B and a random number (nonce)



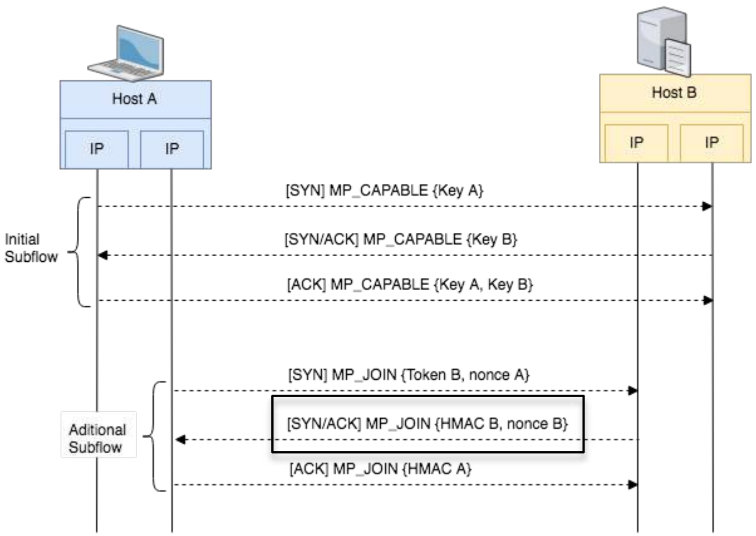
HMAC A = HMAC(Key=(Key A+Key B), Msg=(nonce A+ nonce B))
HMAC B = HMAC(Key=(Key B+Key A), Msg=(nonce B+ nonce A))

Best Next-hop
0



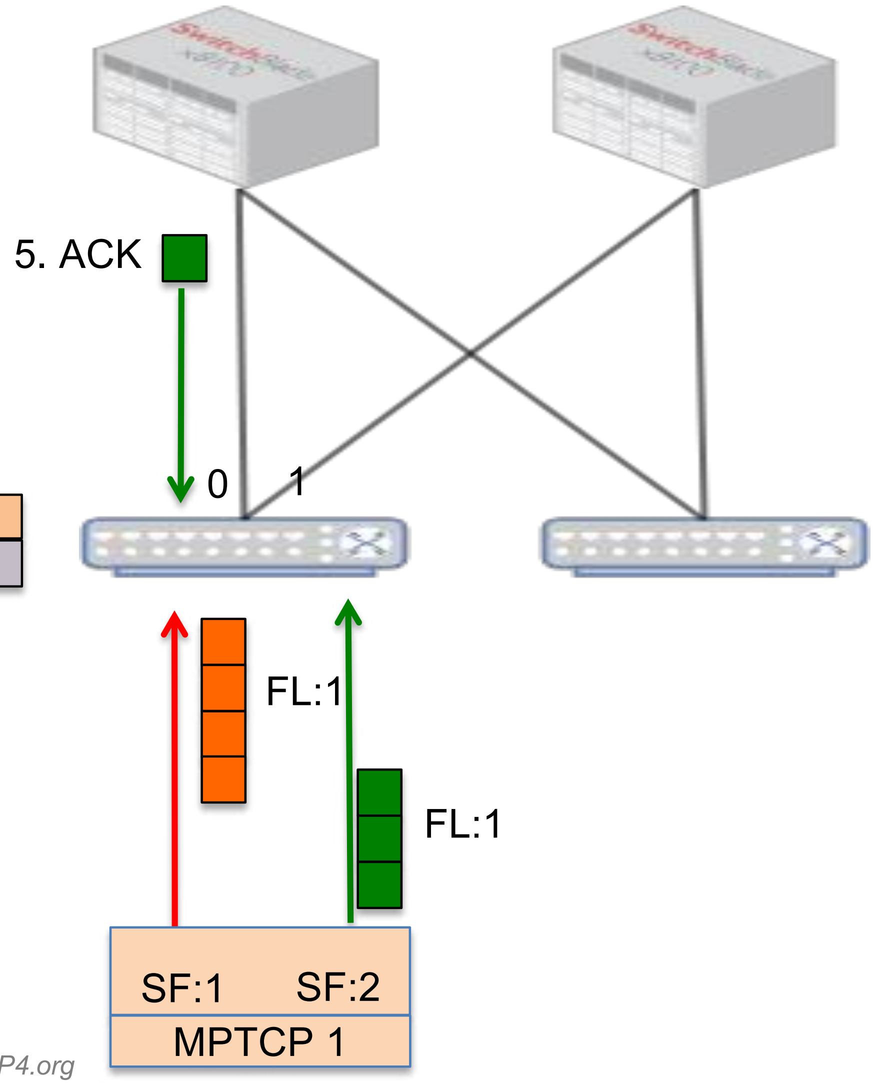
MP-HULA – MPTCP Identification Problem

MPTCP receives the generated Token A and validates it.



HMAC A = HMAC(Key=(Key A+Key B), Msg=(nonce A+ nonce B))
HMAC B = HMAC(Key=(Key B+Key A), Msg=(nonce B+ nonce A))

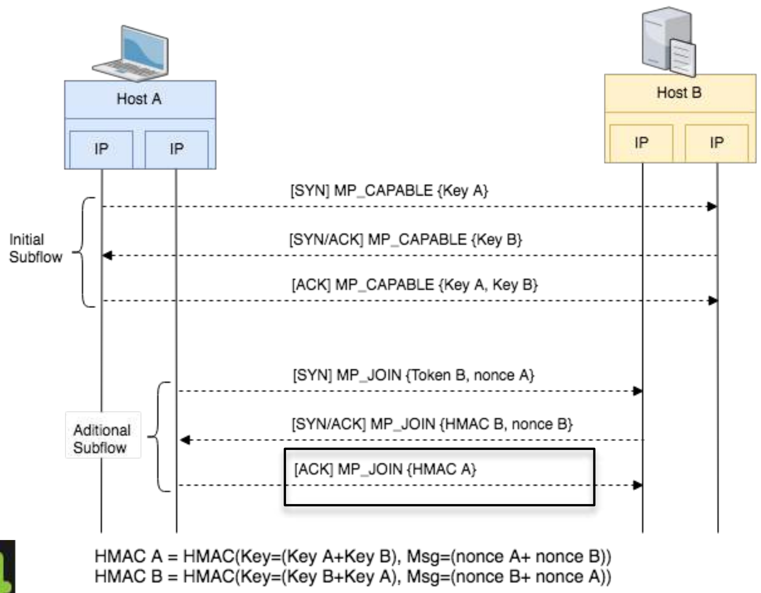
Best Next-hop
0



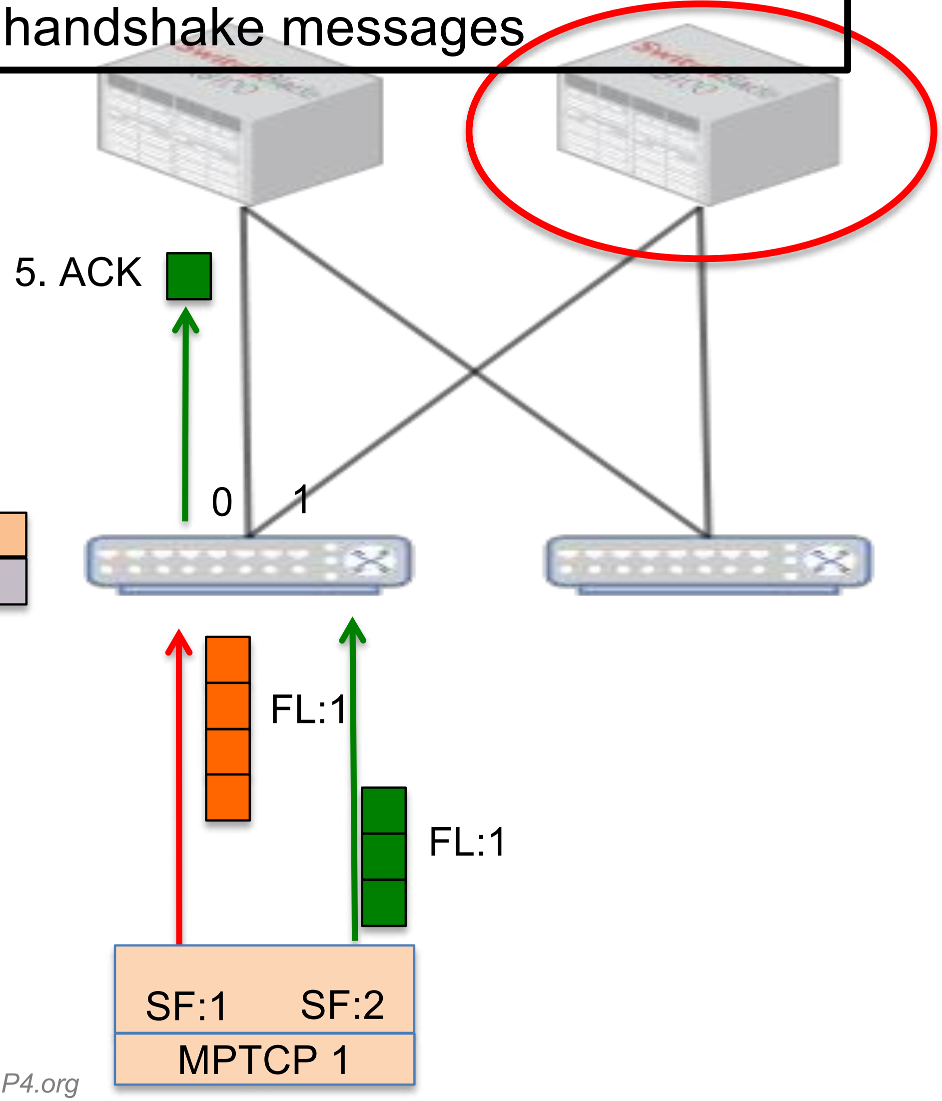
MP-HULA – MPTCP Identification Problem

MPTCP sends the generated authentication code HMAC A and the connection is initiated.

This node is not aware of the 3-handshake messages

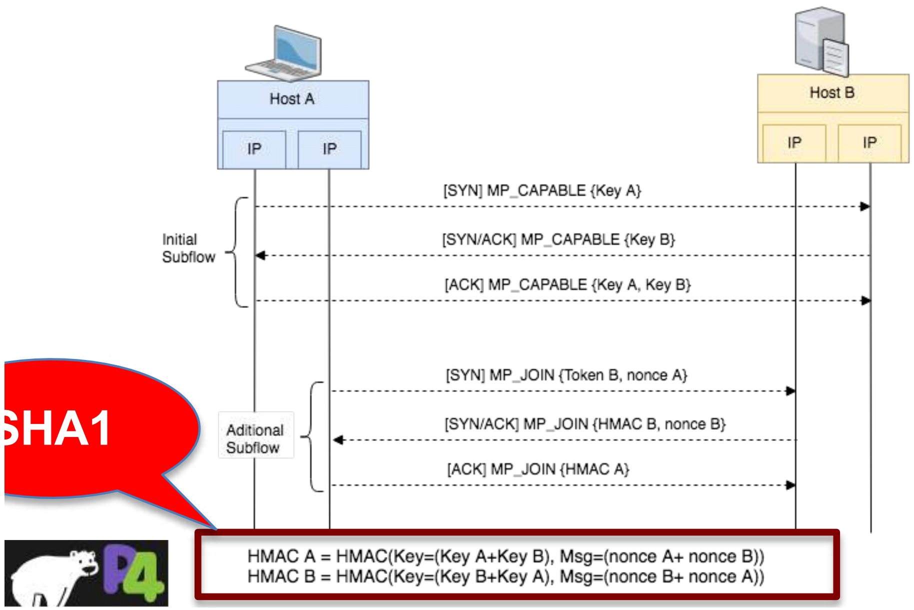


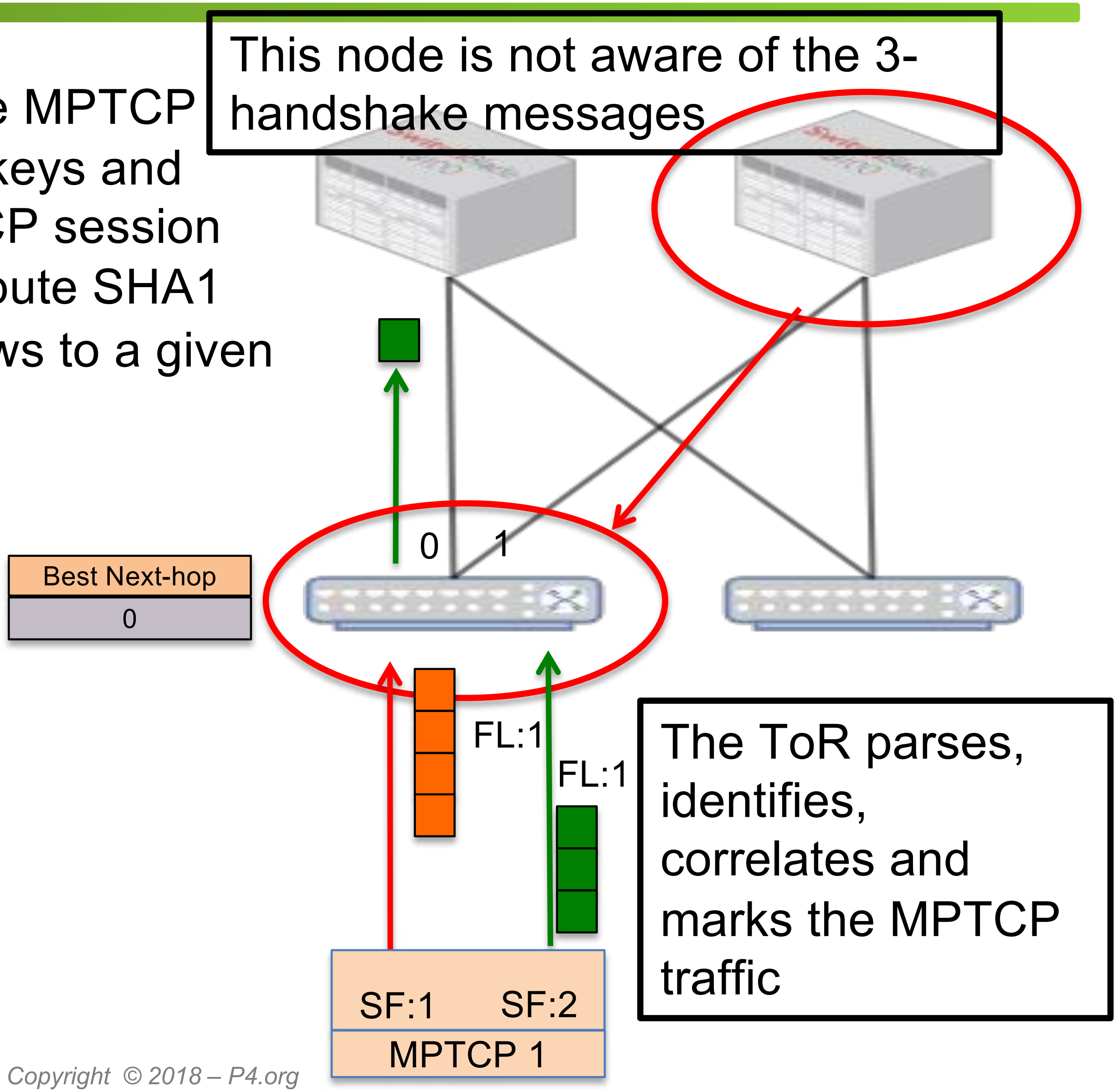
Best Next-hop
0



MP-HULA – Identification and Correlation

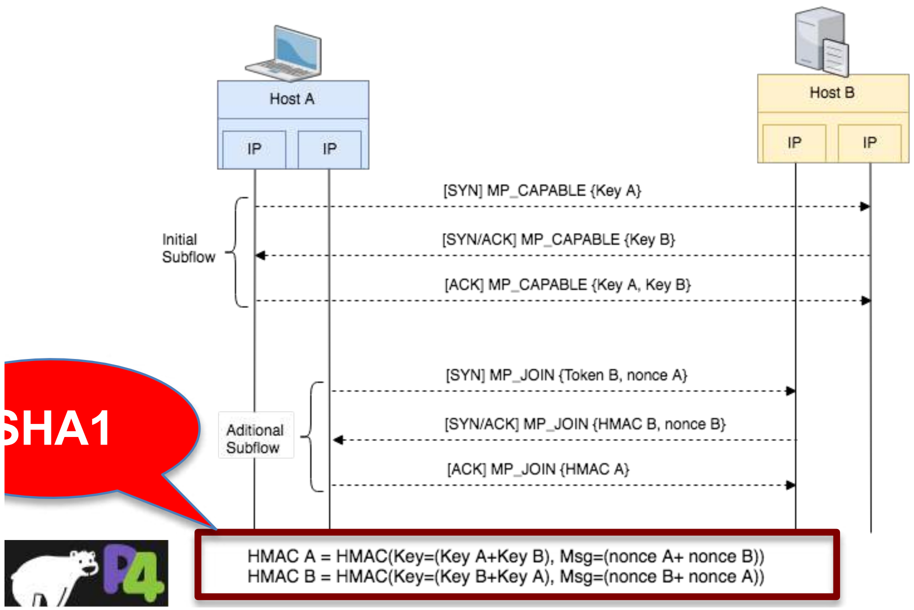
- (1) Parse - The ToR parses the MPTCP option messages carrying the keys and tokens to (2) identify the MPTCP session using external function to compute SHA1
- (3) The ToR correlates sub-flows to a given MPTCP connection

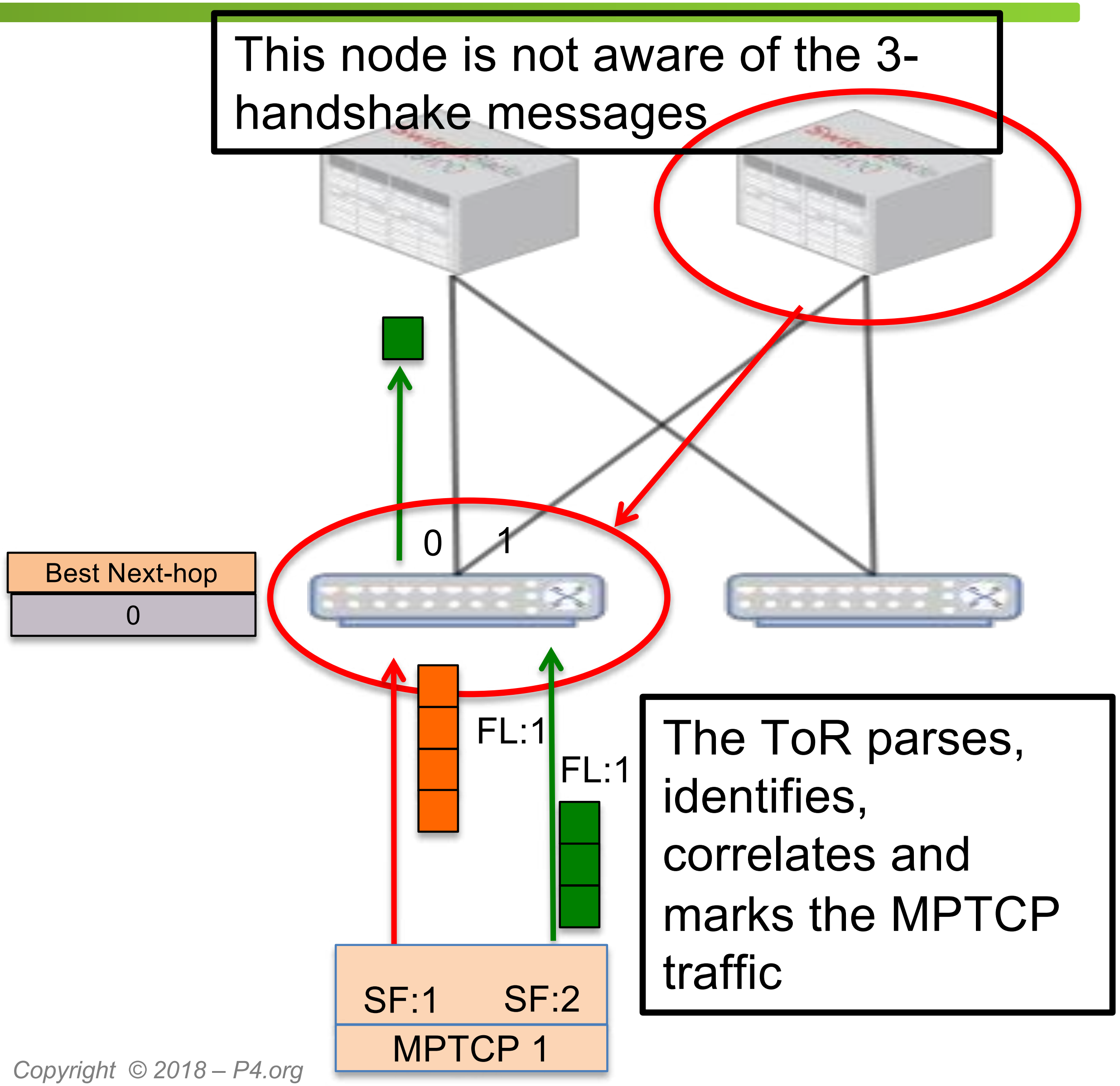




MP-HULA – Identification and Correlation

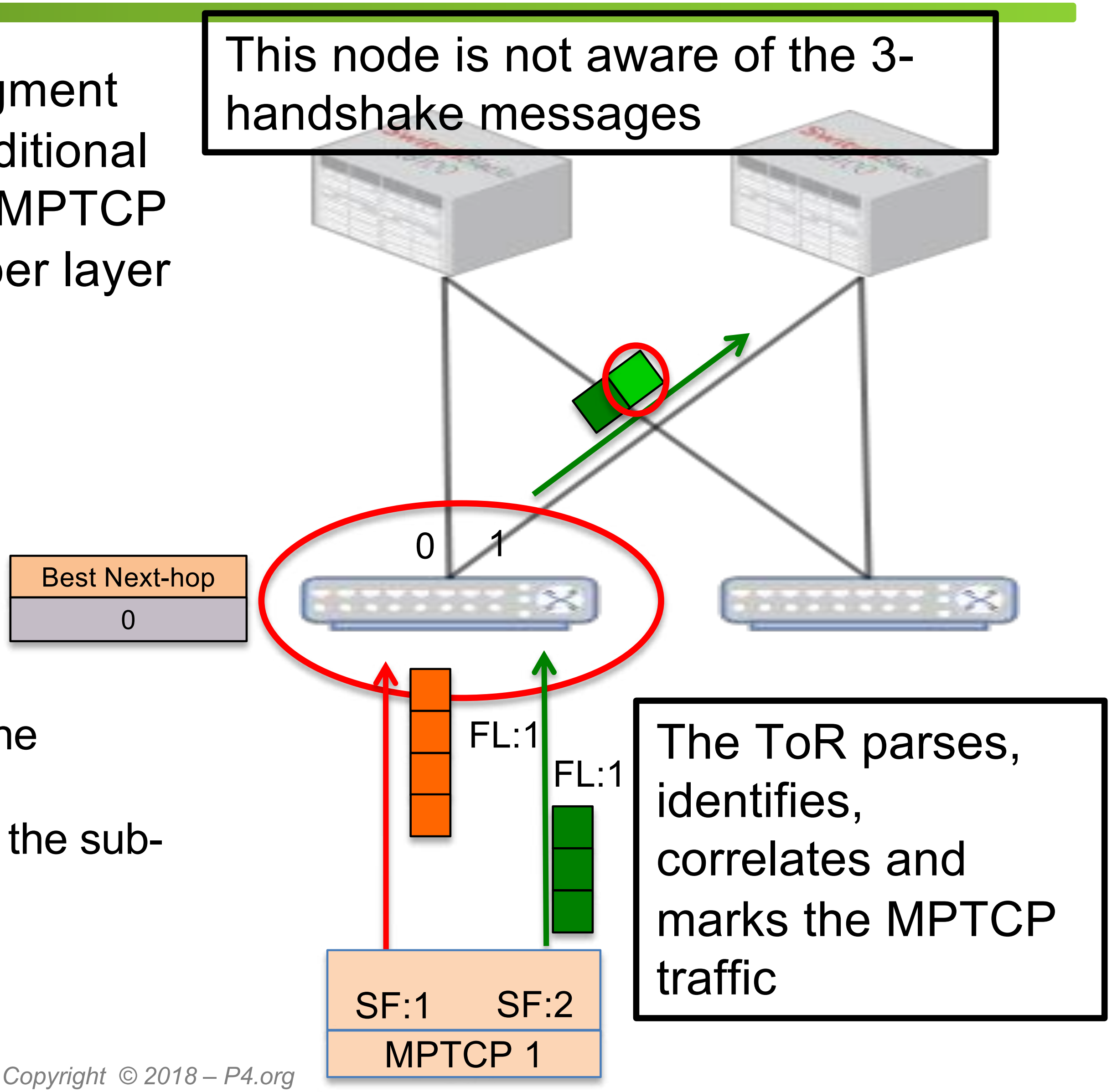
P4 primitives
Programmable Parsing
RW packet metadata
RW access to stateful memory
Comparison/arithmetic operators
External function





MP-HULA – Marking

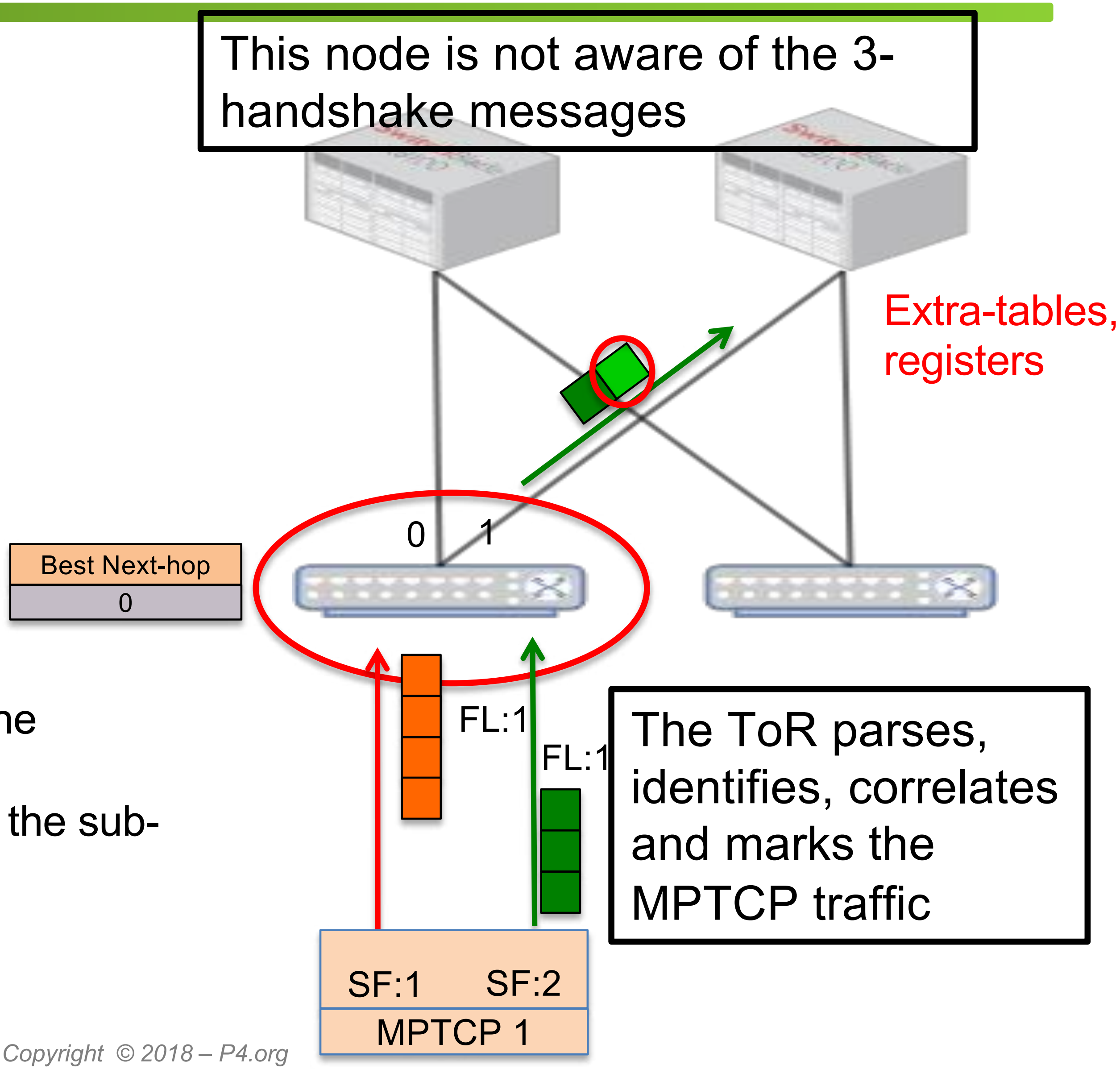
- (4) Marking - ToR needs to augment MPTCP data packets by an additional header to uniquely identify the MPTCP connection and sub-flow to upper layer switches.



- **MPTCP_ID (64 bits)** to identify the MPTCP connection
- **Sub-flow_num(4bits)** to identify the sub-flow number within the MPTCP connection

MP-HULA – Marking

P4 primitives
New header format
RW packet metadata
RW access to stateful memory

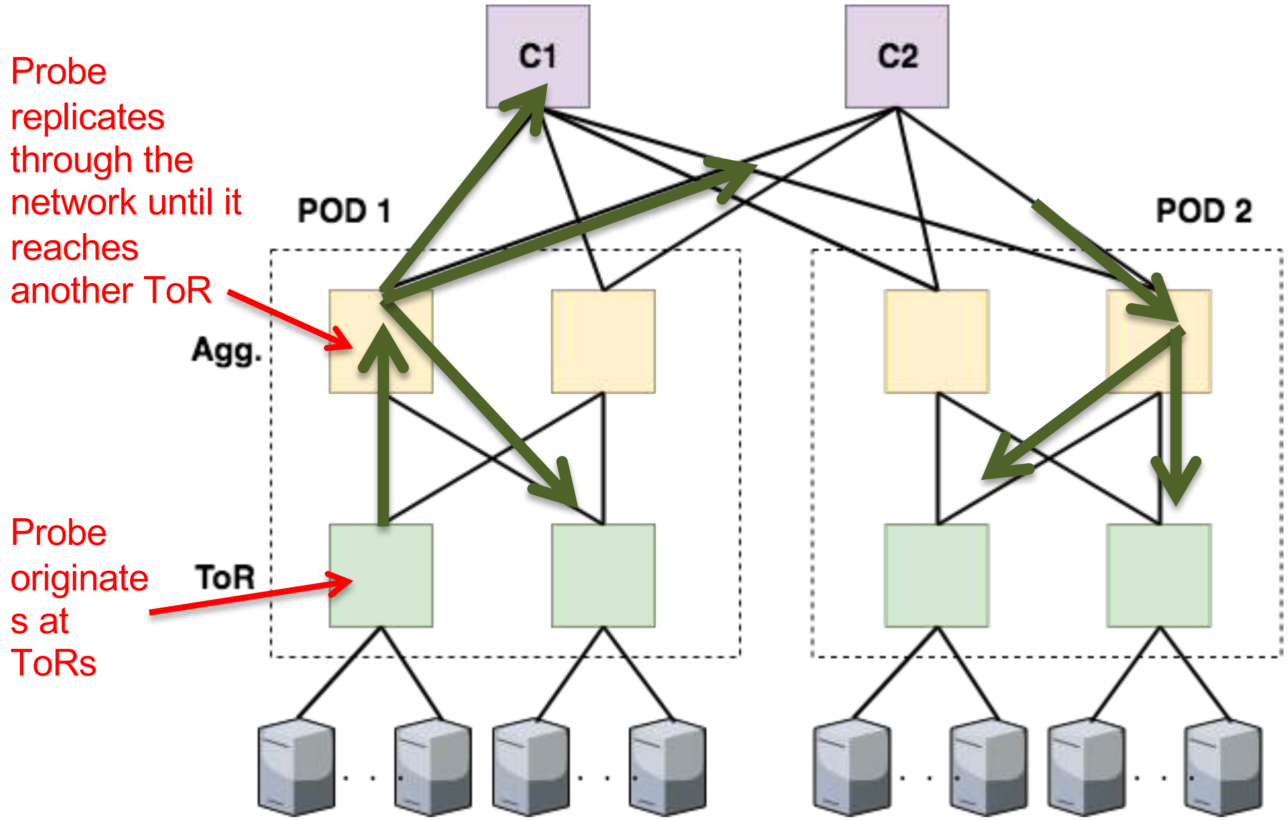


- **MPTCP_ID (64 bits)** to identify the MPTCP connection
- **Sub-flow_num(4bits)** to identify the sub-flow number within the MPTCP connection

Our Approach – MP-HULA

- **MP-HULA Probe Processing**
  - Extended HULA approach to collect **k-**path utilization

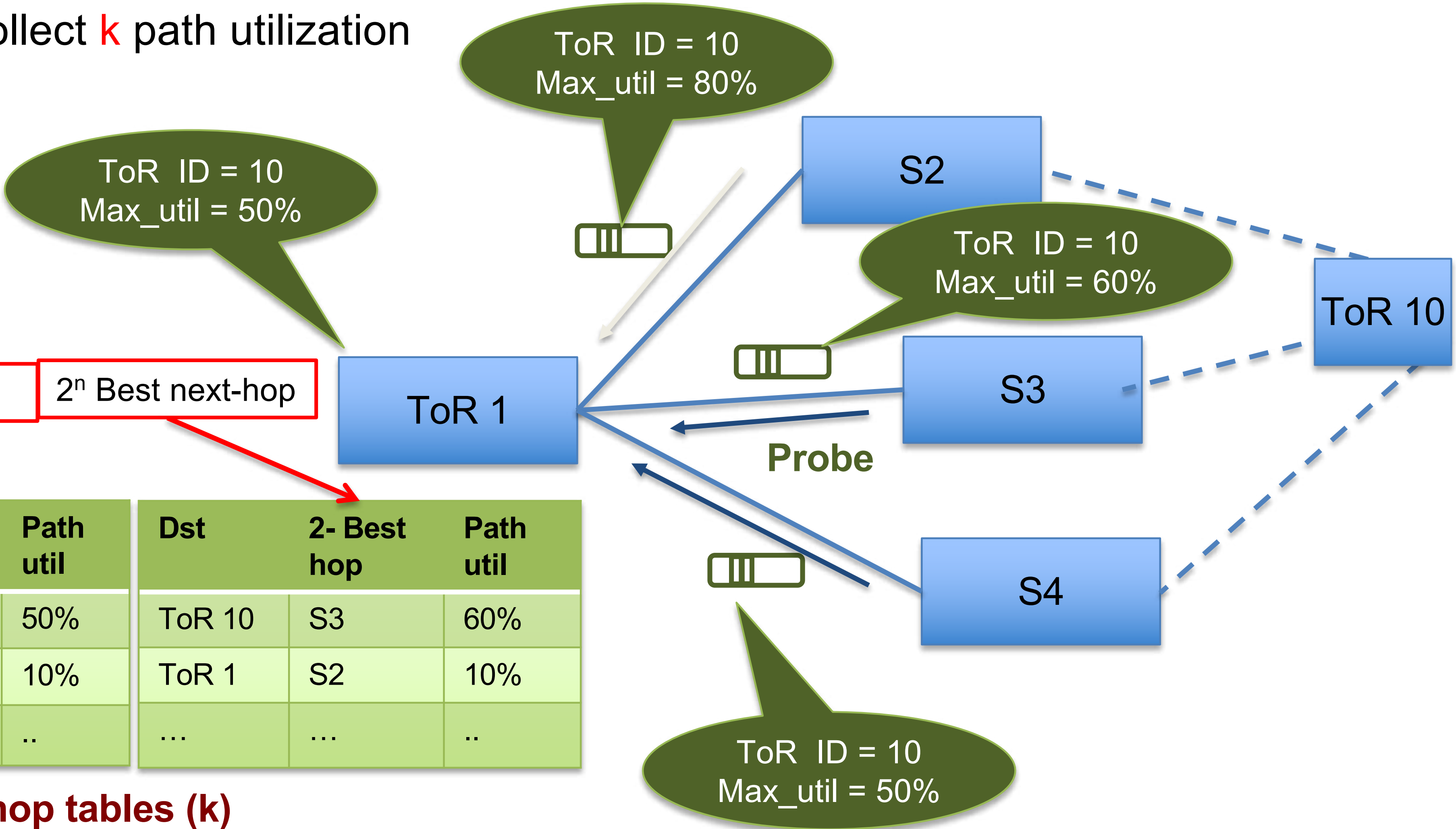
P4 primitives
New header format
Programmable Parsing
RW packet metadata
Comparison/arithmetic operators



Each switch maintains a link utilization estimator per switch port based on an exponential moving average generator (EWMA)

Our Approach – MP-HULA

- **MP-HULA Probe Processing**
  - Collect k path utilization



1^{st} Best next-hop | 2^{n} Best next-hop

Dst	1- Best hop	Path util
ToR 10	S4	50%
ToR 1	S2	10%
…	…	..

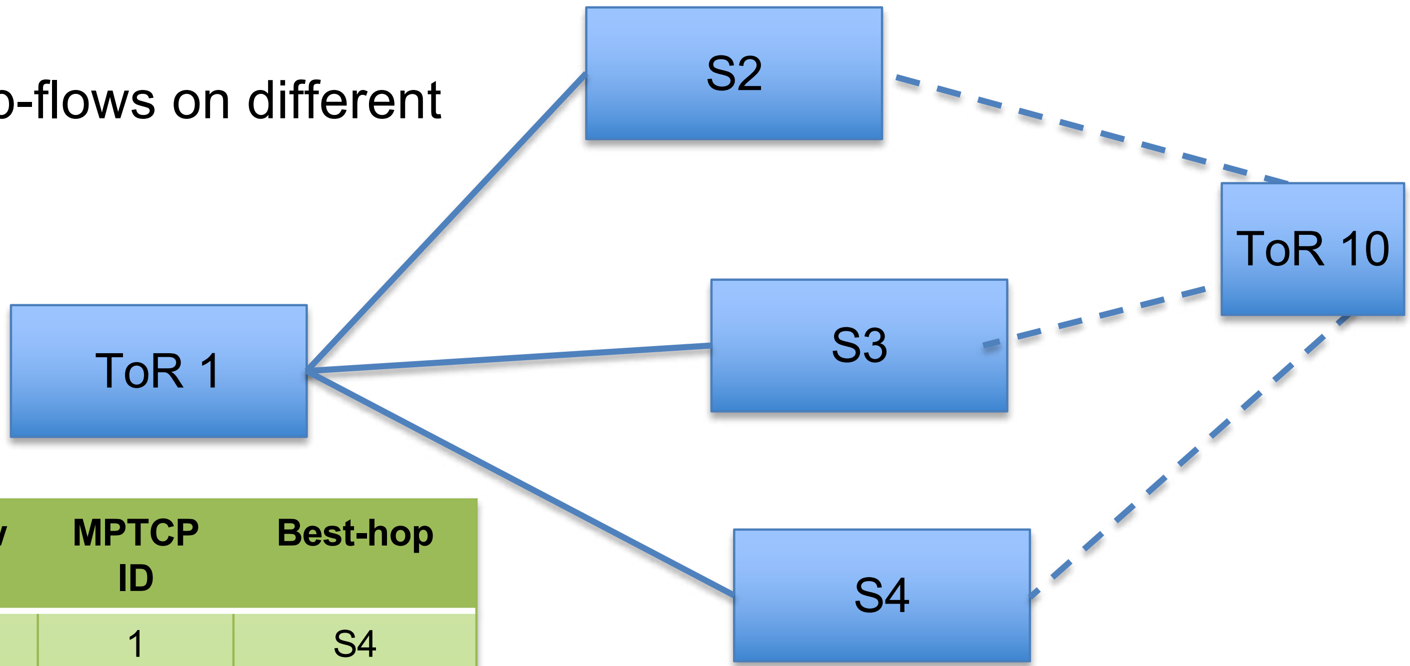
Dst	2- Best hop	Path util
ToR 10	S3	60%
ToR 1	S2	10%
…	…	..

Best hop tables (k)

Our Approach – MP-HULA

- **MP-HULA MP-TCP**
  - Switches load balance flowlet
  - Correlates MPTCP sub-flows to connection IDs
  - Routes different sub-flows on different next hops

P4 primitives
RW access to stateful memory
Comparison/arithmetic operators



Flowlet ID	Dest	Timestamp	Sub-flow ID	MPTCP ID	Best-hop
HASH1	TOR10	1	1	1	S4
HASH2	TOR10	2	2	1	S3
…	…	…			

Our Approach – MP-HULA

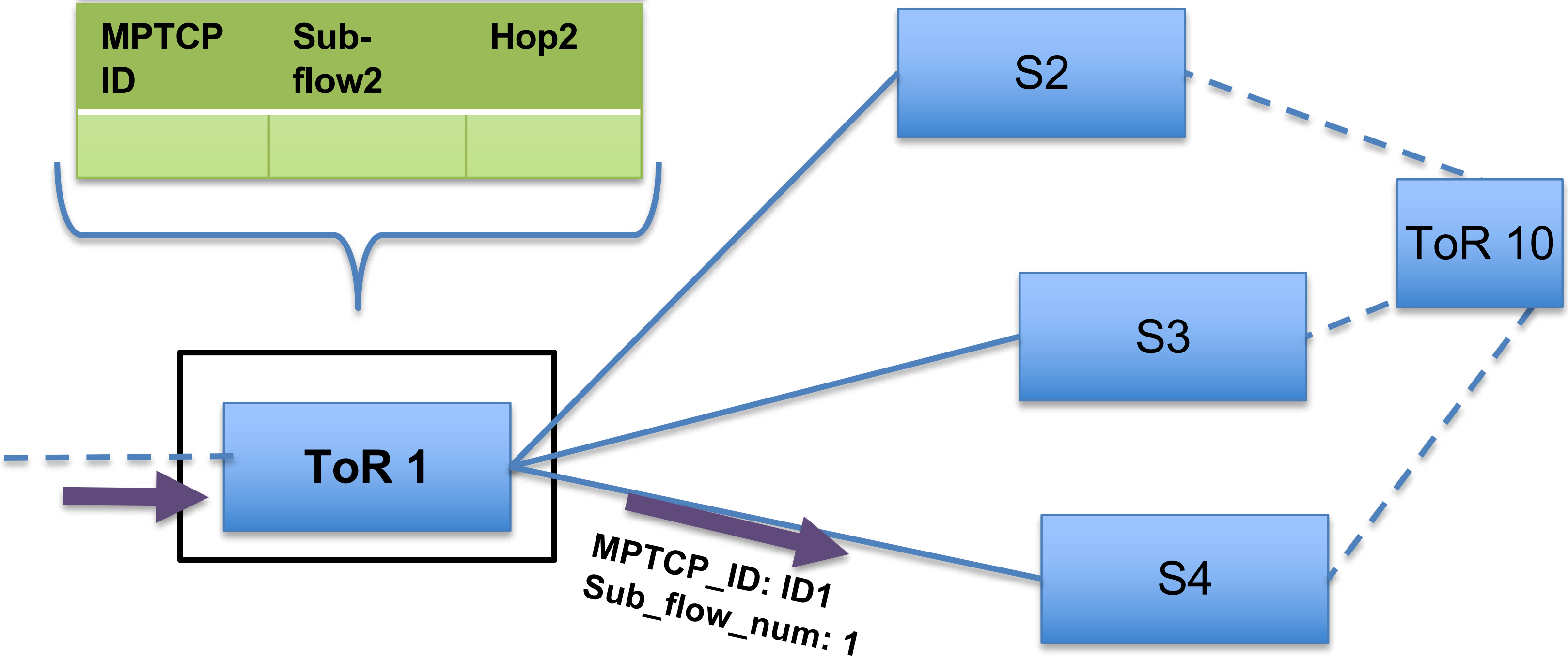
Dst	1- Best hop	Path util
ToR 10	**S4**	**50%**
ToR 1	S2	10%
…	…	

Dst	2- Best hop	Path util
ToR 10	S3	60%
ToR 1	S3	20%
…	…	

Dst	3- Best hop	Path util
ToR 10	S2	80%
ToR 1	S4	30%
…	…	

MPTCP ID	Sub-flow1	Hop1
ID1	1	**S4**

MPTCP ID	Sub-flow2	Hop2



Best hop tables (k)

Our Approach – MP-HULA

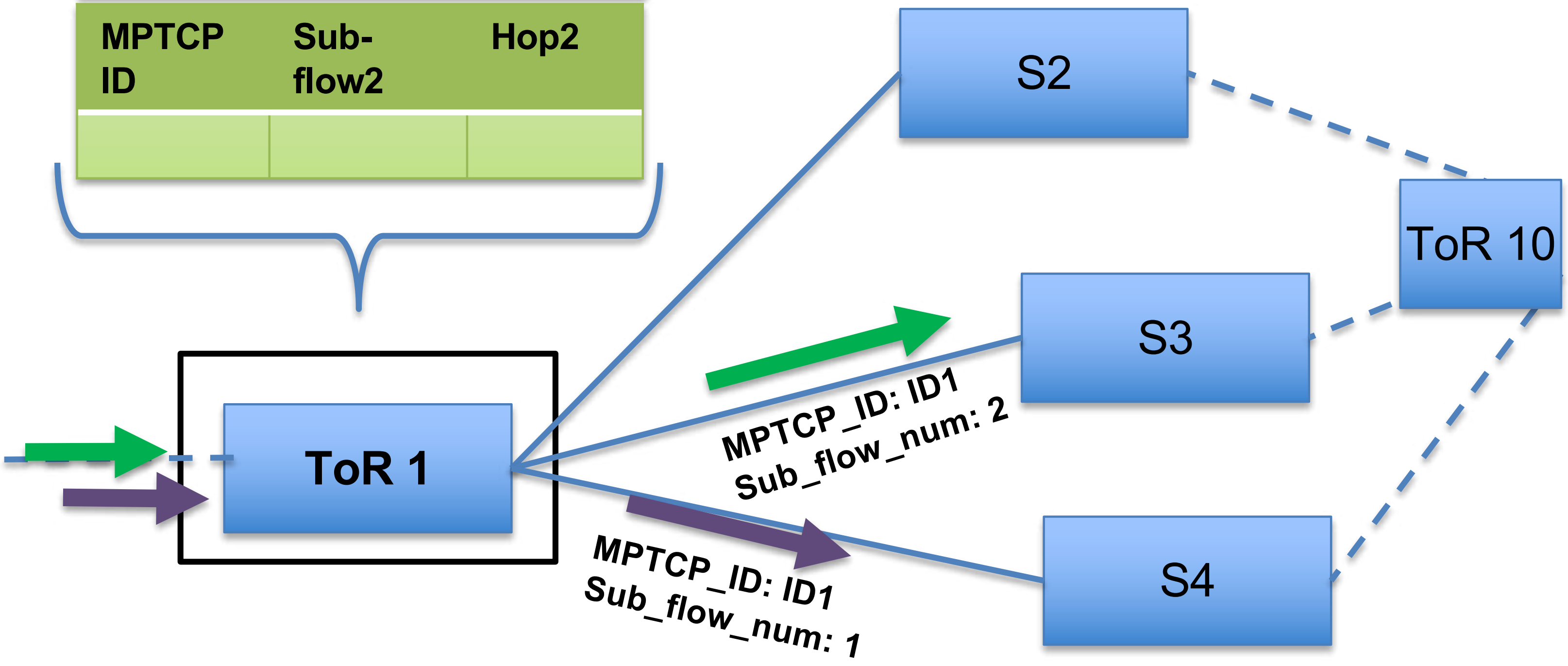
Dst	1- Best hop	Path util
ToR 10	**S4**	**50%**
ToR 1	S2	10%
…	…	

MPTCP ID	Sub-flow1	Hop1
ID1	1	**S4**

MPTCP ID	Sub-flow2	Hop2

Dst	2- Best hop	Path util
ToR 10	S3	60%
ToR 1	S3	20%
…	…	

Dst	3- Best hop	Path util
ToR 10	S2	80%
ToR 1	S4	30%
…	…	



Best hop tables (k)

Our Approach – MP-HULA

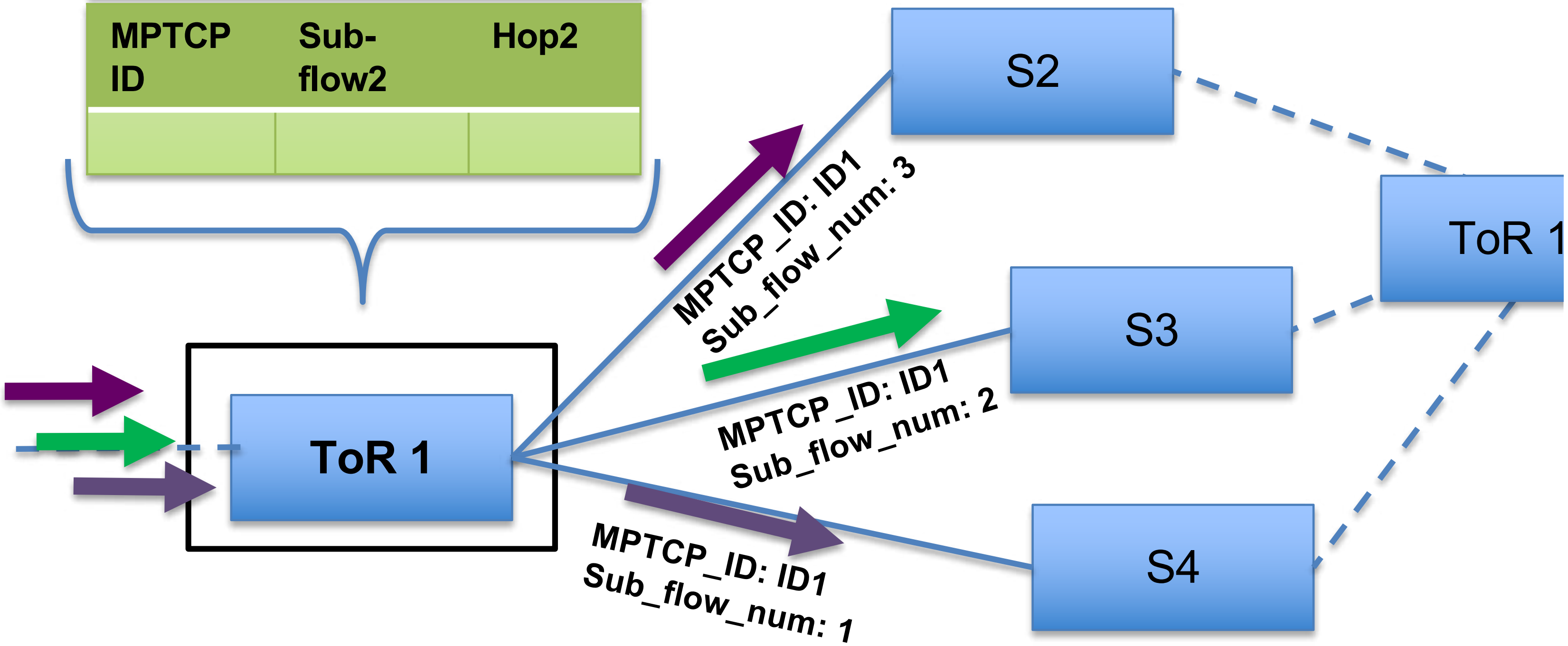
Dst	1- Best hop	Path util
ToR 10	**S4**	**50%**
ToR 1	S2	10%
…	…	

MPTCP ID	Sub-flow1	Hop1
ID1	1	**S4**

MPTCP ID	Sub-flow2	Hop2

Dst	2- Best hop	Path util
ToR 10	S3	60%
ToR 1	S3	20%
…	…	

Dst	3- Best hop	Path util
ToR 10	S2	80%
ToR 1	S4	30%
…	…	



Best hop tables (k)

Our Approach – MP-HULA

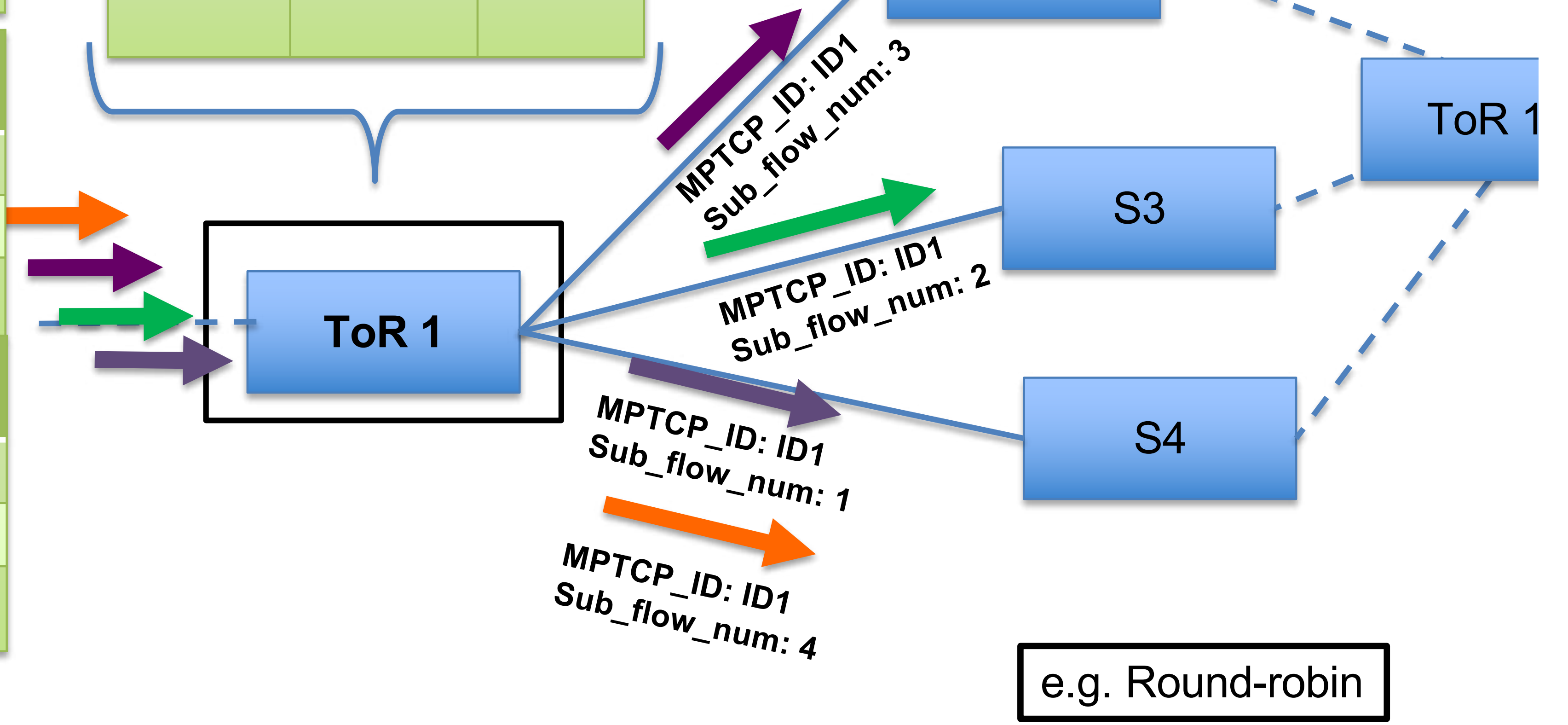
Dst	1- Best hop	Path util
ToR 10	**S4**	**50%**
ToR 1	S2	10%
…	…	

Dst	2- Best hop	Path util
ToR 10	S3	60%
ToR 1	S3	20%
…	…	

Dst	3- Best hop	Path util
ToR 10	S2	80%
ToR 1	S4	30%
…	…	

MPTCP ID	Sub-flow1	Hop1
ID1	1	**S4**

MPTCP ID	Sub-flow2	Hop2



Best hop tables (k)

Evaluation

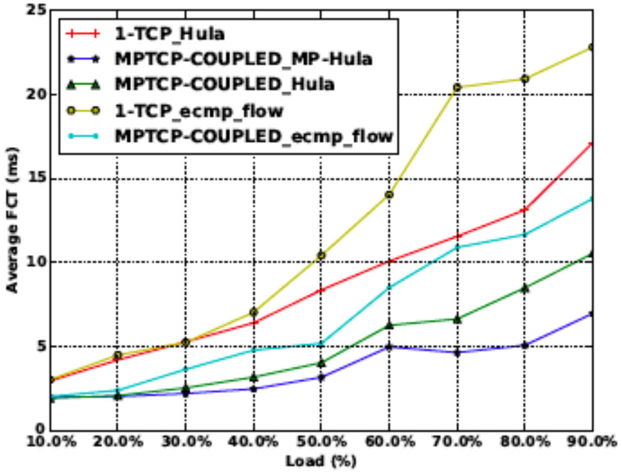


Figure 5: Average FCT using MPTCP-Coupled for web-search traffic

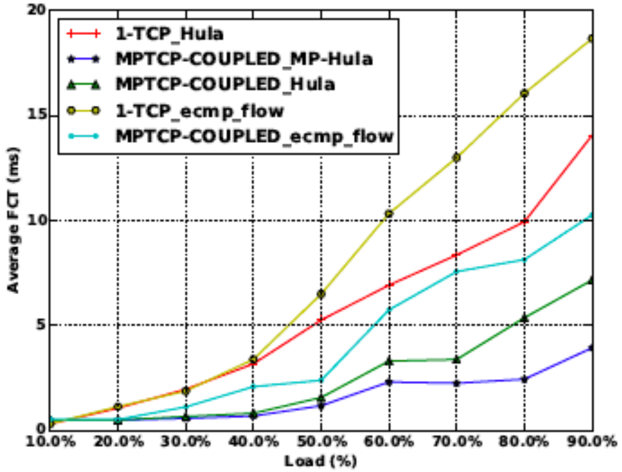


Figure 6: Average FCT for mice flows (<100KB), coupled, web-search

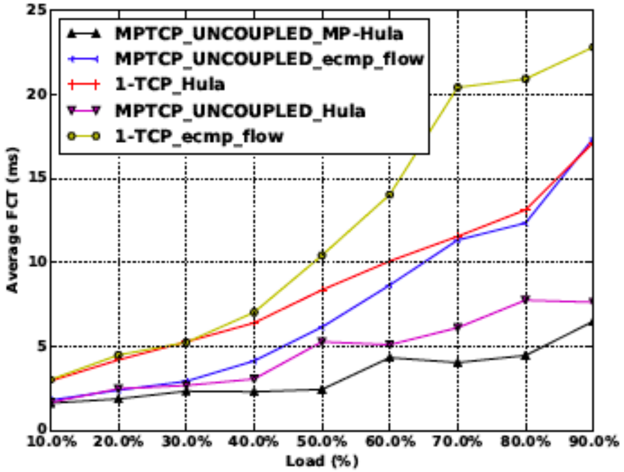


Figure 7: Average FCT using MPTCP-Uncoupled for web-search traffic

Conclusions

- **Data Center Networks**
  - Are crucial for our society
  - Require effective load-balancing
  - Control plane scalability issues
- **Data plane load balancing**
  - Flexible, P4 programmable (e.g. Hula)
  - Can exploit multipath transport protocols (e.g. MP-HULA)
- **Next Course Module…**
  - Starts Monday, April 26th at 17.00-18.30 CET
  - P4 based network monitoring, caching and control
    - Streaming algorithms in P4, e.g. Count-min Sketch and Bloom Filter