Open Software-Defined Wireless Mesh Networks for rural communications

Periplus: An SDN OpenFlow in-band Control Plane

IEEE SA RRSA, Indian Institute of Science, Bangalore, March 2024

Eva M. Castro-Barbero, **Pedro de-las-Heras-Quirós,** Javier Simó-Reigadas, Ignacio-Prieto-Egido
eva.castro@urjc.es, pedro.delasheras@urjc.es javier.simo@urjc.es, ignacio.prieto@urjc.es

Escuela de Ingeniería de Fuenlabrada
Universidad Rey Juan Carlos
Fuenlabrada, Spain

Target Scenario

EHAS

Rural areas are frequently characterized by:

- a low population density
- lower resources
- higher costs of transport and access

Main challenges for telcos are:

- high deployment costs where wired technologies are prohibitively expensive
- high maintenance costs
- lack of maintenance staff
- low income

Proposal: Sharing infrastructure through wireless technology that provides

- advanced QoS support
- robustness and resilience
- basic self-configuration
- easy centralized management

Existing technologies

TUCAN3G (a project deployed in Peru (2013-2016) by the team

- strong in QoS support
- demonstrated to be a valid alternative for operators both technically and economically
- too complex and rigid

Community networks based on distributed mesh networks

- flexible and resilient
- but lack QoS support

Periplus: in-band control plane

Objectives

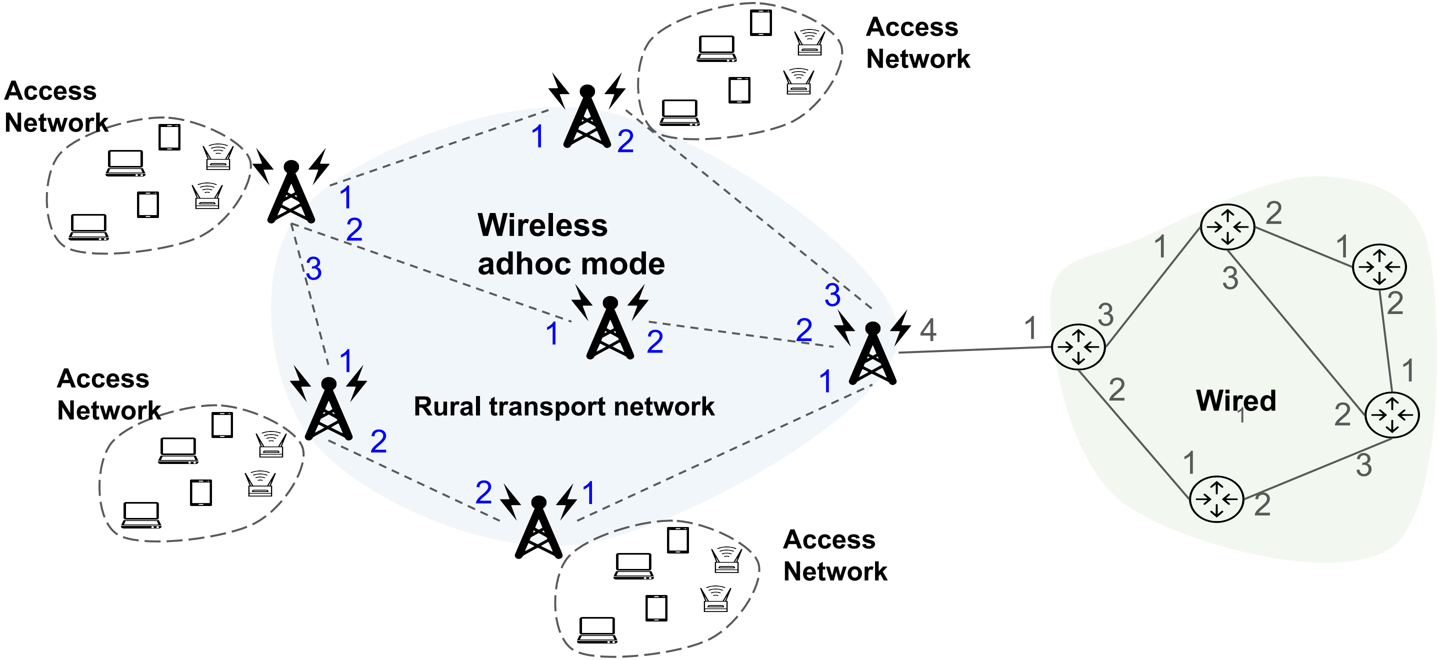
- improved connectivity in rural areas
- support of wireless + wired infrastructure with the same control plane
- enabling advanced QoS support in the data plane
- robustness and resilience
- basic self-configuration
- easy centralized management

Periplus: in-band control plane

- Pure SDN OpenFlow in-band control plane
- Support for multiple SDN controllers
- Forwarding of packets based on Slick Packets: a subgraph is encapsulated between L2 and L3 headers with main and alternate paths
    - Robustness and responsiveness: switches react quickly to link and switch failures without requiring communication with the controller
    - Scalability: amount of OF flows stored in switches is reduced
- Multicontroller support
    - subgraphs scale O(switches of 1 controller)
- Mininet prototype, Ryu, entirely based on OpenFlow, standard Open vSwitch code

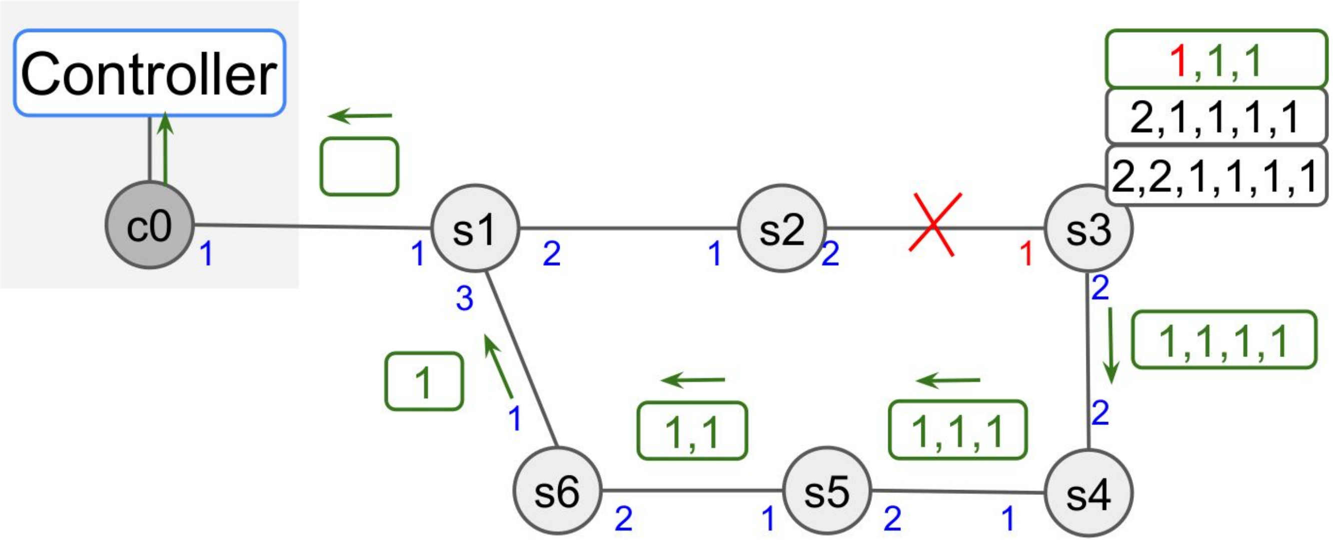
Periplus: in-band control plane

- OpenFlow + Open vSwitch both in Wired & Wireless nodes
- 802.11 adhoc mode
  - Rewriting of src & dst MAC addresses on each hop
  - Each time an 802.11 frame from a new wireless node is rcvd through the same wireless interface, a **new virtual port** is assigned to it

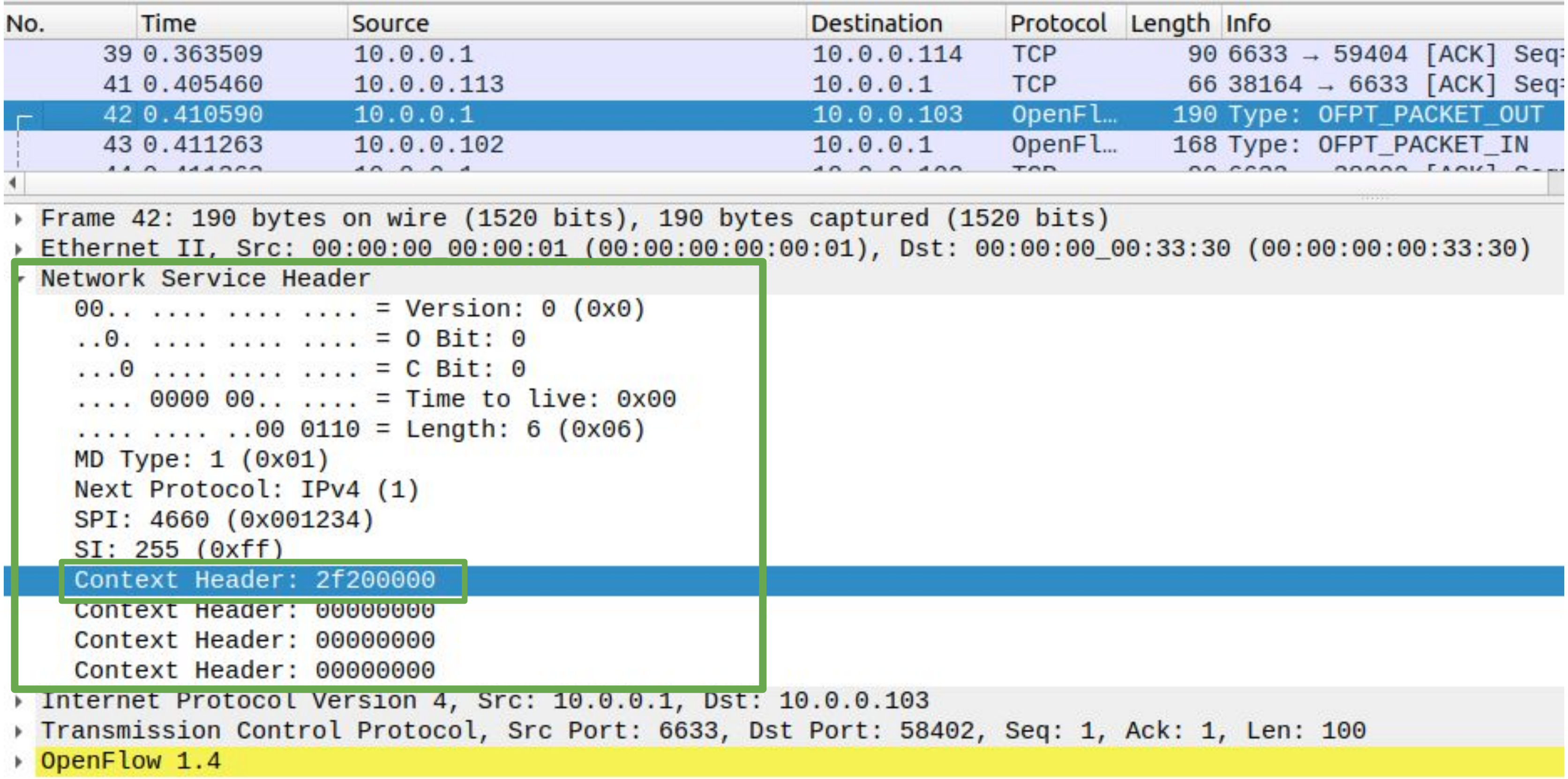


Graph forwarding (Slick Packets)

from s3 to controller (port 1 of s3 is inactive)



Path coding: NSH Header

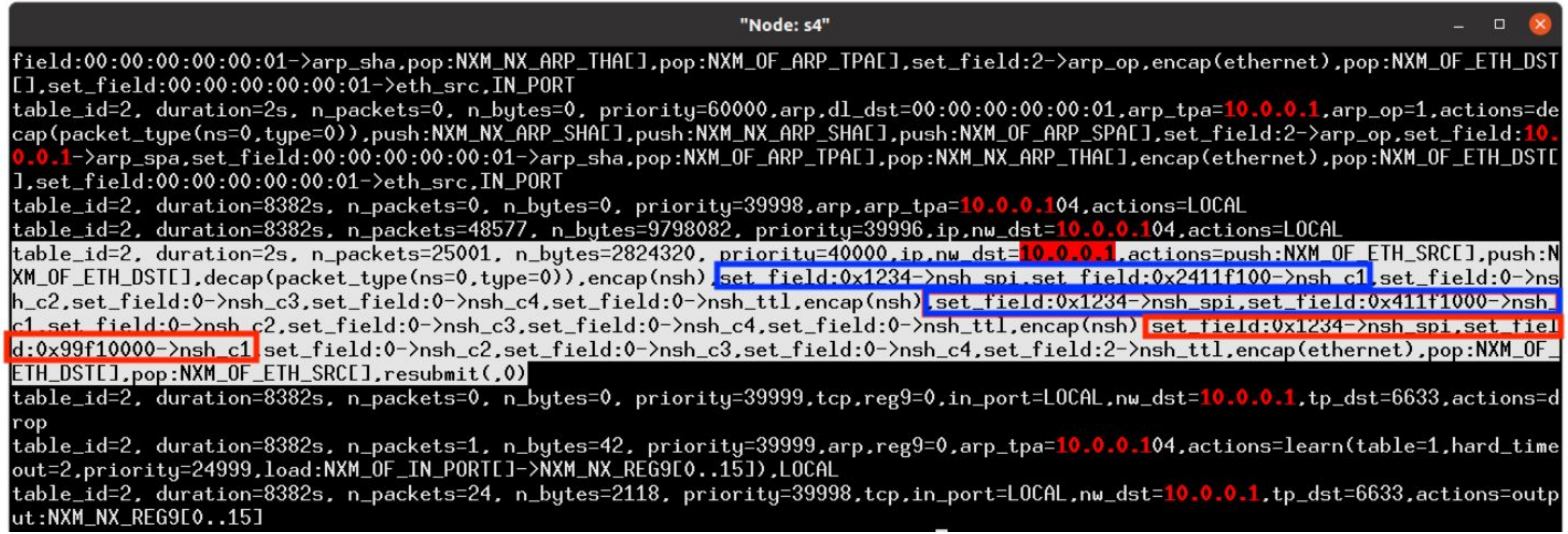


```
No.   Time      Source       Destination  Protocol  Length Info
   39 0.363509  10.0.0.1     10.0.0.114   TCP           90 6633 → 59404 [ACK] Seq=
   41 0.405460  10.0.0.113   10.0.0.1     TCP           66 38164 → 6633 [ACK] Seq=
   42 0.410590  10.0.0.1     10.0.0.103   OpenFl…      190 Type: OFPT_PACKET_OUT
   43 0.411263  10.0.0.102   10.0.0.1     OpenFl…      168 Type: OFPT_PACKET_IN
```

```
▸ Frame 42: 190 bytes on wire (1520 bits), 190 bytes captured (1520 bits)
▸ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:33:30 (00:00:00:00:33:30)
▾ Network Service Header
    00.. .... .... .... = Version: 0 (0x0)
    ..0. .... .... .... = O Bit: 0
    ...0 .... .... .... = C Bit: 0
    .... 0000 00.. .... = Time to live: 0x00
    .... .... ..00 0110 = Length: 6 (0x06)
    MD Type: 1 (0x01)
    Next Protocol: IPv4 (1)
    SPI: 4660 (0x001234)
    SI: 255 (0xff)
    Context Header: 2f200000
    Context Header: 00000000
    Context Header: 00000000
    Context Header: 00000000
▸ Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.103
▸ Transmission Control Protocol, Src Port: 6633, Dst Port: 58402, Seq: 1, Ack: 1, Len: 100
▸ OpenFlow 1.4
```

OpenFlow implementation in unmodified Open vSwitch

OF flows in a switch:



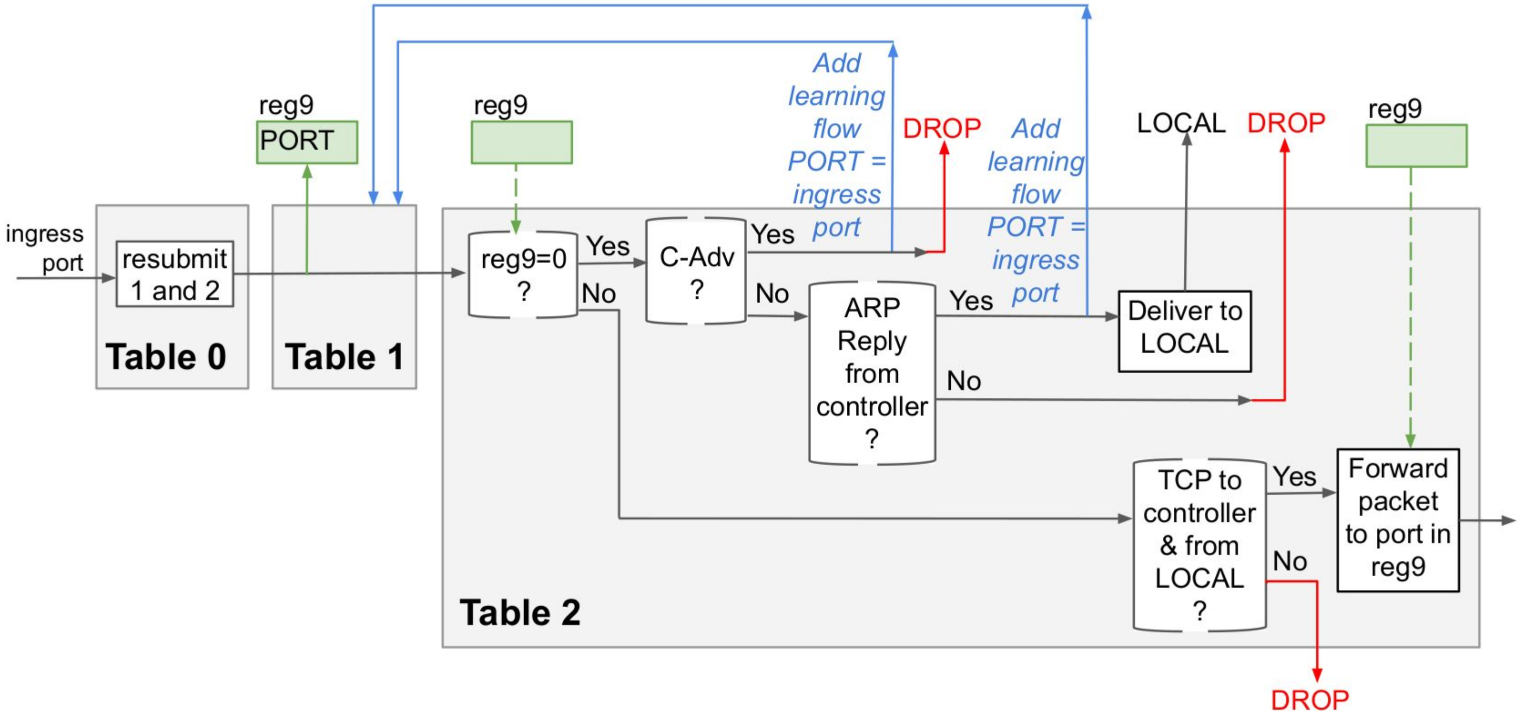
```
field:00:00:00:00:00:01->arp_sha,pop:NXM_NX_ARP_THA[],pop:NXM_OF_ARP_TPA[],set_field:2->arp_op,encap(ethernet),pop:NXM_OF_ETH_DST
[],set_field:00:00:00:00:00:01->eth_src,IN_PORT
table_id=2, duration=2s, n_packets=0, n_bytes=0, priority=60000,arp,dl_dst=00:00:00:00:00:01,arp_tpa=10.0.0.1,arp_op=1,actions=de
cap(packet_type(ns=0,type=0)),push:NXM_NX_ARP_SHA[],push:NXM_NX_ARP_SHA[],push:NXM_OF_ARP_SPA[],set_field:2->arp_op,set_field:10.
0.0.1->arp_spa,set_field:00:00:00:00:00:01->arp_sha,pop:NXM_OF_ARP_TPA[],pop:NXM_NX_ARP_THA[],encap(ethernet),pop:NXM_OF_ETH_DST[
],set_field:00:00:00:00:00:01->eth_src,IN_PORT
table_id=2, duration=8382s, n_packets=0, n_bytes=0, priority=39998,arp,arp_tpa=10.0.0.104,actions=LOCAL
table_id=2, duration=8382s, n_packets=48577, n_bytes=9798082, priority=39996,ip,nw_dst=10.0.0.104,actions=LOCAL
table_id=2, duration=2s, n_packets=25001, n_bytes=2824320, priority=40000,ip,nw_dst=10.0.0.1,actions=push:NXM_OF_ETH_SRC[],push:N
XM_OF_ETH_DST[],decap(packet_type(ns=0,type=0)),encap(nsh),set_field:0x1234->nsh_spi,set_field:0x2411f100->nsh_c1,set_field:0->ns
h_c2,set_field:0->nsh_c3,set_field:0->nsh_c4,set_field:0->nsh_ttl,encap(nsh),set_field:0x1234->nsh_spi,set_field:0x411f1000->nsh_
c1,set_field:0->nsh_c2,set_field:0->nsh_c3,set_field:0->nsh_c4,set_field:0->nsh_ttl,encap(nsh),set_field:0x1234->nsh_spi,set_fiel
d:0x99f10000->nsh_c1,set_field:0->nsh_c2,set_field:0->nsh_c3,set_field:0->nsh_c4,set_field:2->nsh_ttl,encap(ethernet),pop:NXM_OF_
ETH_DST[],pop:NXM_OF_ETH_SRC[],resubmit(,0)
table_id=2, duration=8382s, n_packets=0, n_bytes=0, priority=39999,tcp,reg9=0,in_port=LOCAL,nw_dst=10.0.0.1,tp_dst=6633,actions=d
rop
table_id=2, duration=8382s, n_packets=1, n_bytes=42, priority=39999,arp,reg9=0,arp_tpa=10.0.0.104,actions=learn(table=1,hard_time
out=2,priority=24999,load:NXM_OF_IN_PORT[]->NXM_NX_REG9[0..15]),LOCAL
table_id=2, duration=8382s, n_packets=24, n_bytes=2118, priority=39998,tcp,in_port=LOCAL,nw_dst=10.0.0.1,tp_dst=6633,actions=outp
ut:NXM_NX_REG9[0..15]
```

Booting Switch needs to discover controller:

Sends ARP through all ports

Discovers port to controller through rcvd ARP reply or through C-Adv

Then sends SYN through port that leads to any controller



ingress port
resubmit 1 and 2
Table 0
reg9
PORT
Table 1
reg9
reg9=0 ?
Yes
No
C-Adv ?
Yes
No
Add learning flow PORT = ingress port
DROP
ARP Reply from controller ?
Yes
No
Add learning flow PORT = ingress port
Deliver to LOCAL
LOCAL
DROP
reg9
TCP to controller & from LOCAL ?
Yes
No
Forward packet to port in reg9
DROP
Table 2

Bootstrap messages (s3 switch):

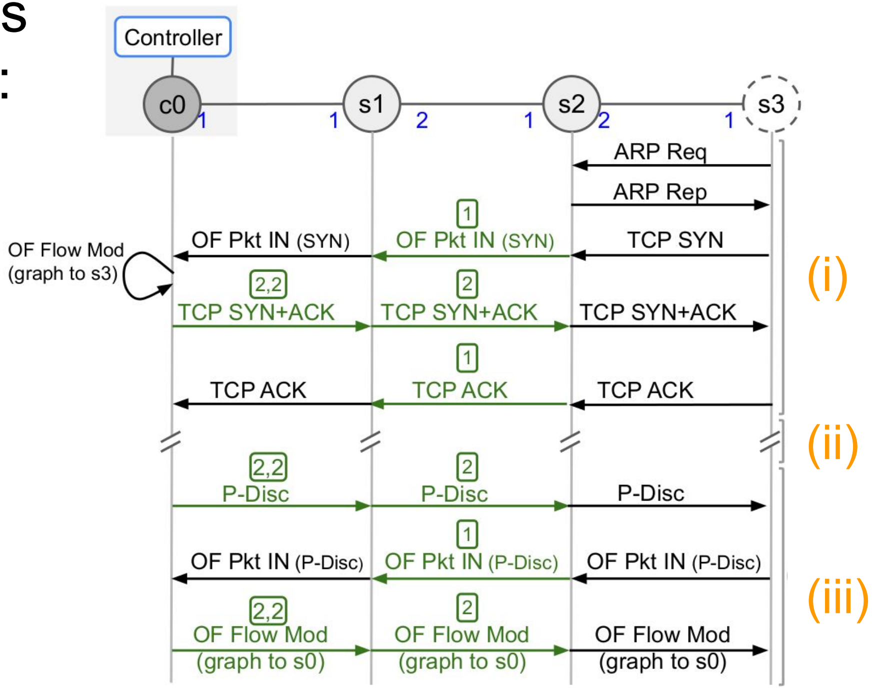
(i) ARP req. target=controller to all ports
Already MANAGED neighbor switch:

- does Proxy ARP
- then TCP 3-way handshake

(ii) OpenFlow session establishment

(iii) Port discovery:

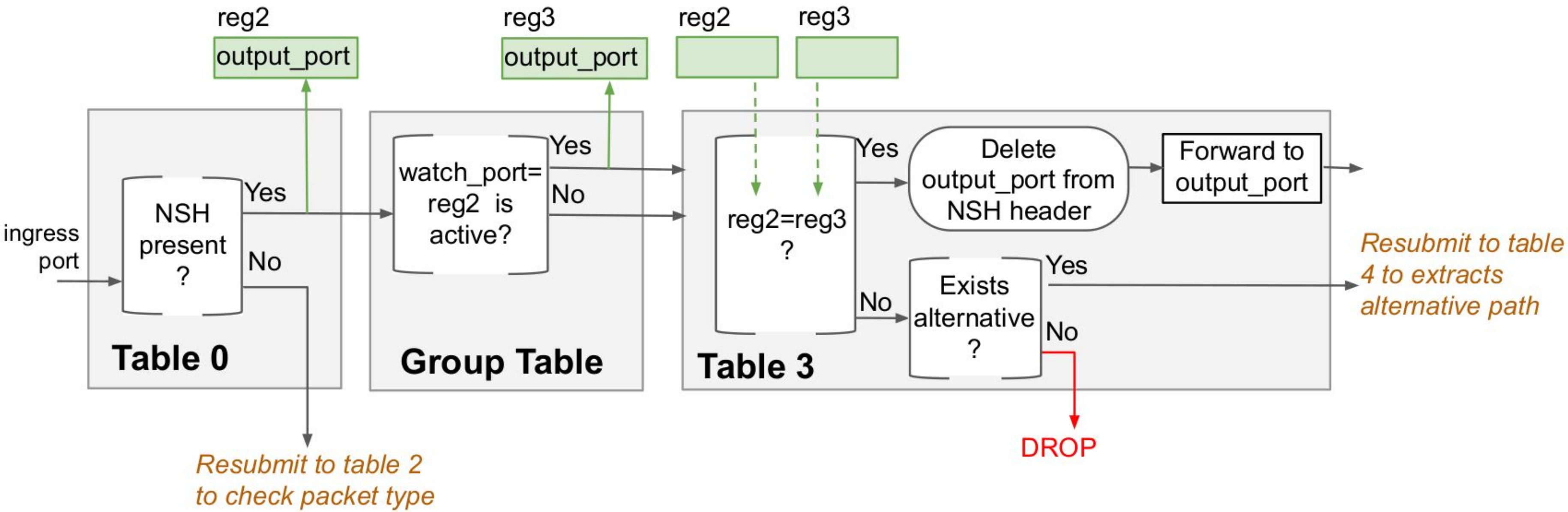
(port 1 of s3 ~ root port)



Switch s3 is MANAGED

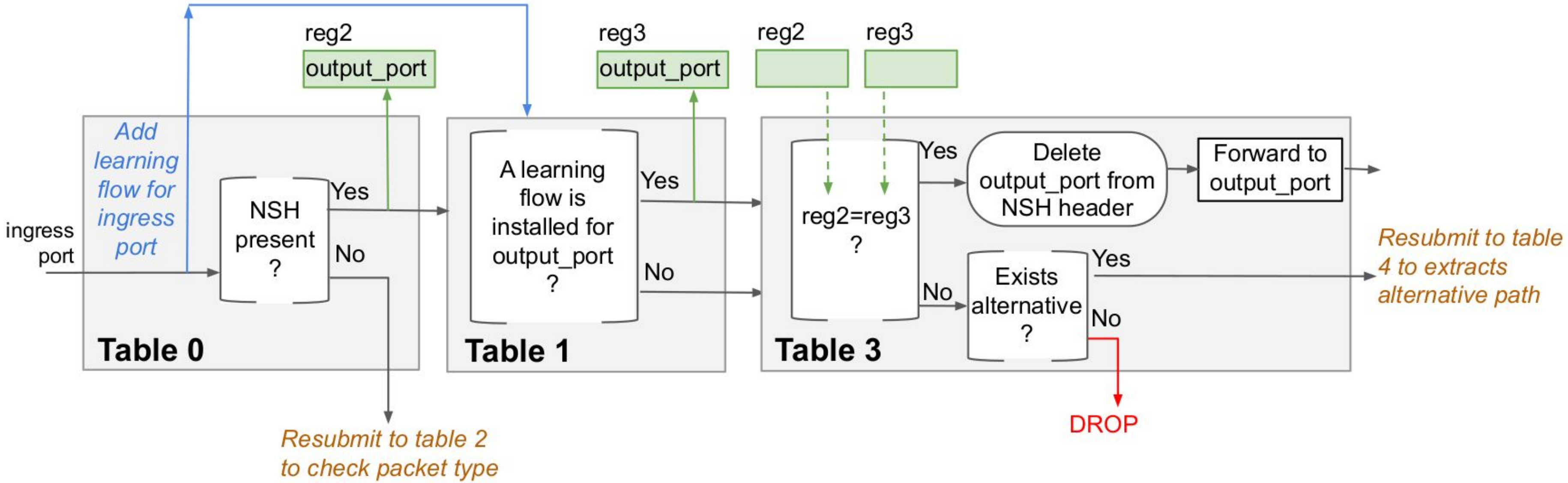
Link/Switch fault detection:

BFD port monitoring checks if output_port is active
If not: commute to alternate path in subgraph



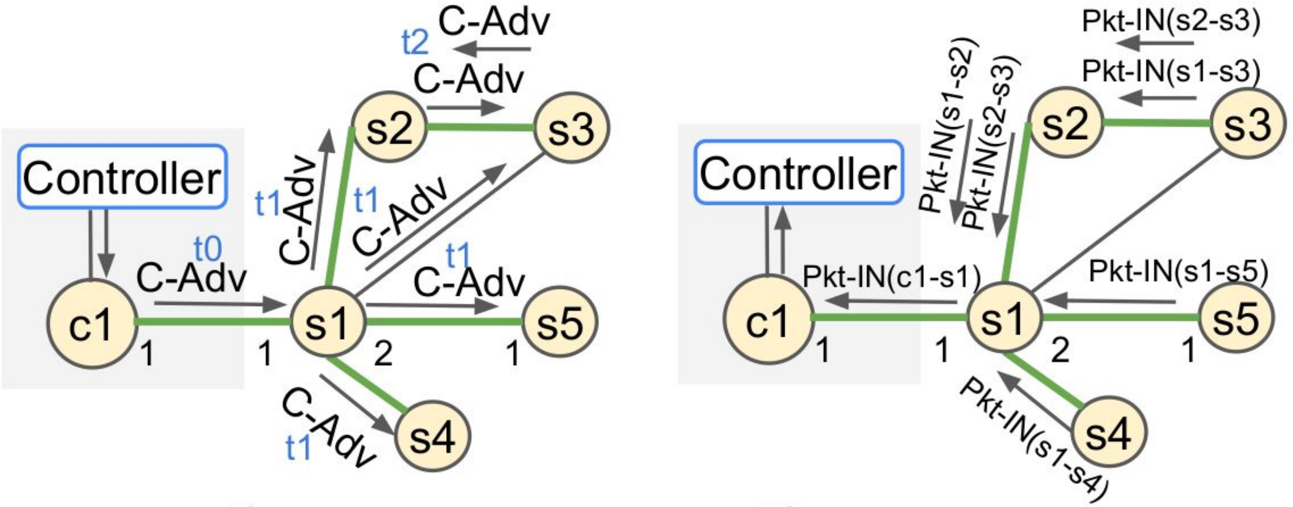
Link/Switch fault detection:

No BFD?
ITD (Input traffic detection)


reg2
output_port
reg3
output_port
reg2
reg3
Add learning flow for ingress port
ingress port
NSH present ?
Yes
No
Resubmit to table 2 to check packet type
Table 0
A learning flow is installed for output_port ?
Yes
No
Table 1
reg2=reg3 ?
Yes
No
Delete output_port from NSH header
Forward to output_port
Exists alternative ?
Yes
No
Resubmit to table 4 to extracts alternative path
DROP
Table 3

New protocol alternative to LLDP/OFDP

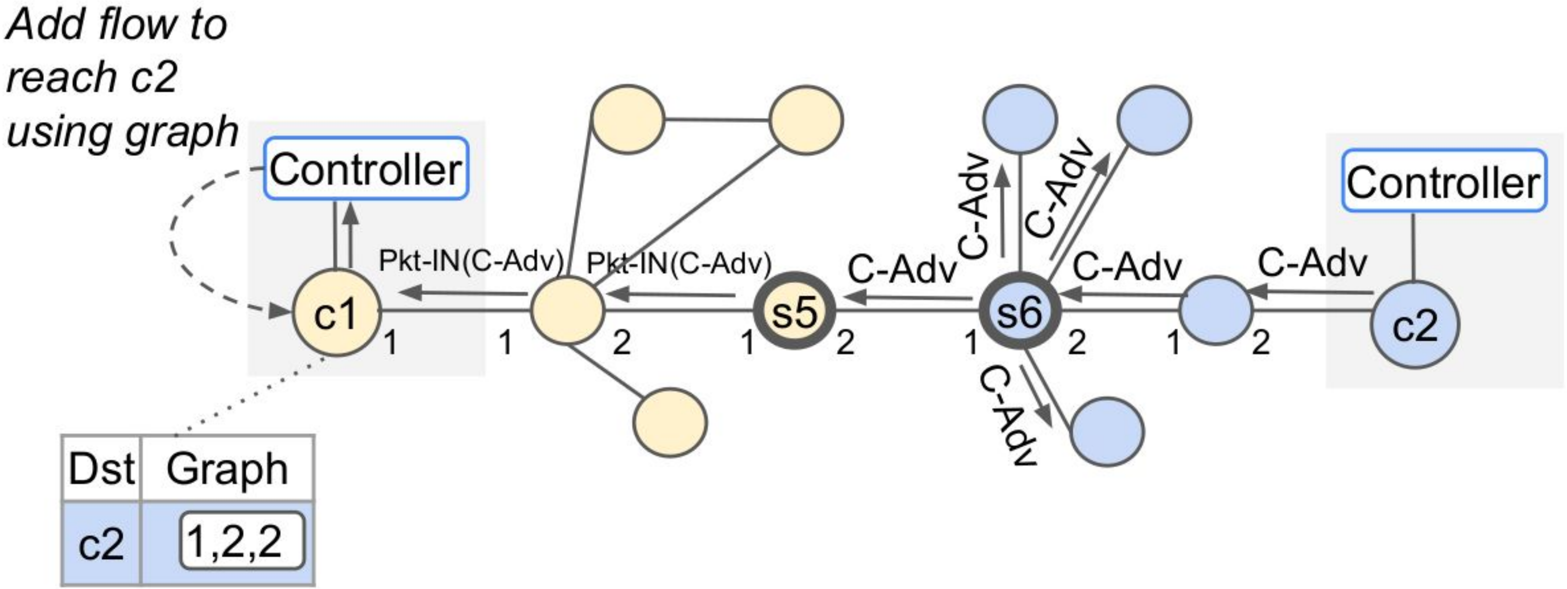
Purpose: Discovery of links not in the tree discovered in bootstrap
How: controlled flooded C-Adv msgs



i)

ii)

Multicontrollers: discovery of neighbor controllers



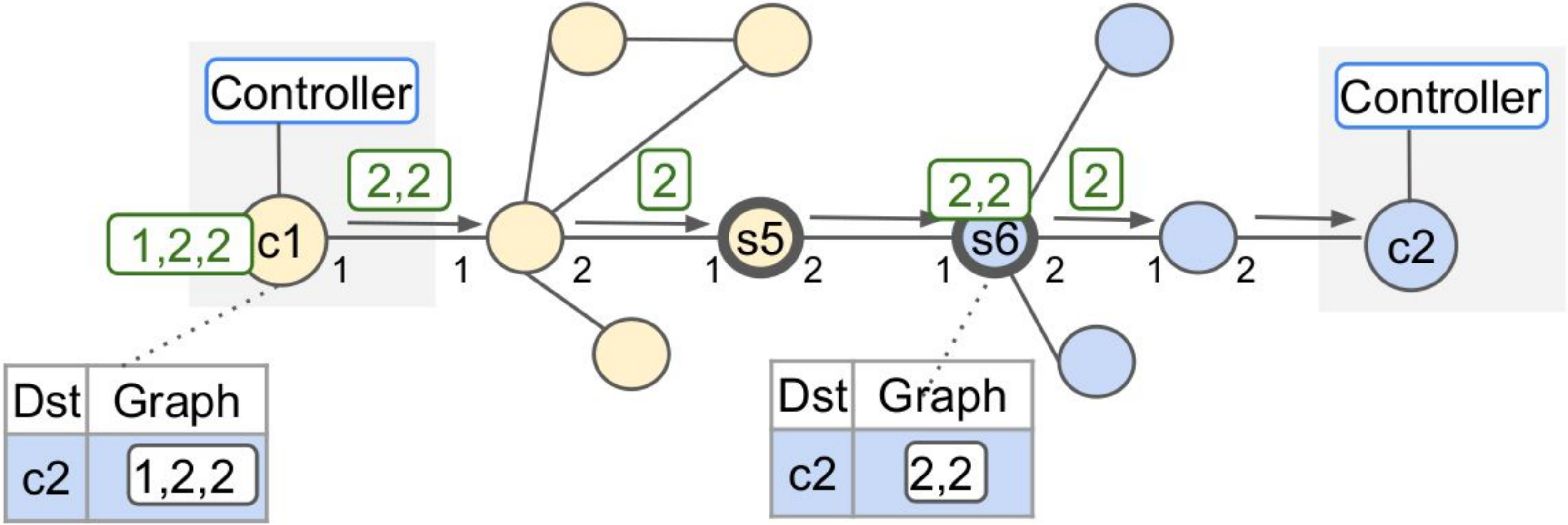
Add flow to reach c2 using graph

Dst	Graph
c2	1,2,2

Multicontrollers: Forwarding between neighbor controllers

Scaling: size of subgraph is O(diameter of net of 1 controller)

- routing subgraph inserted by c1 only until frontier switch s5
- s6 inserts new subgraph with routes towards c2

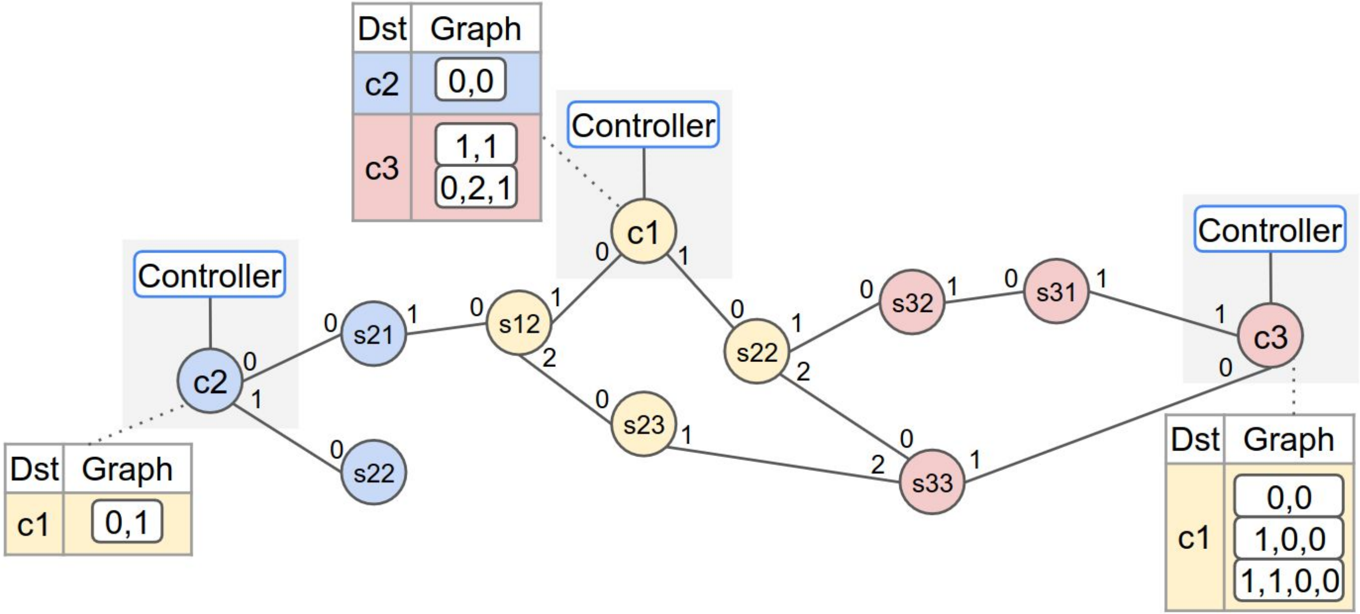


Dst	Graph
c2	1,2,2

Dst	Graph
c2	2,2

Multicontrollers: controllers that are not neighbors

- Path vector protocol (not shown) run between c1, c2, c3
- Enables c1 & c2 to discover that they can reach each other through c1 network



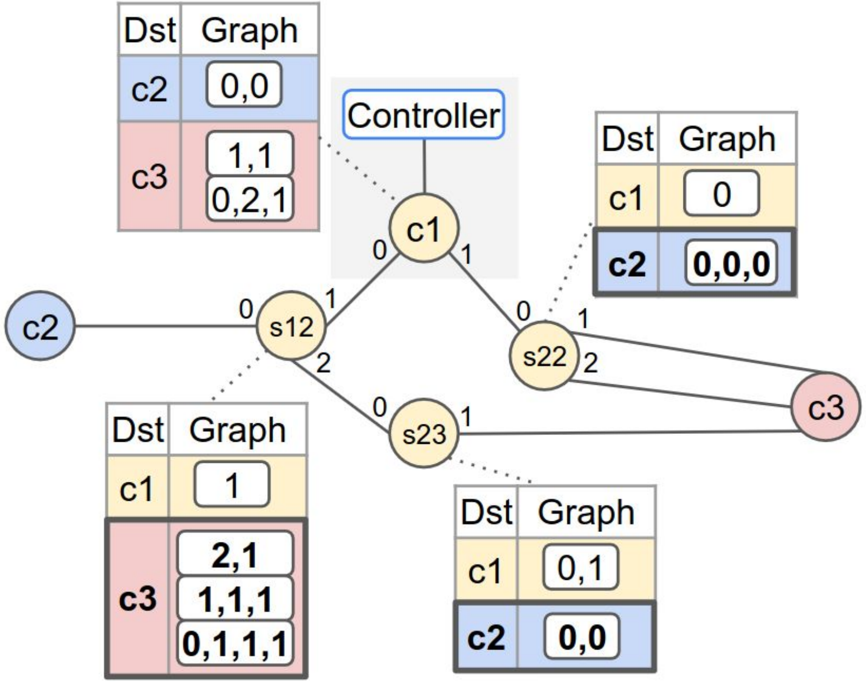
Dst	Graph
c2	0,0
c3	1,1 0,2,1

Dst	Graph
c1	0,1

Dst	Graph
c1	0,0 1,0,0 1,1,0,0

Multicontrollers: controllers that are not neighbors

- c1 is neighbor of c2 and c3
- so it can install routing subgraphs in frontier switches s12, s23, s22
- this enables routing between c2 ⇔ c3

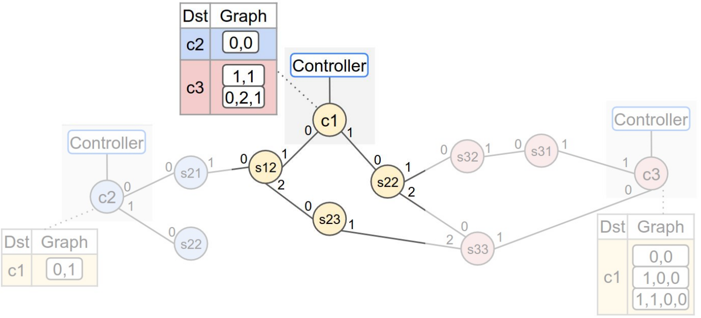


Dst	Graph
c2	0,0
c3	1,1 0,2,1

Dst	Graph
c1	0
c2	**0,0,0**

Dst	Graph
c1	1
c3	**2,1** **1,1,1** **0,1,1,1**

Dst	Graph
c1	0,1
c2	**0,0**



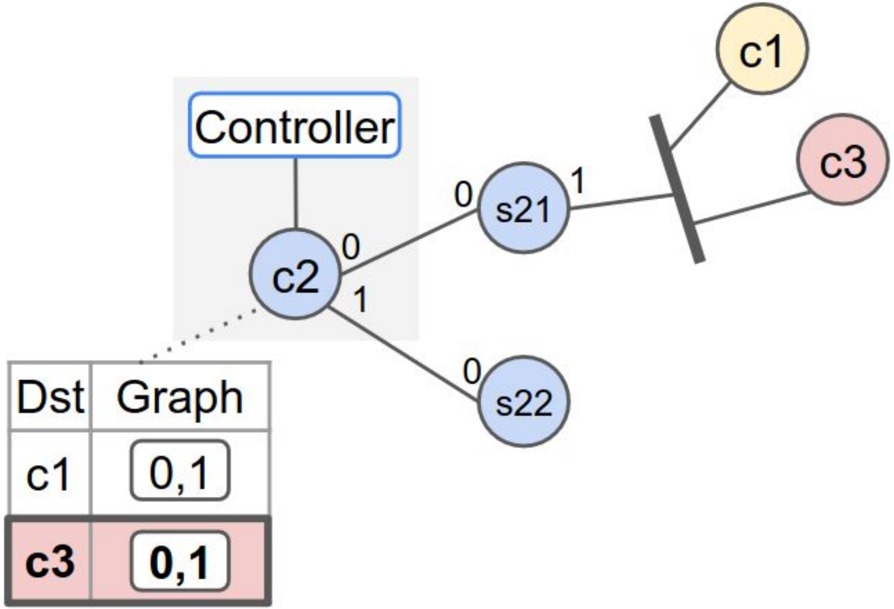
Dst	Graph
c2	0,0
c3	1,1 0,2,1

Dst	Graph
c1	0,1

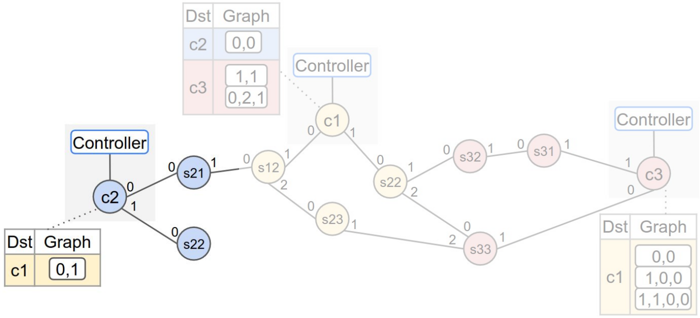
Dst	Graph
c1	0,0 1,0,0 1,1,0,0

Multicontrollers: controllers that are not neighbors

Routing subgraph from c2 to reach c1 and c3



Dst	Graph
c1	0,1
c3	**0,1**



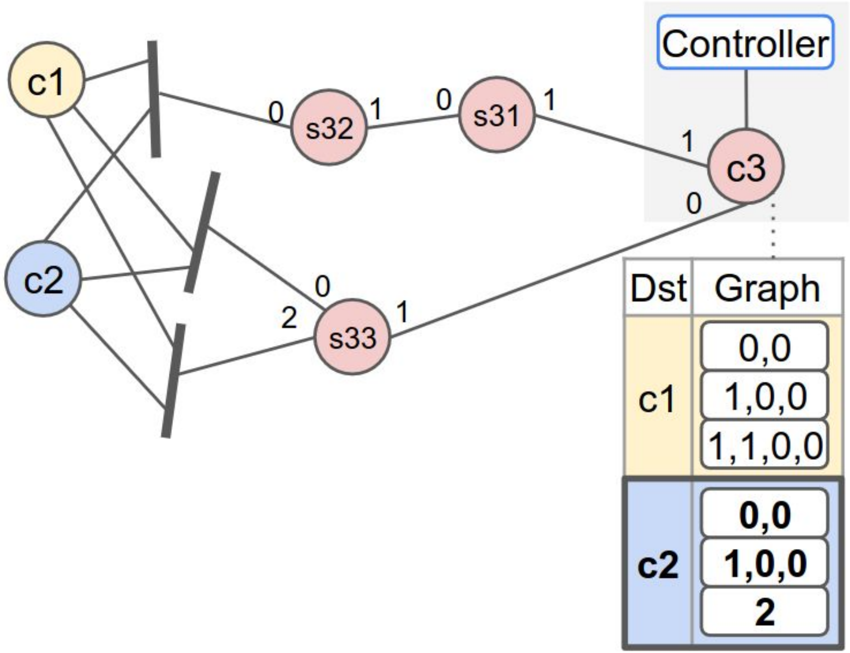
Dst	Graph
c2	0,0
c3	1,1 0,2,1

Dst	Graph
c1	0,1

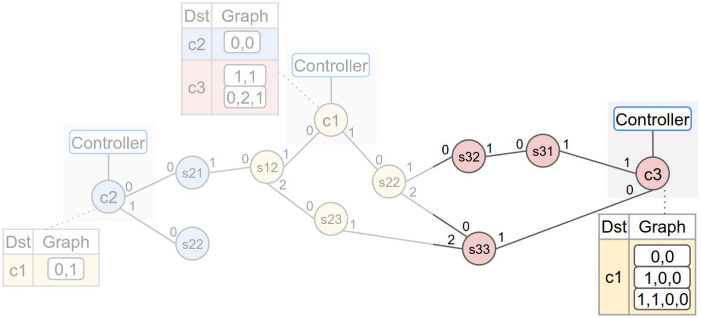
Dst	Graph
c1	0,0 1,0,0 1,1,0,0

Multicontrollers: controllers that are not neighbors

Routing subgraph from c3 to reach c1 and c2



Dst	Graph
c1	0,0 1,0,0 1,1,0,0
c2	**0,0** **1,0,0** **2**



Dst	Graph
c2	0,0
c3	1,1 0,2,1

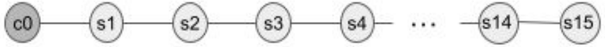
Dst	Graph
c1	0,1

Dst	Graph
c1	0,0 1,0,0 1,1,0,0

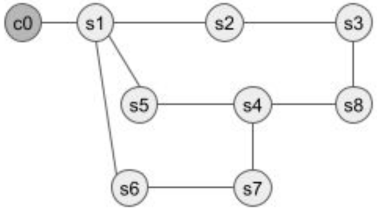
Testbed

- RyU implementation of controller
- Open vSwitch
- Mininet WiFi
- TLA+ specs of protocols

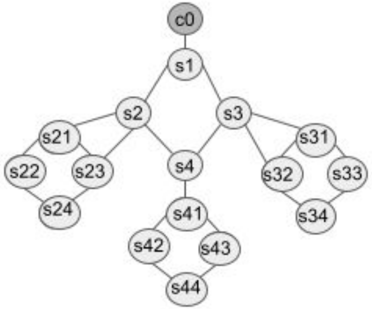
Testing in topologies using one controller



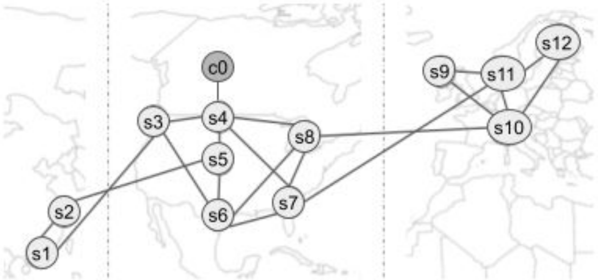
(i) Linear



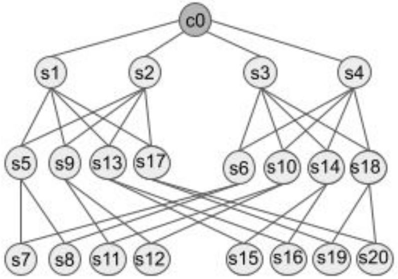
(ii) Simple



(iii) Mesh

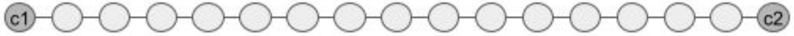


(iv) B4

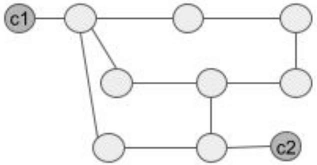


(v) Clos

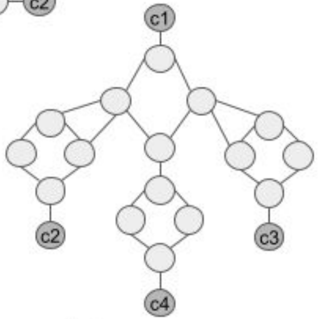
Testing in topologies using multiple controllers



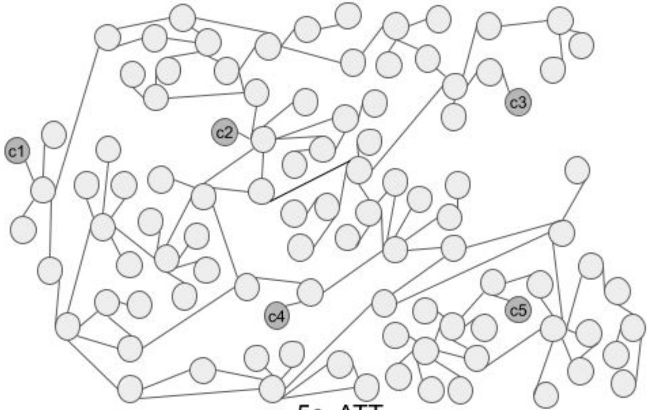
(i) 2c_Linear



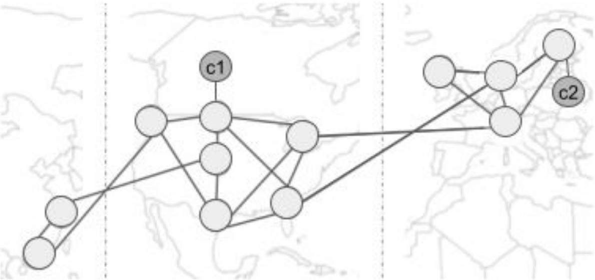
(ii) 2c_Simple



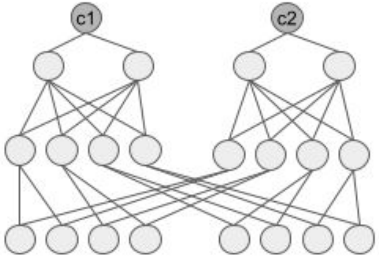
(iii) 4c_Mesh



5c_ATT

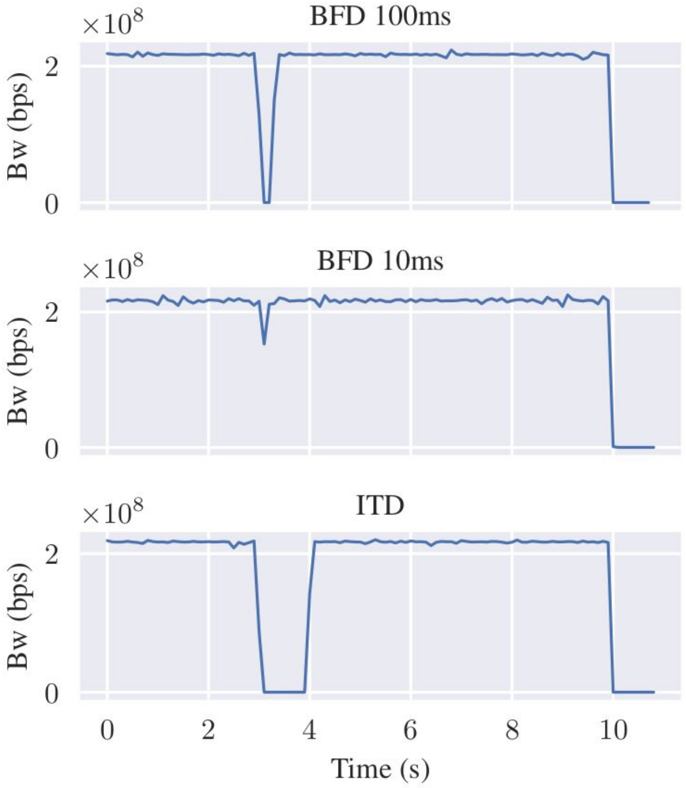


(iv) 2c_B4

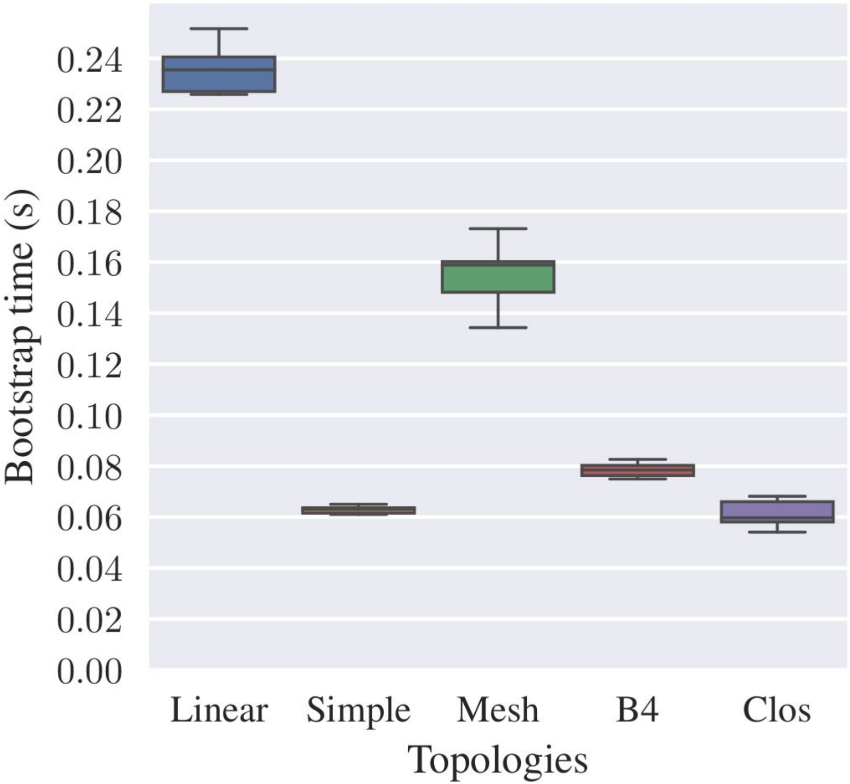


(v) 2c_Clos

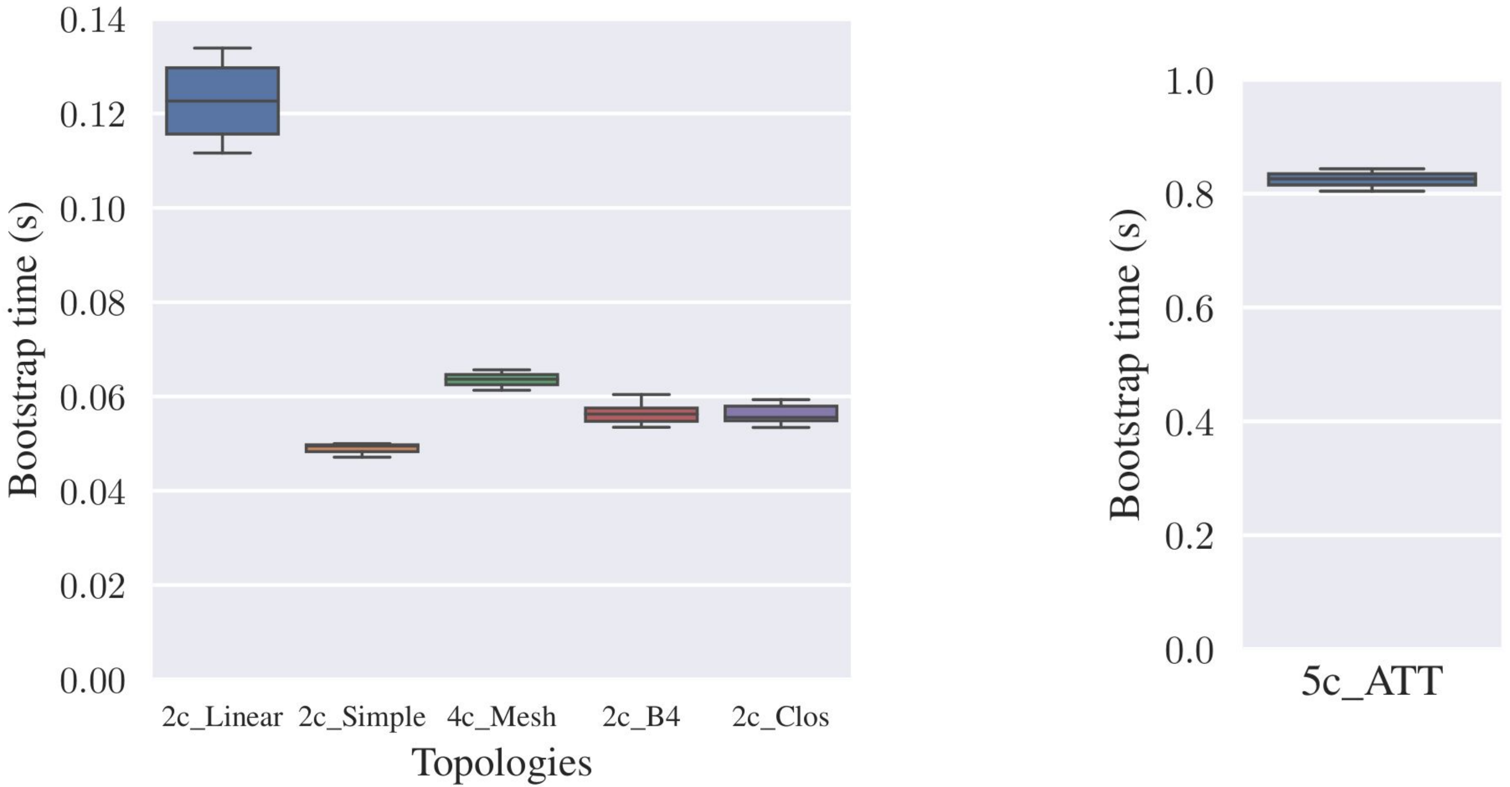
Throughput fall due to node/link failure.



Time until all switches MANAGED in topologies with one controller

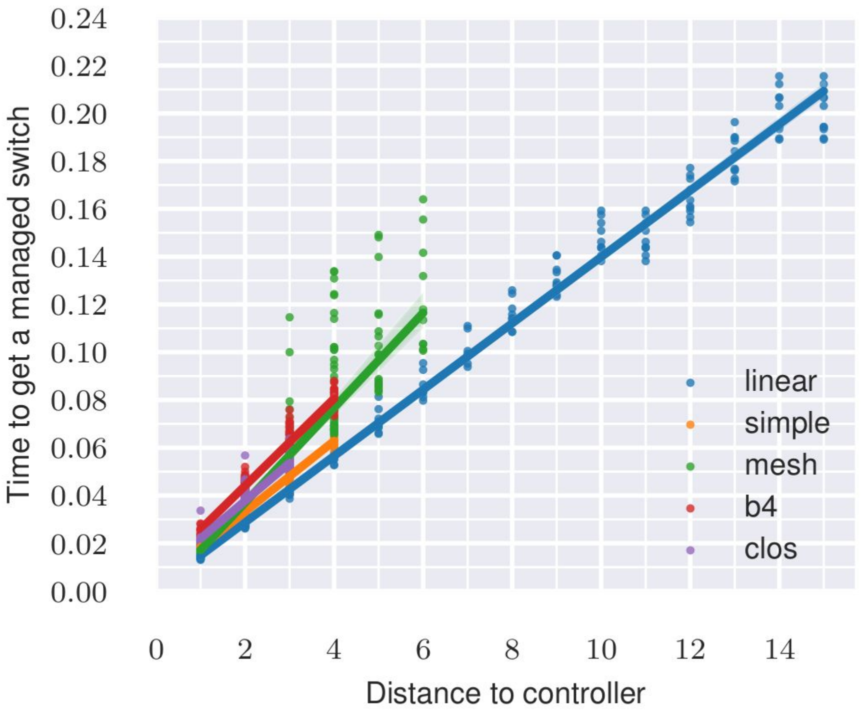


Time until all switches MANAGED in topologies with multiple controllers



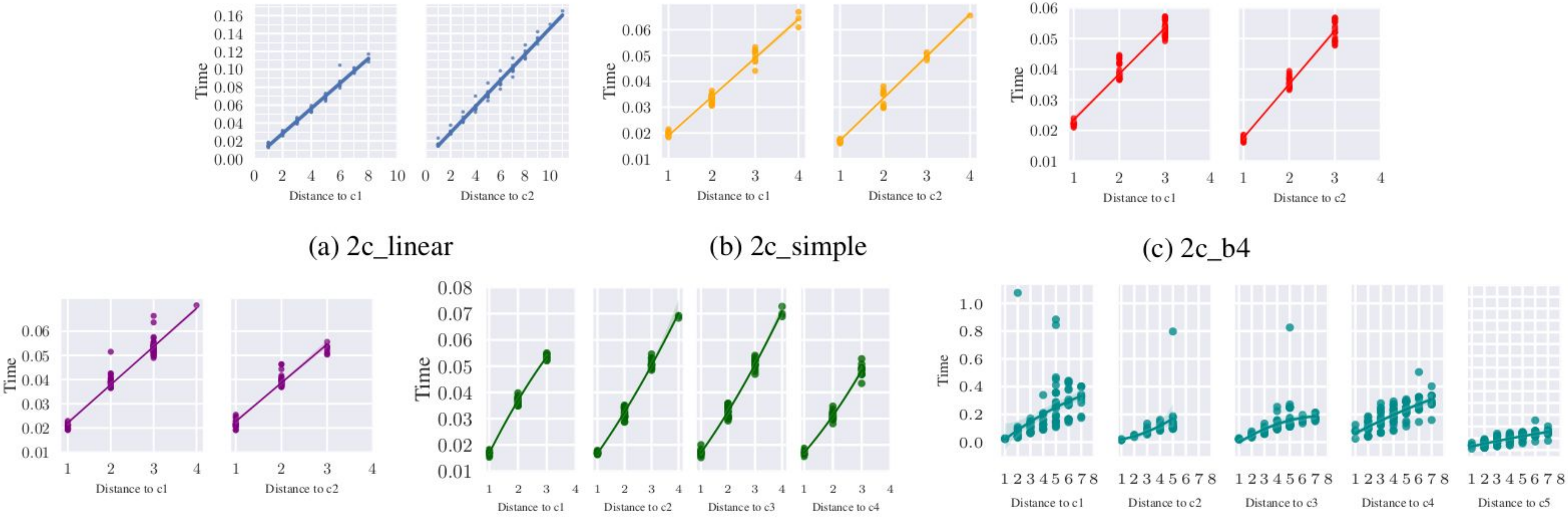
Time until a switch is managed vs distance to controller

topologies with 1 controller



Time until each switch is MANAGED vs distance to controller

topologies with multiple controllers



(a) 2c_linear

(b) 2c_simple

(c) 2c_b4

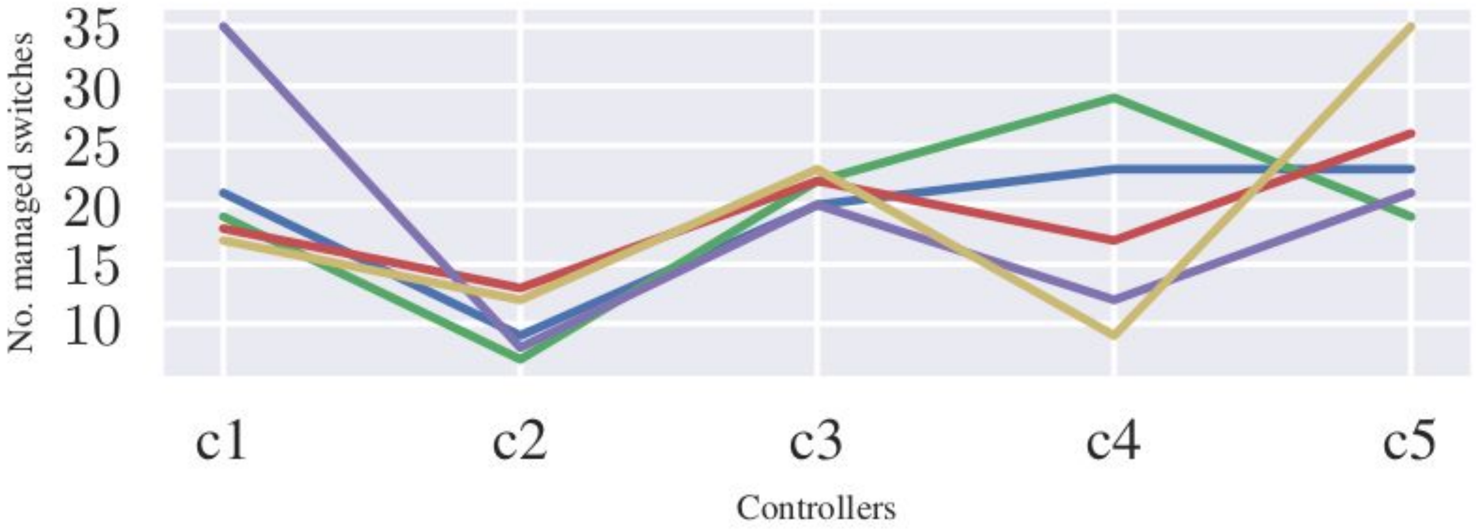
(d) 2c_clos

(e) 4c_mesh

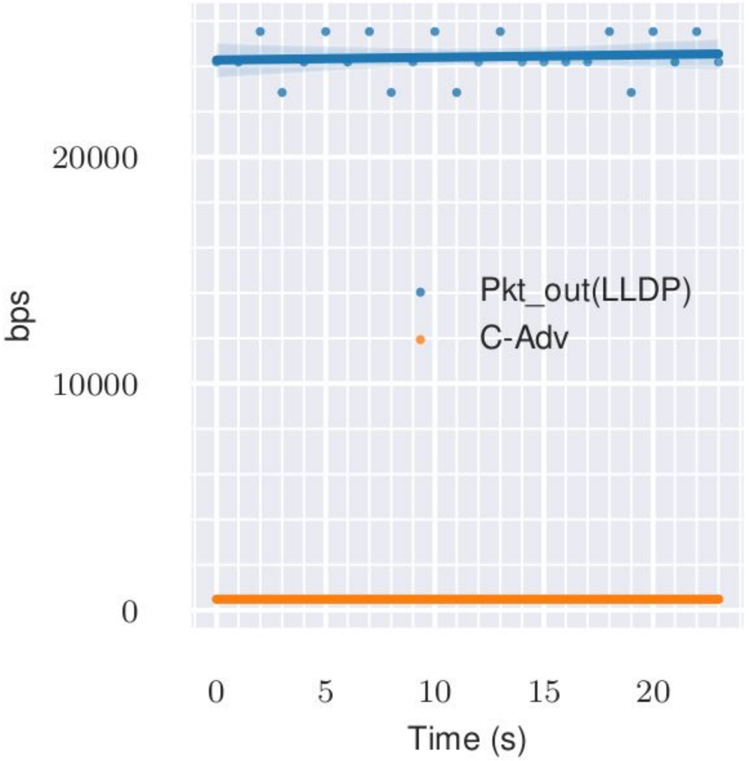
(f) 5c_att

Number of switches per controller in 5c_ATT topology

each line is a different execution



C-Adv messages vs LLDP


bps
● Pkt_out(LLDP)
● C-Adv
Time (s)