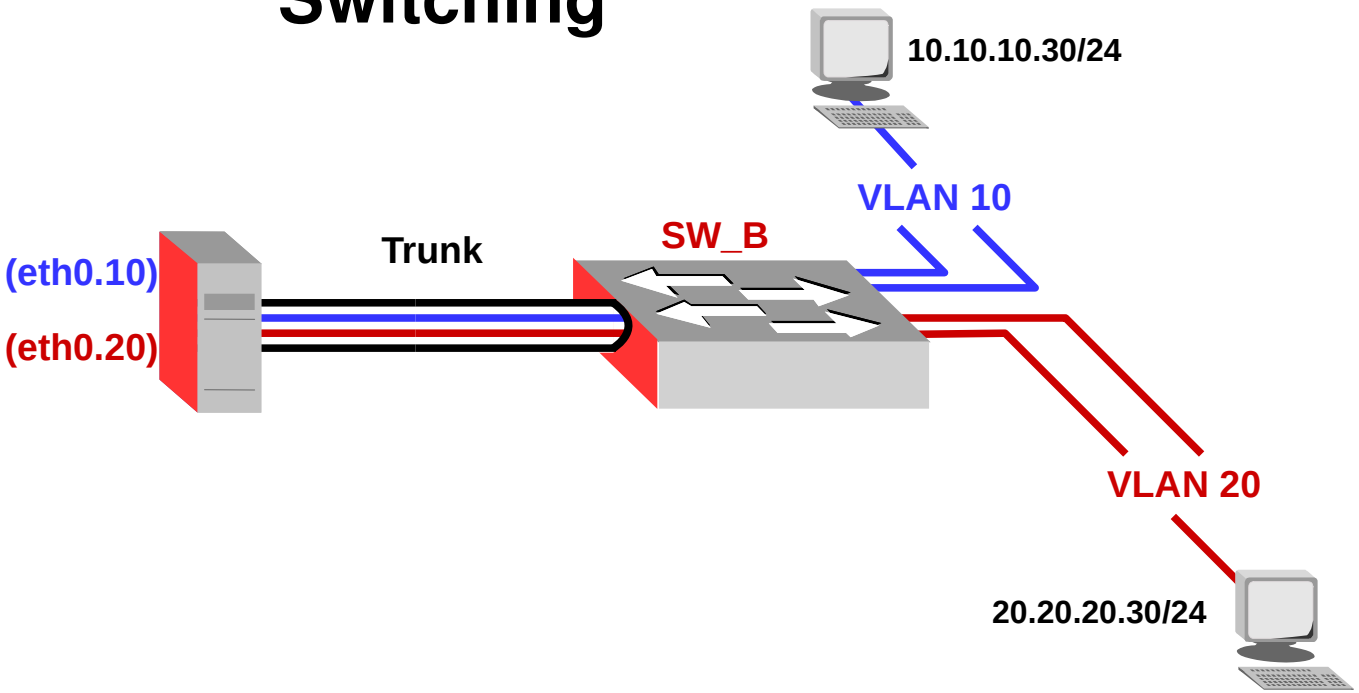
TEL3214
Computer Communication Networks

Lecture 5

Switching



Diarmuid Ó Briain
CEng, FIEI, FIET, CISSP

diarmuid@obriain.com

Bridging

- Device used to connect two or more LAN segments (collision domain) that use identical LAN (MAC Layer) protocols.
- Acts as an address filter.
- The bridge does not modify the contents of the frames and does not add anything to the frame.
- The bridge operates at layer 2 of the OSI model.

Bridging

- Look at the destination address field.
- Compares the address to its address tables for all its ports.
- If it finds the address associated with a port, it sends the frame out on that port
- If it does not find an address, it sends the frame out on all ports.

Layer 2 Switching

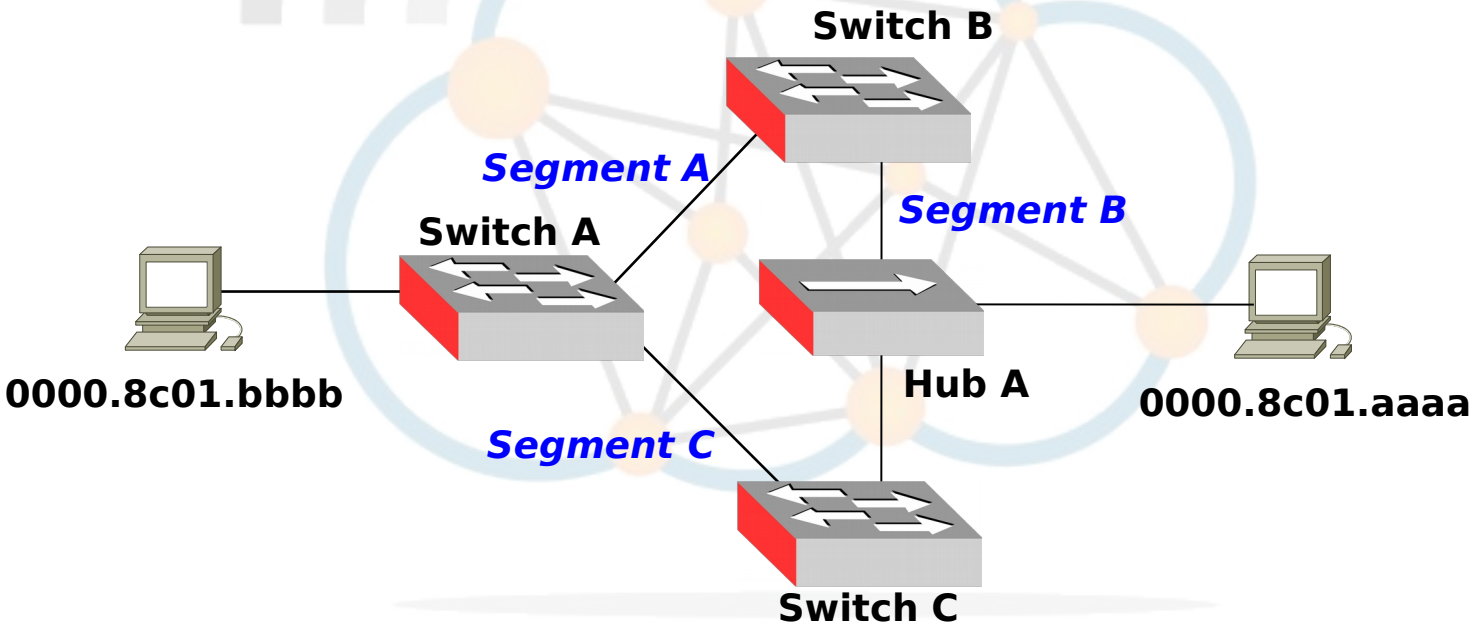
- Operate at Layer 2 (DLC) of the OSI Model.
- Switches forward frames based on the MAC layer address (the actual NIC address).
- Forward frames with very low delay time.
- IEEE 802.1d Spanning Tree Algorithm (STP).
- IEEE 802.1w Rapid Spanning Tree Algorithm (RSTP) allowing redundant switches in the network.
- Switches will forward "broadcast" traffic to all LAN segments attached to them.

Transparent Bridging

- Transparent bridging allows a switch to learn everything it needs to know about the location of nodes on the network without the network administrator having to add static entries.
- Transparent bridging has five parts:
  - Learning
  - Flooding
  - Filtering
  - Forwarding
  - Ageing.

Broadcast Storm

- Redundant Loops.
- Broadcast flooding.



Spanning Tree Protocol

- To prevent broadcast storms and other unwanted side effects of looping, Digital Equipment Corporation (DEC) created the Spanning Tree Protocol (STP)
- It has been standardised as the 802.1d specification by the Institute of Electrical and Electronic Engineers (IEEE).
- It is enhanced with the 802.1w Rapid STP (RSTP).

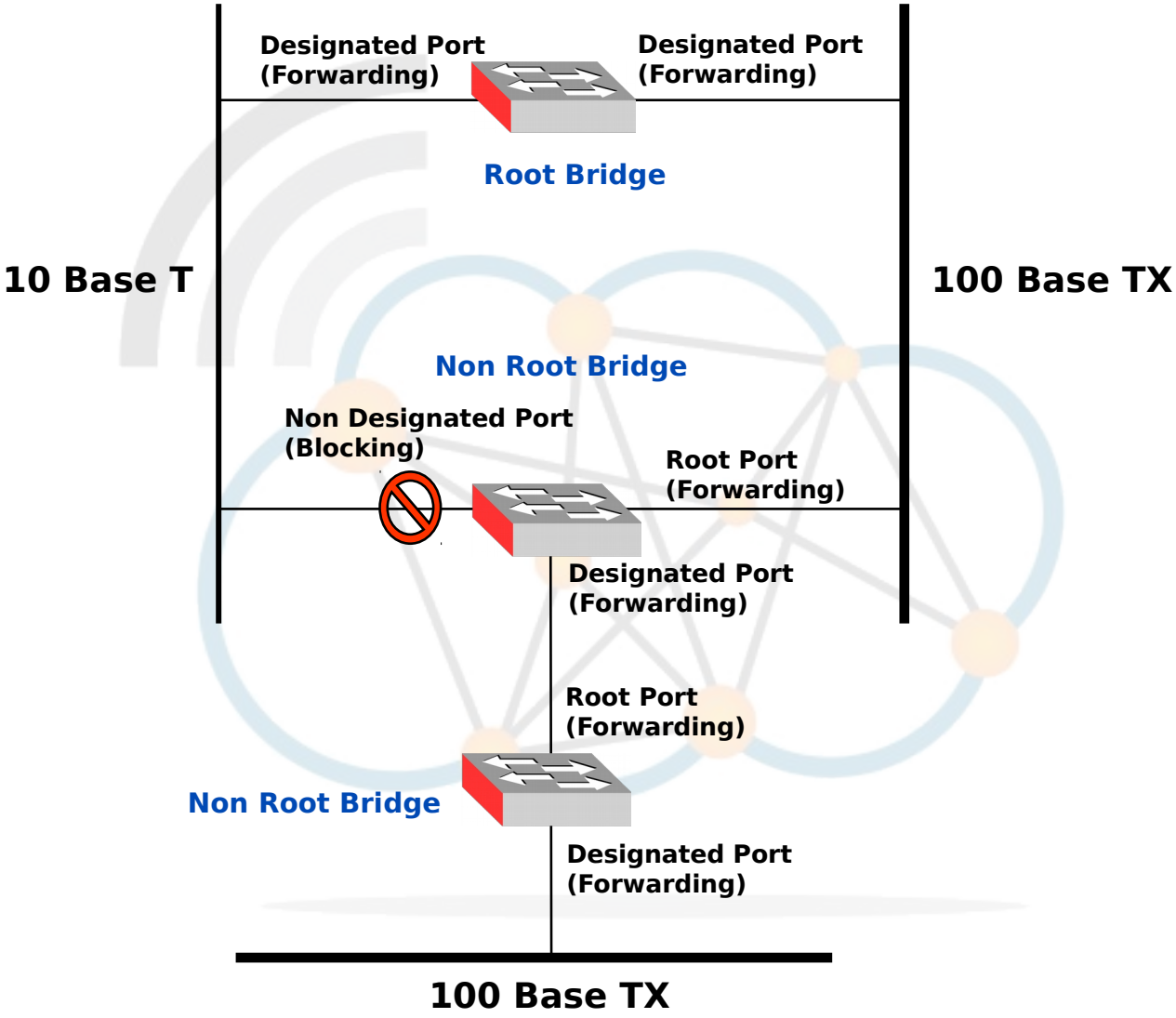
Spanning Tree Protocol

- A spanning tree uses the Spanning Tree Algorithm (STA),
  - Senses that the switch has more than one way to communicate with a node
  - Determines which way is best and blocks out the other path(s).
- It also keeps track of the other path(s), just in case the primary path is unavailable.

Spanning Tree Protocol - Path Cost

Data rate	STP Cost (802.1D-1998)	STP Cost (802.1t-2001)
4 Mbit/s	250	5000000
10 Mbit/s	100	2000000
16 Mbit/s	62	1250000
100 Mbit/s	19	200000
1 Gbit/s	4	20000
2 Gbit/s	3	10000
10 Gbit/s	2	2000

Spanning Tree



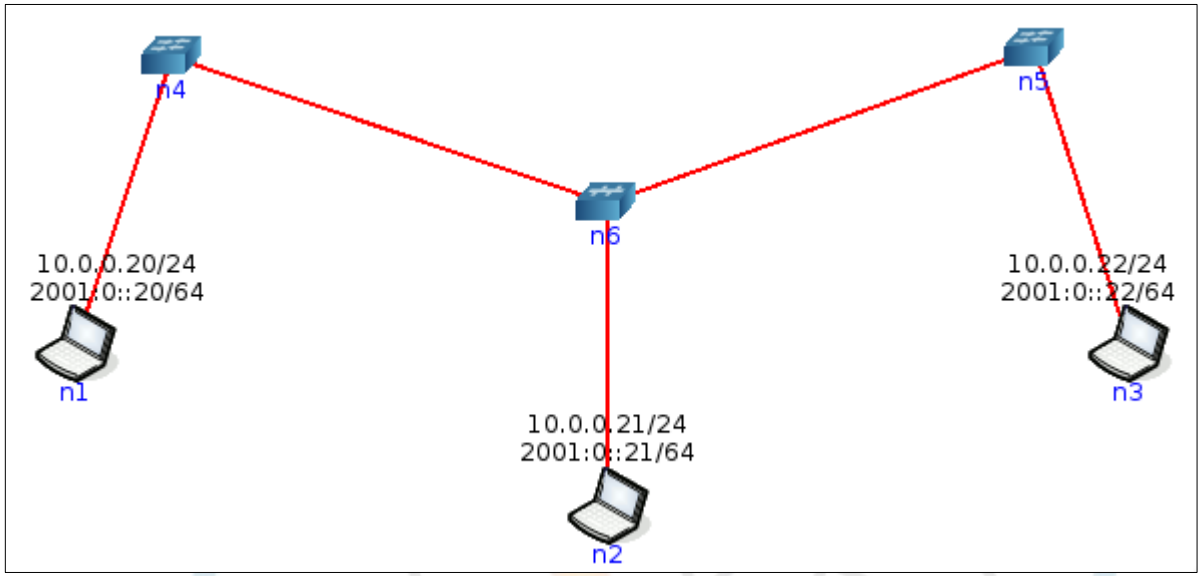
Bridge Protocol Data Units (BPDU)

- Bridge Protocol Data Units (BPDU).
  - Root BID
    - This is the BID of the current root bridge (lowest BID).
  - Path cost to root bridge
    - This determines how far away the root bridge is.
  - Sender BID
    - This is the BID of the switch that sends the BPDU.
  - Port ID
    - This is the actual port on the switch that the BPDU was sent from.

Bridge Utilities

```
# brctl --help
Usage: brctl [commands]
commands:
      addbr           <bridge>                    add bridge
      delbr           <bridge>                    delete bridge
      addif           <bridge> <device>           add interface to bridge
      delif           <bridge> <device>           delete interface from bridge
      hairpin         <bridge> <port> {on|off}    turn hairpin on/off
      setageing       <bridge> <time>             set ageing time
      setbridgeprio   <bridge> <prio>             set bridge priority
      setfd           <bridge> <time>             set bridge forward delay
      sethello        <bridge> <time>             set hello time
      setmaxage       <bridge> <time>             set max message age
      setpathcost     <bridge> <port> <cost>      set path cost
      setportprio     <bridge> <port> <prio>      set port priority
      show            [ <bridge> ]                show a list of bridges
      showmacs        <bridge>                    show a list of mac addrs
      showstp         <bridge>                    show bridge stp info
      stp             <bridge> {on|off}           turn stp on/off
```

Bridged network



root@n1:/tmp/pycore.41149/n1.conf# **ping -c1 10.0.22**
PING 10.0.22 (10.0.0.22) 56(84) bytes of data.
64 bytes from 10.0.0.22: icmp_seq=1 ttl=64 time=0.035 ms

--- 10.0.22 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.035/0.035/0.035/0.000 ms

Capturing on 'eth0'
1 0.000000 10.0.0.20 -> 10.0.0.22 ICMP 98 Echo request id=0x0018, seq=1/256, ttl=64
2 0.000015 10.0.0.22 -> 10.0.0.20 ICMP 98 Echo reply id=0x0018, seq=1/256, ttl=64
3 5.002660 00:00:00:aa:00:00 -> 00:00:00:aa:00:02 ARP 42 Who has 10.0.0.22? Tell 10.0.0.20
4 5.002661 00:00:00:aa:00:02 -> 00:00:00:aa:00:00 ARP 42 Who has 10.0.0.20? Tell 10.0.0.22
5 5.002679 00:00:00:aa:00:02 -> 00:00:00:aa:00:00 ARP 42 10.0.0.22 is at 00:00:00:aa:00:02
6 5.002680 00:00:00:aa:00:00 -> 00:00:00:aa:00:02 ARP 42 10.0.0.20 is at 00:00:00:aa:00:00

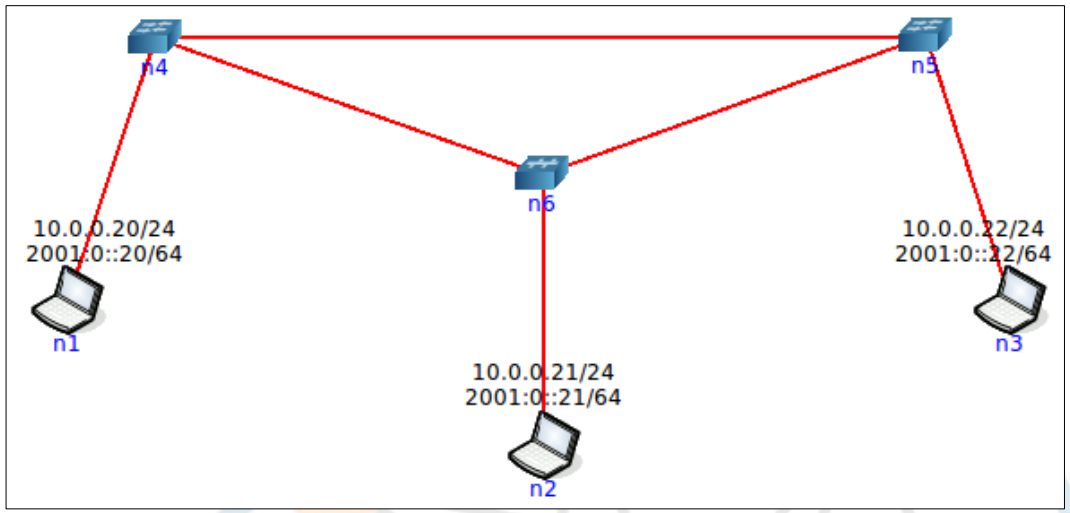
Review the bridges

```
# brctl show
 bridge namebridge id    STP enabled  interfaces
 b.4.26     8000.0a388d18a46b   no     veth2.0.26
                                       veth4.5.26
                                       veth4.6.26
 b.5.26     8000.1a089586bfc5   no     veth1.0.26
                                       veth5.4.26
 b.6.26     8000.1228663b99b8   no     veth3.0.26
                                       veth6.4.26
```

Review the bridges

```
root@NTE-i386:~# sudo brctl showmacs b.4.26
port no mac addr     is local? ageing timer
   1 0a:38:8d:18:a4:6b   yes       0.00
   2 8a:bb:4a:15:9c:5e   yes       0.00
   3 a2:56:5d:c8:0c:b9   yes       0.00

root@NTE-i386:~# sudo brctl showmacs b.5.26
port no mac addr     is local? ageing timer
   1 1a:08:95:86:bf:c5   yes       0.00
   2 1e:80:83:c9:57:3f   yes       0.00

root@NTE-i386:~# sudo brctl showmacs b.6.26
port no mac addr     is local? ageing timer
   1 12:28:66:3b:99:b8   yes       0.00
   2 ea:ef:21:4c:67:99   yes       0.00
```

Bridged network with a loop



- What are the likely problems with this network?
- How can you resolve them?

Review the bridges

```
root@NTE-i386:~# brctl show

bridge name  bridge id    STP enabled  interfaces
b.4.29    8000.12f0b2666390   no     veth2.0.29
                                     veth4.5.29
                                     veth4.6.29
b.5.29    8000.2243edabfea4   no     veth1.0.29
                                     veth5.4.29
                                     veth5.6.29
b.6.29    8000.66d04659b2ec   no     veth3.0.29
                                     veth6.4.29
                                     veth6.5.29
```

Enable STP

```
root@NTE-i386:~# brctl stp b.4.29 on

root@NTE-i386:~# brctl stp b.6.29 on
```

Review the bridges

```
root@NTE-i386:~# brctl show

bridge name   bridge id          STP enabled   interfaces
b.4.29        8000.12f0b2666390   yes          veth2.0.29
                                               veth4.5.29
                                               veth4.6.29
b.5.29        8000.2243edabfea4   no           veth1.0.29
                                               veth5.4.29
                                               veth5.6.29
b.6.29        8000.66d04659b2ec   yes          veth3.0.29
                                               veth6.4.29
                                               veth6.5.29
```

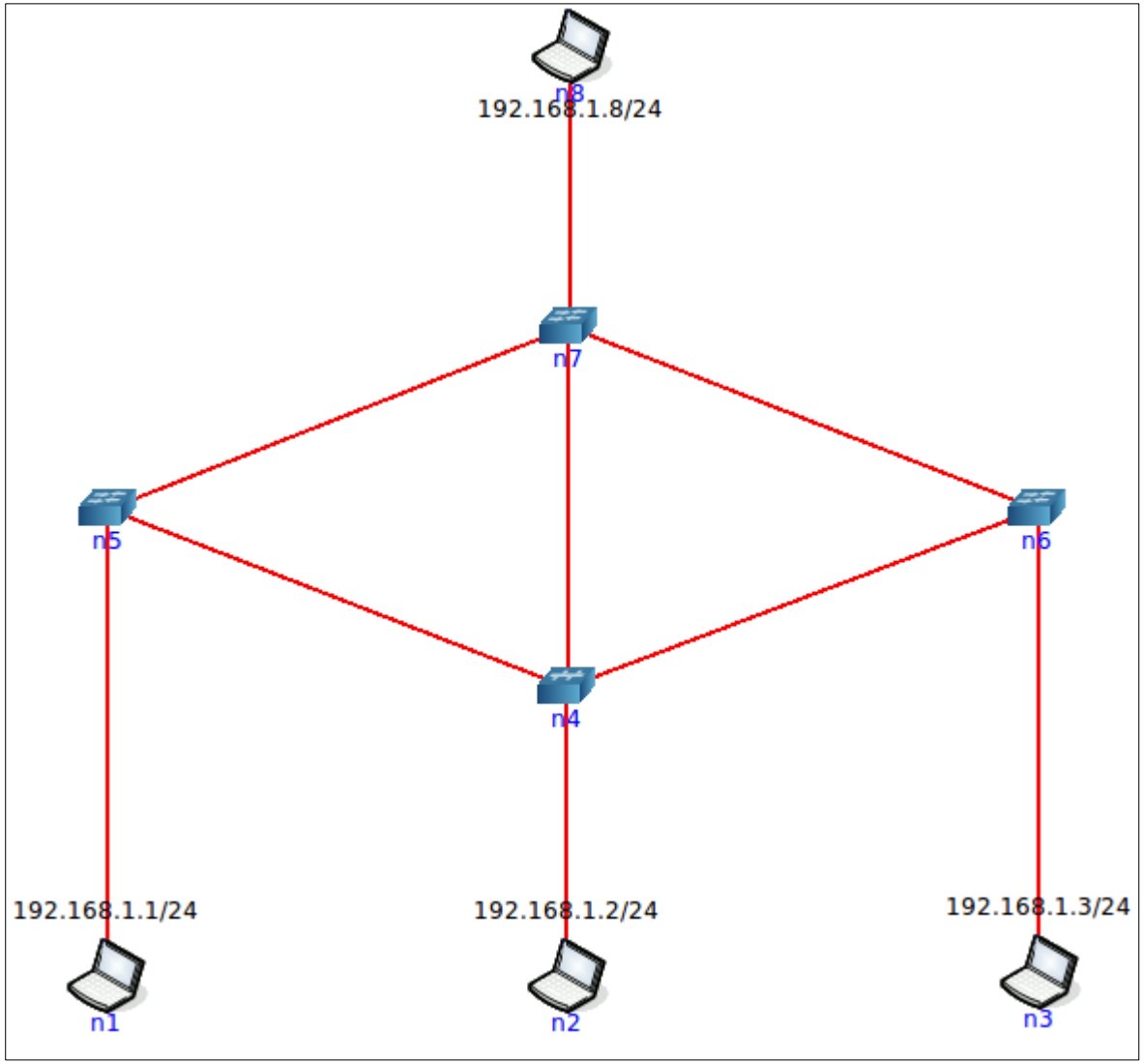
Spanning Tree Protocol

```
Frame: 52 bytes on wire (416 bits), on interface 0
IEEE 802.3 Ethernet
Logical-Link Control
Spanning Tree Protocol
    Protocol Identifier: Spanning Tree Protocol (0x0000)
    Protocol Version Identifier: Spanning Tree (0)
    BPDU Type: Configuration (0x00)
    BPDU flags: 0x00
        0... .... = Topology Change Acknowledgement: No
        .... ...0 = Topology Change: No
    Root Identifier: 32768 / 0 / 12:f0:b2:66:63:90
        Root Bridge Priority: 32768
        Root Bridge System ID Extension: 0
        Root Bridge System ID: 12:f0:b2:66:63:90 (12:f0:b2:66:63:90)
    Root Path Cost: 0
    Bridge Identifier: 32768 / 0 / 12:f0:b2:66:63:90
        Bridge Priority: 32768
        Bridge System ID Extension: 0
        Bridge System ID: 12:f0:b2:66:63:90 (12:f0:b2:66:63:90)
    Port identifier: 0x8003
    Message Age: 0
    Max Age: 20
    Hello Time: 2
    Forward Delay: 2
```

Review the bridges

```
root@NTE-i386:~# sudo brctl showmacs b.4.29
  port no mac addr      is local? ageing timer
     2 12:f0:b2:66:63:90   yes        0.00
     3 36:ac:5e:6f:00:73   yes        0.00
     1 3e:53:b4:a5:c6:ed   yes        0.00

  root@NTE-i386:~# sudo brctl showmacs b.5.29
  port no mac addr      is local? ageing timer
     1 22:43:ed:ab:fe:a4   yes        0.00
     2 26:ae:6f:53:fc:84   yes        0.00
     1 3e:53:b4:a5:c6:ed   no         0.86
     3 f2:e1:fb:cb:c7:86   yes        0.00

  root@NTE-i386:~# sudo brctl showmacs b.6.29
  port no mac addr      is local? ageing timer
     3 3e:53:b4:a5:c6:ed   no         0.15
     1 66:d0:46:59:b2:ec   yes        0.00
     3 92:55:07:58:7a:03   yes        0.00
     2 b6:c9:d0:0b:4e:f9   yes        0.00
```

Switching Lab



Virtual LANs (VLAN)

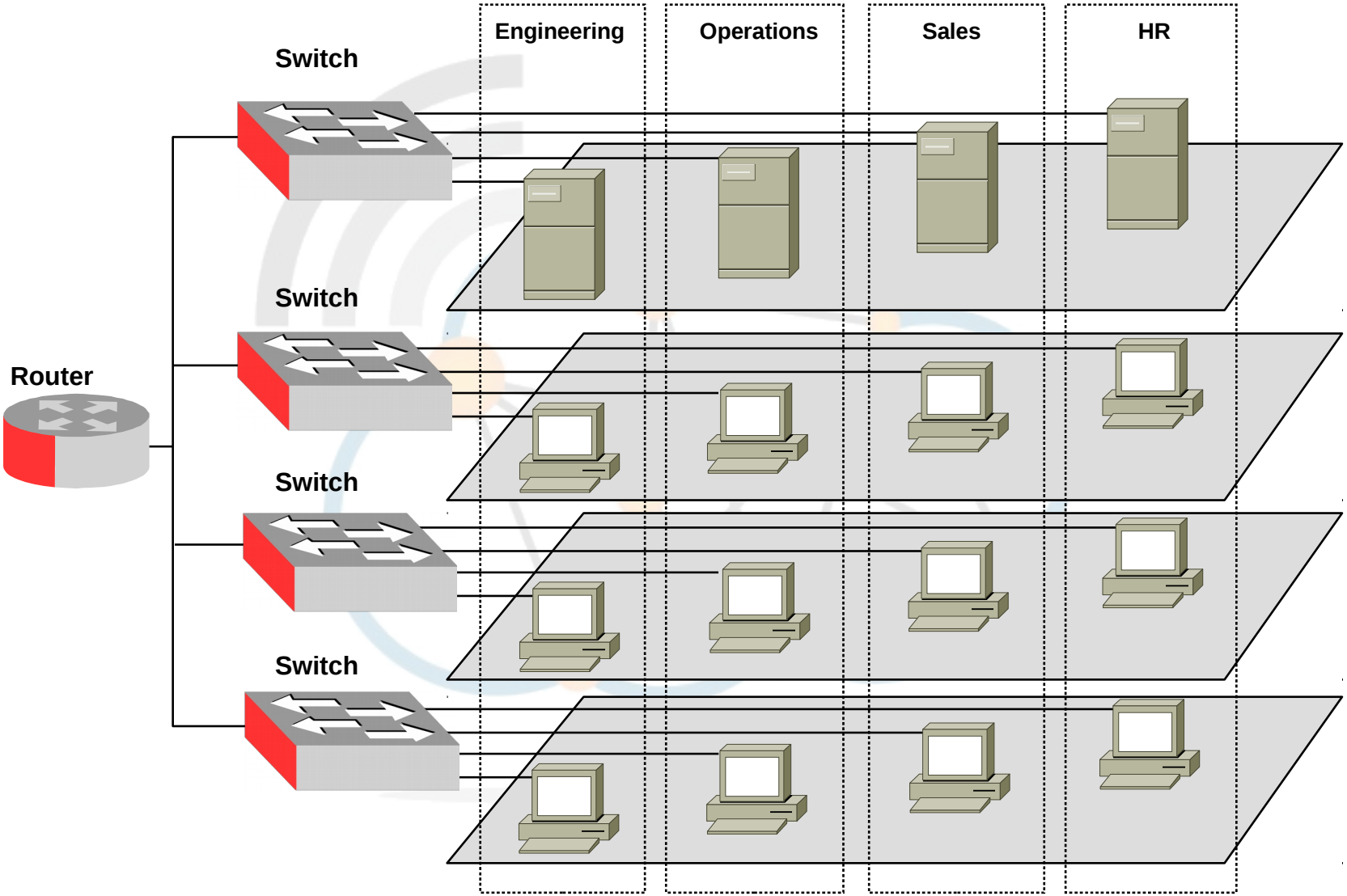
Diarmuid Ó Briain
CEng, FIEI, FIET, CISSP

diarmuid@obriain.com

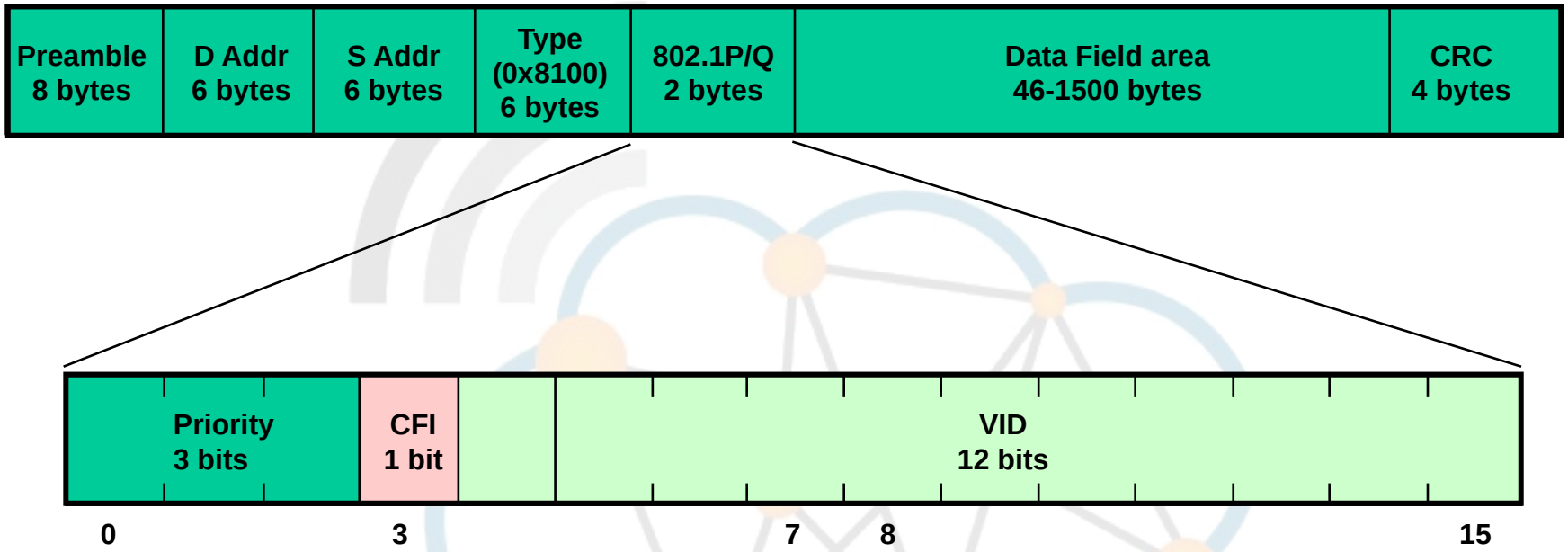
Virtual LANs (VLANs)

- Allow network managers to group switch ports and users connected to them into logically defined communities of interest.
- These groupings can be coworkers within the same department, a cross-functional product team, or diverse users sharing the same network application or software.
- VLANs completely remove the physical constraints of workgroup communications across the enterprise.

VLANs


Engineering
Operations
Sales
HR
Switch
Switch
Switch
Switch
Router

802.1P/Q Tag in Ethernet Frame

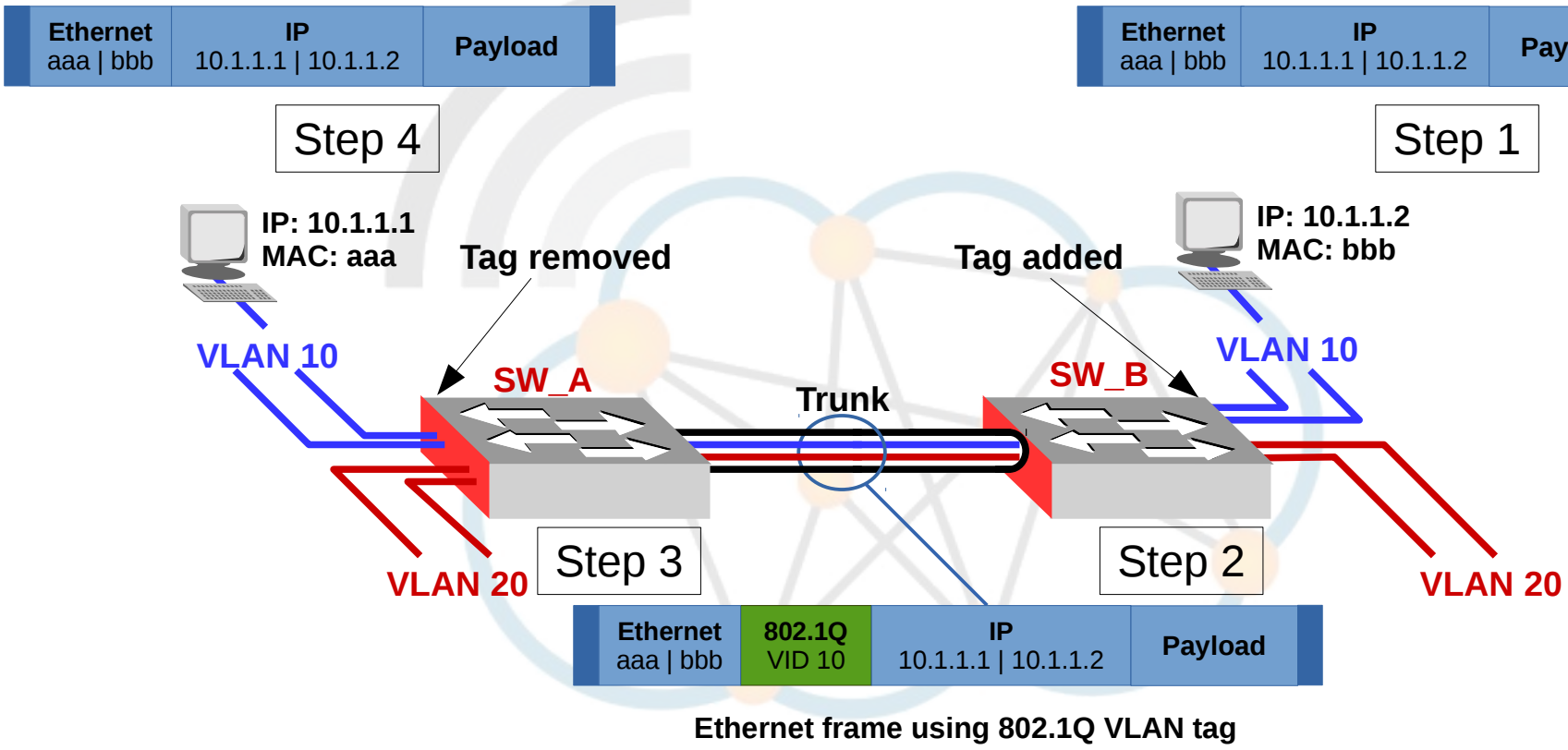


CRC - Cyclical Redundancy Check

CFI - Canonical Format Indicator is a single-bit flag, always set to zero for Ethernet switches.

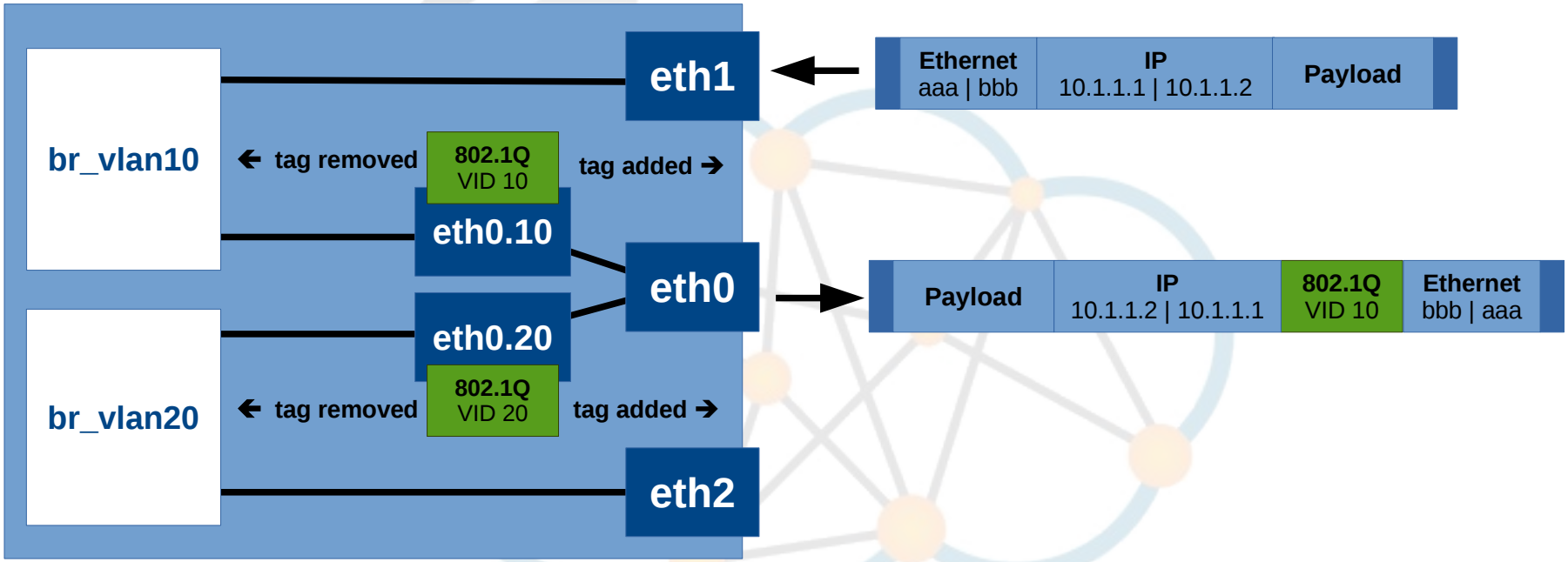
VID - VLAN Identifier 2^{12} = 4096 VLANs. VID 0 is used to identify priority frames and value 4095 (FFF) is reserved, maximum VLAN configurations are 4,094.

VLAN Explicit Tagging



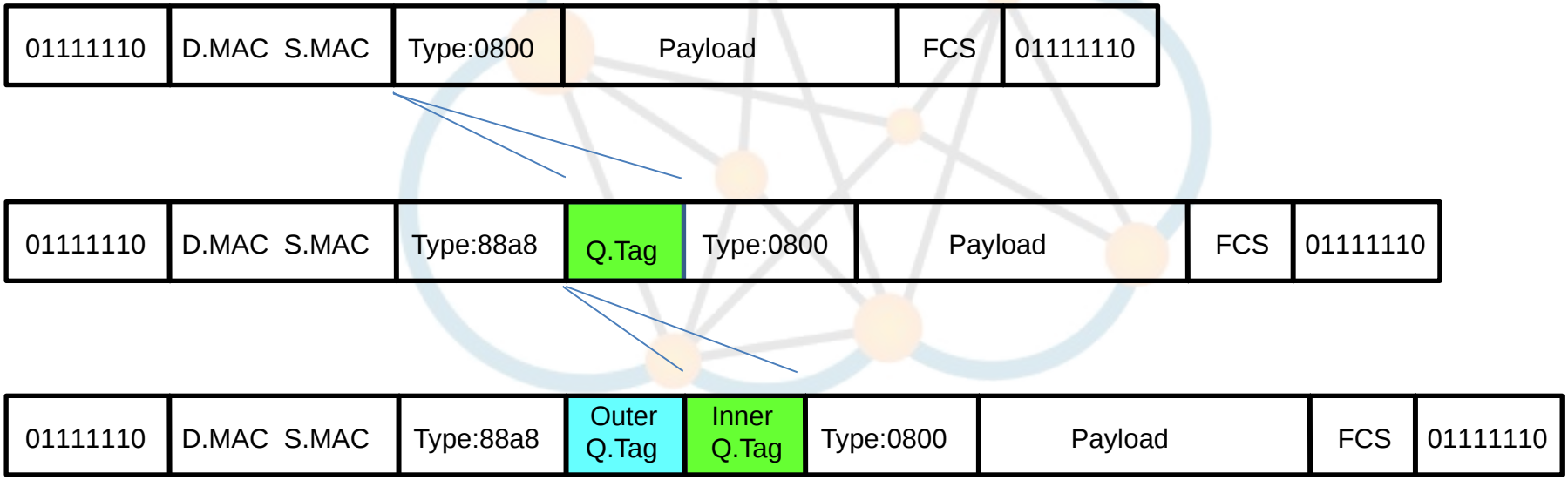
Ethernet frame using 802.1Q VLAN tag

Ethernet frame using 802.1Q VLAN tag

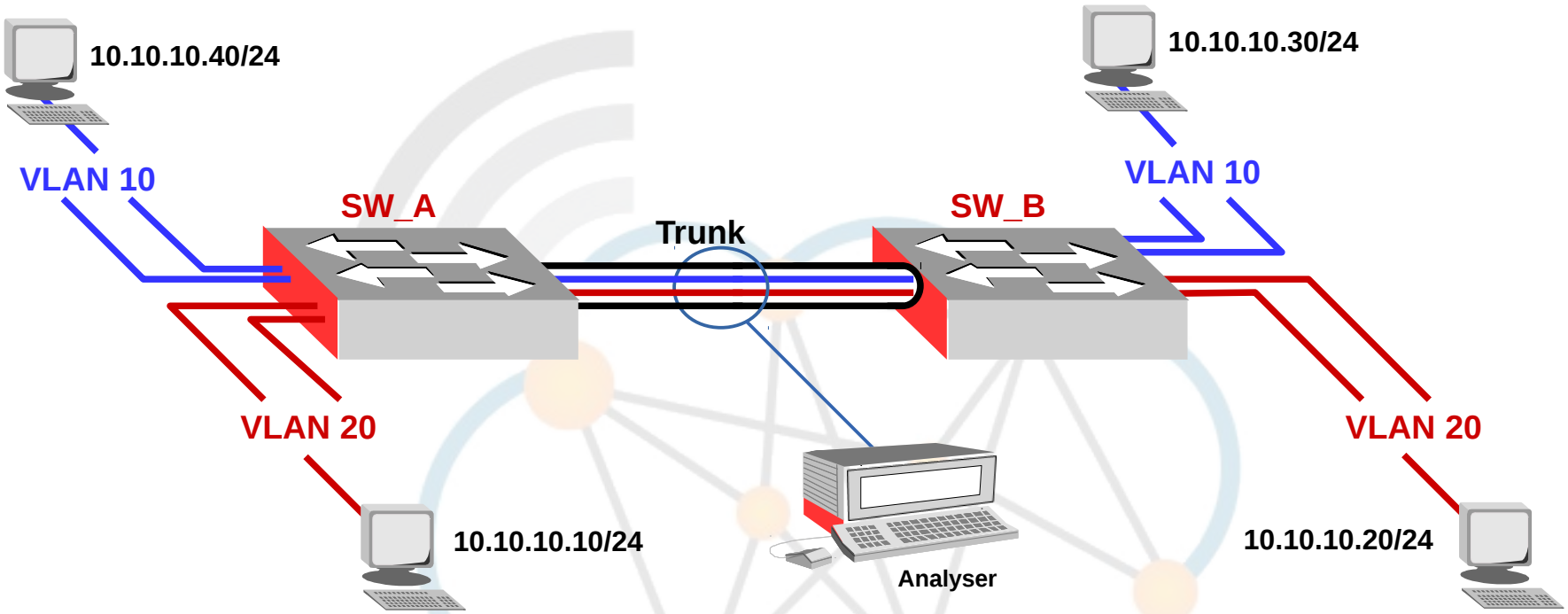


802.1ad (Q-inQ)

- Amendment to IEEE 802.1Q to allow VLANs within VLANs, Provider bridging, Stacked VLANs or simply QinQ or Q-in-Q.



Provider tagging



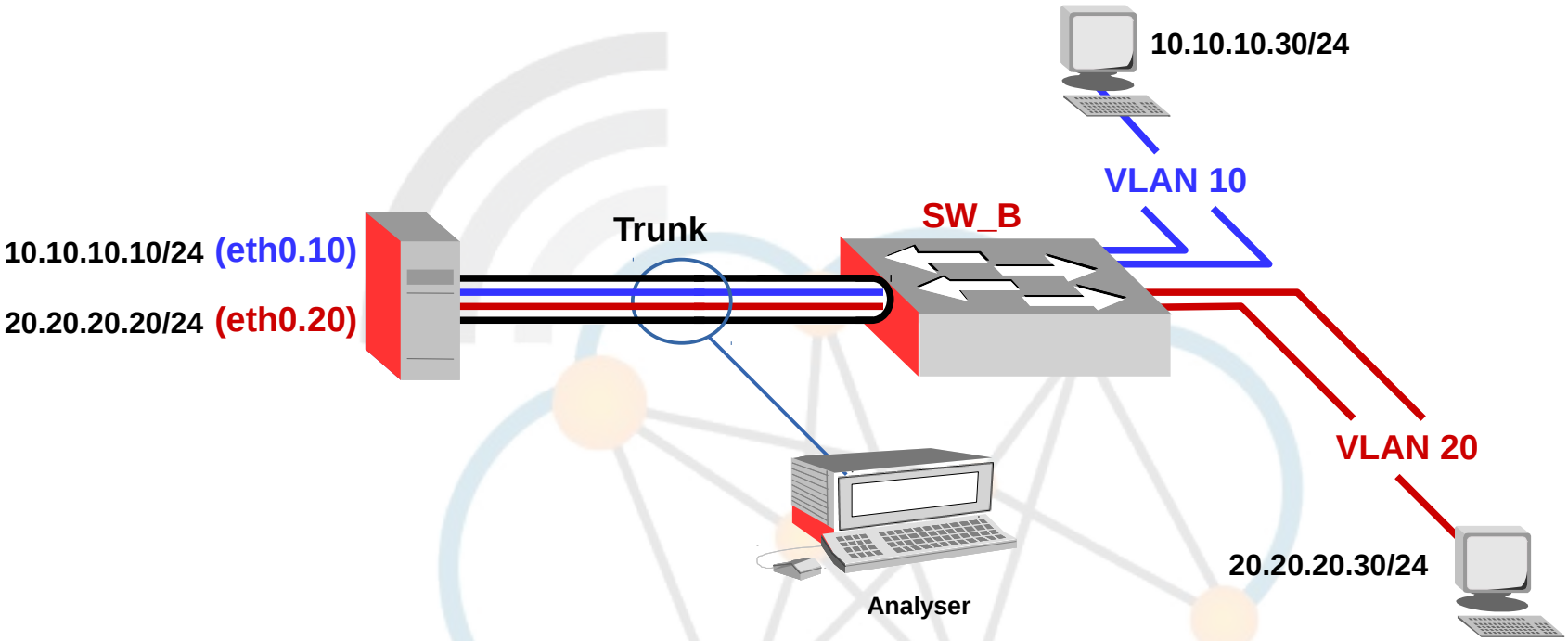
Ethernet frame using 802.1Q VLAN tag

01111110	**Ethernet** Dst MAC, Src MAC	**802.1Q** VID 10	**IP** Dst IP, Src IP	**Payload**	01111110

Ethernet frame using 802.1ad VLAN tag

01111110	**Ethernet** Dst MAC, Src MAC	**802.1ad** S-tag 10	**IP** Dst IP, Src IP	**Payload**	01111110

VLANs on GNU/Linux



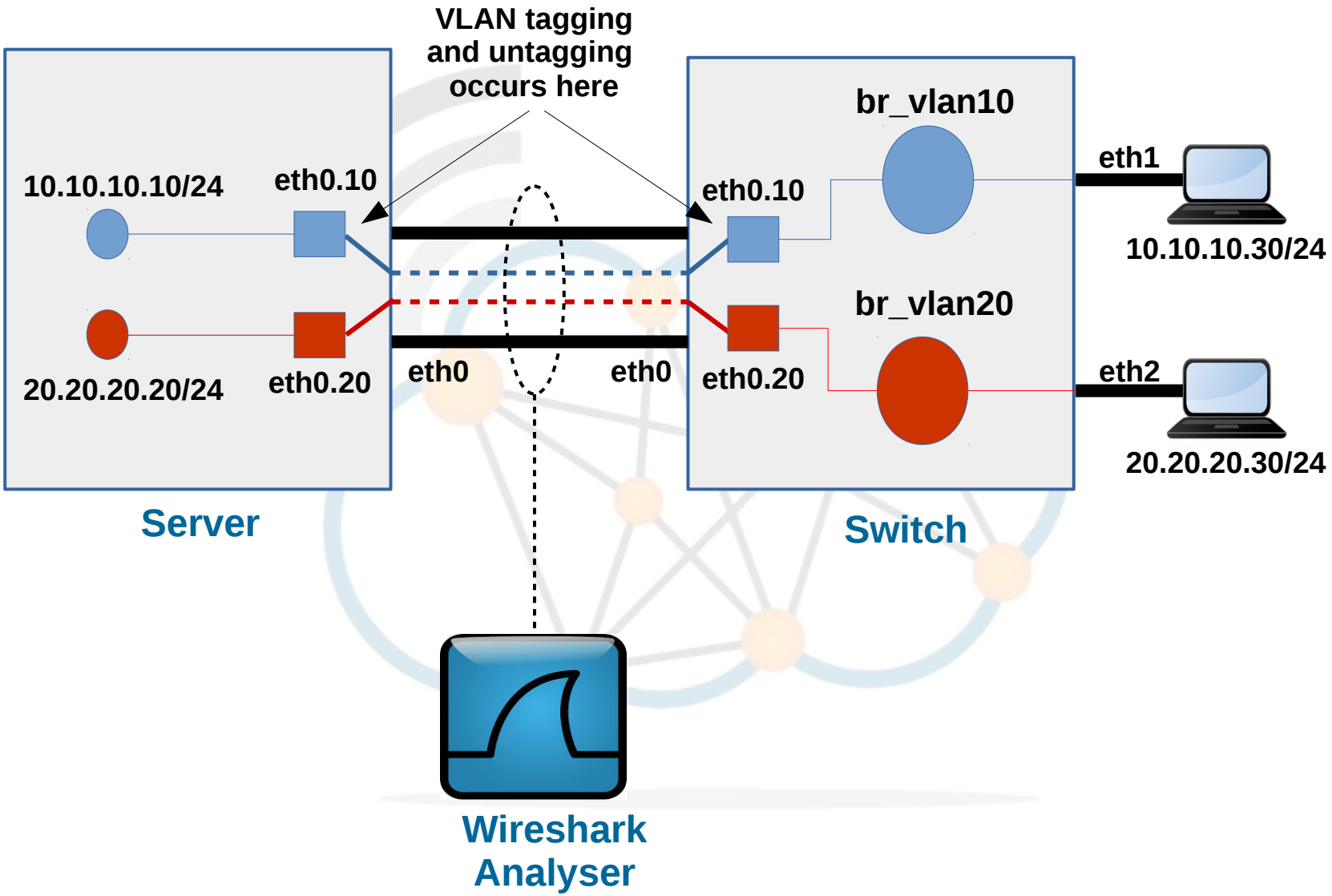
Ethernet frame using 802.1Q VLAN tag

01111110	**Ethernet** Dst MAC, Src MAC	**802.1Q** VID 10	**IP** Dst IP, Src IP	**Payload**	01111110

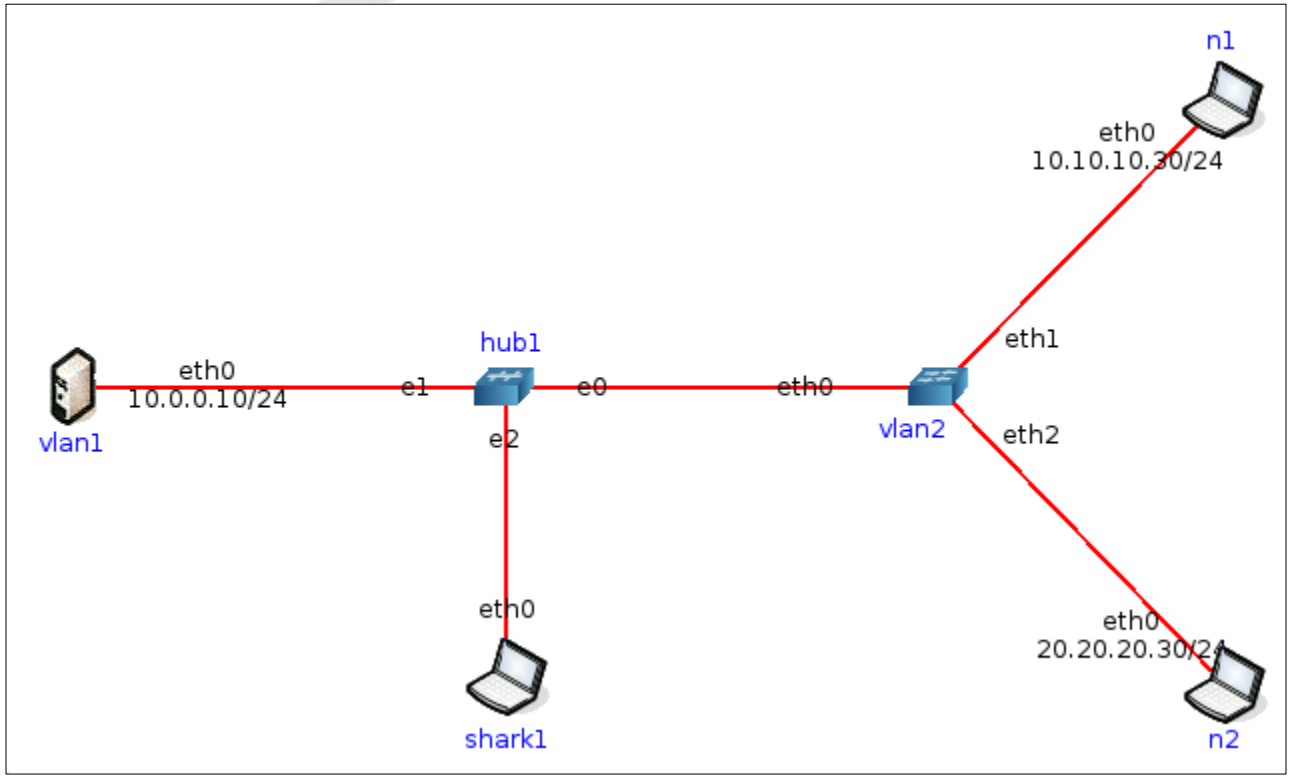
Ethernet frame using 802.1ad VLAN tag

01111110	**Ethernet** Dst MAC, Src MAC	**802.1ad** S-tag 10	**IP** Dst IP, Src IP	**Payload**	01111110

VLAN example logical diagram



VLAN Example



Confirm 8021q kernel module is loaded

- Enable 802.1q on the NTE Server before running the emulation.

```
root@NTE-i386:~# lsmod |grep 8021q

root@NTE-i386:~# modprobe 8021q

root@NTE-i386:~# lsmod |grep 8021q
8021q          18824  0
garp           13025  1 8021q
```

Check interface and add sub-interfaces

```
root@vlan1:~# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN mode DEFAULT group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether 00:00:00:aa:00:00 brd ff:ff:ff:ff:ff:ff
```

```
root@vlan1:~# ip link add link eth0 name eth0.10 type vlan id 10
root@vlan1:~# ip link add link eth0 name eth0.20 type vlan id 20
```

Review the sub-interfaces

```
root@vlan1:~# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN mode DEFAULT group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether 00:00:00:aa:00:00 brd ff:ff:ff:ff:ff:ff
3: eth0.10@eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop
state DOWN mode DEFAULT group default link/ether
00:00:00:aa:00:00 brd ff:ff:ff:ff:ff:ff
4: eth0.20@eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop
state DOWN mode DEFAULT group default link/ether
00:00:00:aa:00:00 brd ff:ff:ff:ff:ff:ff
```

Add IP address and enable sub-interfaces

```
root@vlan1:~# ip addr add 10.10.10.10/24 dev eth0.10
root@vlan1:~# ip addr add 20.20.20.20/24 dev eth0.20
```

Bring up the interface and its new sub-interfaces.

```
root@vlan1:~# ip link set dev eth0 up
root@vlan1:~# ip link set dev eth0.10 up
root@vlan1:~# ip link set dev eth0.20 up
```

Review the sub-interfaces again

```
root@vlan1:~# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN mode DEFAULT group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether 00:00:00:aa:00:00 brd ff:ff:ff:ff:ff:ff
3: eth0.10@eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop
state UP mode DEFAULT group default link/ether
00:00:00:aa:00:00 brd ff:ff:ff:ff:ff:ff
4: eth0.20@eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop
state UP mode DEFAULT group default link/ether
00:00:00:aa:00:00 brd ff:ff:ff:ff:ff:ff
```

Configure Switch n2

```
root@vlan2:~# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN mode DEFAULT group default link/loopback
00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UP mode DEFAULT group default qlen 1000
link/ether 00:00:00:aa:00:02 brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UP mode DEFAULT group default qlen 1000
link/ether 00:00:00:aa:00:03 brd ff:ff:ff:ff:ff:ff
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UP mode DEFAULT group default qlen 1000
link/ether 00:00:00:aa:00:05 brd ff:ff:ff:ff:ff:ff
```

Create sub-interfaces for VLANs

```
root@vlan2:~# ip link add link eth0 name eth0.10 type vlan id 10
root@vlan2:~# ip link add link eth0 name eth0.20 type vlan id 20
root@vlan2:~# ip link set dev eth0.10 up
root@vlan2:~# ip link set dev eth0.20 up
```

Check new sub-interfaces

```
root@vlan2:~# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN mode DEFAULT group default link/loopback
00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0.10@eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop
state UP mode DEFAULT group default link/ether
00:00:00:aa:00:02 brd ff:ff:ff:ff:ff:ff
3: eth0.20@eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop
state UP mode DEFAULT group default link/ether
00:00:00:aa:00:02 brd ff:ff:ff:ff:ff:ff
4: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UP mode DEFAULT group default qlen 1000
link/ether 00:00:00:aa:00:02 brd ff:ff:ff:ff:ff:ff
5: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UP mode DEFAULT group default qlen 1000
link/ether 00:00:00:aa:00:03 brd ff:ff:ff:ff:ff:ff
6: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UP mode DEFAULT group default qlen 1000
link/ether 00:00:00:aa:00:05 brd ff:ff:ff:ff:ff:ff
```

Create bridges to link sub and actual interfaces

Create bridges for each VLAN.

```
root@vlan2:~# brctl addbr br_vlan10
root@vlan2:~# brctl addbr br_vlan20
```

Add the sub-interfaces and the appropriate interfaces on *n2* to their respective bridges.

```
root@vlan2:~# brctl addif br_vlan10 eth0.10 eth1
root@vlan2:~# brctl addif br_vlan20 eth0.20 eth2
```

Bring the bridges to a state of UP.

```
root@vlan2:~# ip link set dev br_vlan10 up
root@vlan2:~# ip link set dev br_vlan20 up
```

Test the VLANs

Ping from Host n2 to 20.20.20.20.

```
Frame: 102 bytes on wire (816 bits), on interface 0
Ethernet II, Src: 00:00:00_aa:00:06, Dst: 00:00:00_aa:00:00
    Type: 802.1Q Virtual LAN (0x8100)
802.1Q Virtual LAN, PRI: 0, CFI: 0, ID: 20
    Type: IP (0x0800)
Internet Protocol Version 4, Src: 20.20.20.30, Dst: 20.20.20.20
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xf83a [correct]
    Identifier (BE): 20 (0x0014)
    Identifier (LE): 5120 (0x1400)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    [Response frame: 2]
    Timestamp from icmp data: Feb 20, 2016 08:55:17.339026000 GMT
    Data (48 bytes)
```

802.1ad Q-in-Q

Diarmuid Ó Briain
CEng, FIEI, FIET, CISSP

diarmuid@obriain.com

802.1ad configuration

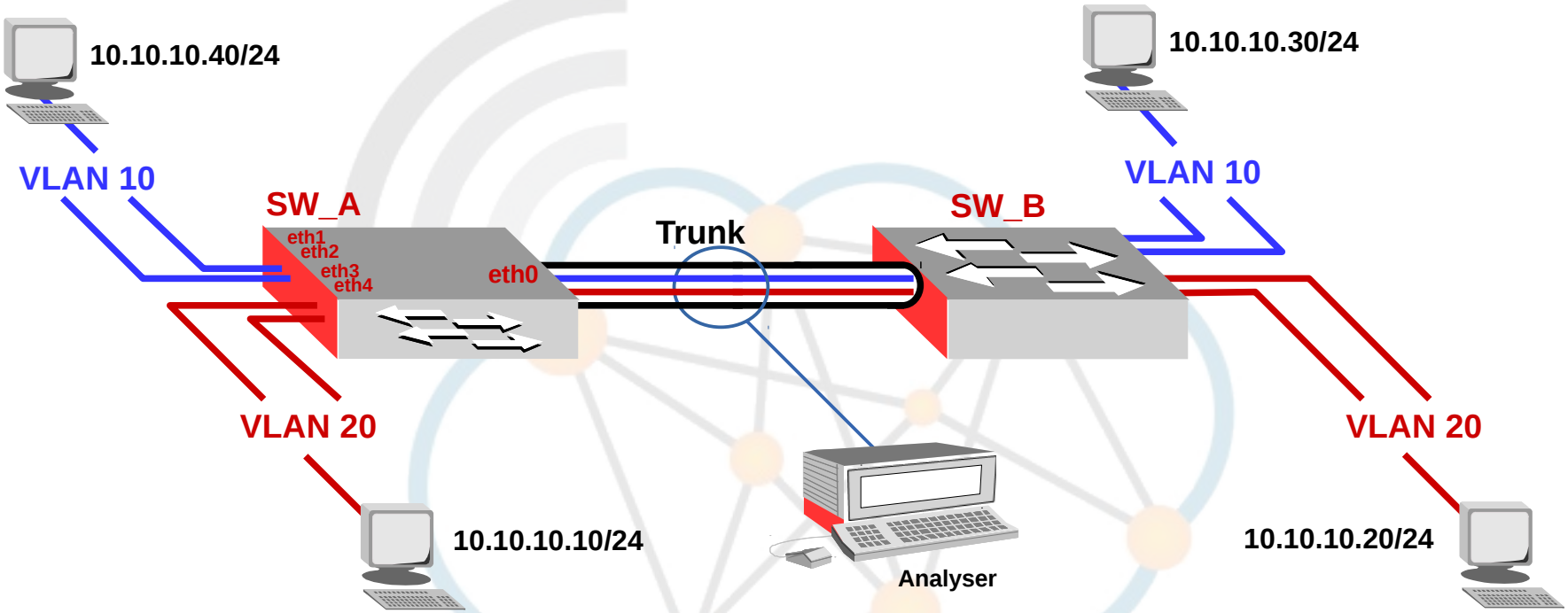
Difference is the addition of the proto field.

```
# ip link add link eth0 name eth0.10 type vlan proto 802.1ad id 10
# ip link add link eth0 name eth0.20 type vlan proto 802.1ad id 20

# ip addr add 10.10.10.10/24 dev eth0.10
# ip addr add 20.20.20.20/24 dev eth0.20

# ip link set dev eth0 up
# ip link set dev eth0.10 up
# ip link set dev eth0.20 up
```

802.1ad packet trace

```
Frame: 102 bytes on wire (816 bits)
Ethernet II, Src: 00:12:3f:dc:ab:47, Dst: d4:ca:6d:61:dd:89
    Destination: d4:ca:6d:61:dd:89
    Source: 00:12:3f:dc:ab:47
    Type: 802.1ad Provider Bridge (Q-in-Q) (0x88a8)
IEEE 802.1ad, ID: 10
    000. .... .... .... = Priority: 0
    ...0 .... .... .... = DEI: 0
    .... 0000 0000 1010 = ID: 10
    Type: IP (0x0800)
Internet Protocol Version 4, Src: 10.10.10.10, Dst: 10.10.10.30
Internet Control Message Protocol
```

IEEE 802.1ad support on GNU/Linux



Ethernet frame using 802.1ad VLAN tag

01111110	**Ethernet** Dst MAC, Src MAC	**802.1ad** S-tag 10	**IP** Dst IP, Src IP	**Payload**	01111110

802.1ad Switch configuration

Create the VLAN subinterfaces to deal with the incoming trunk interface containing the VLANs on physical interface *eth0*.

```
# ip link add link eth0 name eth0.10 type vlan proto 802.1ad id 10
# ip link add link eth0 name eth0.20 type vlan proto 802.1ad id 20

# ip link set dev eth0 up
# ip link set dev eth0.10 up
# ip link set dev eth0.20 up
```

Bring up the interfaces that connect to the LANs.

```
# ip link set dev eth1 up
# ip link set dev eth2 up
# ip link set dev eth3 up
# ip link set dev eth4 up
```

802.1ad Provider configuration

Create bridges to link the VLANs to their appropriate interfaces.

```
# brctl addbr br_vlan_10
# brctl addbr br_vlan_20
```

Assign interfaces to the various bridges.

```
# brctl addif br_vlan_10 eth0.10 eth1 eth2
# brctl addif br_vlan_20 eth0.20 eth3 eth4
```

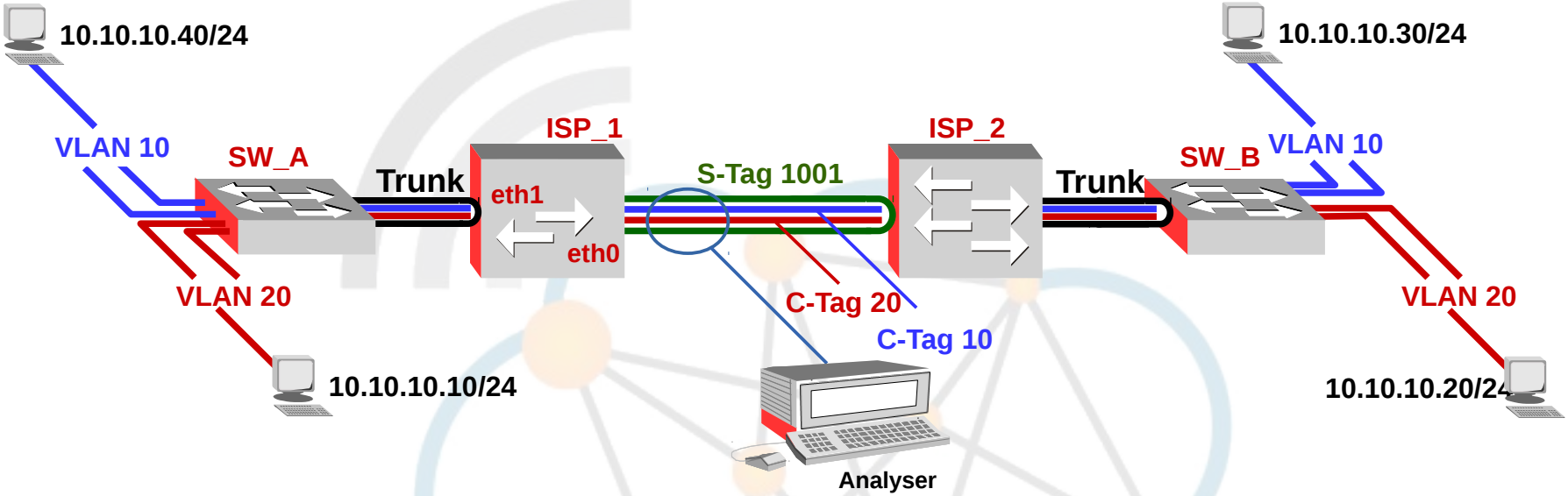
Bring up the bridges.

```
# ip link set dev br_vlan_10 up
# ip link set dev br_vlan_20 up
```

802.1ad Provider configuration

```
Frame: 78 bytes on wire (624 bits)
Ethernet II, Src: d4:ca:6d:61:dd:89, Dst: 00:0c:42:8b:73:e4
    Type: 802.1Q Virtual LAN (0x8100)
802.1Q Virtual LAN, PRI: 0, CFI: 0, ID: 1001
    000. .... .... .... = Priority: Best Effort (default) (0)
    ...0 .... .... .... = CFI: Canonical (0)
    .... 0011 1110 1001 = ID: 1001
    Type: 802.1ad Provider Bridge (Q-in-Q) (0x88a8)
IEEE 802.1ad, ID: 10
    000. .... .... .... = Priority: 0
    ...0 .... .... .... = DEI: 0
    .... 0000 0000 1010 = ID: 10
    Type: IP (0x0800)
Internet Protocol Version 4, Src: 10.10.10.30, Dst: 10.10.10.40
Internet Control Message Protocol
```

GNU/Linux as a Service Provider bridge



Ethernet frame using 802.1ad outer S-tag and 802.1Q inner VLAN tag

01111110	**Ethernet** Dst MAC, Src MAC	**802.1ad** S-tag 1001	**802.1Q** VID 10	**IP** Dst IP, Src IP	**Payload**	01111110

Ethernet frame using 802.1ad outer S-tag and inner C-tag

01111110	**Ethernet** Dst MAC, Src MAC	**802.1ad** S-tag 1001, C-tag 10	**IP** Dst IP, Src IP	**Payload**	01111110

802.1ad Provider configuration

Create a sub-interface on eth0 to handle the VLAN S-tag 1001 and bring the physical and sub-interfaces up.

```
# ip link add link eth0 name eth0.1001 type vlan proto 802.1ad id 1001
# ip link set dev eth0 up
# ip link set dev eth0.1001 up
```

Bring up the eth1 interface which will be connected to the trunk from *SW_A*.

```
# ip link set dev eth1 up
```

Create a bridge *br_vlan_1001* and put the *eth0.1001* sub-interface and *eth1* into it.
Then bring the bridge up.

```
# brctl addbr br_vlan_1001
# brctl addif br_vlan_1001 eth0.1001 eth1

# ip link set dev br_vlan_1001 up
```

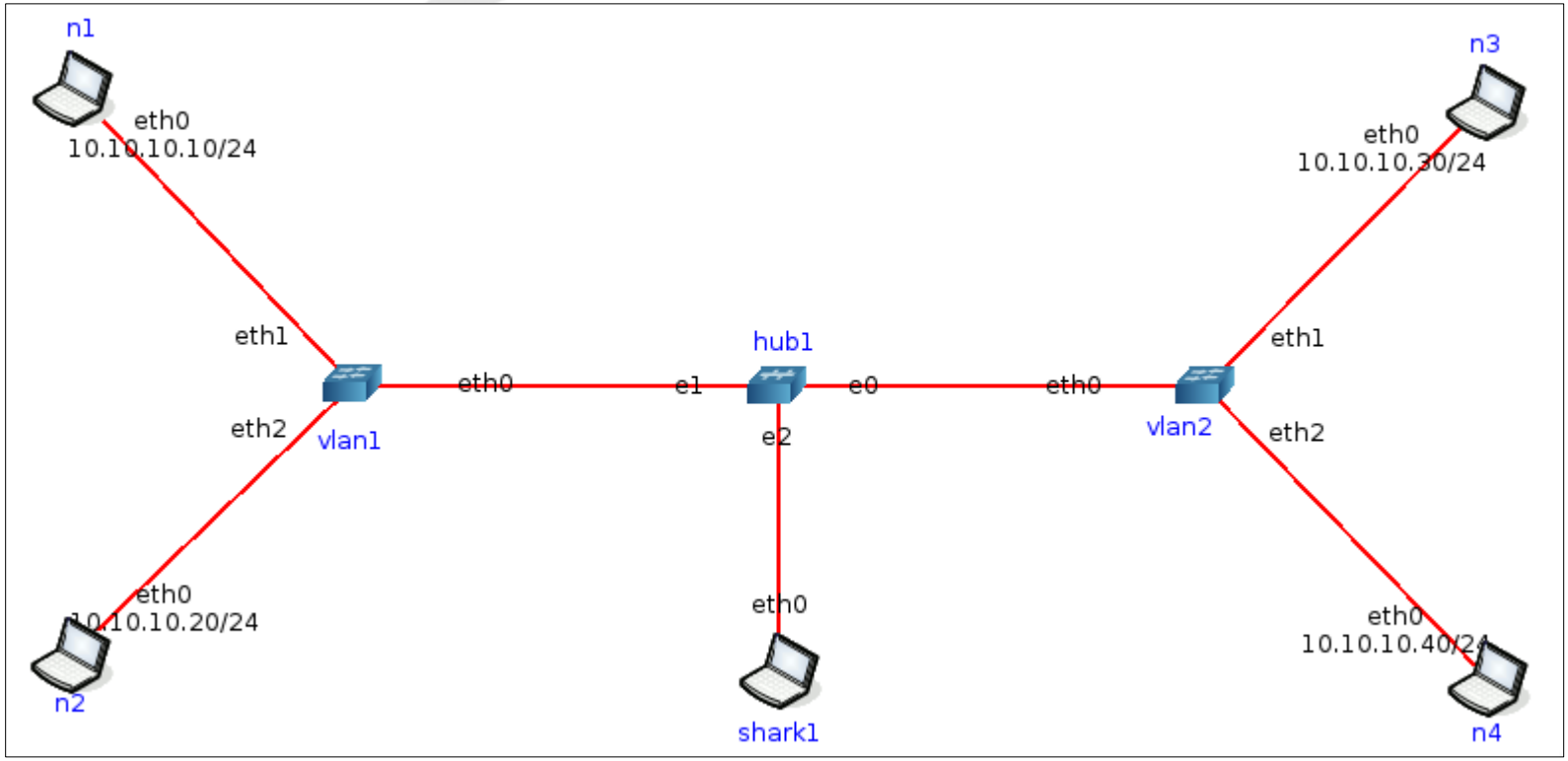
802.1ad packet trace – 802.1Q in 802.1ad

```
Frame: 106 bytes on wire (848 bits)
Ethernet II, Src: d4:ca:6d:61:dd:89, Dst: 00:12:3f:dc:ab:47
    Destination: 00:12:3f:dc:ab:47
    Source: d4:ca:6d:61:dd:89
    Type: 802.1ad Provider Bridge (Q-in-Q) (0x88a8)
IEEE 802.1ad, ID: 1001
    000. .... .... .... = Priority: 0
    ...0 .... .... .... = DEI: 0
    .... 0011 1110 1001 = ID: 1001
    Type: 802.1Q Virtual LAN (0x8100)
802.1Q Virtual LAN, PRI: 0, CFI: 0, ID: 10
    000. .... .... .... = Priority: Best Effort (default) (0)
    ...0 .... .... .... = CFI: Canonical (0)
    .... 0000 0000 1010 = ID: 10
    Type: IP (0x0800)
Internet Protocol Version 4, Src: 10.10.10.40, Dst: 10.10.10.30
Internet Control Message Protocol
```

802.1ad packet trace – 802.1ad C and S tags

```
Frame: 106 bytes on wire (848 bits)
Ethernet II, Src: d4:ca:6d:61:dd:89, Dst: 00:12:3f:dc:ab:47
    Destination: 00:12:3f:dc:ab:47
    Source: d4:ca:6d:61:dd:89
    Type: 802.1ad Provider Bridge (Q-in-Q) (0x88a8)
IEEE 802.1ad, S-VID: 1001, C-VID: 10
    000. .... .... .... = Priority: 0
    ...0 .... .... .... = DEI: 0
    .... 0011 1110 1001 = ID: 1001
    000. .... .... .... = Priority: 0
    ...0 .... .... .... = DEI: 0
    .... 0000 0000 1010 = ID: 10
    Type: IP (0x0800)
Internet Protocol Version 4, Src: 10.10.10.40, Dst: 10.10.10.30
Internet Control Message Protocol
```

VLAN Lab


n1
eth0
10.10.10.10/24
eth1
eth2
vlan1
eth0
e1
hub1
e2
e0
eth0
vlan2
eth1
eth2
n3
eth0
10.10.10.30/24
eth0
10.10.10.20/24
n2
eth0
shark1
eth0
10.10.10.40/24
n4

Thank you

Diarmuid Ó Briain
CEng, FIEI, FIET, CISSP

diarmuid@obriain.com