Experiences in Realizing Large Robust Software Defined Networks

Ravi Manghirmalani
Ramesh Subrahmaniam

Controller Offload

- SDN mandates a split of control and forwarding plane
- The control and forwarding planes communicate through a TCP connection
- Every lookup miss in the forwarding plane causes the controller and forwarding to exchange three messages
  - Packet-In Missed information to Controller
  - Flow-Mod Install a Rule in the forwarding table
  - Packet-Out Forward the packet to outgoing port
- The rate of arrival of new flows will significantly impact this TCP connection
- Applications like Firewall would cause an increased communication between the control and forwarding plane

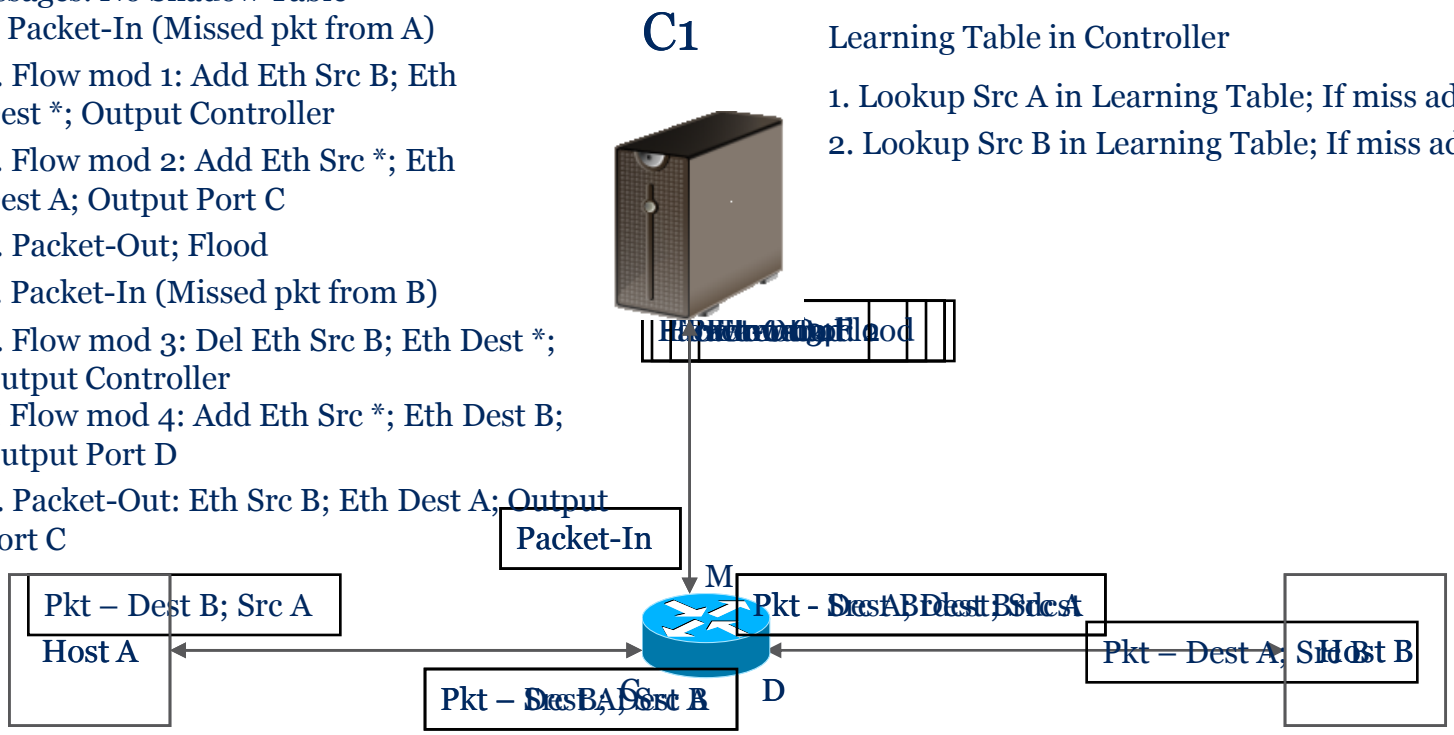
Controller Offload

- Our solution is to implement a shadow table that
  - supports flexible rules
  - allows switches to install rules without involving controller
  - allows switches to monitor sessions
- New Message to Modify Shadow Table instead of the OpenFlow Table
- Algorithm change:
  - Lookup Shadow Table in the case of OpenFlow Table miss
  - If Match in shadow table execute action
  - If Miss in shadow table send packet-in to controller

Controller Offload – Bridging (No Shadow Table)

Messages: No Shadow Table

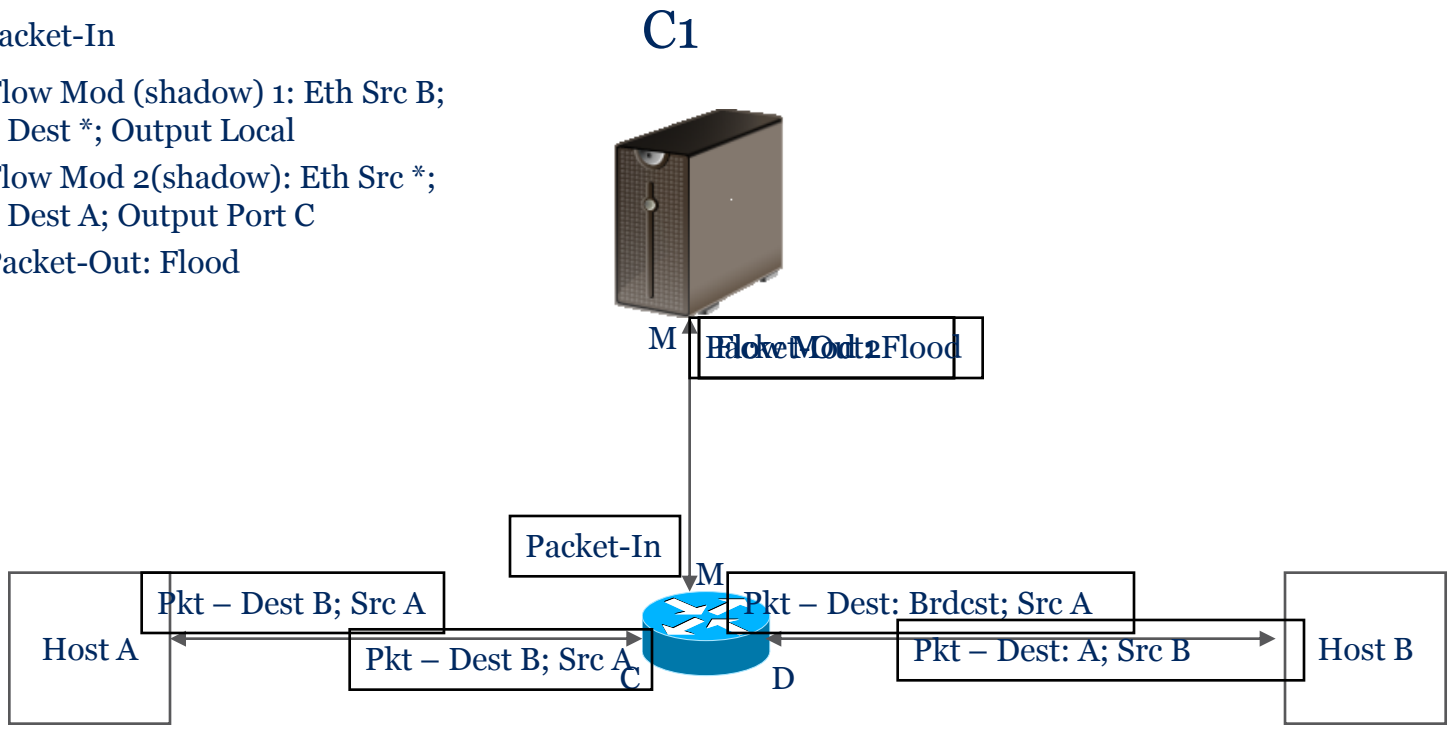
1. Packet-In (Missed pkt from A)
2. Flow mod 1: Add Eth Src B; Eth Dest *; Output Controller
3. Flow mod 2: Add Eth Src *; Eth Dest A; Output Port C
4. Packet-Out; Flood
5. Packet-In (Missed pkt from B)
6. Flow mod 3: Del Eth Src B; Eth Dest *; Output Controller
7. Flow mod 4: Add Eth Src *; Eth Dest B; Output Port D
8. Packet-Out: Eth Src B; Eth Dest A; Output Port C



Controller Offload – Bridging (With Shadow Table)

Messages: With Shadow Table

1. Packet-In
2. Flow Mod (shadow) 1: Eth Src B; Eth Dest *; Output Local
3. Flow Mod 2(shadow): Eth Src *; Eth Dest A; Output Port C
4. Packet-Out: Flood



Learning Table in Switch

1. Lookup Src A in Learning Table; If miss add A port C
2. Lookup Src B in Learning Table; If miss add B port D

Shadow Table Handler installs rule:

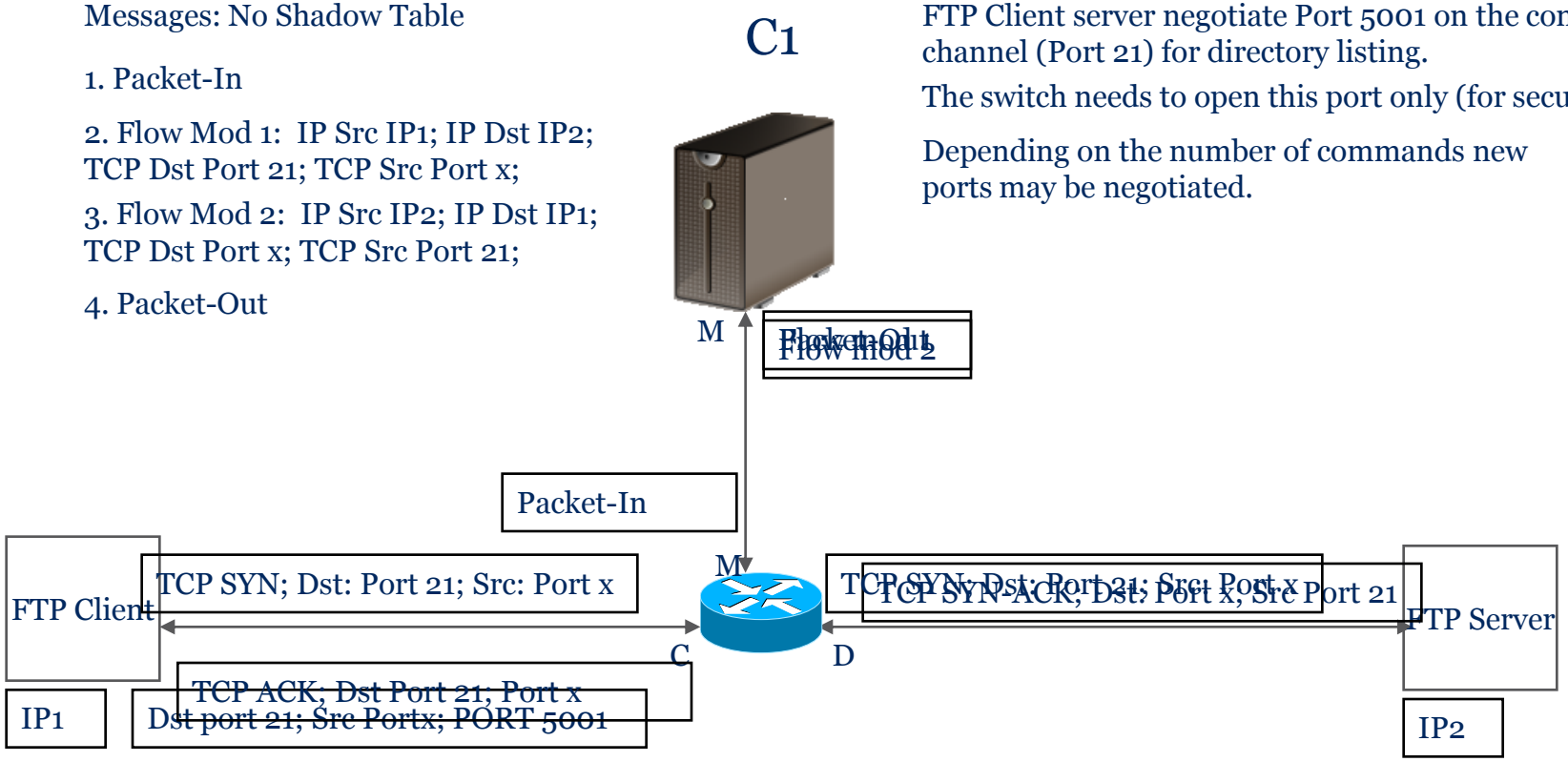
Eth Src *; Dest B; Output Port D

Shadow Table Handler forwards packet out of port C

Controller Offload – Session Monitoring (No Shadow Table)

Messages: No Shadow Table

1. Packet-In
2. Flow Mod 1: IP Src IP1; IP Dst IP2; TCP Dst Port 21; TCP Src Port x;
3. Flow Mod 2: IP Src IP2; IP Dst IP1; TCP Dst Port x; TCP Src Port 21;
4. Packet-Out



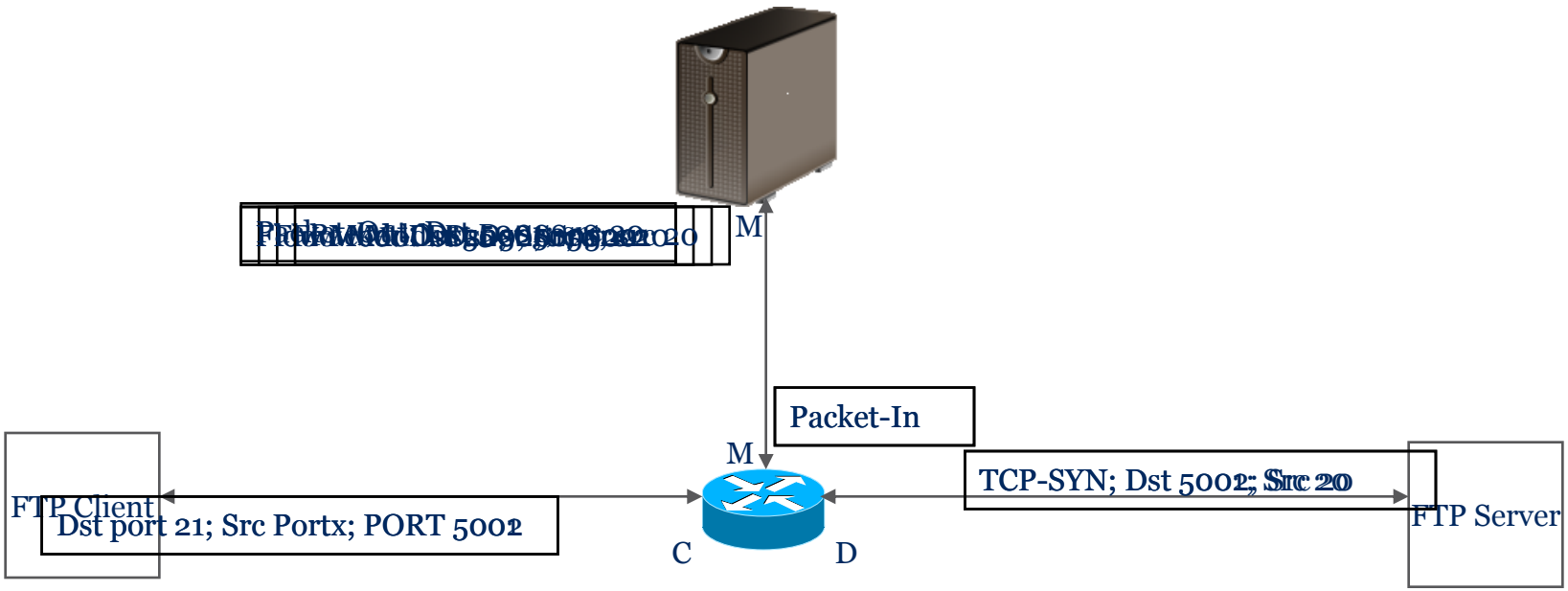
FTP Client server negotiate Port 5001 on the control channel (Port 21) for directory listing.

The switch needs to open this port only (for security).

Depending on the number of commands new ports may be negotiated.

Controller Offload – Session Monitoring (No Shadow Table)

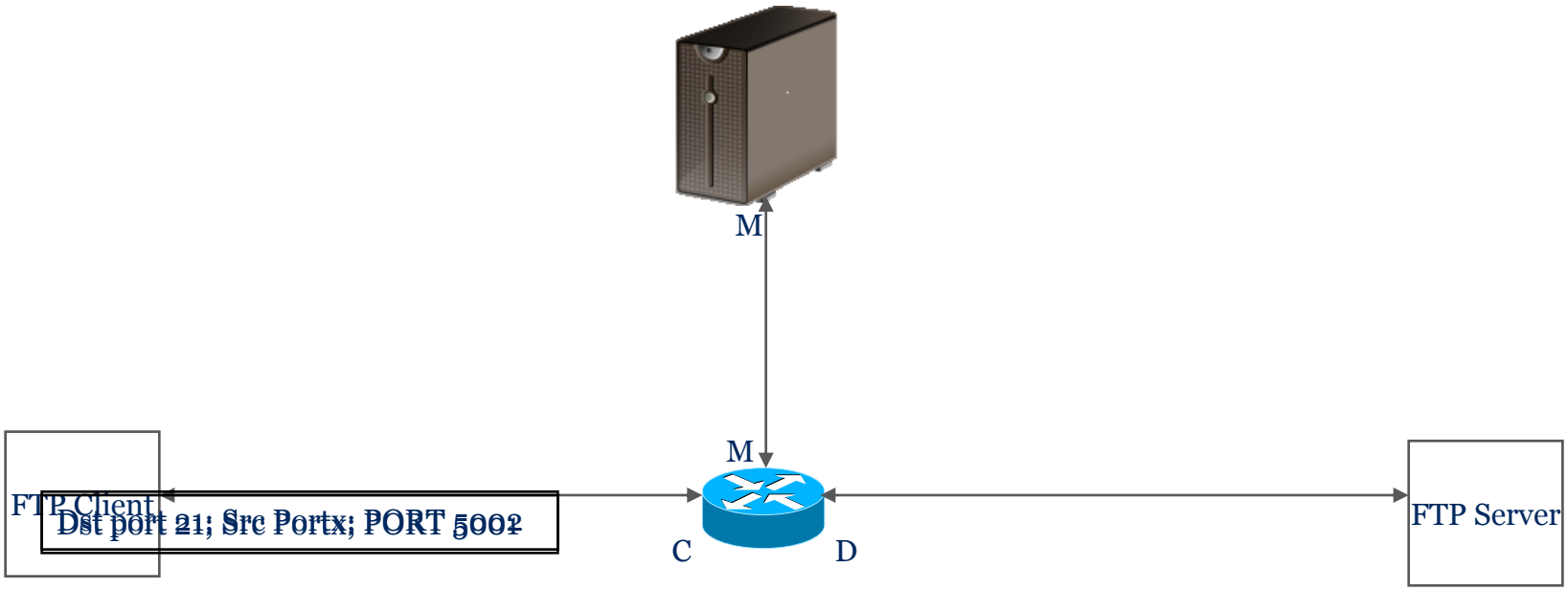
Method 1



Every new data channel port that is negotiated on the control channel will cause a miss in the OpenFlow table and hence 4 messages across the OpenFlow channel to the controller.

Controller Offload – Session Monitoring (No Shadow Table)

Method 2



Every conversation packet on the control channel (port 21) is sent to the controller so that it can parse the PORT command and send a Flow mod to install flow rule for the matching data channel port

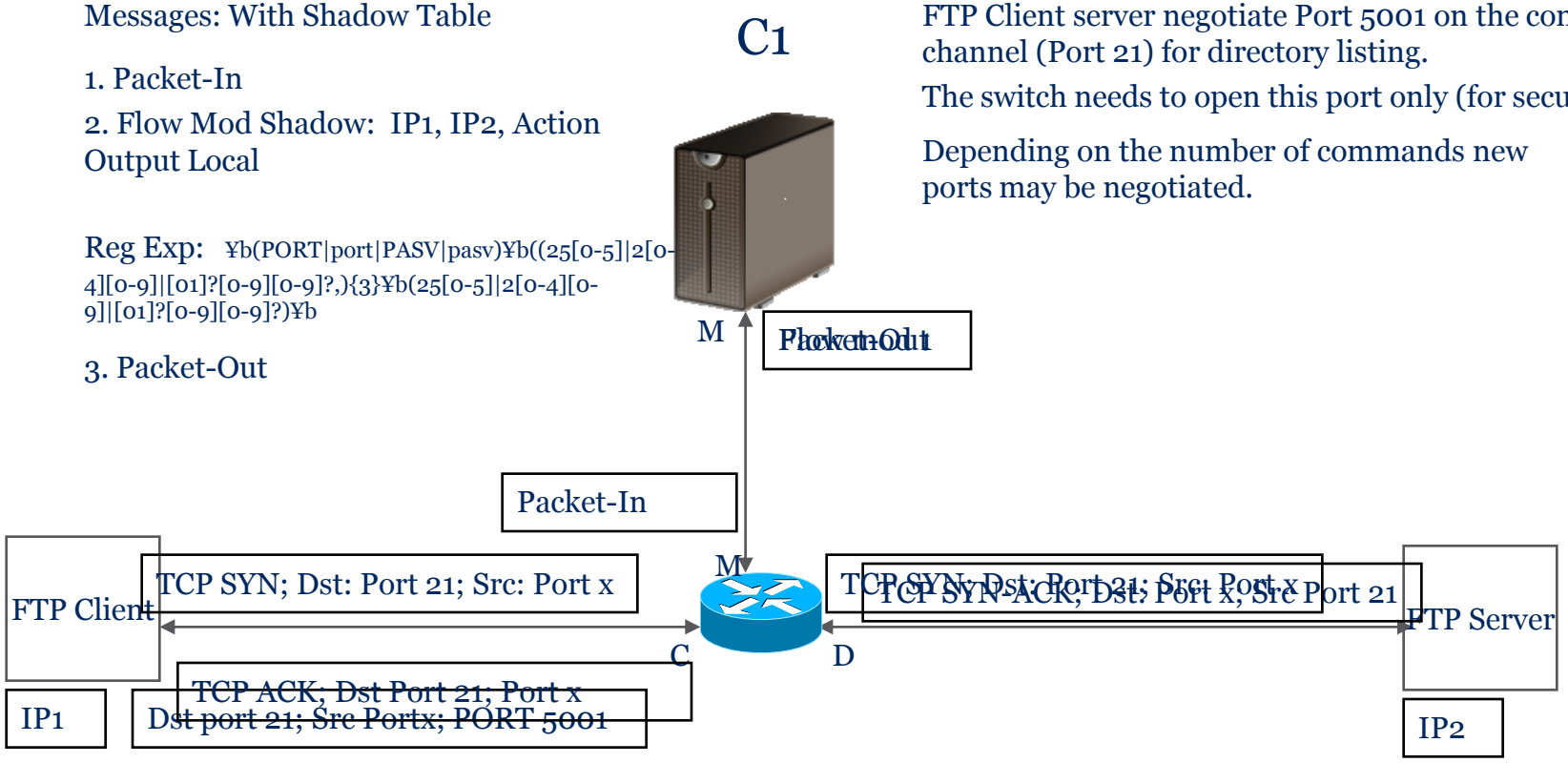
Controller Offload – Session Monitoring (With Shadow Table)

Messages: With Shadow Table

1. Packet-In
2. Flow Mod Shadow: IP1, IP2, Action Output Local

Reg Exp: ¥b(PORT|port|PASV|pasv)¥b((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?,){3}¥b(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)¥b

3. Packet-Out



FTP Client server negotiate Port 5001 on the control channel (Port 21) for directory listing.

The switch needs to open this port only (for security).

Depending on the number of commands new ports may be negotiated.

Controller Offload – Session Monitoring (With Shadow Table)

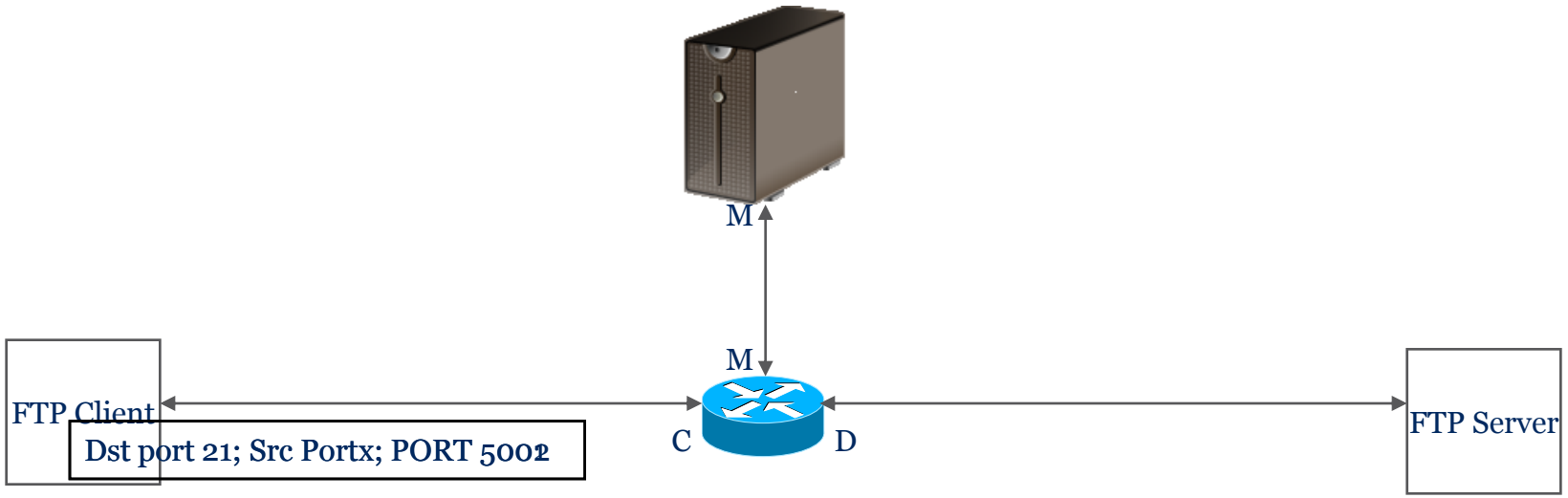
Monitoring with Shadow Table

If IP Src == Client IP and Dest IP == Server IP

Or IP Src == Server IP and Dest IP == Client IP

Trigger Regex Engine with

¥b(PORT|port|PASV|pasv)¥b((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?,){3}¥b(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)¥b



Every conversation packet on the control channel (port 21) is parsed by the regex engine with the expression passed to the shadow table in the switch

It also installs rule for the data channel port being negotiated (both directions)

Forwards the control channel packet to the client/server

Controller Offload – Flexible Rule Matching

ERICSSON

- OpenFlow does not support port/address ranges to be matched
- Takes an inordinate number of rules to support all ports/addresses in a range. Should be done by enumerating the ports/addresses with a rule for every port/address combination
- Port/Address ranges with certain ports/addresses to be excluded is even harder
- Greatly increases controller to switch OpenFlow channel communication

Controller Offload – Flexible Rule Matching

- The offload mechanism rules can be expressed by the key words range, list and exclude, ip, port, mac, source and dest

- The rules with these keywords can be parsed into data structures to be implemented in the shadow table

- These rules will optimize the number of rules in the OpenFlow table

Controller Offload – Flexible Rule Matching

Rules such as the ones shown could be implemented very efficiently by using the shadow table mechanism.

- a. Rule: source IP Dest IP Port range N1 to N2 exclude N, N1 < N < N2

  Example: source 172.16.10.100 dest 171.16.23.1 dest port range 10 to 30 exclude port 20

- b. Rule: dest IP address Range A.B.C.0/24 to A.B.C.D/24 exclude A.B.C.P, 0 < P < D

  Example: source 172.16.10.0/24 to 172.16.10.20/24 exclude 172.16.10.2 source port 49200 dest port 50000

- c. Rule: list IP A.B.C.D, E.F.G.H, A1.B1.C1.D1 port list p1, p2, p3

  Example: list ip 172.16.10.1, 172.16.10.3, 172.16.10.4 port list 22, 23, 24, 25

Multiple Controllers

- High degrees of Scale and reliability demand a cluster of controllers acting as one
- Need to balance the Forwarding Element (FE) connection load between the multiple controllers
- Need to re-balance connections in the case of controller failures
- In some cases need to communicate the notion of Master controller to the FE

Multiple Controllers

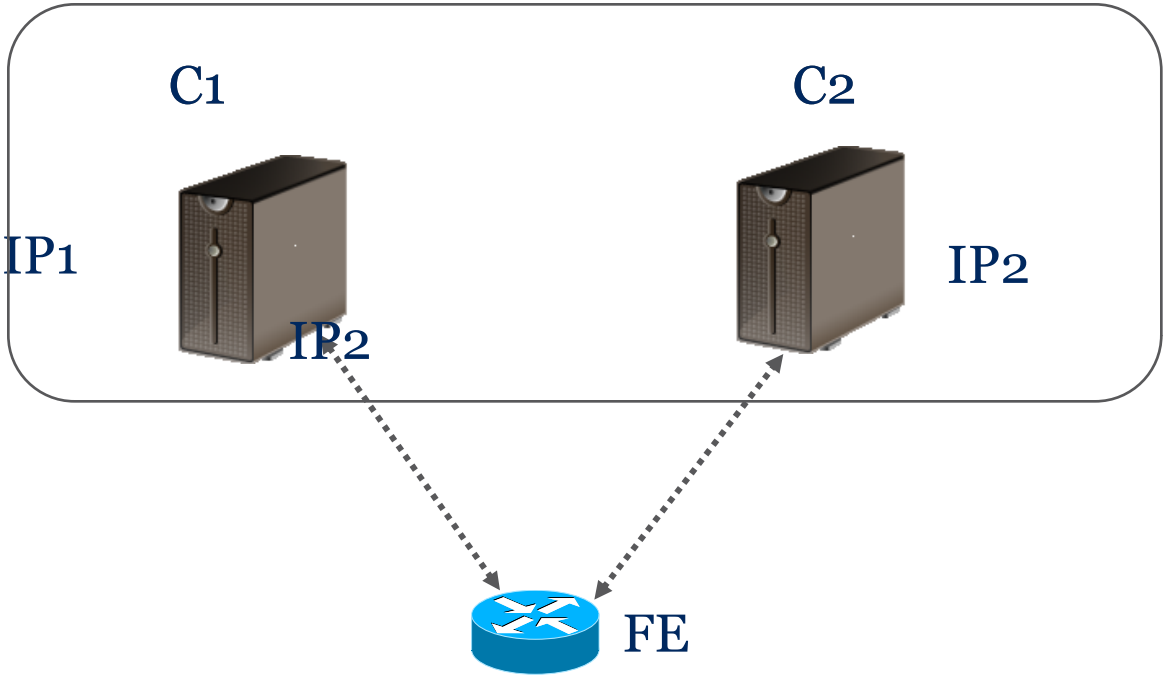
- Existing OpenFlow method to Reject Connection
  - No response to the ECHO_REQUEST message
  - This increases the time to converge on required information
- The Forwarding Element has to implement logic to connect with the appropriate controller upon failures
- Control Plane Load Balancing cannot be implemented

Multiple Controllers

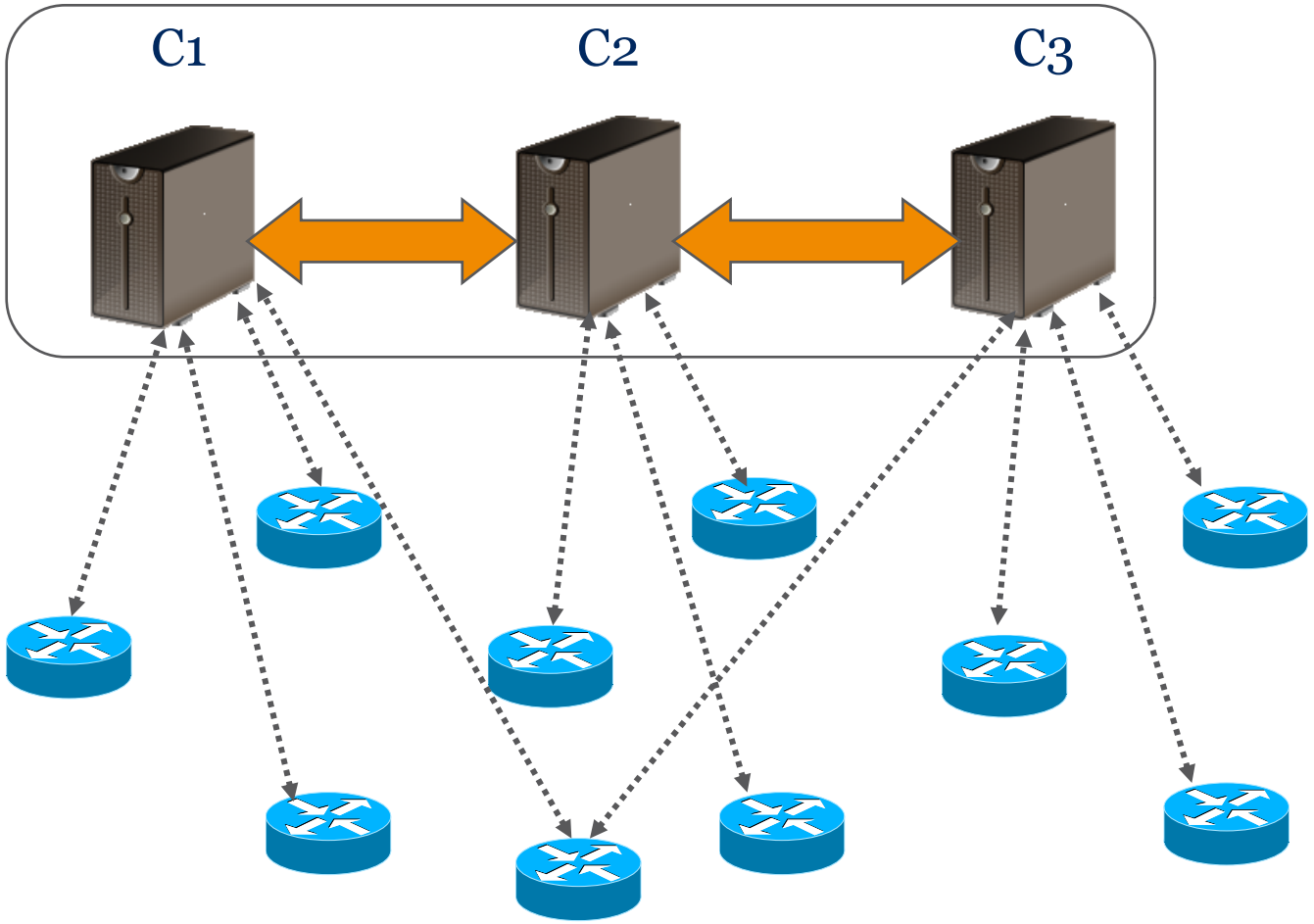
- One solution is to explicitly configure each Forwarding Element with the required information and this quickly devolves into a configuration nightmare

- Our solution is to extend the OpenFlow with a simple reject message

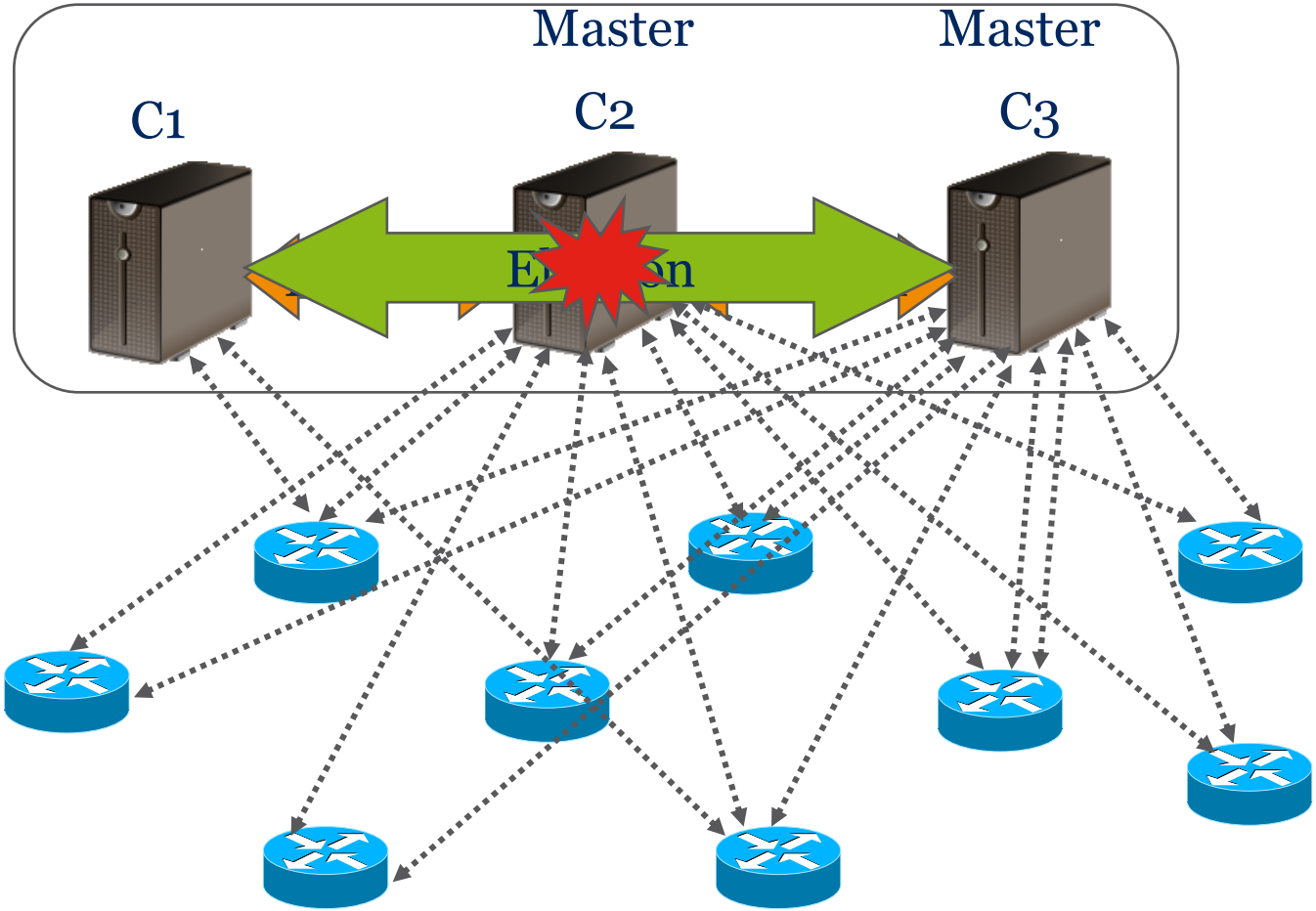
Multiple Controllers – Redirect/Reject Connection



Multiple Peer Controllers – Load Balancing



Multiple Controllers – Master Failure



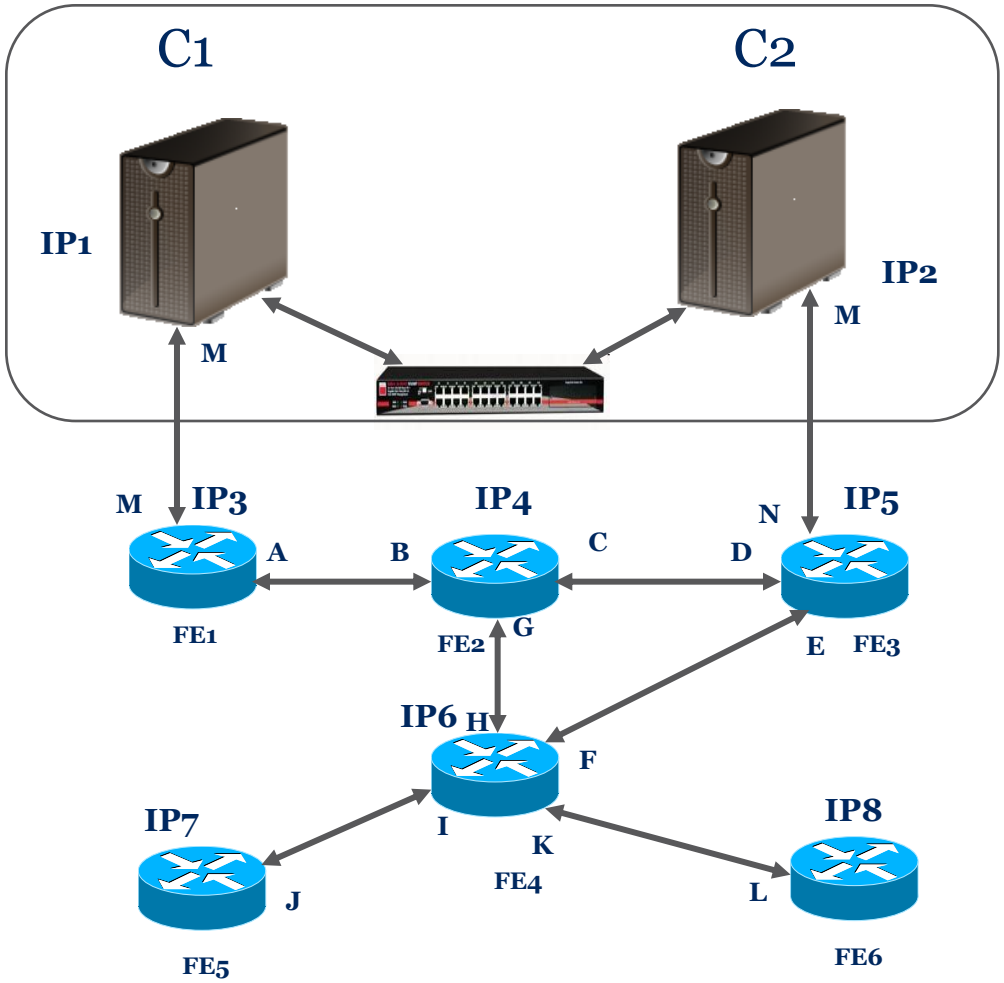
Establishing Controller Connectivity

- OpenFlow specification is unclear about the connection initiator.
- Either Controller or the FE can initiate the OpenFlow connection
- An IP address configuration is required on the Controller and the FE
- The FE needs to be configured with the IP address of the controller
- The Controller and FE are assumed to be in the same IP subnet or broadcast domain.
- The Controller and FE are connected to each other via their respective management ports to the same switch

Establishing Controller Connectivity

- Solution:
- Uses Open Flow forwarding
- Is Distributed and scalable
- Uses LLDP to exchange information
- Runs an algorithm to build a database of controllers and the ports through which they can be reached
- Uses the database to find an optimal path to the controller
- Populates forwarding rules based on this path
- This design accommodates out-of-band and in-band deployments

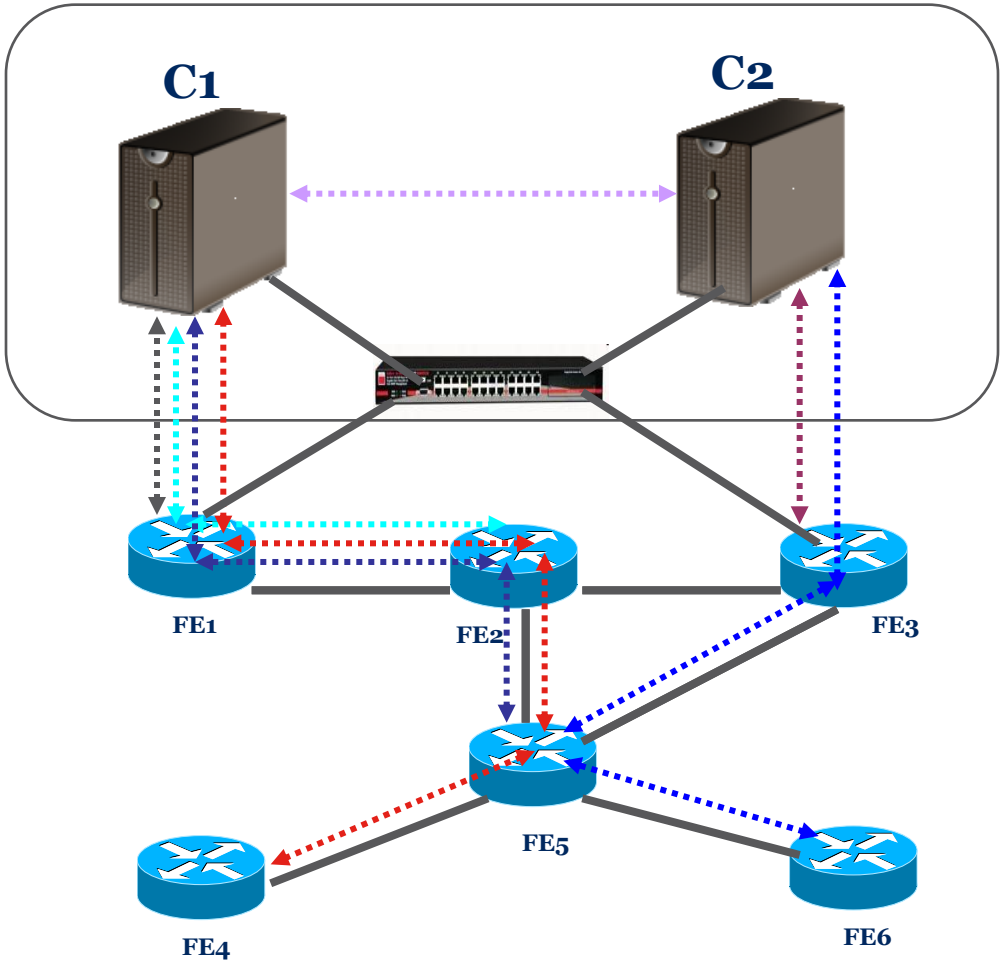
Establishing Controller Connectivity



Controller and FE exchanging LLDP messages

Establishing Controller Connectivity

In-Band Example of Logical Connections



Configured Controller

FE1 -> IP1

FE2 -> IP1

FE3 -> IP2

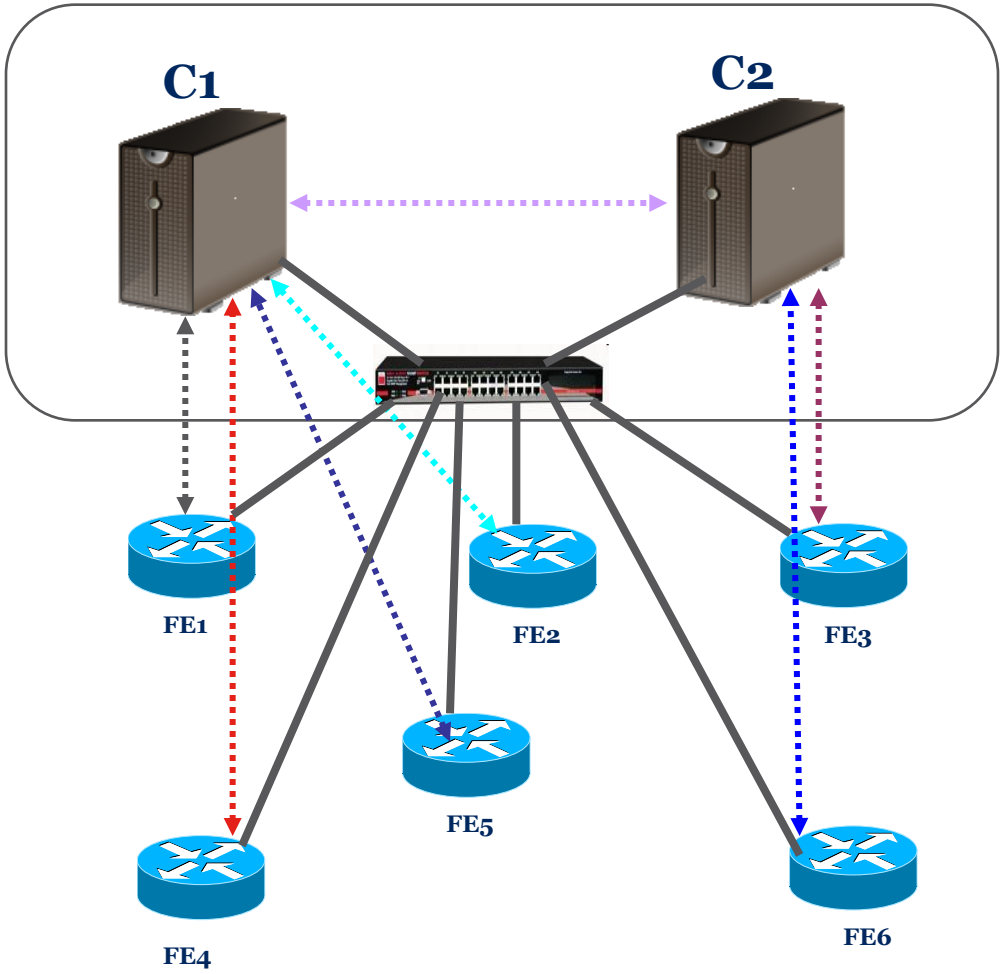
FE4 -> IP1

FE5 -> IP1

FE6 -> IP2

Establishing Controller Connectivity

Out-Of-Band Example of Logical Connections



Configured Controller

FE1 -> IP1

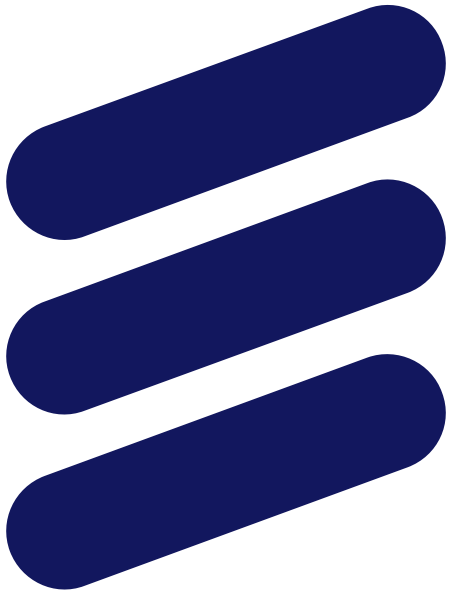
FE2 -> IP1

FE3 -> IP2

FE4 -> IP1

FE5 -> IP1

FE6 -> IP2



ERICSSON