P802.1Qcz D0.5
2nd Task Group Ballot
Editor's Report and Discussion

Paul Congdon

January 2020

Summary

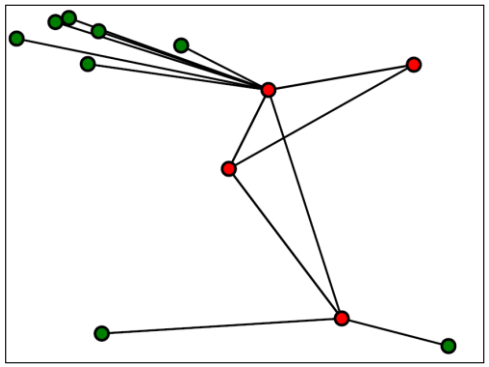
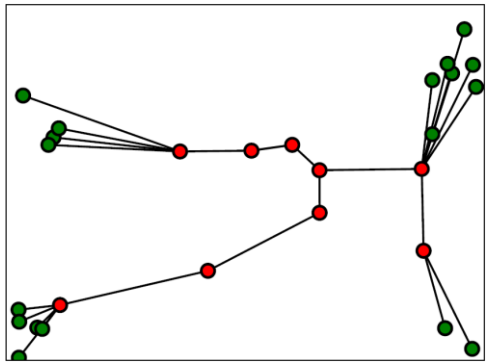
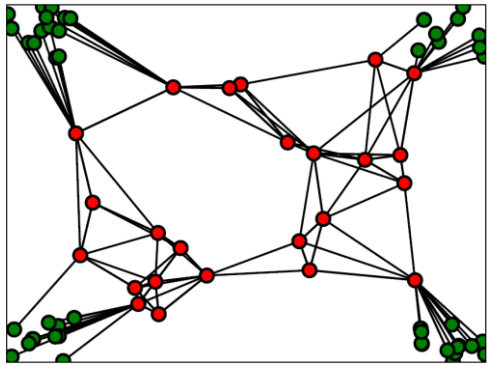
- Ballot details
  - Yes 6 75.00%
  - No 2 25.00%
  - Voting Yes or No 8 100.00%
  - Abs. Time 5 16.67%
  - Abs. Expertise 16 53.33%
  - Abs. Other 0 0.00%
  - Respondents 30
  - Voting members 27
  - Non-voting 3
  - No. of commenters 4 13.33%
  - No. of comments 96
- Comments proposed to approve without discussion
  - 91->103, 105-110, 112-120, 122-125, 127, 130-153, 155, 157-164, 167, 169-172, 174-176, 178, 182, 184-185
- Priority comments to discuss (with supporting material)
  - 166, 168
- Priority comments to discuss
  - 104, 111, 126, 128, 154, 177, 179-181, 183, 186
- Lower priority comments to discuss
  - 121, 129, 156, 165, 173
- Current proposed disposition posted:
  - http://www.ieee802.org/1/files/private/cz-drafts/d0/802-1Qcz-d0-5-pdis-v01.pdf

Comment 168 – TR Algorithm

- The trUpdate() algorithm in the current draft has a flaw in it.
- Two independent implementations have found and corrected the issue.
- Details of validation follow

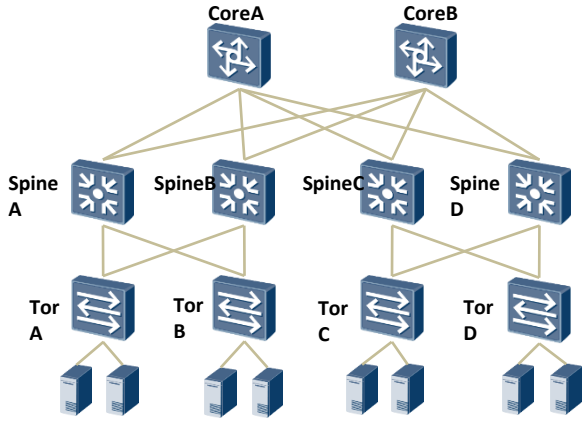
Validating TR algorithm

- Variants of the algorithm were simulated by a C program that runs LLDP between nodes in the topology:
  - paul@ubuntu:~/Code/tr-sim$ ./tr-sim -h
    usage: ./tr-sim
    ./tr-sim -a num ==> set algorithm number (defaults to 0)
    ./tr-sim -d num ==> set start-up delay max (random 0 to num)
    ./tr-sim -f file ==> topology configuration file
    ./tr-sim -h ==> show help
    ./tr-sim -m num ==> maximum simulation time in ticks
- Random topologies were generated by an open source python script (https://github.com/cesarghali/topology-generator/blob/master/topo-gen.py).

Test Environment Introduction

- Validate the TR algorithm in the real lab environment. The topology is a Layer 3 CLOS network
  - ✓ Network
  Spine&Core: Huawei CE8850 tomahawk2 switch*6, Port 100G
  Tor: Huawei CE6865 trident3 switch*4, Port 100G
  - ✓ Server
  ubuntu , Huawei 2288HV3*8, NIC Mellanox CX5*8, 100G
- Perform Topology Recognition(TR) program on all servers and Switches
- LLDP is turned on all nodes



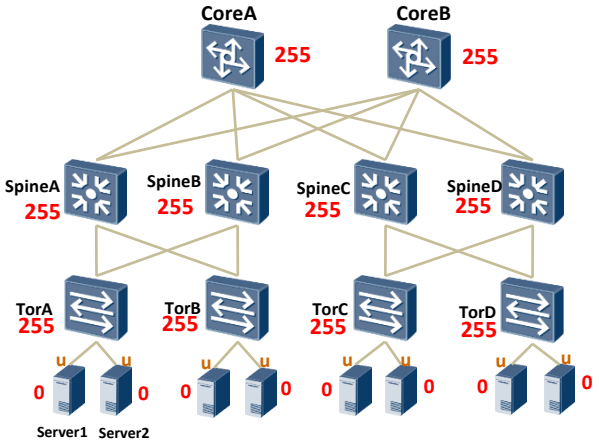
Test environment is shown in the figure above

Perform Topology Recognition program at all servers and Switches

98.5.3.1 trInit()

The trInit() procedure initializes the controlling variables to a known state after system initialization or a restart of the TR functionality. The procedure performs the following:

a) If trDeviceType is 0, specifying a non-relay end station or server then
   1) Set trLevel to 0
   2) Set trPortAttribte to *uplink*.
b) If trDeviceType is not 0
   1) Set trLevel to 255, specifying *unknown*
   2) Set trPortOrientation to 255, specifying *unknown*
c) Call trSet() to cause the transmission of an LLDPDU to peers



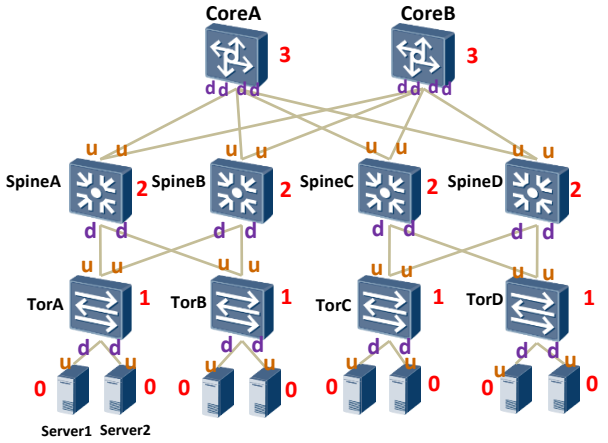
98.5.3.1 trInit()

98.5.3.2 trSet()

Calls the somethingChangedLocal() procedure defined in IEEE Std 802.1AB which cause the transmission of an LLDPDU. The trLevel and trPortOrientation variables map to objects in the IEEE 802.1/LLDP extension MIB (D.5). A change to the single system wide trLevel variable will cause the transmission of an LLDPDU on each participating port. A change to the per-port trPortOrientation variable will cause the transmission of an LLDPDU on the associated port.

98.5.3.3 trUpdate()

The trUpdate() procedure is invoked when the somethingChangedRemote() procedure defined in IEEE Std 802.1AB determines that fields a received LLDP Topology Recognition TLV have changed. The procedure is responsible for updating the local TR variables and calling trSet() according to the following algorithm:



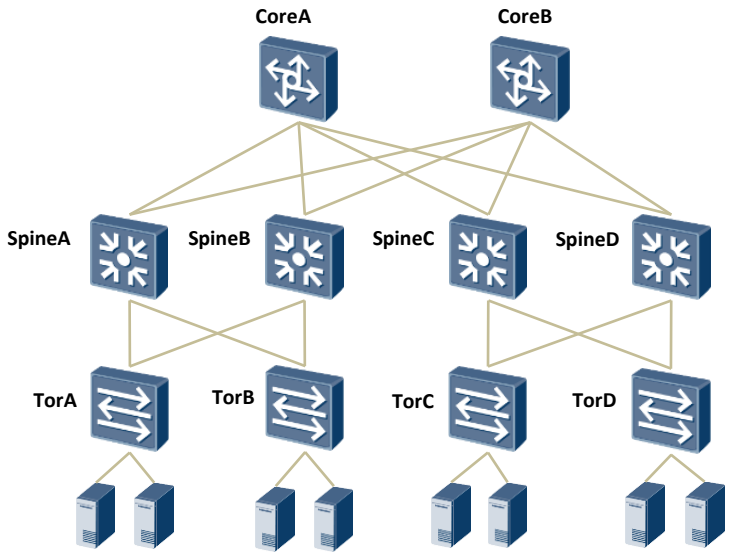
98.5.3.3 trUpdate()

Legend:
Number: switch level
d: downlink
u: uplink
c: crosslink

Test Cases

- Basic convergence of Layer-3 CLOS
- Disconnect the link between Spine and Tor
- Disconnect multiple links between Spine and Tor
- Add a link between 2 Tors
- Remove a Tor
- Add a Core to all Spines

Note: Detail slides in backup

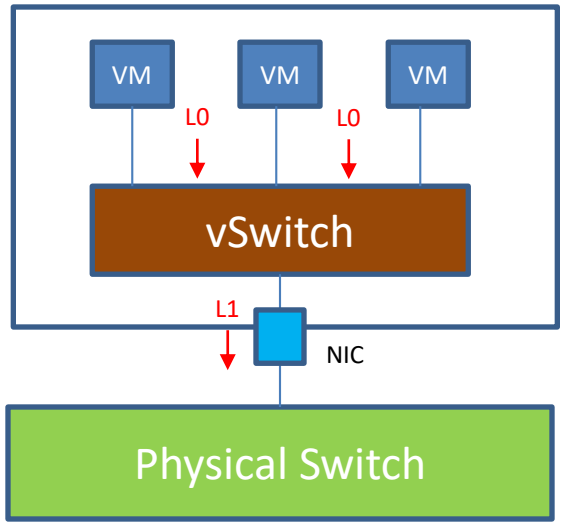


Validated Algorithm

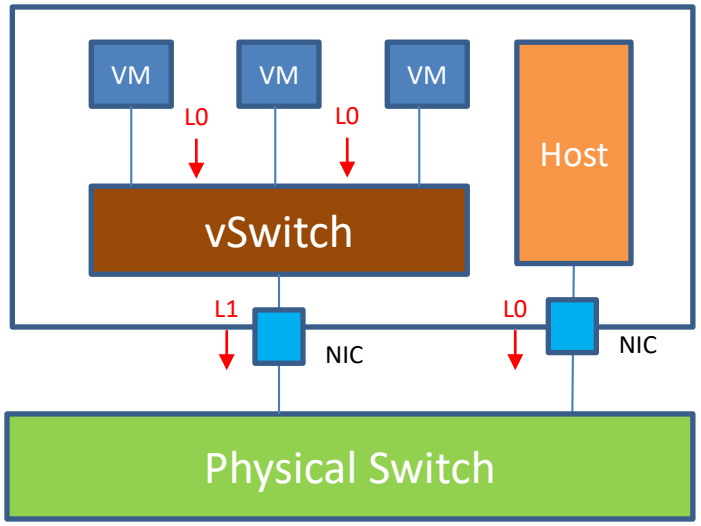
```
if (the receiving device is a server)                                   # Servers are always at 0
      return;
if (the sending device doesn't know its level)                          # Sender isn't providing any info
      return;
if (if the sending and receive device are at the same level)            # Crosslink case
      if (receiving port is already a crosslink)
         return;
      else
          set receiving port to crosslink
if (if the sending level is one greater than the receiving level)       # Uplink case
      if (receiving port is already an uplink)
         return;
      else
         set receiving port to uplink
if (if the sending level is one less than the receiving level)          # Downlink case
      if (receiving port is already a downlink)
          return;
      else
          set the receiving port to downlink
If (the sending level is less than the receiving level minus 1) OR (the receiving level is unknown)
      set the receiving level to the sending level plus 1               # Works because unknown = -1
      set the receiving port to downlink
      set all other ports on the receiving device to unknown

Call trSet() for any ports that had something change locally
```

Comment 166 - Virtualization

- The term level refers to end systems a level 0. With virtualization technologies end systems may have virtual bridges and levels inside them. These interfaces can be exposed but it is more normal that they are tunneled between servers. However the question arises that if a virtual bridge interface is enables and LLDP is turned on a) would this impact topology recognition? Also would there be vulnerability to an implementation that spoofed



Physical Switch is L2



Physical Switch is L1

Draft Plan

- Produce D0.6 based on Q-base provided by John Messenger
  - Resolved comments
  - Add missing YANG
  - Add missing PICs
- Run 3rd TG Ballot prior to March Plenary
- Motion to move to WG Ballot in March

Backup

Test Result for Topology Recognition Algorithms

Yongxian Chen
Xiang Yu

IEEE 802.1 Interim meeting
Jan, Geneva, 2020

Test Case – Topology Recognition Procedures

Local TR variables
- DeviceType
- Level
- PortOrientation

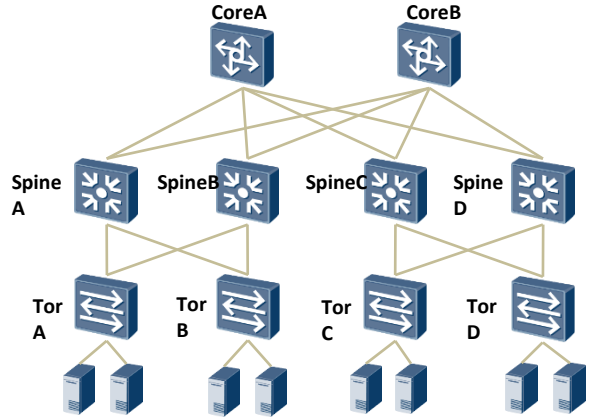


Figure1

Step 1: All Servers and Switches perform trInit()
Servers: If DeviceType is 0,
Set Level to *0*
Set PortOrientation to *uplink*
Switches:If DeviceType is not 0
Set Level to *255*
Set PortOrientation to *255*

Step 2: All Servers and Switches perform transmission of an LLDPDU to peers

Step 3: If DeviceType is 0, return // Server, do nothing.

Step 4: If the received Level is unknown, return //the peer is not providing additional information

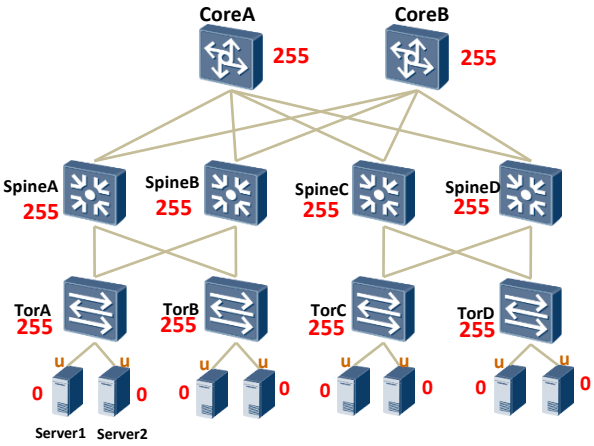


Figure2

Step 5.1: On TorA, receive the LLDPDU from Server1, Server2,SpineA and SpineB
If DeviceType is not 0 // Switch, Router
If (the received Level is less than Level – 1) or (Level is unknown)
Set TorA Level to the received Level plus one // 0+1 = 1
So do TorB,C,D

Step 5.2: On SpineA, receive the LLDPDU from TorA, TorB, CoreA and CoreB
If DeviceType is not 0
If (the received Level is less than Level – 1) or (Level is unknown)
Set SpineA Level to the received Level plus one // 1+1 =2
So do SpineB,C,D

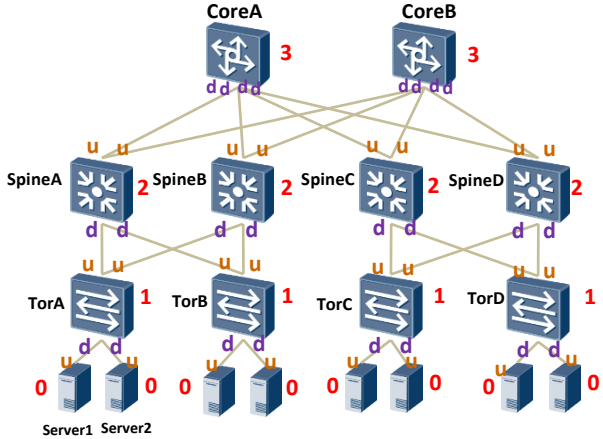


Figure3

Step 5.3: On CoreA, receive the LLDPDU from SpineA, SpineB, SpineC and SpineD
If DeviceType is not 0
If (the received Level is less than Level – 1) or (Level is unknown)
Set CoreA Level to the received Level plus one // 2+1 =3
So do CoreB

Step 6: Levels on those devices are converged. Then set the portOrientation
If the received Level is known and is less than Level - 1
set PortOrientation of the receiving port to *downlink*
If the received Level is known with a value of Level + 1
set PortOrientation of the receiving port to *uplink*
If the received Level is known with a value of Level
set PortOrientation of the receiving port to *crosslink*

Legend:
Number: switch level
d: downlink
u: uplink
c: crosslink

Test Case 1 – Disconnect the link between Spine and Tor

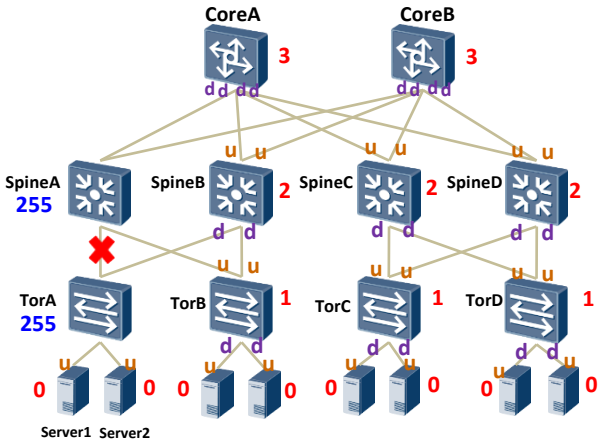


Figure1

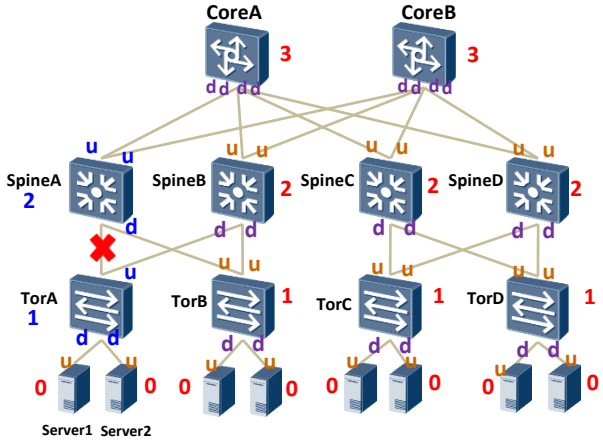


Figure2

Step 1： Disconnect the link between SpineA and TorA

Step 2： SpineA and TorA detect the link state change, Spine A and TorA will do trInit().
In SpineA and TorA
If DeviceType is not 0
Set Level to 255
Set PortOrientation to 255

Step 3： All Servers and Switches perform transmission of an LLDPDU to peers

Step 4: If DeviceType is 0, return // Server, do nothing.

Step 5: If the received Level is unknown, return //the peer is not providing additional information

Step 6.1: On TorA, receive the LLDPDU from Server1, Server2 and SpineB
If DeviceType is not 0 // Switch, Router
If (the received Level is less than Level - 1) or (Level is unknown)
Set TorA Level to the received Level plus one // 0+1 = 1

Step 6.2: On SpineA, receive the LLDPDU from TorB and CoreA, CoreB
If DeviceType is not 0 // Switch, Router
If (the received Level is less than Level -1) or (Level is unknown)
Set SpineA Level to the received Level plus one // 1+1 = 2

Step 7: Levels on those devices are converged. Then set the portOrientation
If the received Level is known and is less than Level - 1
set PortOrientation of the receiving port to *downlink*
If the received Level is known with a value of Level + 1
set PortOrientation of the receiving port to *uplink*
If the received Level is known with a value of Level
set PortOrientation of the receiving port to *crosslink*

Legend:
Number: switch level
d: downlink
u: uplink
c: crosslink

Test Case 2 – Disconnect the links between Spine and Tor

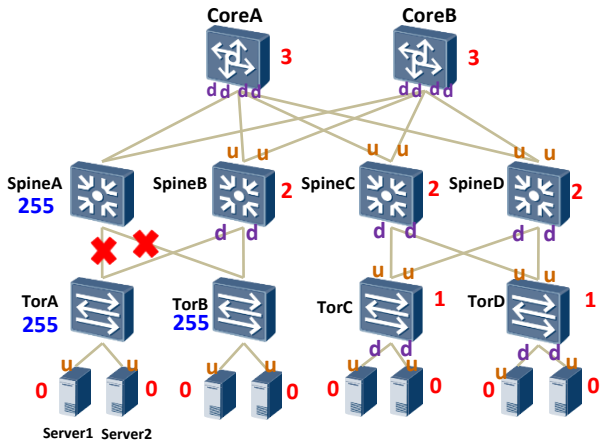


Figure1

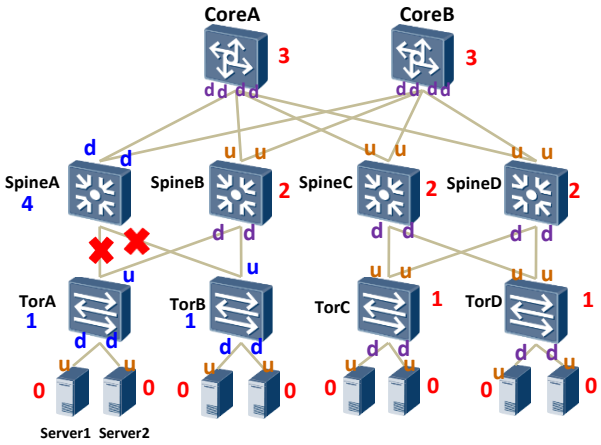


Figure2

Step 1： Disconnect the links between SpineA and TorA, TorB

Step 2： SpineA and TorA, TorB detect the link state change, Spine A and TorA,TorB will do trInit().

In SpineA and TorA, TorB
If DeviceType is not 0
Set Level to 255
Set PortOrientation to 255

Step 3： All Servers and Switches perform transmission of an LLDPDU to peers

Step 4: If DeviceType is 0, return // Server, do nothing.

Step 5: If the received Level is unknown, return //the peer is not providing additional information

Step 6.1: On TorA, receive the LLDPDU from Server1, Server2 and SpineB
If DeviceType is not 0 // Switch, Router
If (the received Level is less than Level - 1) or (Level is unknown)
Set TorA Level to the received Level plus one // 0+1 = 1
So do TorB

Step 6.2: On SpineA, receive the LLDPDU from CoreA and CoreB
If DeviceType is not 0 // Switch, Router
If (the received Level is less than Level -1) or (Level is unknown)
Set SpineA Level to the received Level plus one // 1+1 = 2

Step 7: Levels on those devices are converged. Then set the portOrientation
If the received Level is known and is less than Level - 1
set PortOrientation of the receiving port to *downlink*
If the received Level is known with a value of Level + 1
set PortOrientation of the receiving port to *uplink*
If the received Level is known with a value of Level
set PortOrientation of the receiving port to *crosslink*

Legend:
Number: switch level
d: downlink
u: uplink
c: crosslink

Test Case 3 – Add a link between 2 Tors

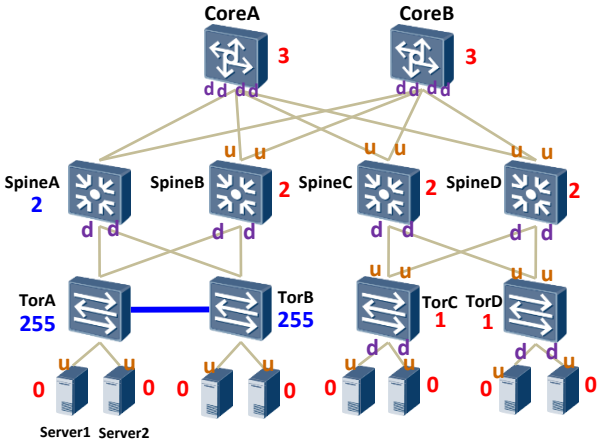


Figure1

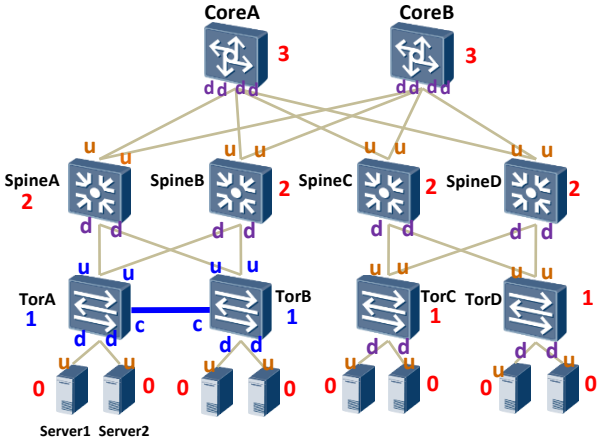


Figure2

Step 1：Add a link between TorA and TorB

Step 2：TorA and TorB detect the link state change, TorA and TorB will do trInit().
In TorA and TorB
If DeviceType is not 0
Set Level to 255
Set PortOrientation to 255

Step 3：All Servers and Switches perform transmission of an LLDPDU to peers

Step 4: If DeviceType is 0, return // Server, do nothing.

Step 5: If the received Level is unknown, return //the peer is not providing additional information

Step 6: On TorA, receive the LLDPDU from Server1, Server2 and SpineB
If DeviceType is not 0 // Switch, Router
If (the received Level is less than Level - 1) or (Level is unknown)
Set TorA Level to the received Level plus one // 0+1 = 1
So do TorB

Step 7: Levels on those devices are converged. Then set the portOrientation
If the received Level is known and is less than Level - 1
set PortOrientation of the receiving port to *downlink*
If the received Level is known with a value of Level + 1
set PortOrientation of the receiving port to *uplink*
If the received Level is known with a value of Level
set PortOrientation of the receiving port to *crosslink*

Legend:
Number: switch level
d: downlink
u: uplink
c: crosslink

Test Case 4 – Remove a Tor

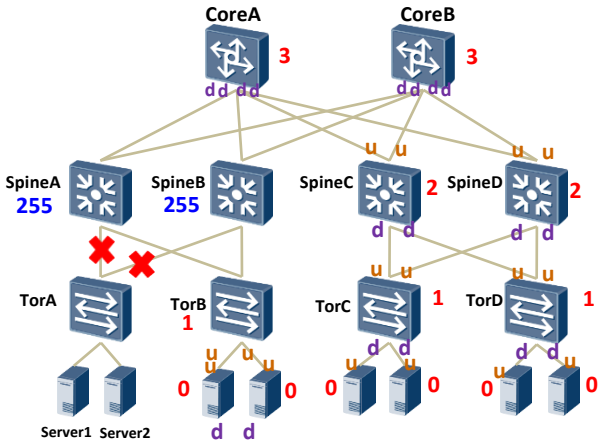


Figure1

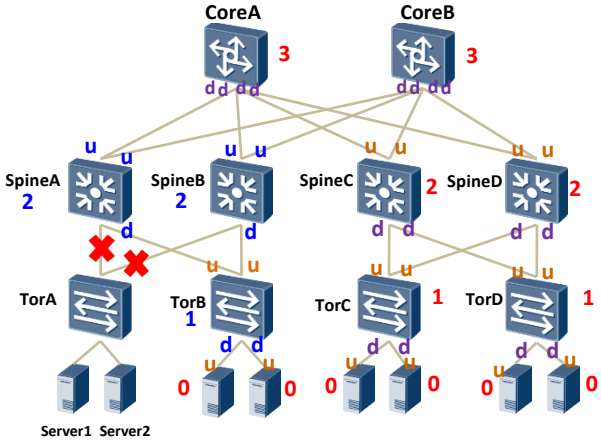


Figure2

Step 1： Disconnect the links between TorA and SpineA, SpineB

Step 2： SpineA and SpineB detect the link state change, SpineA and SpineB will do trInit().
In SpineA and SpineB
If DeviceType is not 0
Set Level to 255
Set PortOrientation to 255

Step 3： All Servers and Switches perform transmission of an LLDPDU to peers

Step 4: If DeviceType is 0, return // Server, do nothing.

Step 5: If the received Level is unknown, return //the peer is not providing additional information

Step 6: On SpineA, receive the LLDPDU from CoreA and CoreB
If DeviceType is not 0 // Switch, Router
If (the received Level is less than Level -1) or (Level is unknown)
Set SpineA Level to the received Level plus one // 1+1 = 2
So do SpineB

Step 7: Levels on those devices are converged. Then set the portOrientation
If the received Level is known and is less than Level - 1
set PortOrientation of the receiving port to *downlink*
If the received Level is known with a value of Level + 1
set PortOrientation of the receiving port to *uplink*
If the received Level is known with a value of Level
set PortOrientation of the receiving port to *crosslink*

Legend:
Number: switch level
d: downlink
u: uplink
c: crosslink

Test Case 5 – Add a Core to all Spines

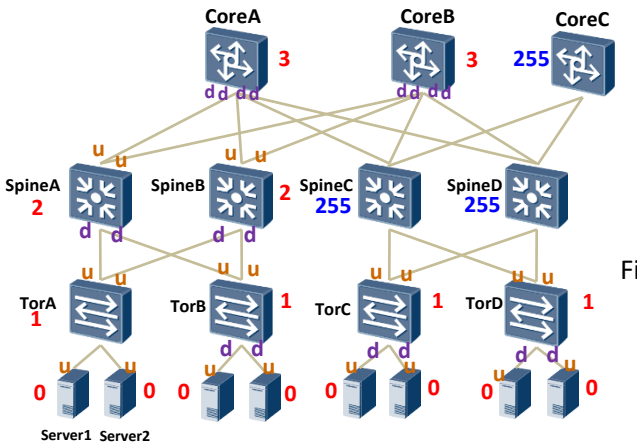


Figure1

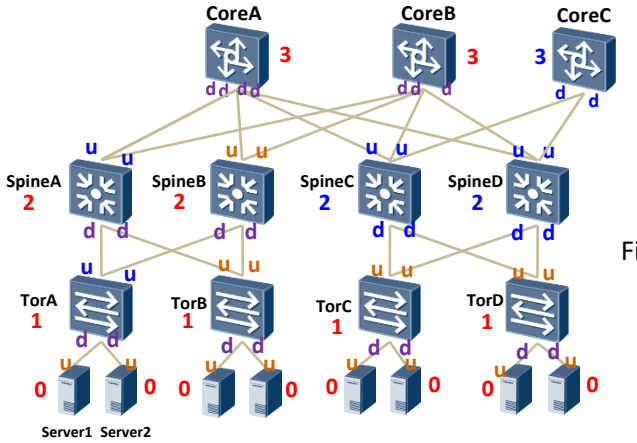


Figure2

Step 1： **Add CoreC to SpineC and SpineD**

Step 2： **SpineA and SpineB detect the link state change, Core C and SpineA, SpineB will do trInit().**

In SpineA, SpineB
If DeviceType is not 0
Set Level to 255
Set PortOrientation to 255

Step 3： **All Servers and Switches perform transmission of an LLDPDU to peers**

Step 4: **If DeviceType is 0, return // Server, do nothing.**

Step 5: **If the received Level is unknown, return //the peer is not providing additional information**

Step 6.1: On SpineC, receive the LLDPDU from TorC, TorD and CoreA, CoreB, CoreC
If DeviceType is not 0 // Switch, Router
If (the received Level is less than Level -1) or (Level is unknown)
Set SpineC Level to the received Level plus one // 1+1 = 2
So do SpineD

Step 6.2: On CoreC, receive the LLDPDU from SpineC, SpineD
If DeviceType is not 0 // Switch, Router
If (the received Level is less than Level -1) or (Level is unknown)
Set CoreC Level to the received Level plus one // 2+1 = 2

Step 7: Levels on those devices are converged. Then set the portOrientation
If the received Level is known and is less than Level - 1
set PortOrientation of the receiving port to *downlink*
If the received Level is known with a value of Level + 1
set PortOrientation of the receiving port to *uplink*
If the received Level is known with a value of Level
set PortOrientation of the receiving port to *crosslink*

Legend:
Number: switch level
d: downlink
u: uplink
c: crosslink