6TiSCH Centralized Scheduling and Multipath Construction

Hao Jiang

September 21st, 2016

Industrial Mentor: Pascal Thubert

Academic Coordinator: Géraldine Texier

Agenda

- The art: 6TiSCH multipath forwarding with BIER-TE (5 min)
- Internship: 6TiSCH multipath control with BIER-TE (8 min) + Demo (3 min)
- Testbed Implementation (2 min)
- Multipath Control Loop Experiments (8 min) + Demo (3 min)
- Conclusion (1 min)
- Q&A

Mission: optimize delivery within time and energy constraints

Bring deterministic properties to Low-power and Lossy Networks (LLNs) based on 6TiSCH (IPv6 over the TSCH mode of IEEE 802.15.4e):

- **Scheduling** over Time Slotted Channel Hopping (TSCH)
- **Packet replication and elimination** over redundant routes with BIER-TE

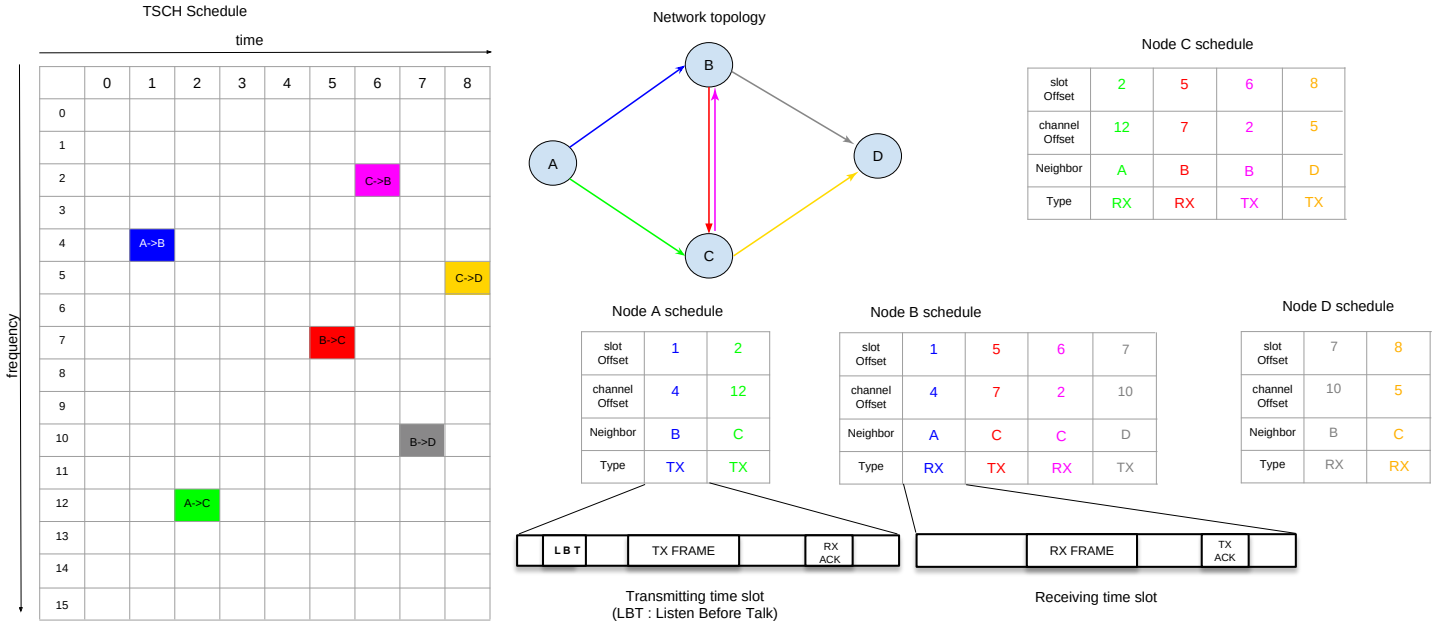


Figure 1: TSCH Schedule

(a)

(COMI) CoAP / DTLS	(PANA)	6LoWPAN ND	RPL	
UDP	ICMP			
IPv6				
6LoWPAN adaptation and compression (HC)				
6top	BIER-TE			
IEEE802.15.4e TSCH				
Radio				

(b)

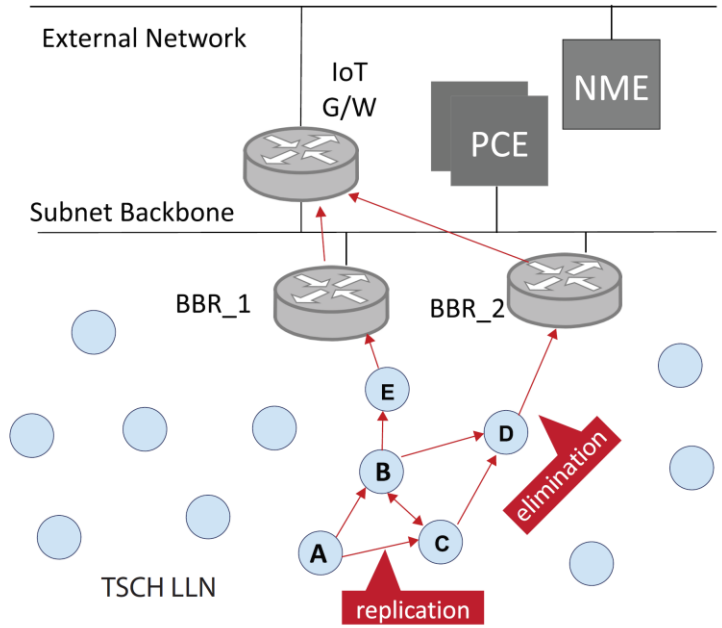


Figure 2: 6TiSCH Protocol Stack (a) and Architecture (b)

The art: 6TiSCH multipath forwarding with BIER-TE

Packet replication and elimination is controlled by BIER-TE (Bit-Indexed Explicit Forwarding – Traffic Engineering):

- Every packet contains a bitString in header with each bit representing an adjacency;
- If a bit of an adjacency is SET to '1' in the bitString, a node would RESET the bit to '0' and send a copy to the adjacency;
- If received multiple copies, a node performs an AND operation on the bitStrings and keeps only one copy.

A BIFT

BitIndex	Adjacency
1	B
2	C

B BIFT

BitIndex	Adjacency
1	A
3	C
4	D

C BIFT

BitIndex	Adjacency
2	A
3	B
5	D

D BIFT

BitIndex	Adjacency
4	B
5	C

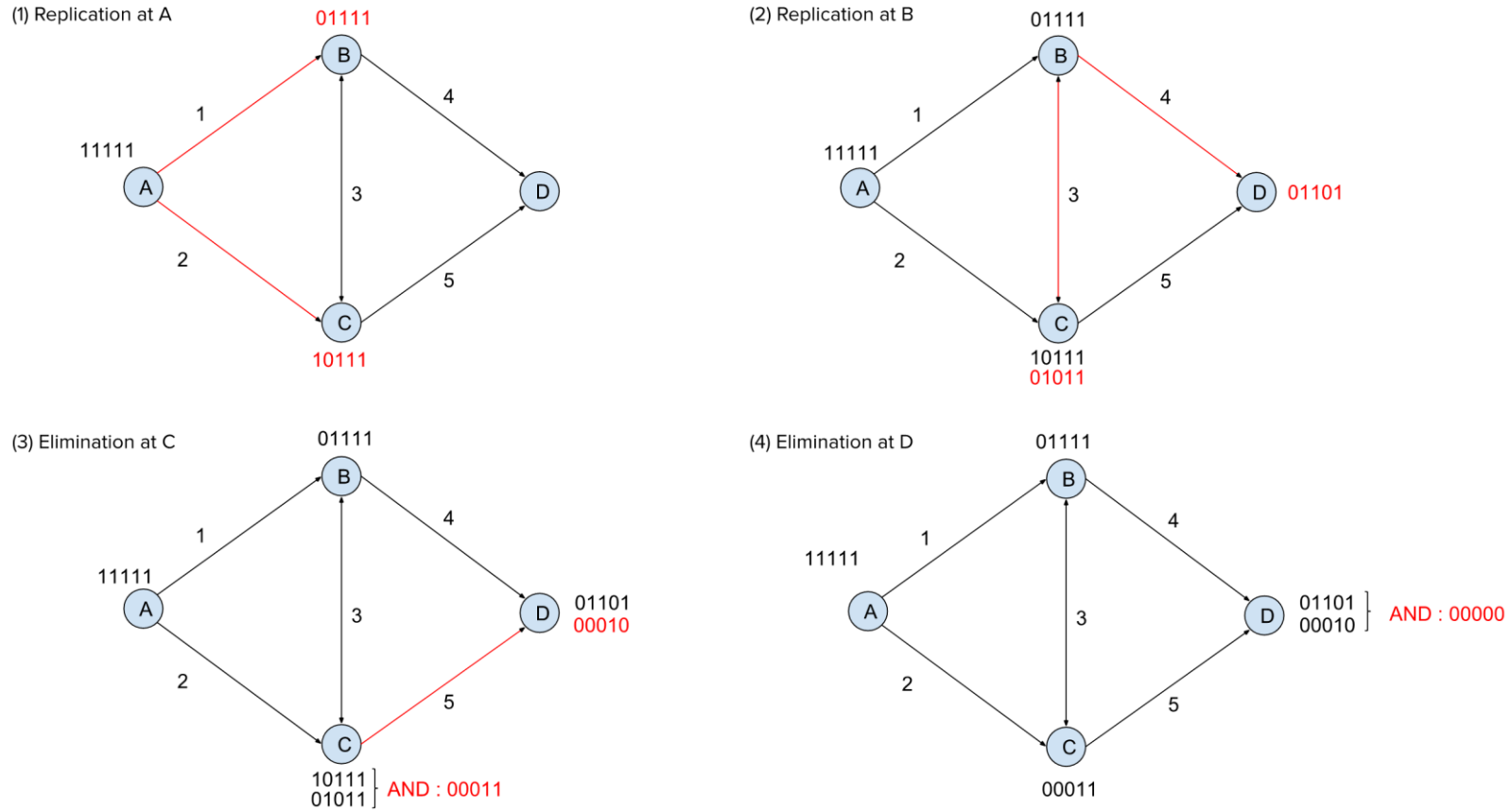
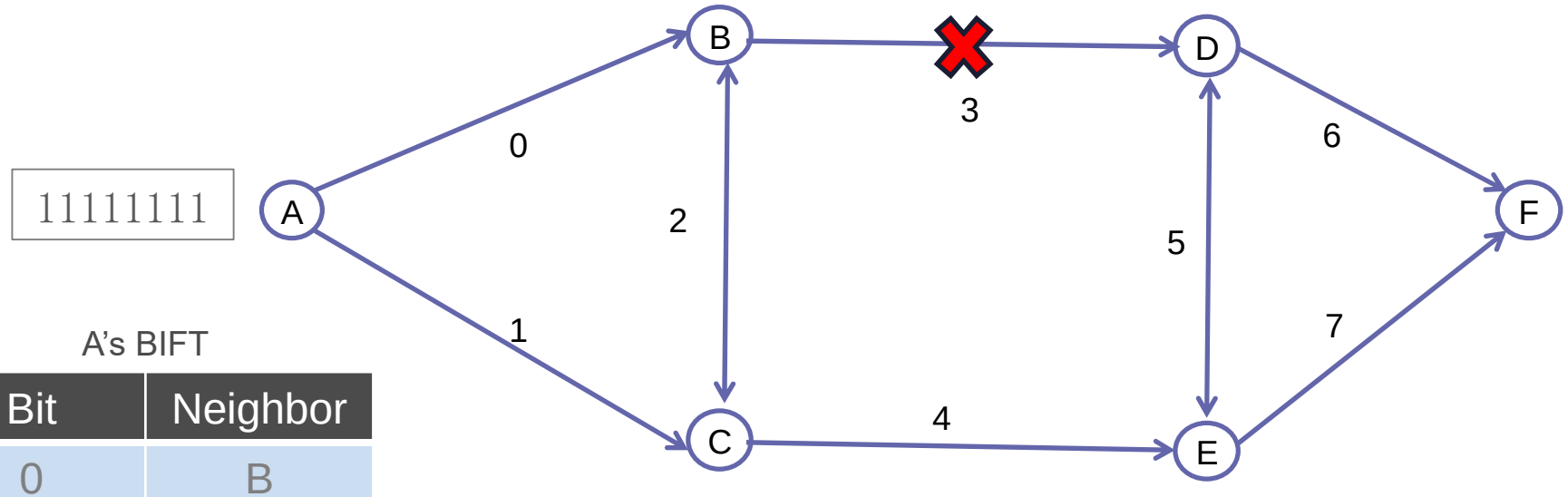


Figure 1: BIER-TE Multipath Forwarding with Full Replication

BIER-TE Multipath Forwarding Rules

Let's assume A has a packet with a bitString '11111111', and the link from B to D is broken.



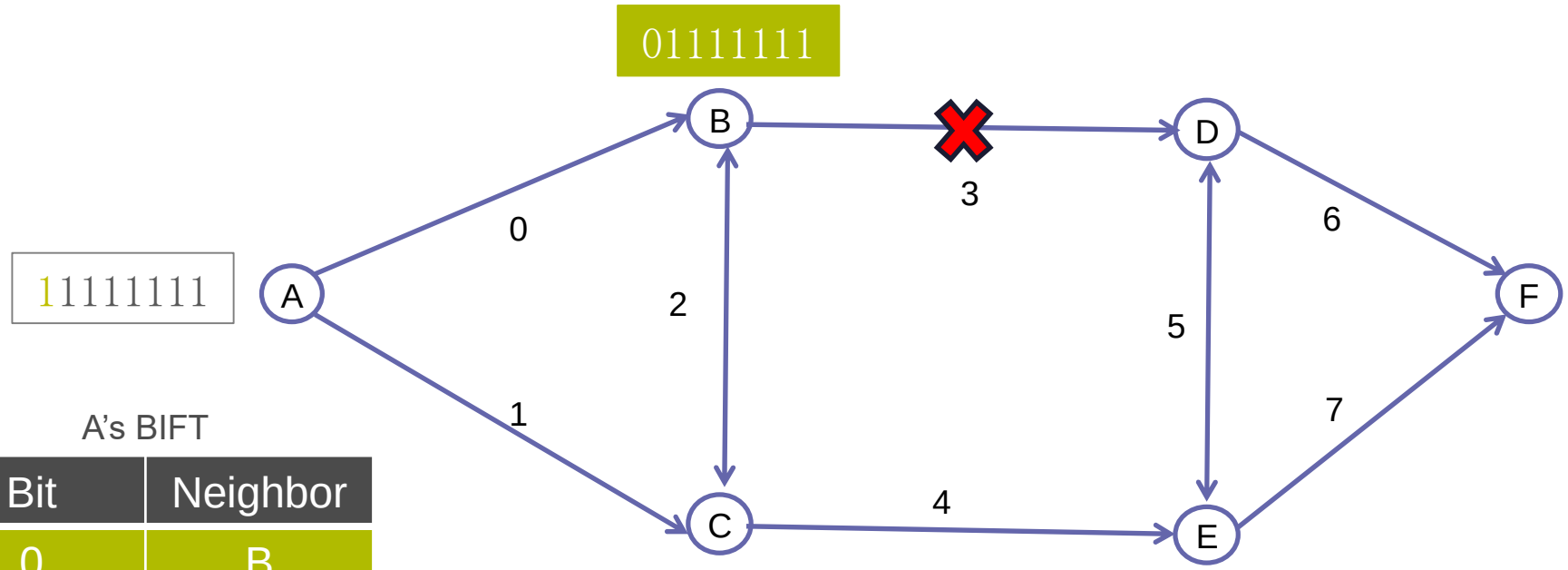
A's BIFT

Bit	Neighbor
0	B
1	C

Schedule:

0	1	2	3	4	5	6	7	8	9
A->B	A->C	B->C	C->B	B->D	C->E	D->E	E->D	D->F	E->F

Slot 0: A resets bit 0 and sends a copy of the packet to B



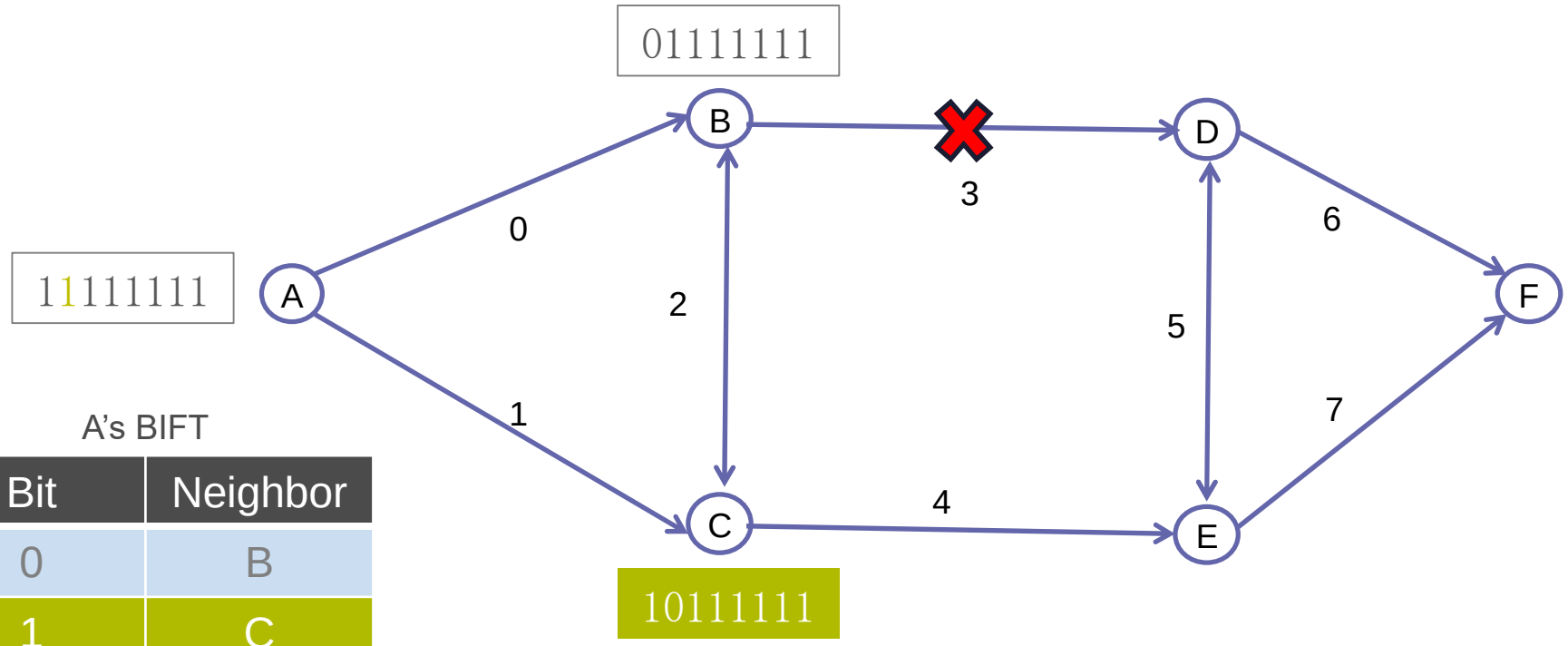
A's BIFT

Bit	Neighbor
0	B
1	C

Schedule:

0	1	2	3	4	5	6	7	8	9
A->B	A->C	B->C	C->B	B->D	C->E	D->E	E->D	D->F	E->F

Slot 1: A resets bit 1 and sends another copy to C



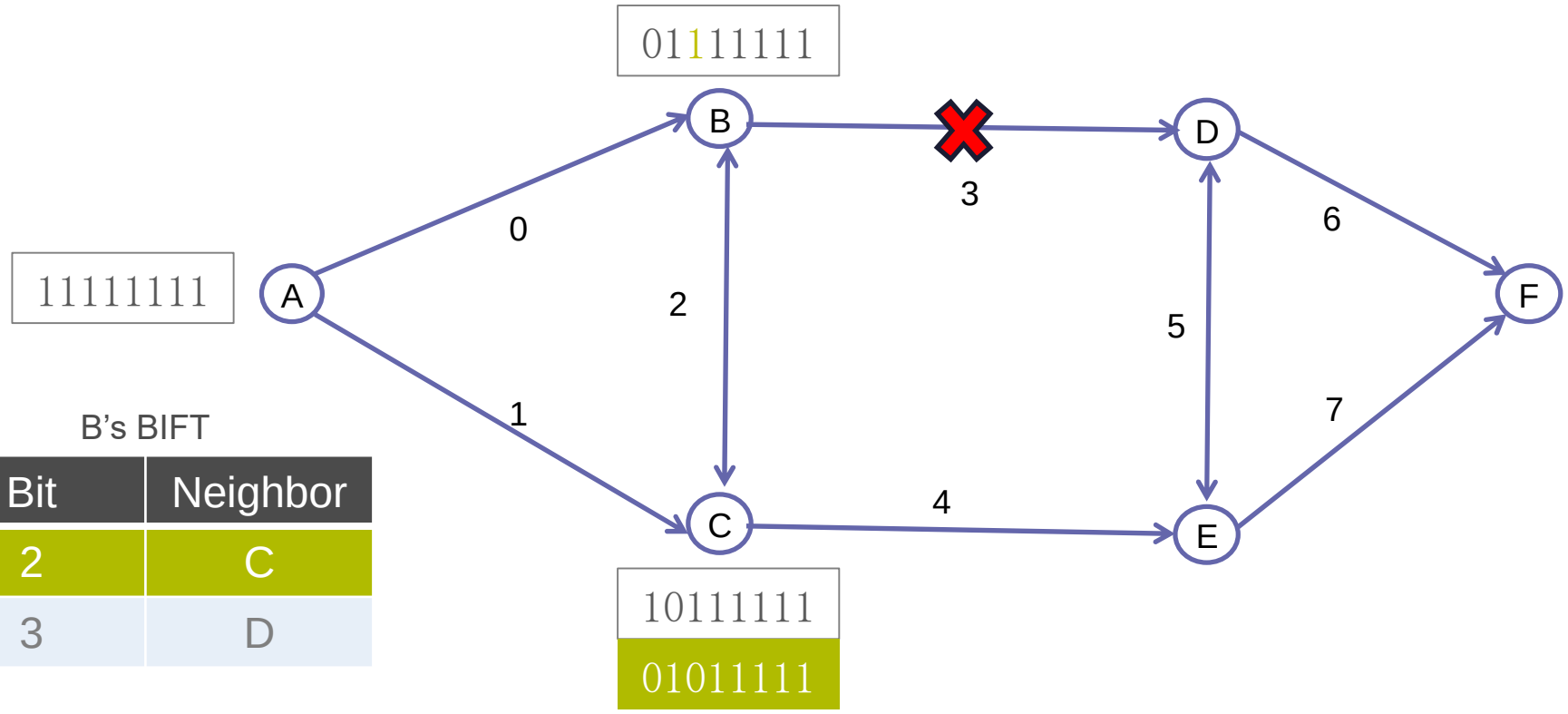
A's BIFT

Bit	Neighbor
0	B
1	C

Schedule:

0	1	2	3	4	5	6	7	8	9
A->B	A->C	B->C	C->B	B->D	C->E	D->E	E->D	D->F	E->F

Slot 2: B resets bit 2 and sends a copy to C



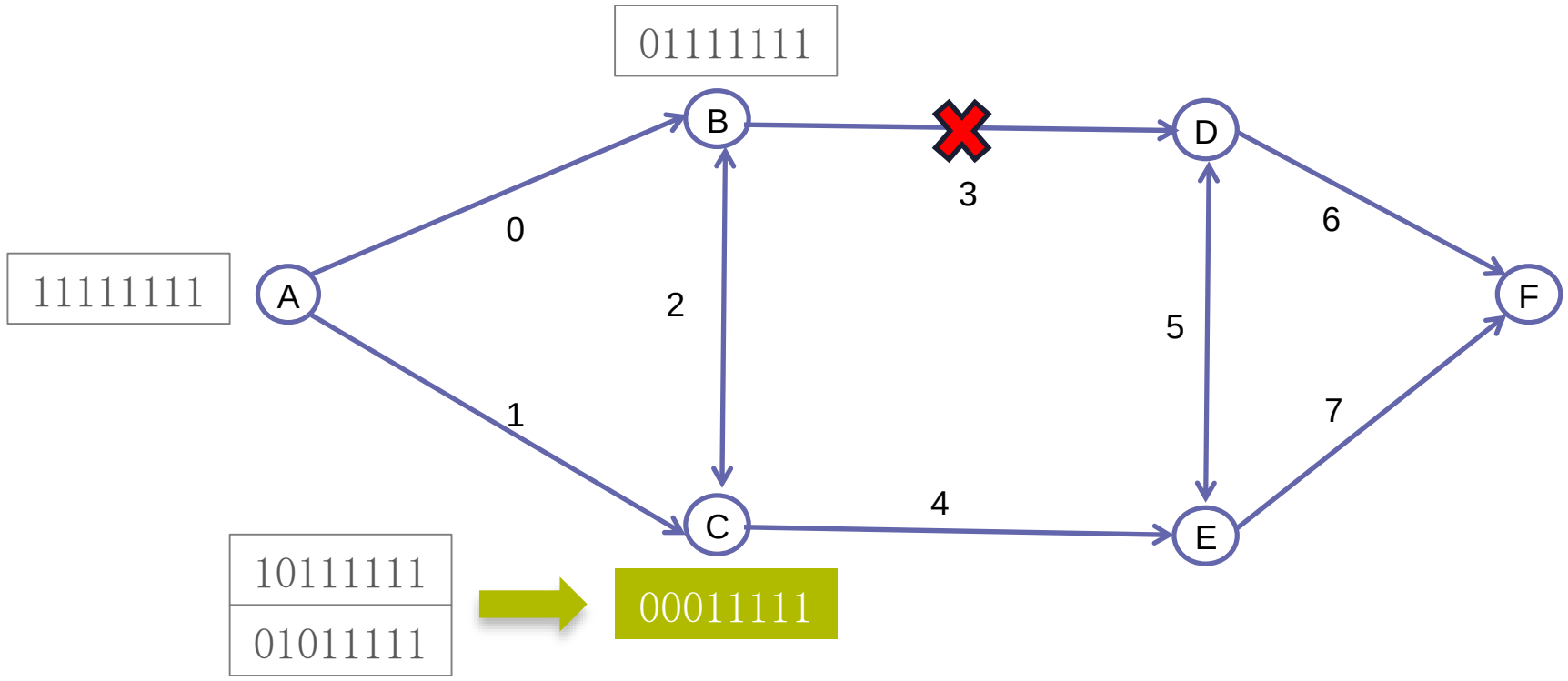
B's BIFT

Bit	Neighbor
2	C
3	D

Schedule:

0	1	2	3	4	5	6	7	8	9
A->B	A->C	B->C	C->B	B->D	C->E	D->E	E->D	D->F	E->F

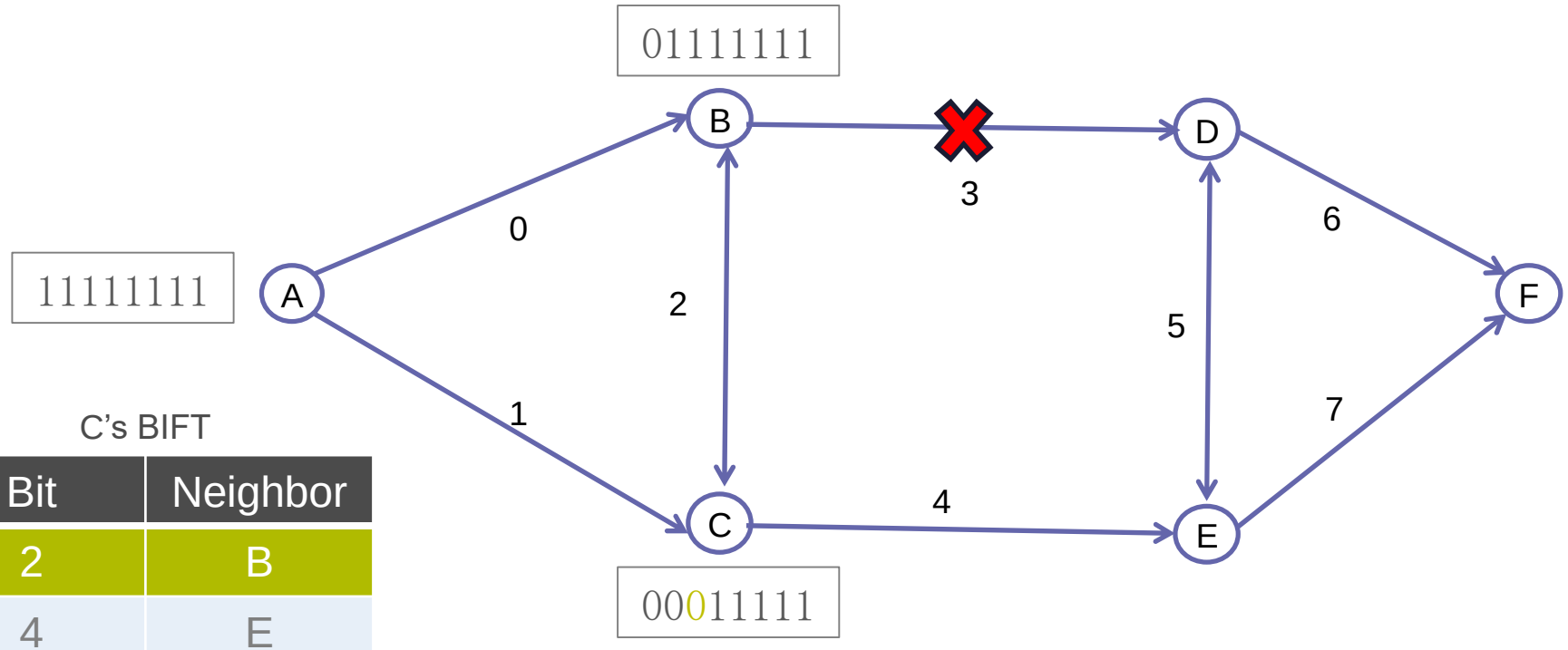
Slot 2: C performs an AND operation on the bitStrings and drops a copy



Schedule:

0	1	2	3	4	5	6	7	8	9
A->B	A->C	B->C	C->B	B->D	C->E	D->E	E->D	D->F	E->F

Slot 3: C does not send a copy to B, as bit 2 is not SET



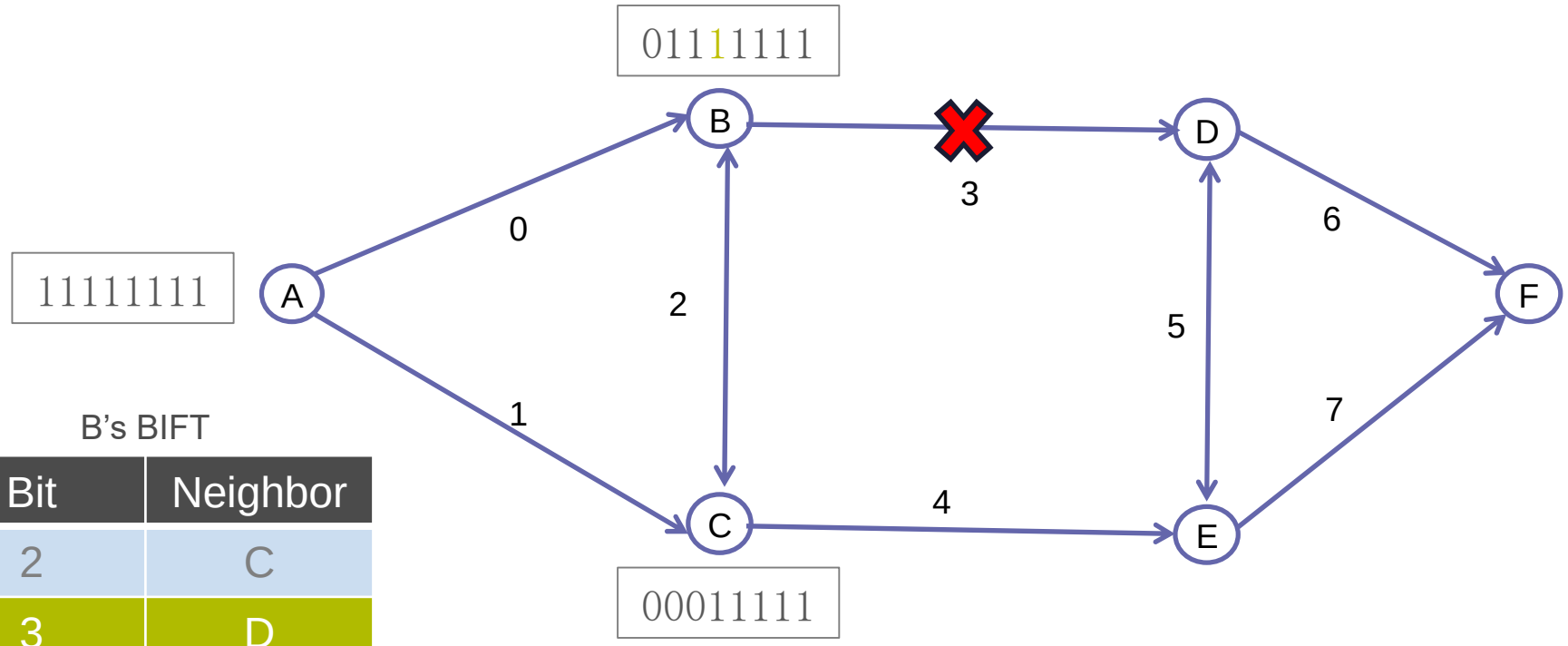
C's BIFT

Bit	Neighbor
2	B
4	E

Schedule:

0	1	2	3	4	5	6	7	8	9
A->B	A->C	B->C	C->B	B->D	C->E	D->E	E->D	D->F	E->F

Slot 4: Transmission fails, and D does not get the message



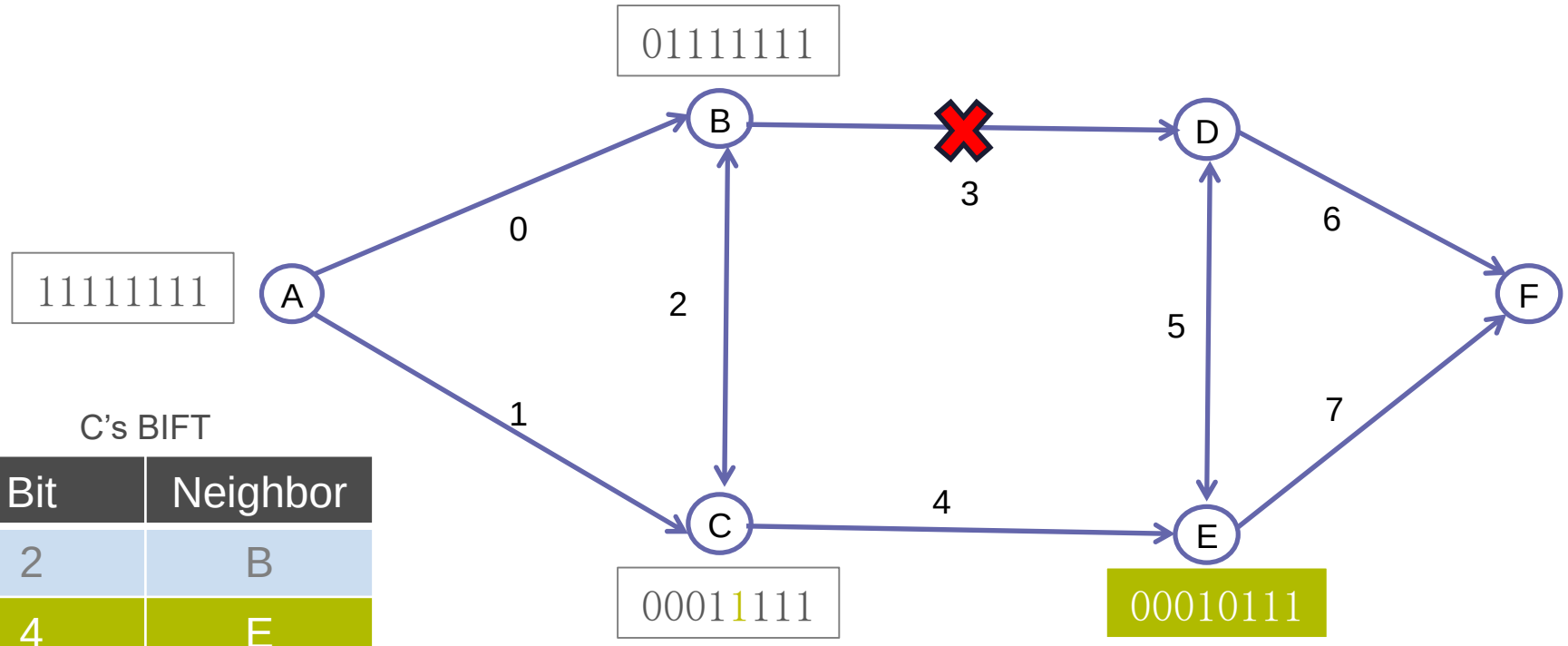
B's BIFT

Bit	Neighbor
2	C
3	D

Schedule:

0	1	2	3	4	5	6	7	8	9
A->B	A->C	B->C	C->B	B->D	C->E	D->E	E->D	D->F	E->F

Slot 5: C resets bit 4 and sends a copy to E



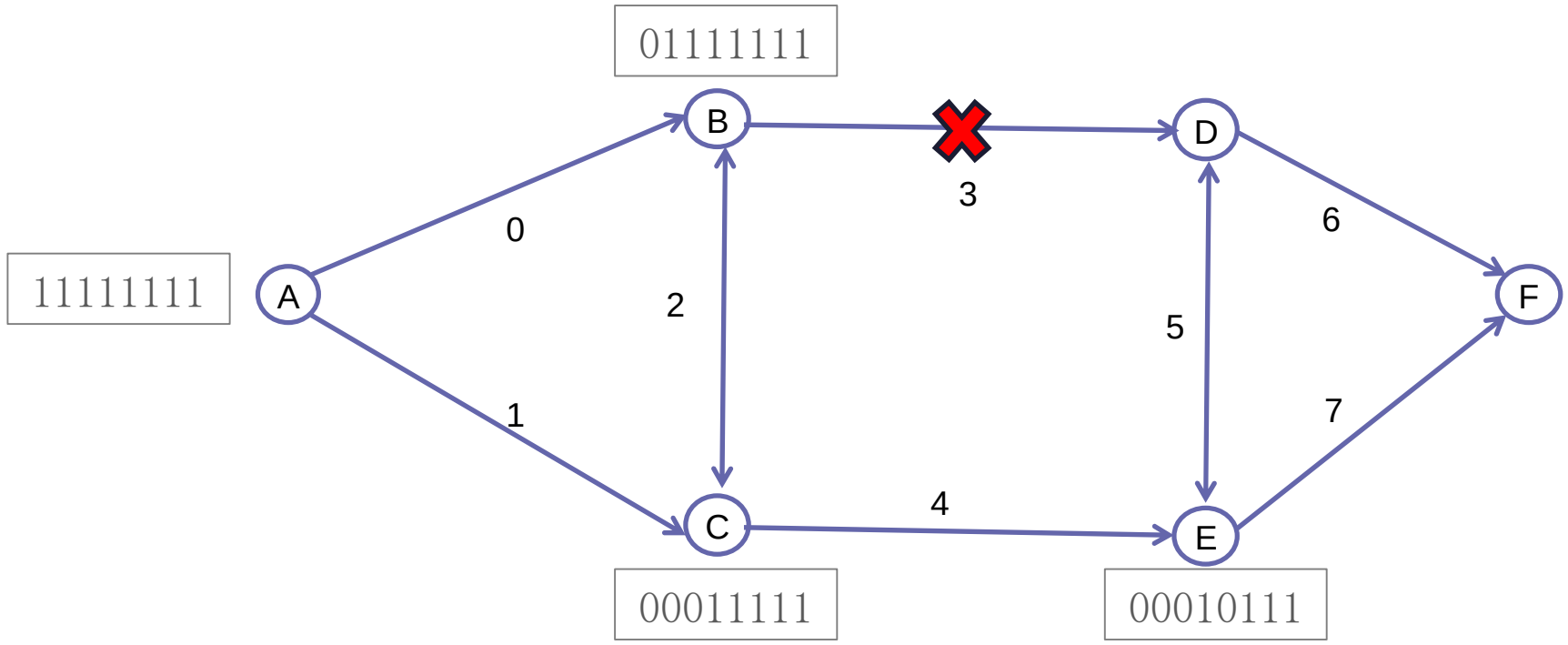
C's BIFT

Bit	Neighbor
2	B
4	E

Schedule:

0	1	2	3	4	5	6	7	8	9
A->B	A->C	B->C	C->B	B->D	C->E	D->E	E->D	D->F	E->F

Slot 6: D has nothing to send



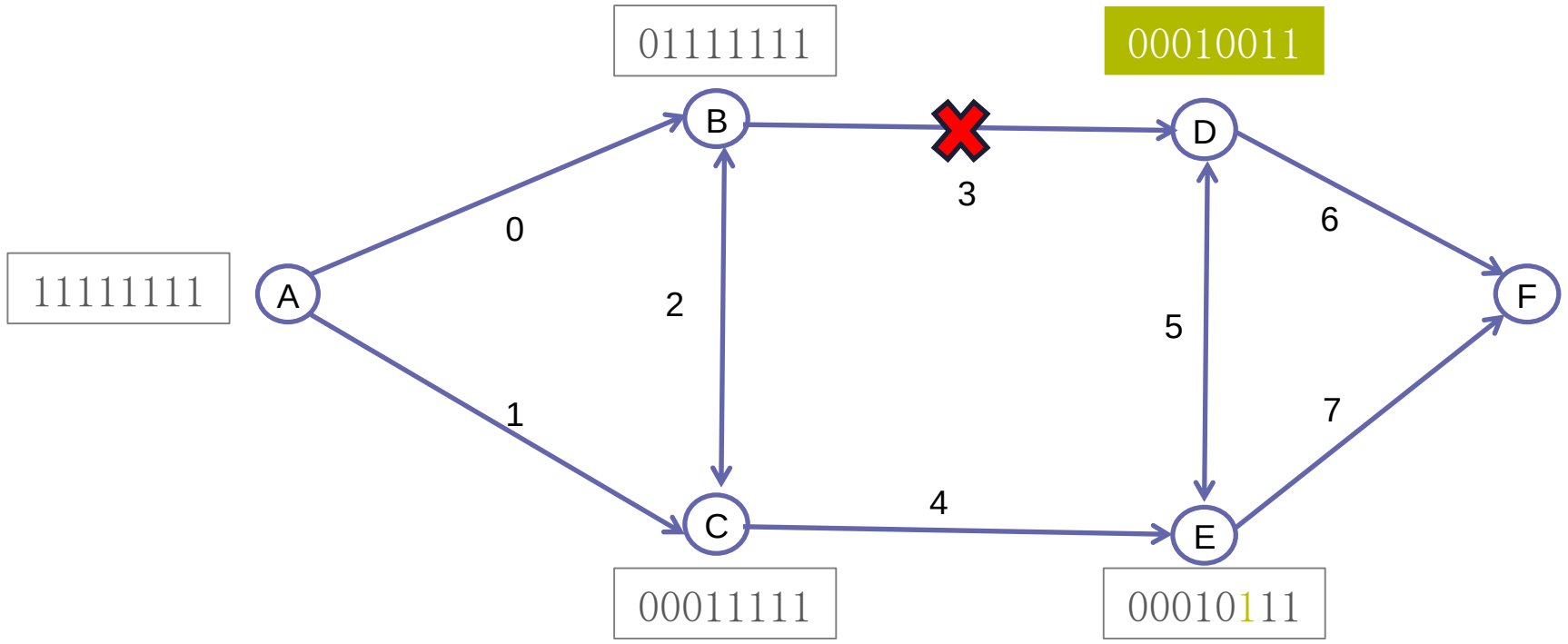
D's BIFT

Bit	Neighbor
5	E
6	F

Schedule:

0	1	2	3	4	5	6	7	8	9
A->B	A->C	B->C	C->B	B->D	C->E	D->E	E->D	D->F	E->F

Slot 7: E resets bit 5 and sends a copy to D



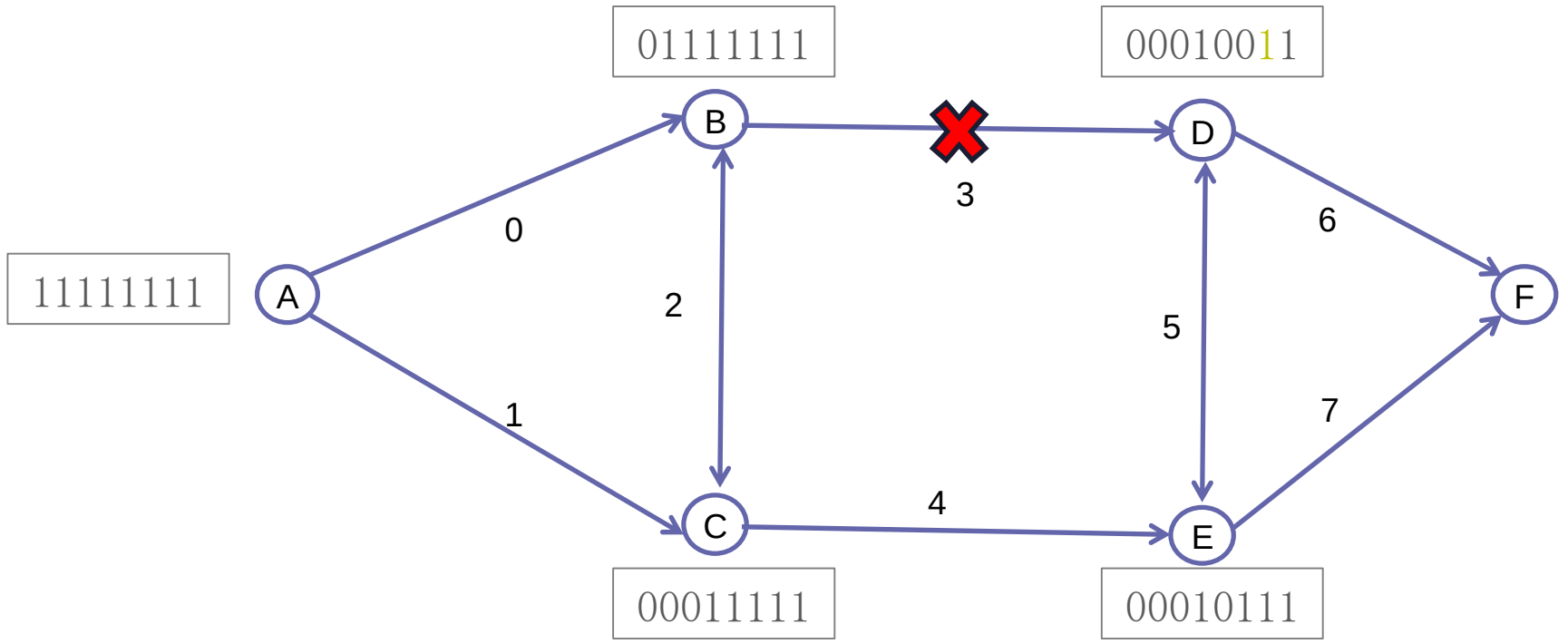
E's BIFT

Bit	Neighbor
5	D
7	F

Schedule:

0	1	2	3	4	5	6	7	8	9
A->B	A->C	B->C	C->B	B->D	C->E	D->E	E->D	D->F	E->F

Slot 8: D resets bit 6 and sends a copy to F



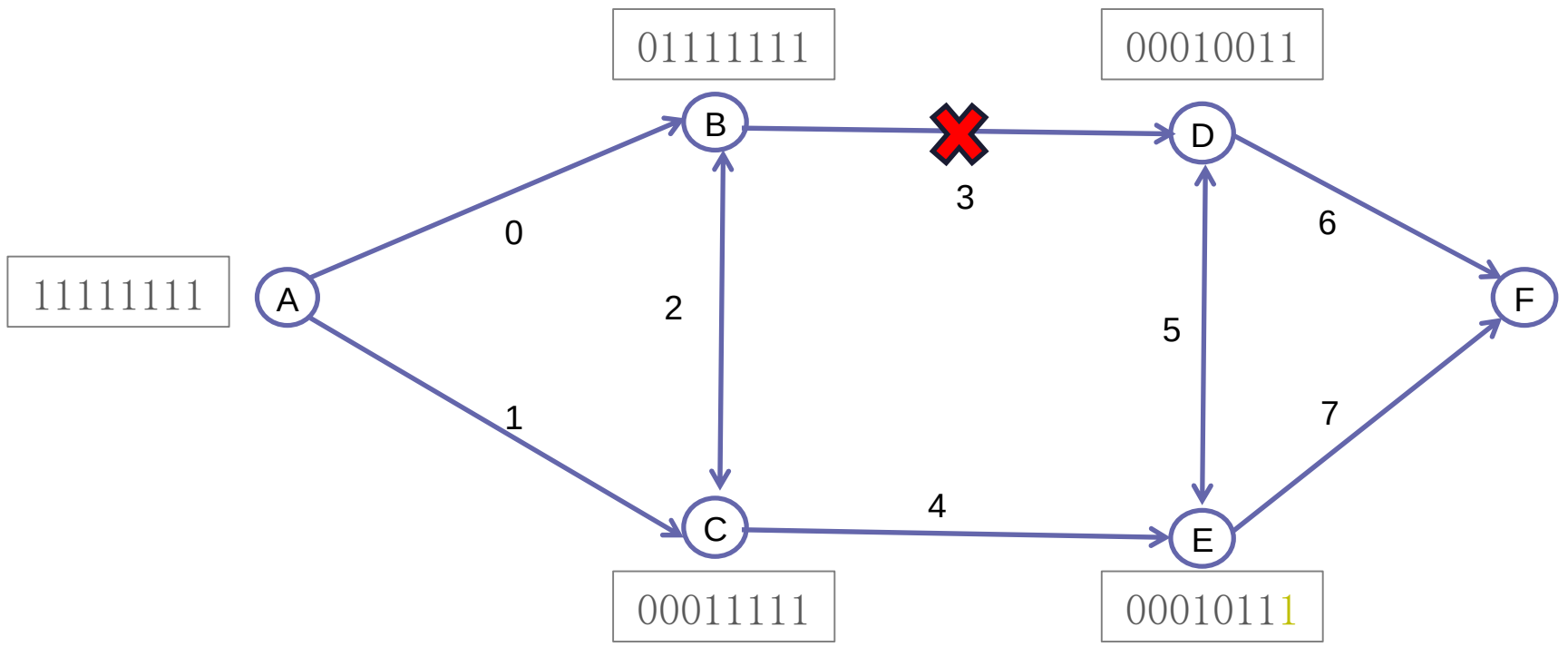
D's BIFT

Bit	Neighbor
5	E
6	F

Schedule:

0	1	2	3	4	5	6	7	8	9
A->B	A->C	B->C	C->B	B->D	C->E	D->E	E->D	D->F	E->F

Slot 9: E resets bit 7 and sends a copy to F



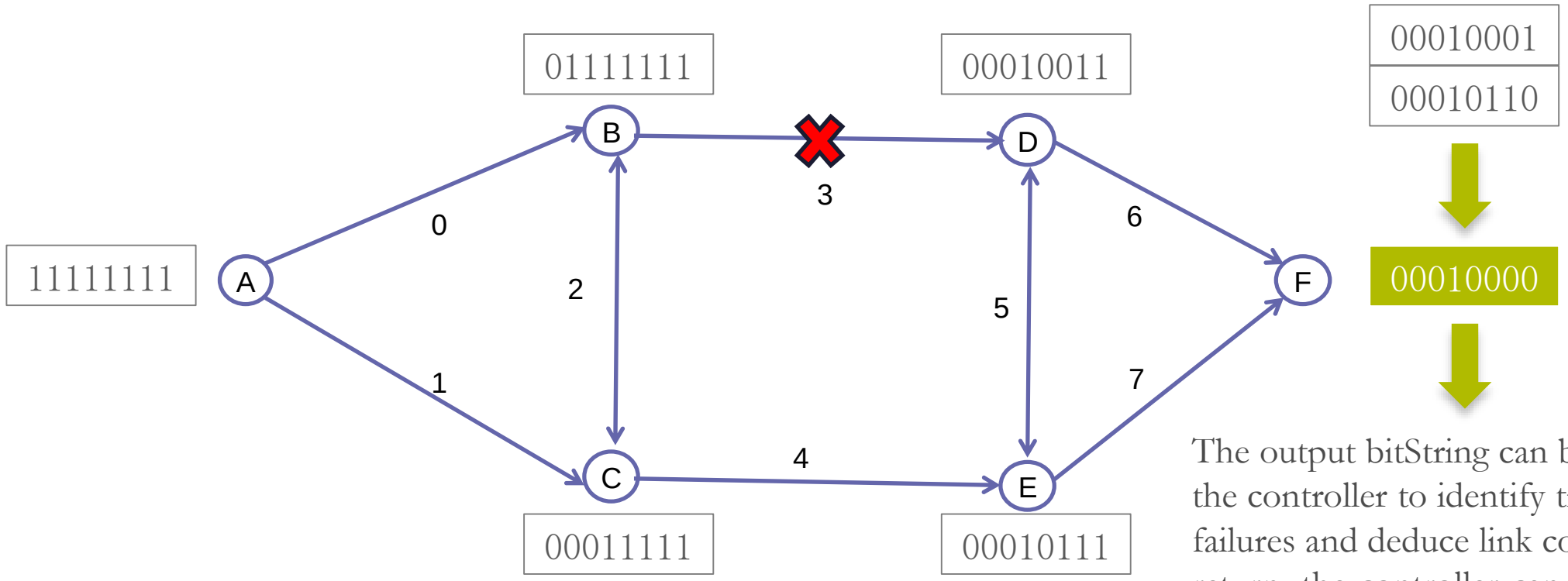
E's BIFT

Bit	Neighbor
5	D
7	F

Schedule:

0	1	2	3	4	5	6	7	8	9
A->B	A->C	B->C	C->B	B->D	C->E	D->E	E->D	D->F	E->F

Slot 9: F performs an AND operation and forwards a copy to upper layer



The output bitString can be passed to the controller to identify transmission failures and deduce link conditions, in return, the controller can accordingly modify the bitString for next packets.

Schedule:

0	1	2	3	4	5	6	7	8	9
A->B	A->C	B->C	C->B	B->D	C->E	D->E	E->D	D->F	E->F

Internship: 6TiSCH centralized scheduling and multipath construction with BIER-TE

Goal

explore centralized operations with a PCE on control plane to improve the delivery ratio while lowering energy budget

Content

Topology Discovery, Multipath Computation, Resource/Track Reservation, BitString-based Multipath Dynamic Control

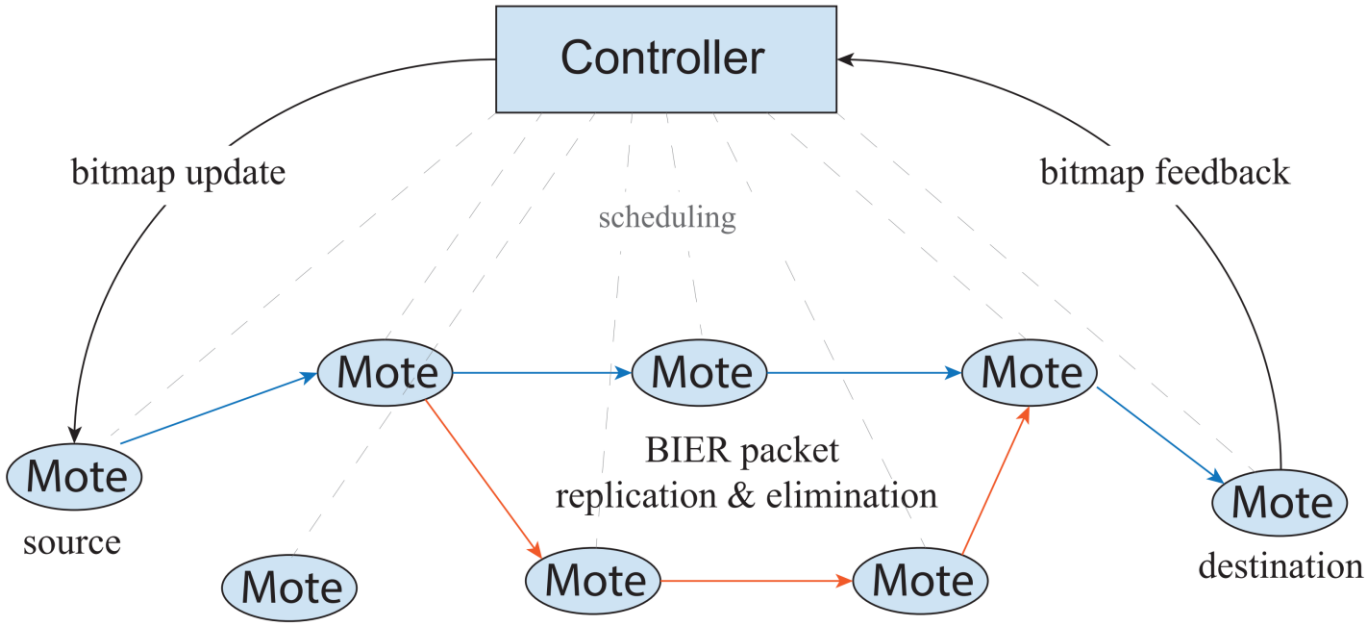


Figure 1: BIER-TE Multipath Control

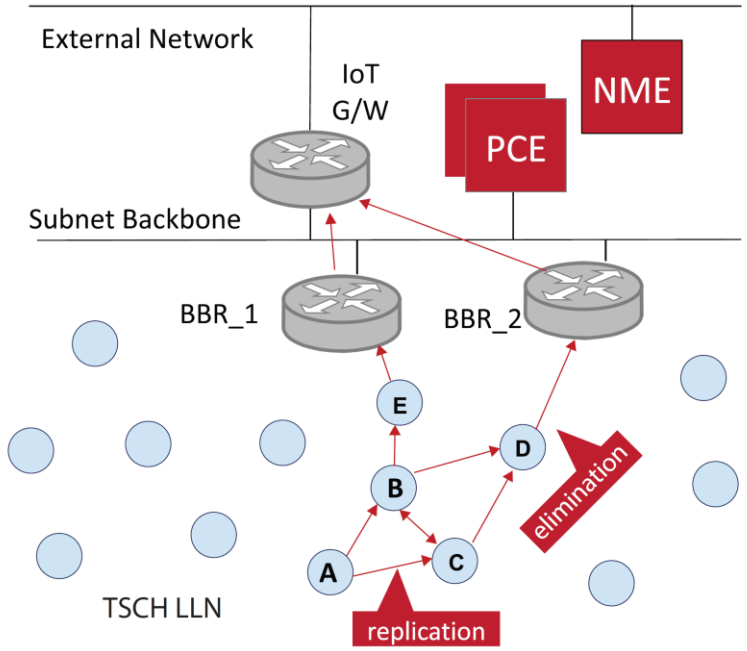


Figure 2: 6TiSCH Centralized Architecture

Topology Learning Track Computation Track Reservation Multipath Control

RPL (IPv6 Routing Protocol for Low-Power and Lossy Networks) Non-storing mode defines:

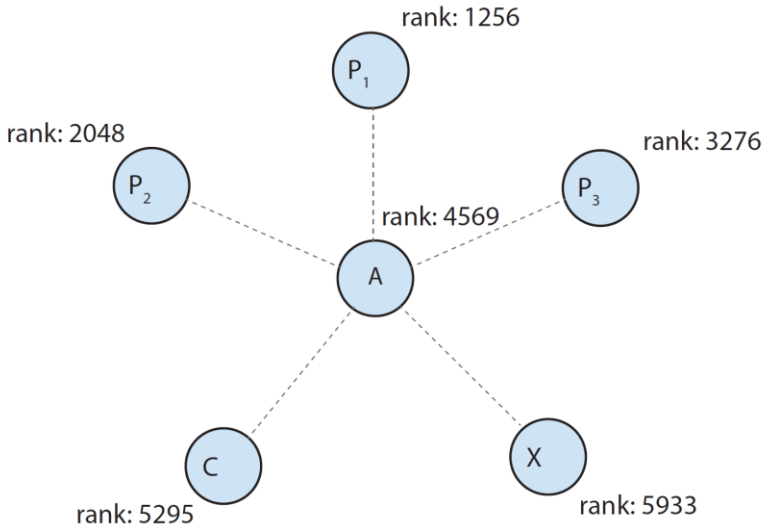
each node periodically unicast a list of its parents to the DAG root, which normally directly connects to the controller.

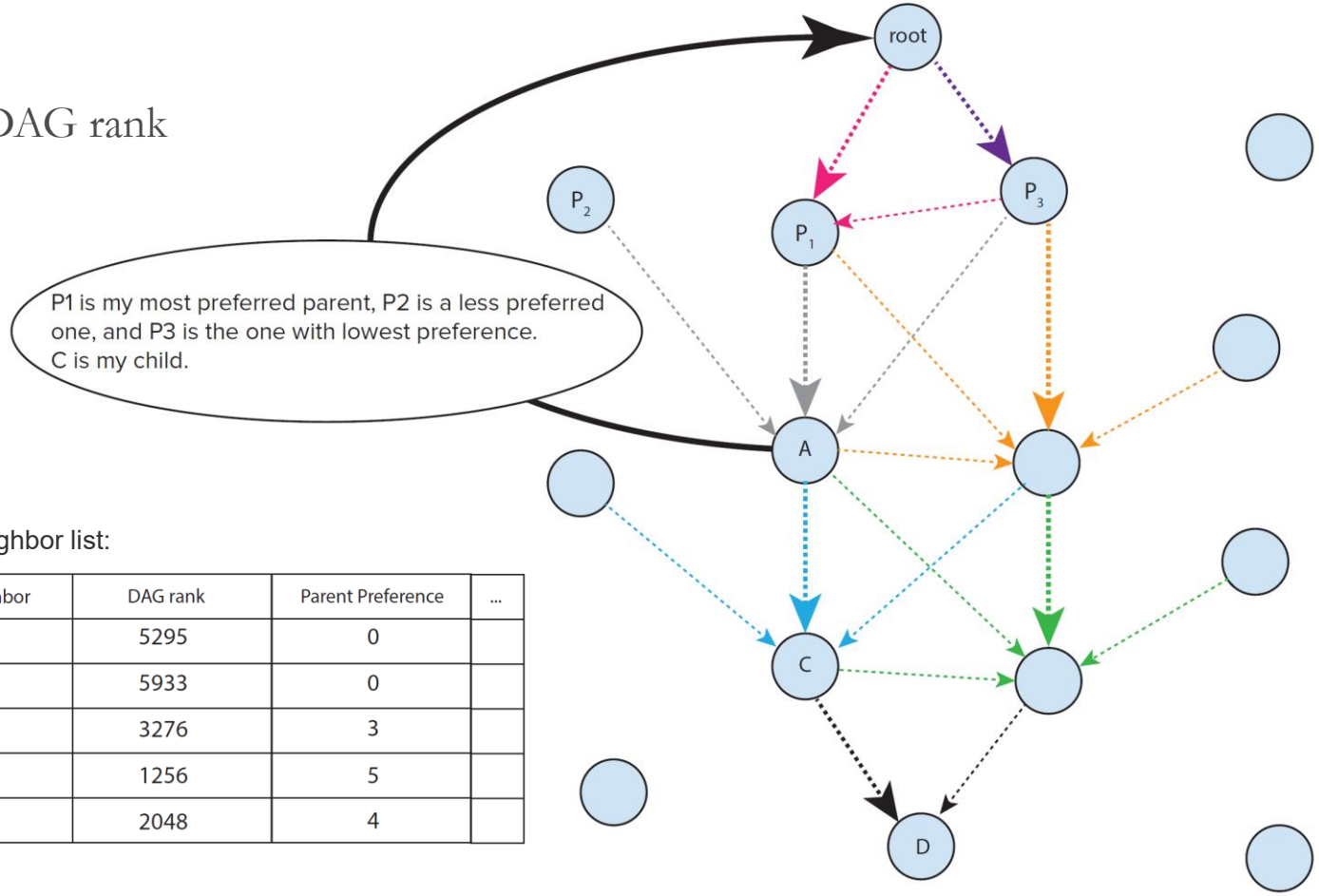
Data plane work

- Multipath Selection Algorithm based on the DAG rank
- RPL DAO message compression

Control plane work

- Topology Monitoring





A's neighbor list:

Neighbor	DAG rank	Parent Preference	...
C	5295	0	
X	5933	0	
P_3	3276	3	
P_1	1256	5	
P_2	2048	4	

Figure 1: Multi-parent Selection and Advertisement

Implementation on OpenWSN open-source code

- Each mote selects a maximum of 5 parents with different preferences;
- A compressed DAO can contain 5 parents' and 1 child's information;
- The controller updates topology information each time receiving a DAO.

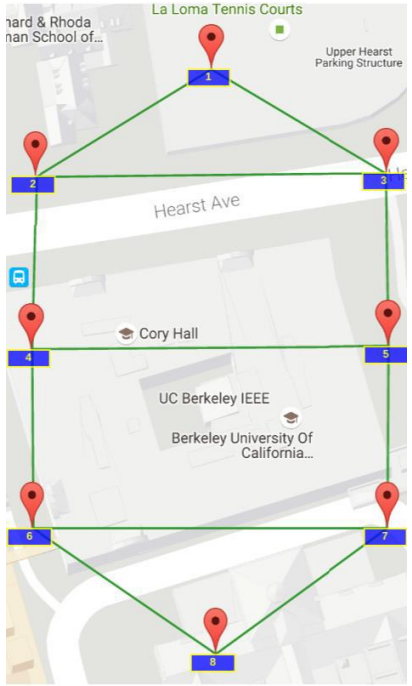


Figure 1: an example network in simulation

```
received RPL DAO from 14-15-92-cc-00-00-00-03
- parents:
   . 14-15-92-cc-00-00-00-01 : 5
- children:
   . 14-15-92-cc-00-00-00-05

received RPL DAO from 14-15-92-cc-00-00-00-05
- parents:
   . 14-15-92-cc-00-00-00-03 : 5
- children:
   . 14-15-92-cc-00-00-00-07

received RPL DAO from 14-15-92-cc-00-00-00-02
- parents:
   . 14-15-92-cc-00-00-00-03 : 4
   . 14-15-92-cc-00-00-00-01 : 5
- children:

received RPL DAO from 14-15-92-cc-00-00-00-07
- parents:
   . 14-15-92-cc-00-00-00-05 : 5
- children:
   . 14-15-92-cc-00-00-00-06

received RPL DAO from 14-15-92-cc-00-00-00-08
- parents:
   . 14-15-92-cc-00-00-00-06 : 4
   . 14-15-92-cc-00-00-00-07 : 5
- children:

received RPL DAO from 14-15-92-cc-00-00-00-06
- parents:
   . 14-15-92-cc-00-00-00-07 : 4
   . 14-15-92-cc-00-00-00-04 : 5
- children:
   . 14-15-92-cc-00-00-00-08

received RPL DAO from 14-15-92-cc-00-00-00-04
- parents:
   . 14-15-92-cc-00-00-00-05 : 4
   . 14-15-92-cc-00-00-00-02 : 5
- children:
   . 14-15-92-cc-00-00-00-06
```

Figure 2: received DAO messages from the nodes

```
▸ Frame 466: 205 bytes on wire (1640 bits), 205 bytes captured (1640 bits) on interface 0
  Raw packet data
▸ Internet Protocol Version 6, Src: bbbb::1, Dst: bbbb::1
▸ User Datagram Protocol, Src Port: 0 (0), Dst Port: 17754 (17754)
▸ ZigBee Encapsulation Protocol, Channel: 20, Length: 125
▸ IEEE 802.15.4 Data, Dst: 14:15:92:cc:00:00:00:01, Src: 14:15:92:cc:00:00:00:02
▸ 6LoWPAN
▸ Internet Protocol Version 6, Src: fe80::1415:92cc:0:7, Dst: fe80::1415:92cc:0:1
▾ Internet Control Message Protocol v6
    Type: RPL Control (155)
    Code: 2 (Destination Advertisement Object)
  ▸ Checksum: 0xd76b [incorrect, should be 0x51e1]
  ▸ Flags: 0x40
    DAO Sequence: 0
    DODAGID: 1415:92cc:0:1::
  ▾ ICMPv6 RPL Option (RPL Target 1415:92cc:0:a::(prefix len: 64))
      Type: RPL Target (5)
      Target: 1415:92cc:0:a::
  ▸ ICMPv6 RPL Option (Transit Information 1415:92cc:0:4::)
  ▸ ICMPv6 RPL Option (Transit Information 1415:92cc:0:6::)
  ▸ ICMPv6 RPL Option (Transit Information 1415:92cc:0:5::)
  ▸ ICMPv6 RPL Option (Transit Information 1415:92cc:0:3::)
  ▾ ICMPv6 RPL Option (Transit Information 1415:92cc:0:2::)
      Type: Transit Information (6)
      Path Control: 5
      Path Sequence: 13
      Parent Address: 1415:92cc:0:2::

0000  60 00 00 00 00 a5 11 08  bb bb 00 00 00 00 00 00   `....... ........
0010  00 00 00 00 00 00 00 01  bb bb 00 00 00 00 00 00   ........ ........
0020  00 00 00 00 00 00 00 01  00 00 45 5a 00 a5 e8 6a   ........ ..EZ...j
0030  45 58 02 01 14 00 01 01  ff 01 01 01 01 01 01 01   EX...... ........
0040  01 02 02 02 02 00 00 00  00 00 00 00 00 00 00 7d   ........ .......}

Frame (205 bytes)   Decompressed 6LoWPAN IPHC (118 bytes)
```

Figure 3: a compressed DAO dissected by revised Wireshark dissector

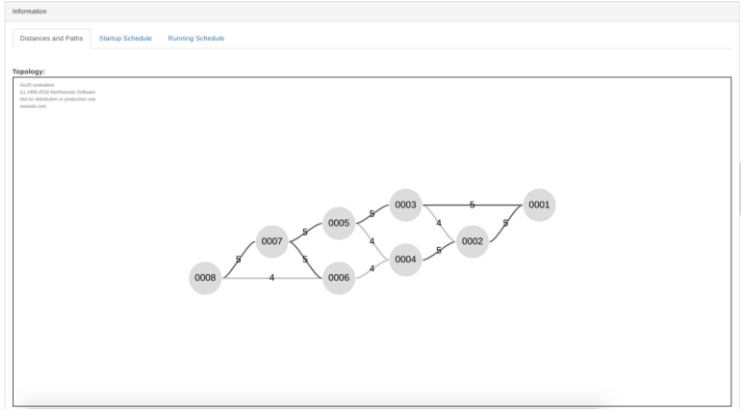


Figure 4: Controller's view of topology learned from RPL

Based on the topology exposed by RPL, a variation of ARCs (Available Routing Constructs) algorithm is used to compute the track, with a general shape of a ladder from the source to the destination.

ARC Basics

An ARC is a loop-free ordered set of nodes and links whereby the data traffic may enter via any node of it but may only leave it through either one of its edges.

Each ARC is composed of 3 subsets {Left Exit}{Intermediate Nodes}{Right Exit}.

The links between intermediate nodes are made reversible (except for the last ARC).

Goal

The idea is to protect each hop of a path with an ARC starting from the destination.

As a result, each node would have 2 alternative outgoing links to reach the destination.

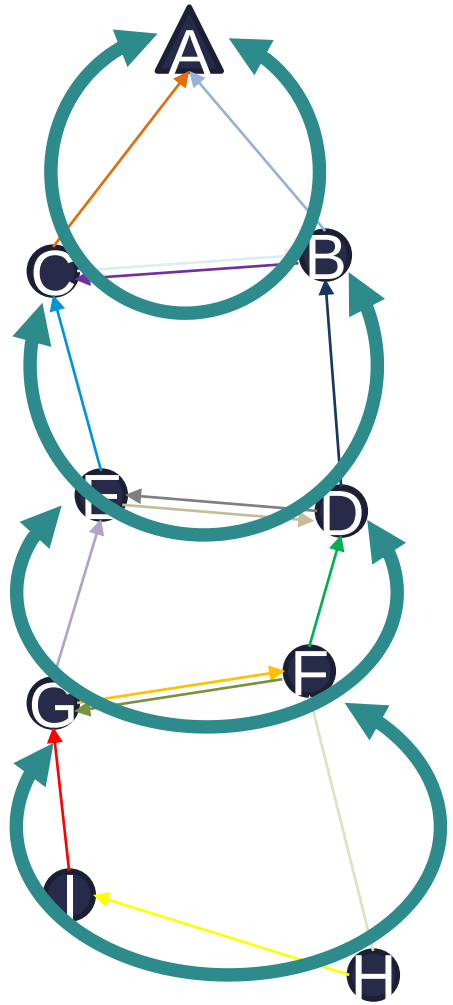


Figure 1: an example track from H to A

After computing the track, the controller:

- signals each node along the track to reserve time slots for each transmission;
- assigns each segment an associated bitPosition/bitIndex in the bitString;
- installs the BIFT (Bit-Indexed Forwarding Table) on each node;
- assigns a dedicated bitString to the ingress node of this track.

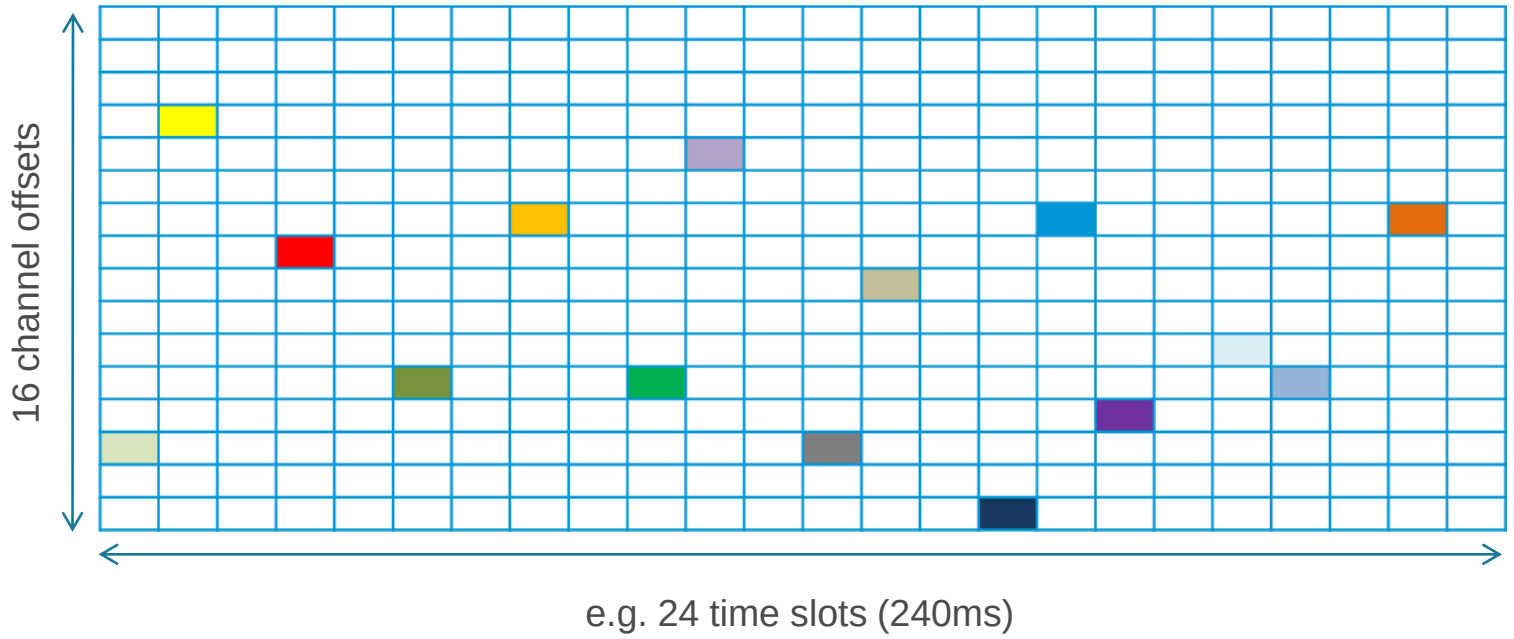


Figure 1: reserved time slots for the track

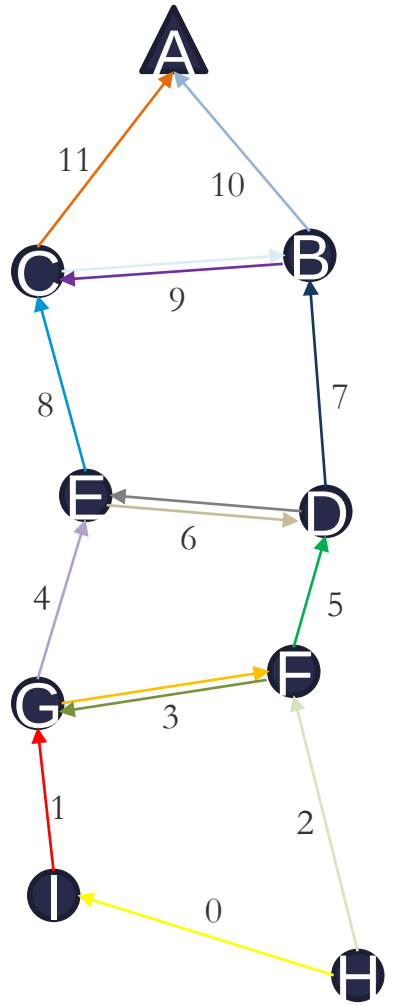


Figure 2: the computed track from H to A

Implementation

- a reversible link means two timeslots for both directions
- first transmission first timeslot
- the bits of a bitString should be enabled according to policies and link conditions

node	type	frameID	action	cell information
A	schedule	1	add	{'type': 'tx', 'slotOffset': 1, 'channelOffset': 4, 'shared': False, 'trackID': 1, 'bitIndex': 0, 'BIER': True}
B	schedule	1	add	{'type': 'rx', 'slotOffset': 1, 'channelOffset': 4, 'shared': False, 'trackID': 1, 'bitIndex': 0, 'BIER': True}

Table 1: example signaling data structure for the reservation of transmission from A to B

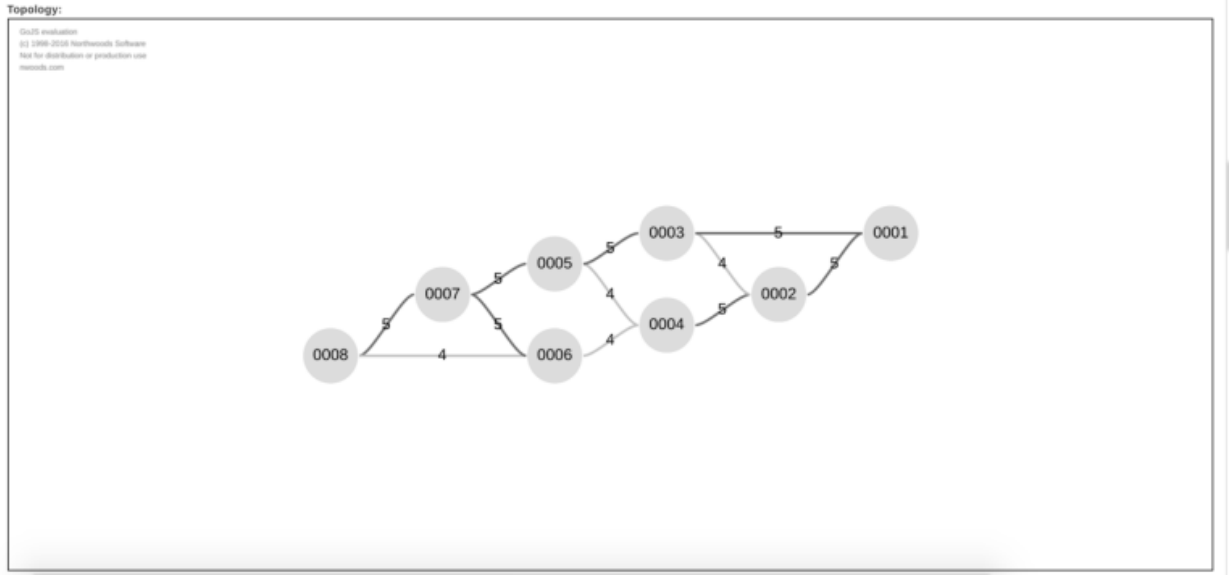


Figure 1: topology learned from RPL

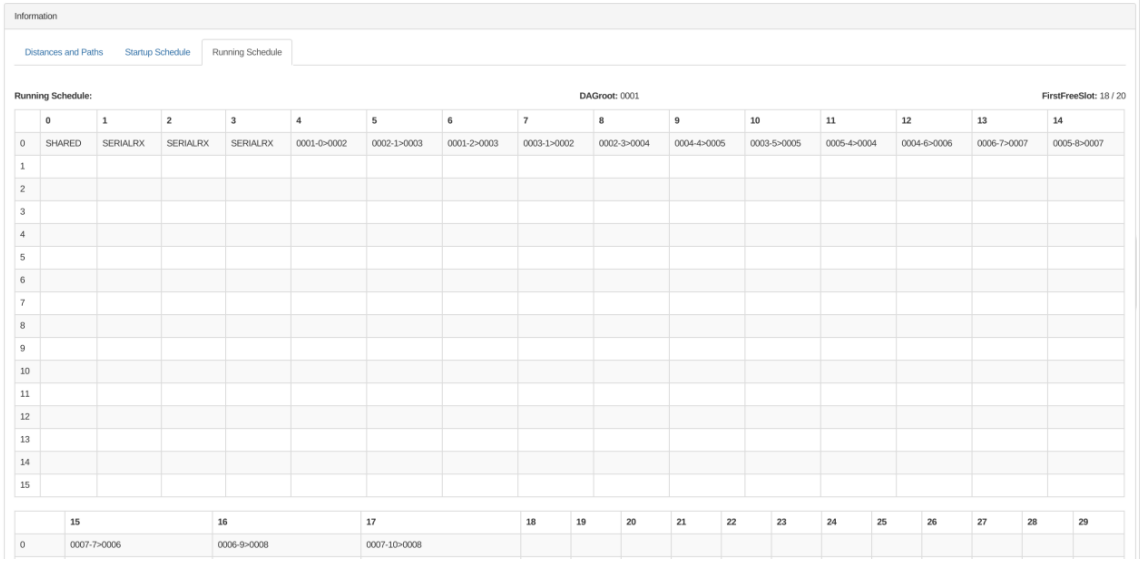


Figure 3: reserved slots for the track

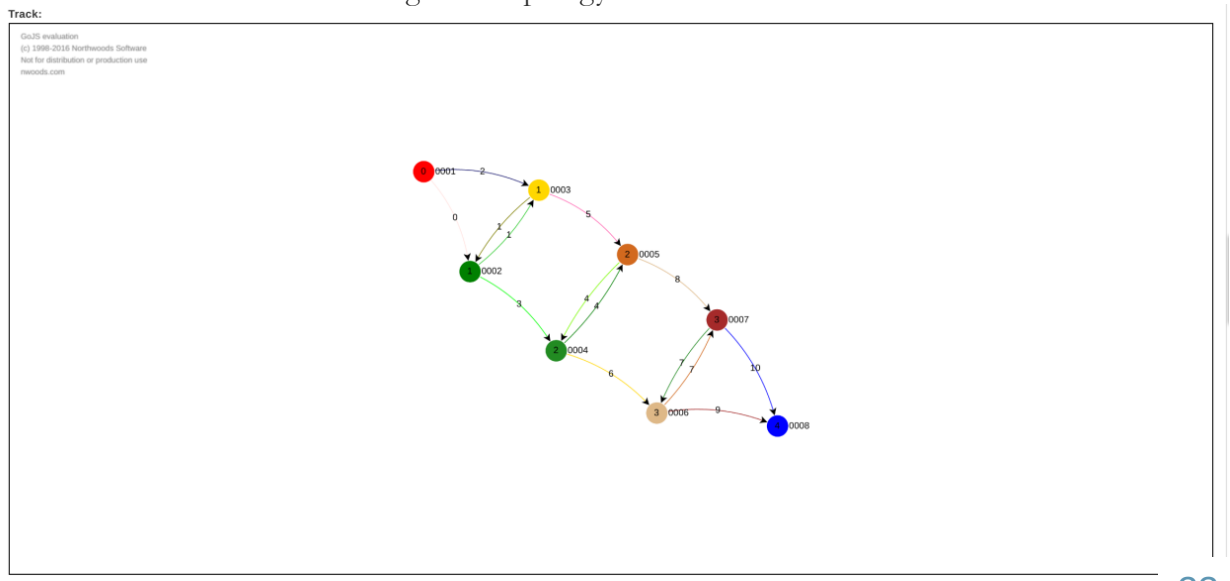
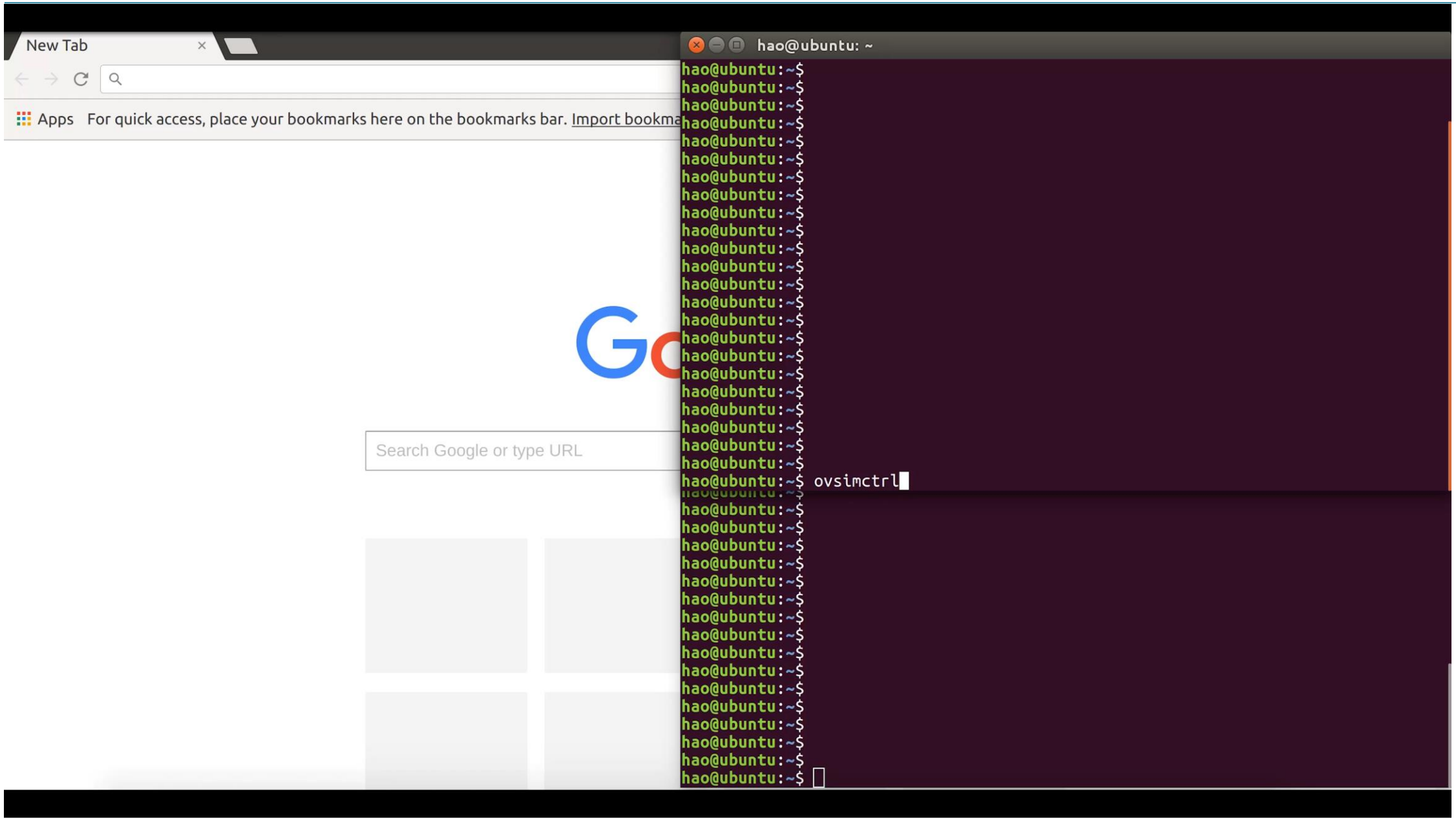


Figure 2: the computed track based on the topology


New Tab
Apps For quick access, place your bookmarks here on the bookmarks bar. Import bookma
Search Google or type URL
hao@ubuntu: ~
hao@ubuntu:~$
hao@ubuntu:~$ ovsimctrl

Goal

Enable multipath/packet replication mechanism only when necessary to save energy while maintaining high reliability.

Steps

- Set up a real hardware testbed;
- Validate experimentally the value of path diversity;
- Analyze when and how the mechanism should be activated;
- Based on the conclusions, design and implement a bitString feedback control loop.

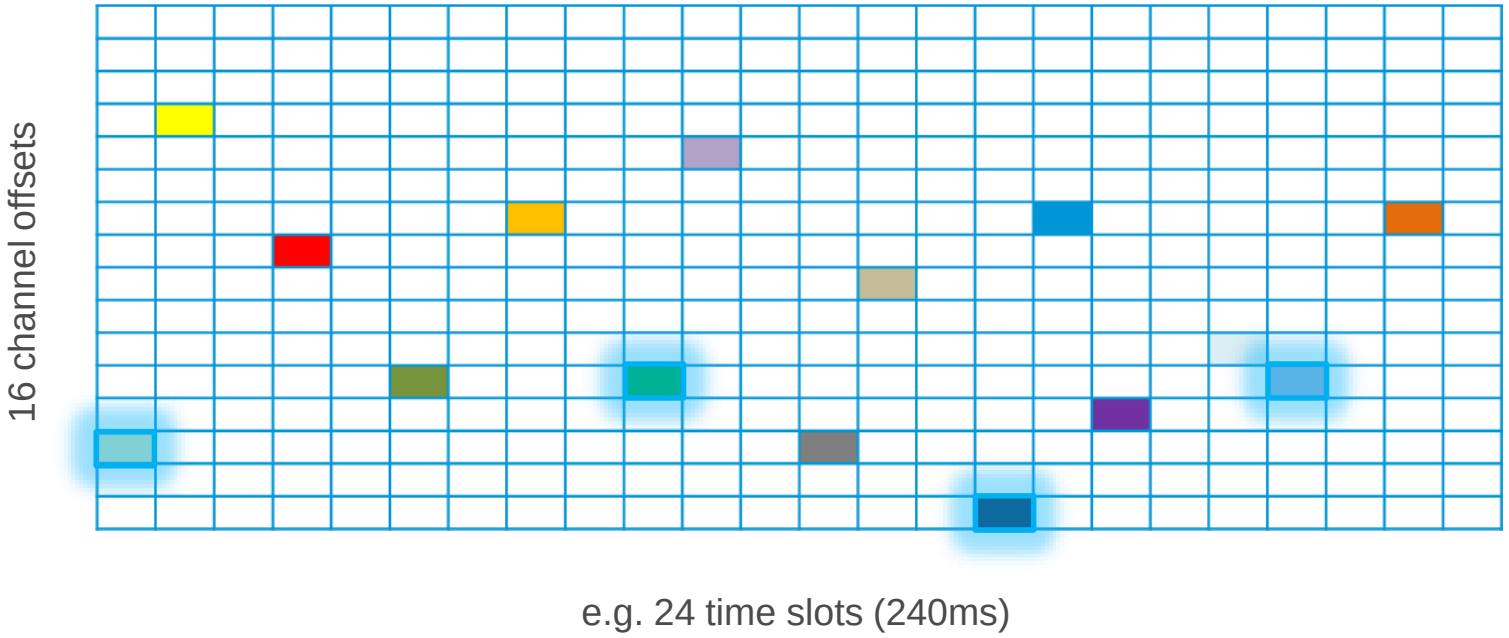


Figure 1: enabled timeslots for single path

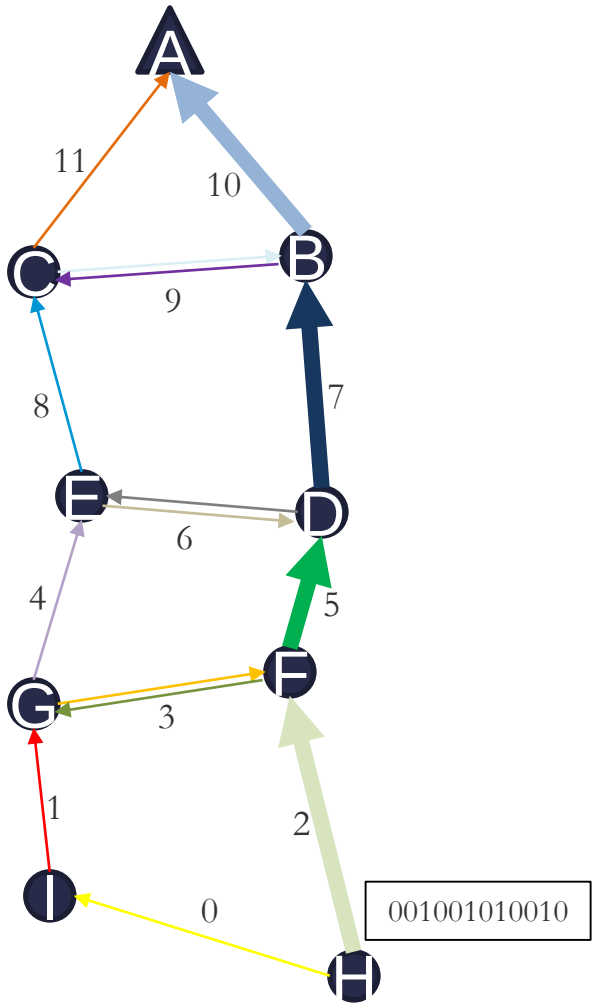


Figure 2: enable single path when link quality is good

Testbed Implementation

- Raspberry Pi 3
- OpenWSN open-source code
- OpenMote hardware

Submitted paper :
Rover: Poor (but Elegant) Man's Testbed.
Zacharie Brodard, Hao Jiang, Tengfei Chang, Thomas Watteyne, Xavier Vilajosana, Pascal Thubert, Geraldine Texier.
ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN), Valletta, Malta, 13-17 November 2016.

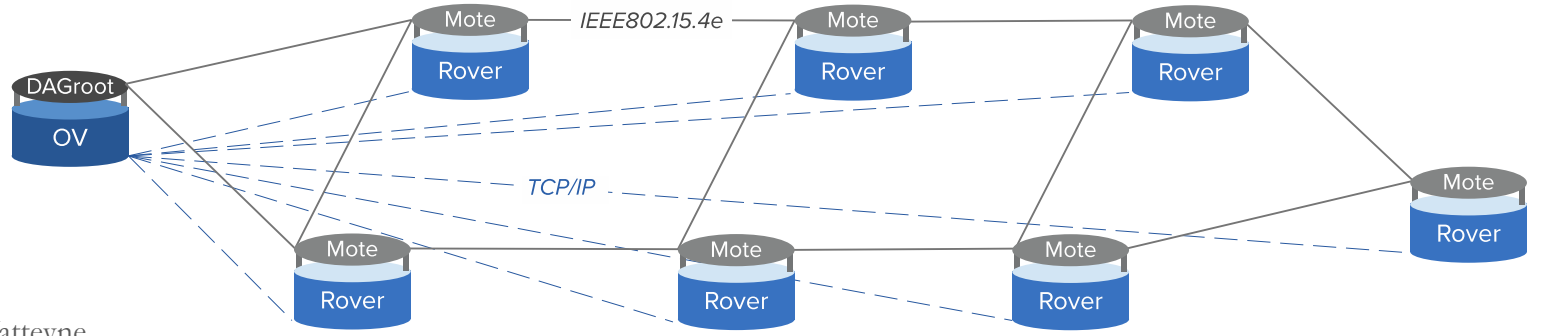


Figure 1: the testbed architecture

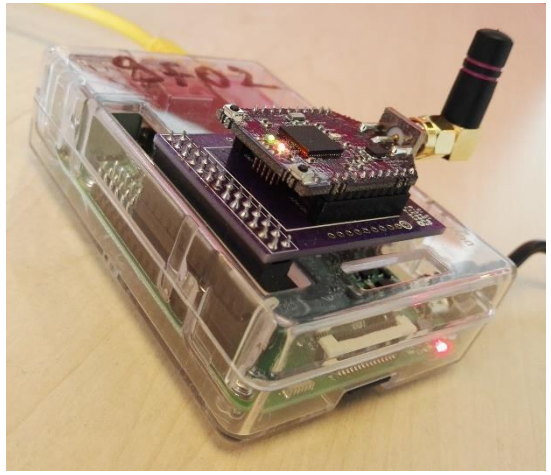


Figure 2: a Rover node

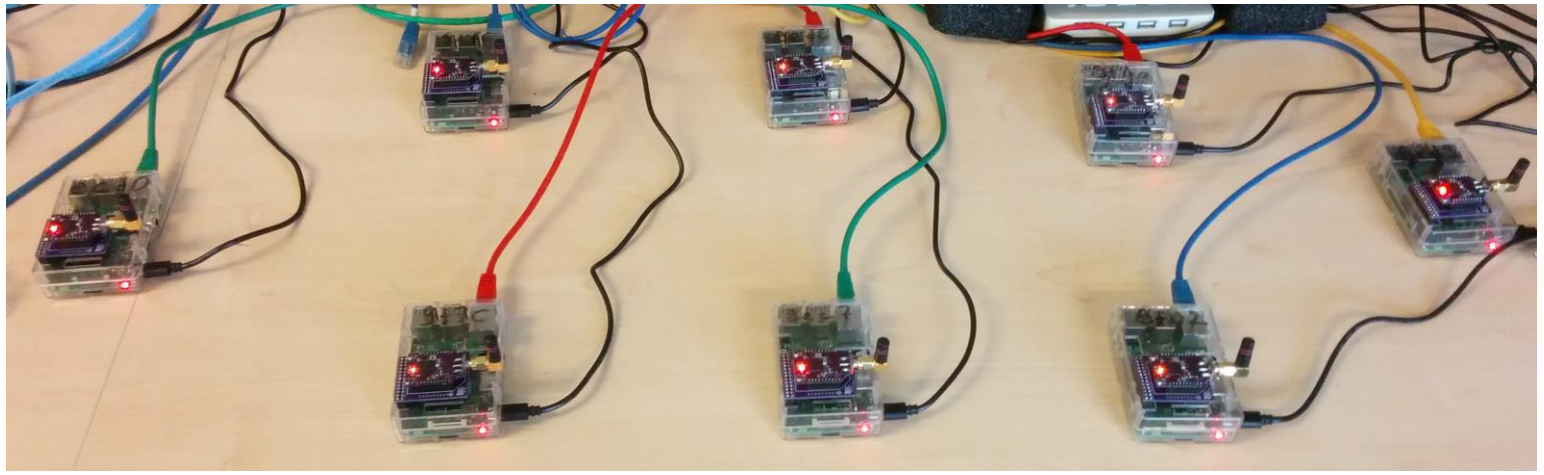


Figure 3: a local view of the testbed

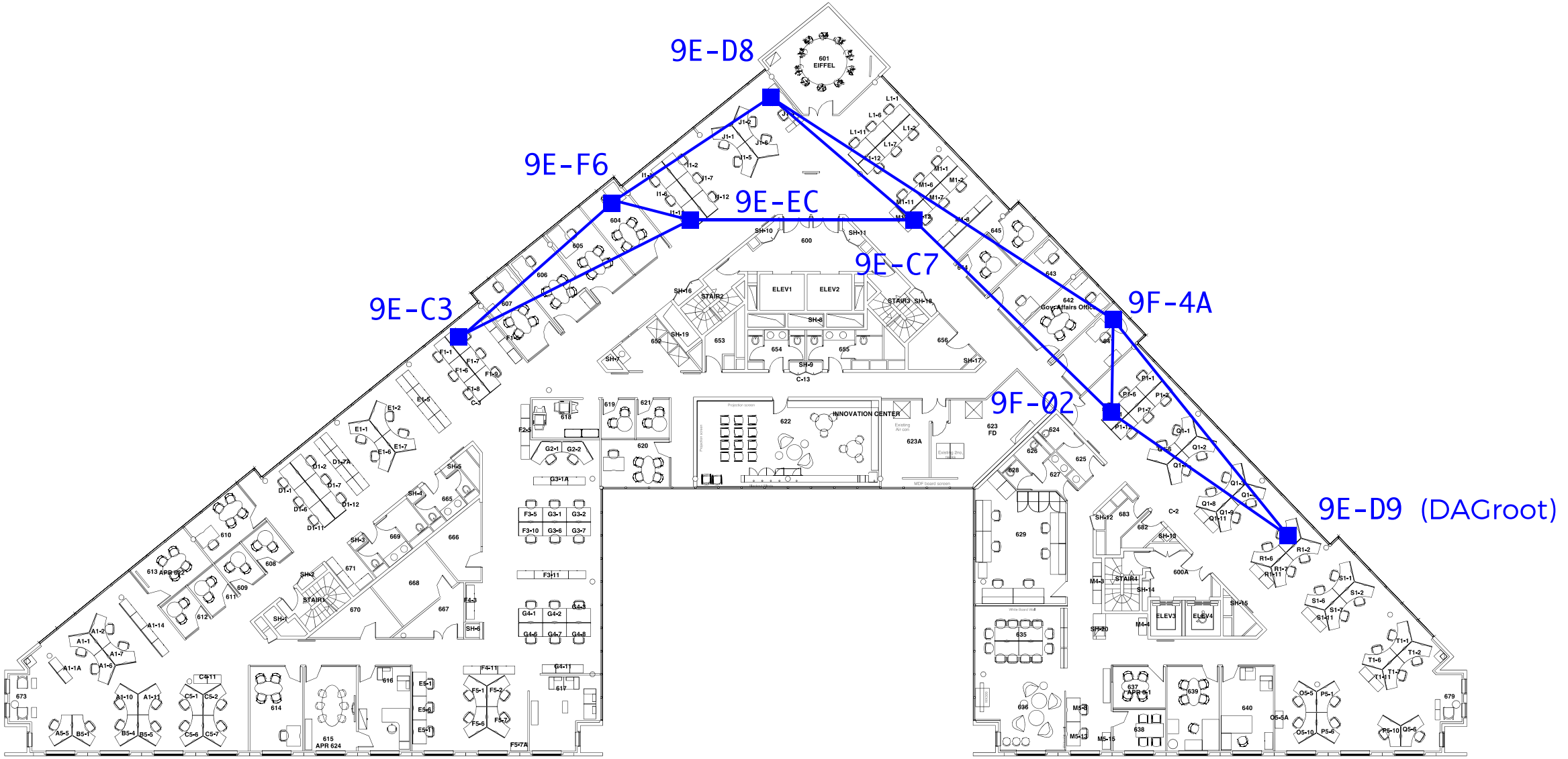


Figure 1: Deployment at Cisco PIRL (Paris ILM Office: 6th Floor)

	Low Latency Multipath without Retries					Low Latency Multipath with Retries								
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	A --> B,C					A --> B,C								
2		B --> D					A --> B,C							
3		C --> E						B --> D						
4			D --> F					C --> E						
5			E --> G						B --> D					
6				F --> H					C --> E					
7					G --> H					D --> F				
8										E --> G				
9											D --> F			
10											E --> G			
11												F --> H		
12													F --> H	
13														G --> H
14														G --> H
15														

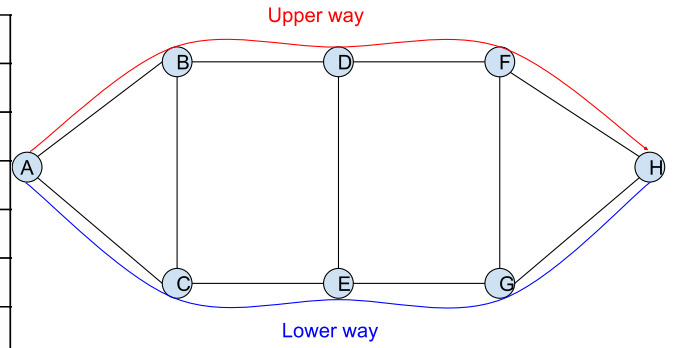
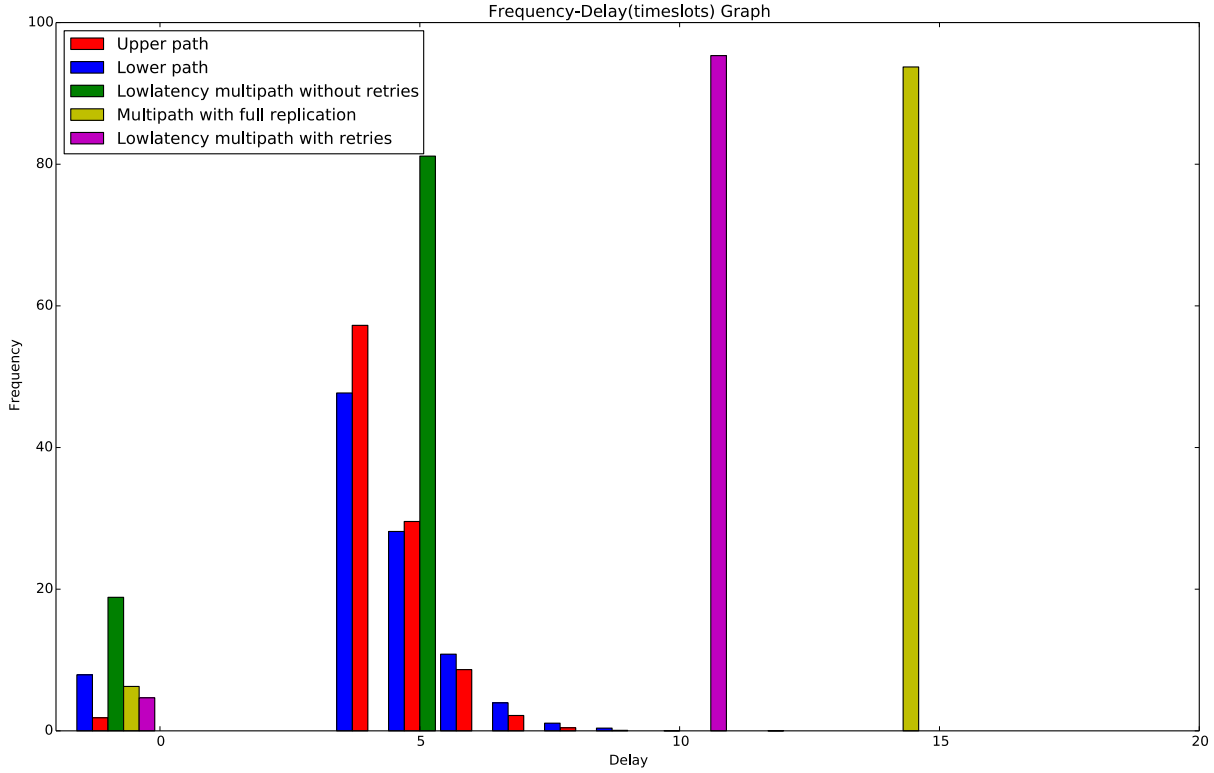


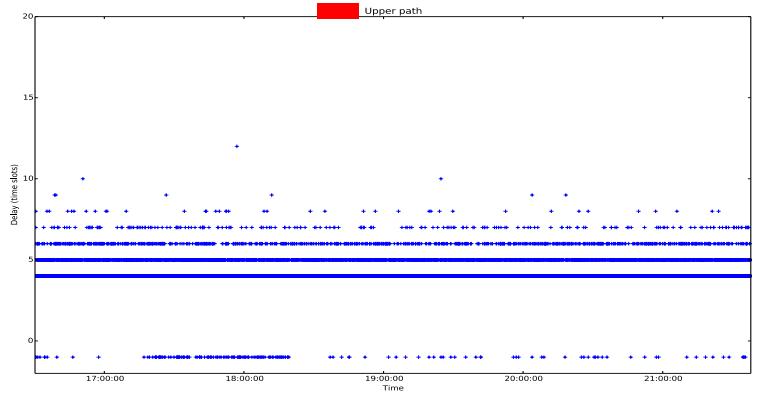
Figure 1: experiment tracks with different scheduling patterns

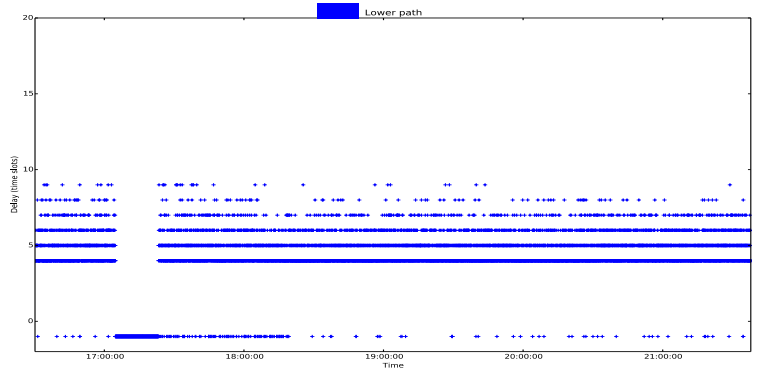
TSCH Schedule:

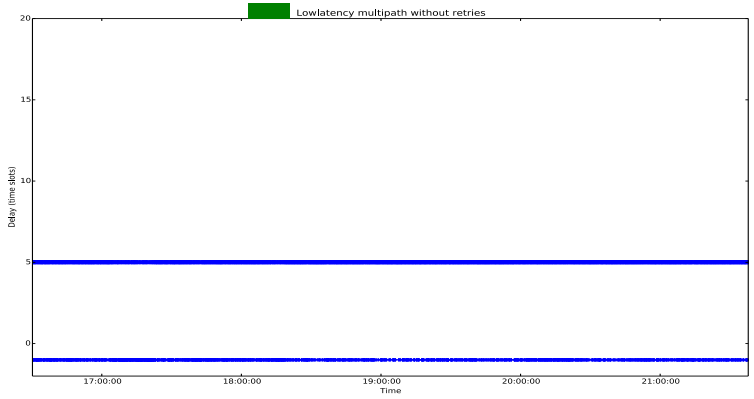
Low latency multipath with retries	Low latency multipath without retries	Multipath with full replication	Upper path with retries	Lower path with retries

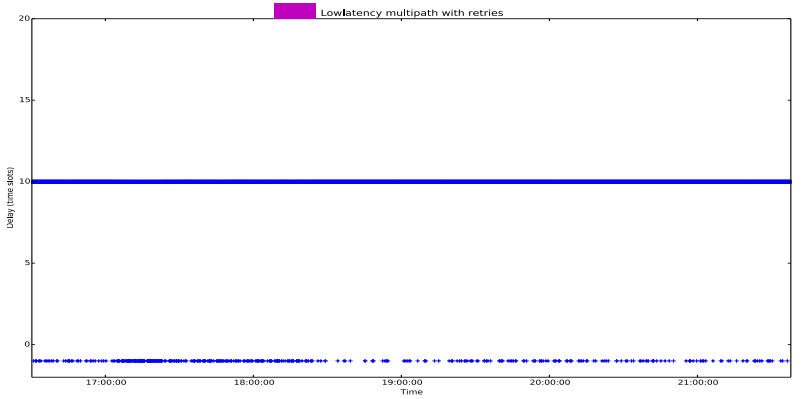
- Loss
- Burstness
- Jitter
- Delay
- Energy
- Reliability

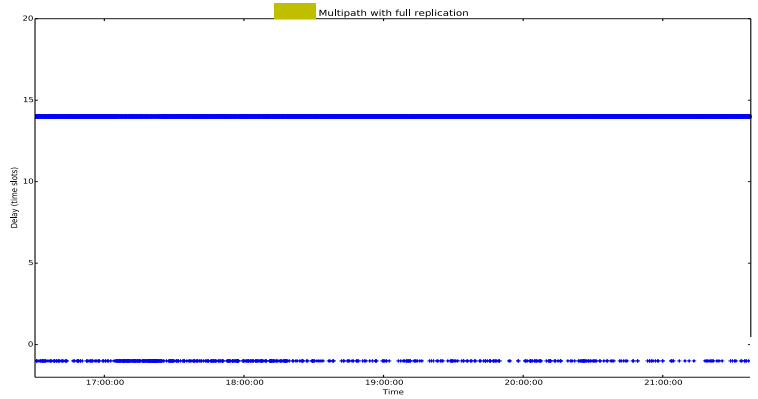












Conclusion

✓ **Multipath** improves delivery by providing additional reliability with spatial diversity.

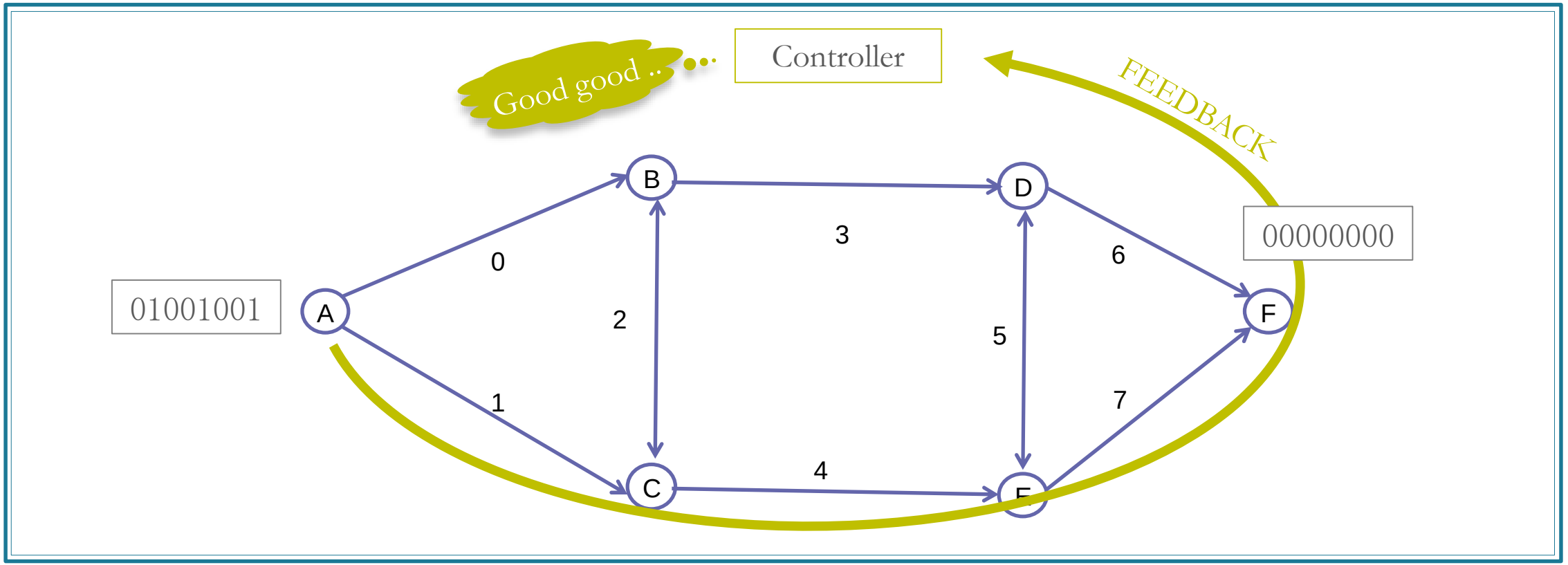
- Multipath reduces burst losses;
- Multipath overcomes node or link failures;
- Multipath introduces no jitter and bounded latency by design;
- Parallel concurrent multipath increases reliability without introducing additional delay;
- Packet replication and elimination introduces delay and more energy consumption.

✓ **Retries** provide more efficient packet delivery with some limitations.

- Retries are highly valuable only with channel hopping;
- Stochastic retries introduce delay and jitter;
- Retries are less immune to burst losses: if a transmission failed, using retries has greater chance to fail again.

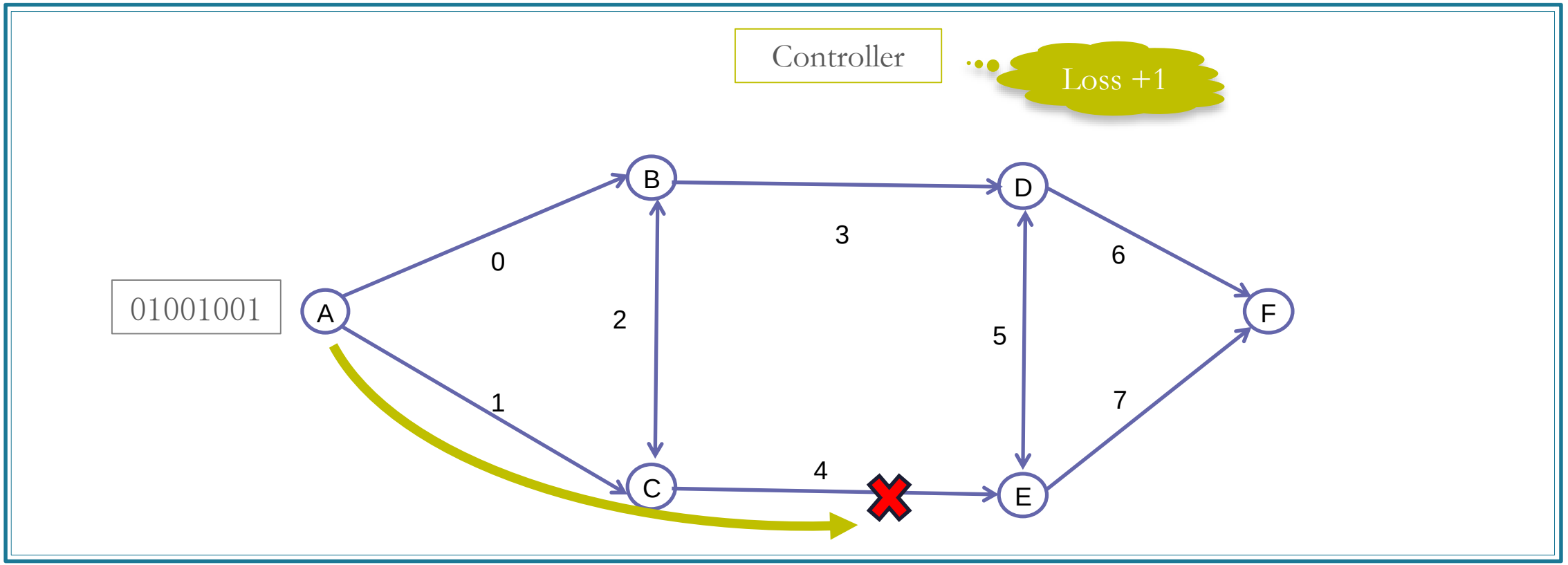
a single path with retries is used for efficient packet delivery and less power consumption, and the multipath mechanism is enabled only when necessary to provide higher reliability and eliminate burst losses.

BitString Control Loop



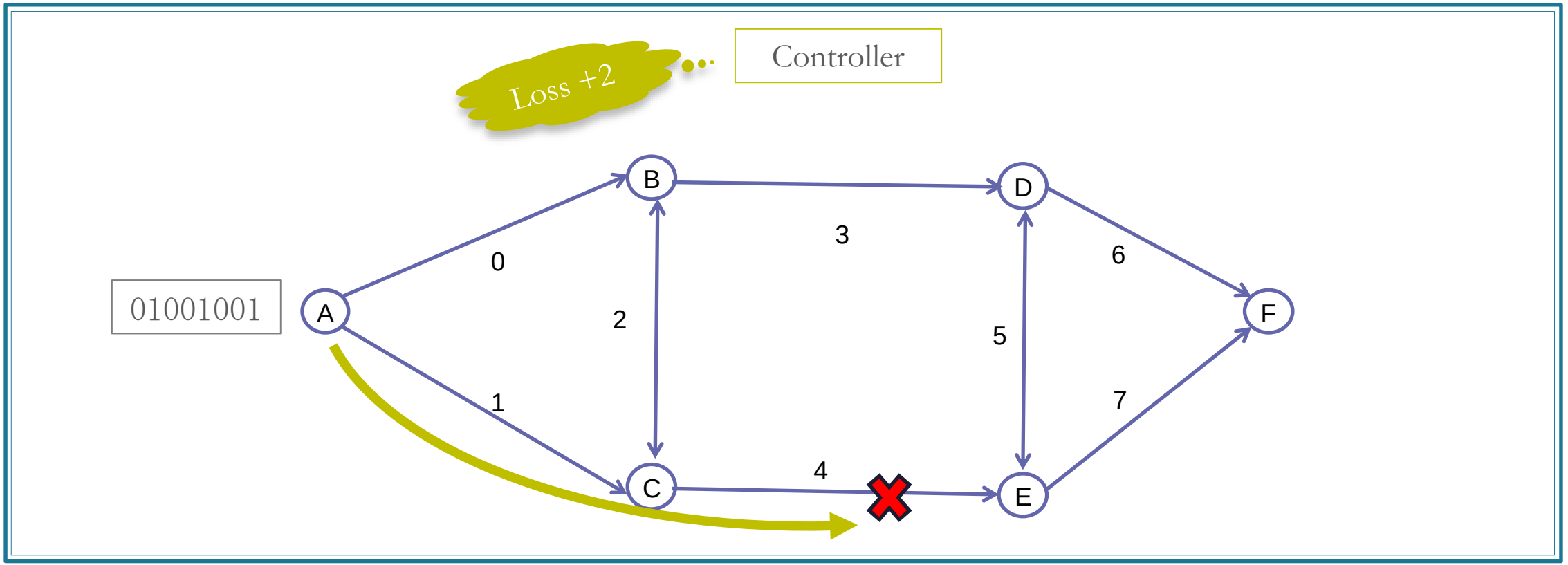
a single path with retries is used for efficient packet delivery and less power consumption, and the multipath mechanism is enabled only when necessary to provide higher reliability and eliminate burst losses.

BitString Control Loop



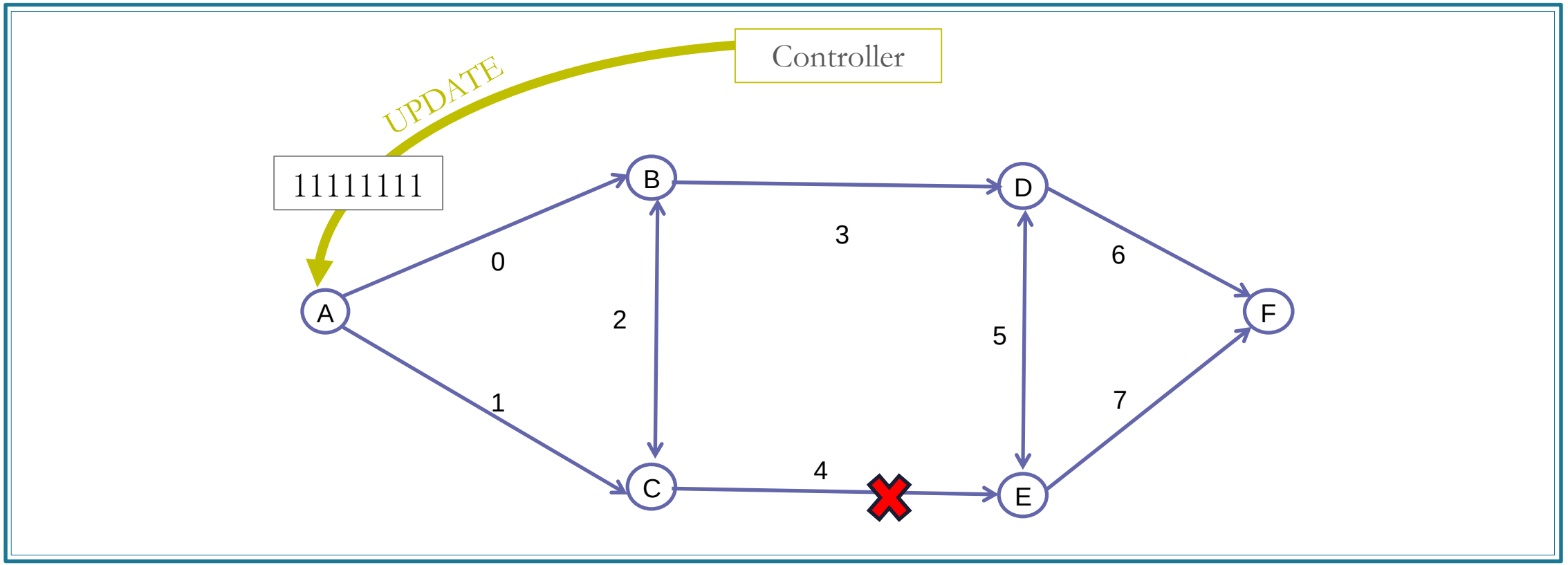
a single path with retries is used for efficient packet delivery and less power consumption, and the multipath mechanism is enabled only when necessary to provide higher reliability and eliminate burst losses.

BitString Control Loop



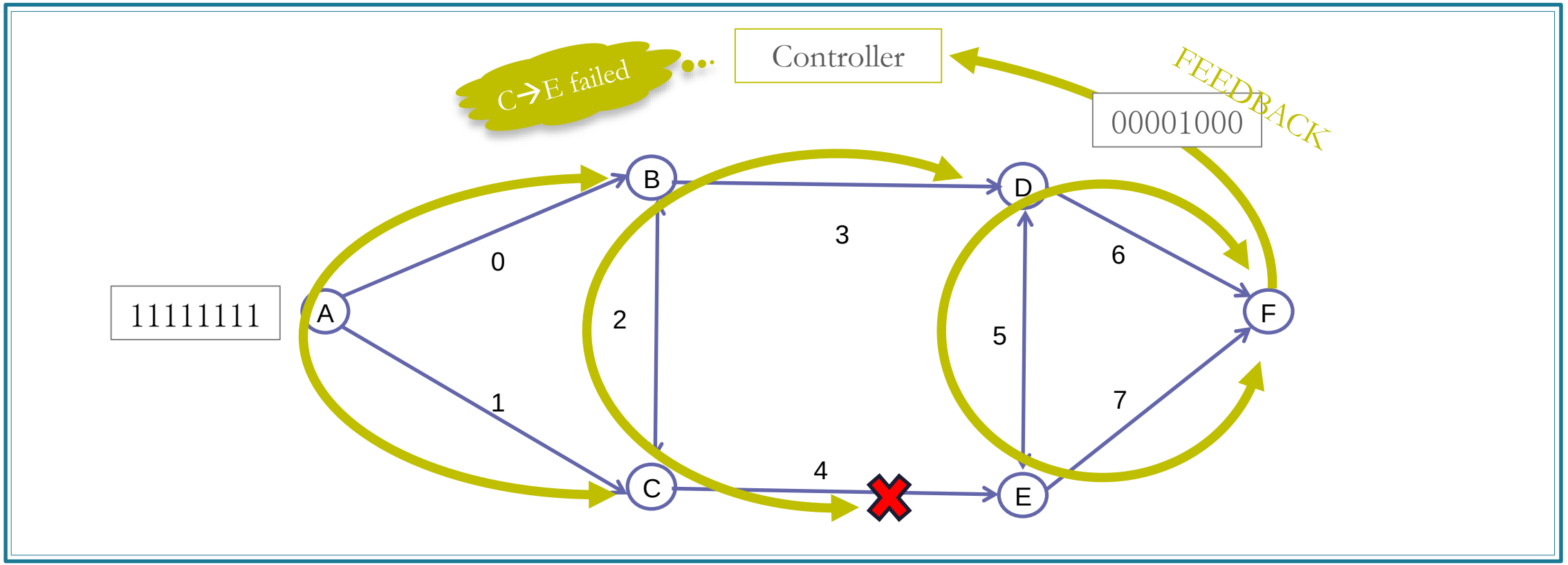
a single path with retries is used for efficient packet delivery and less power consumption, and the multipath mechanism is enabled only when necessary to provide higher reliability and eliminate burst losses.

BitString Control Loop



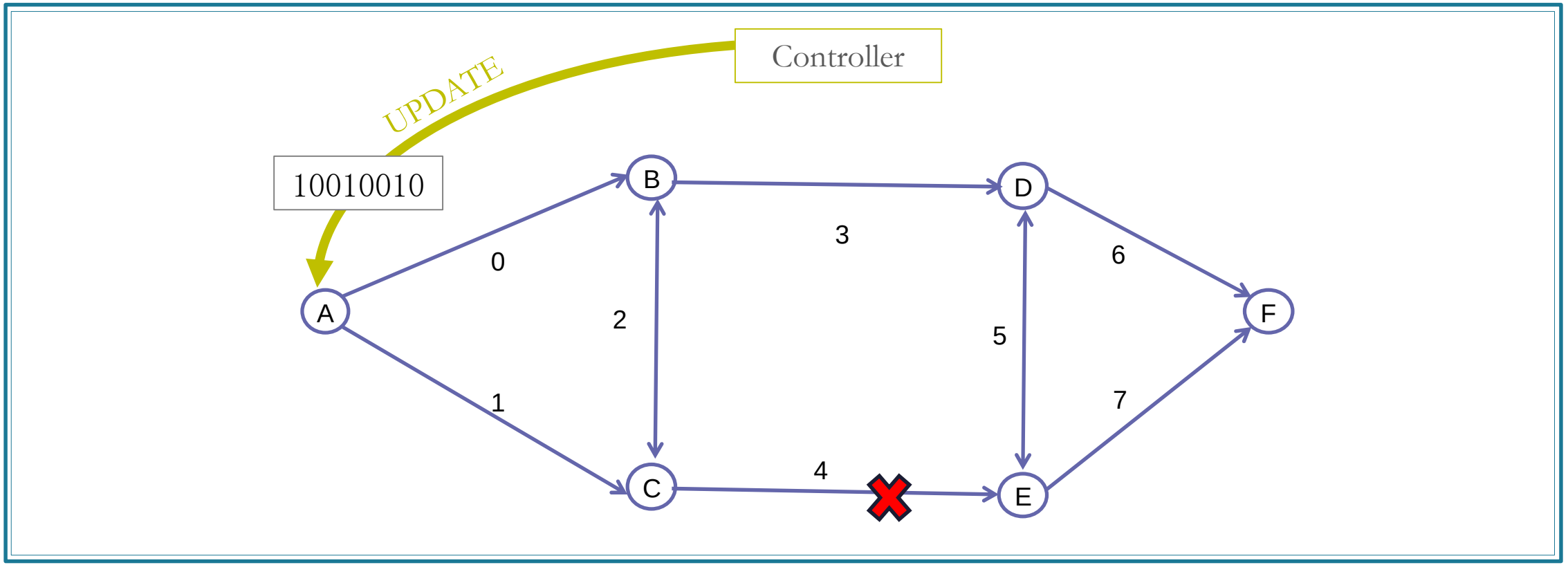
a single path with retries is used for efficient packet delivery and less power consumption, and the multipath mechanism is enabled only when necessary to provide higher reliability and eliminate burst losses.

BitString Control Loop



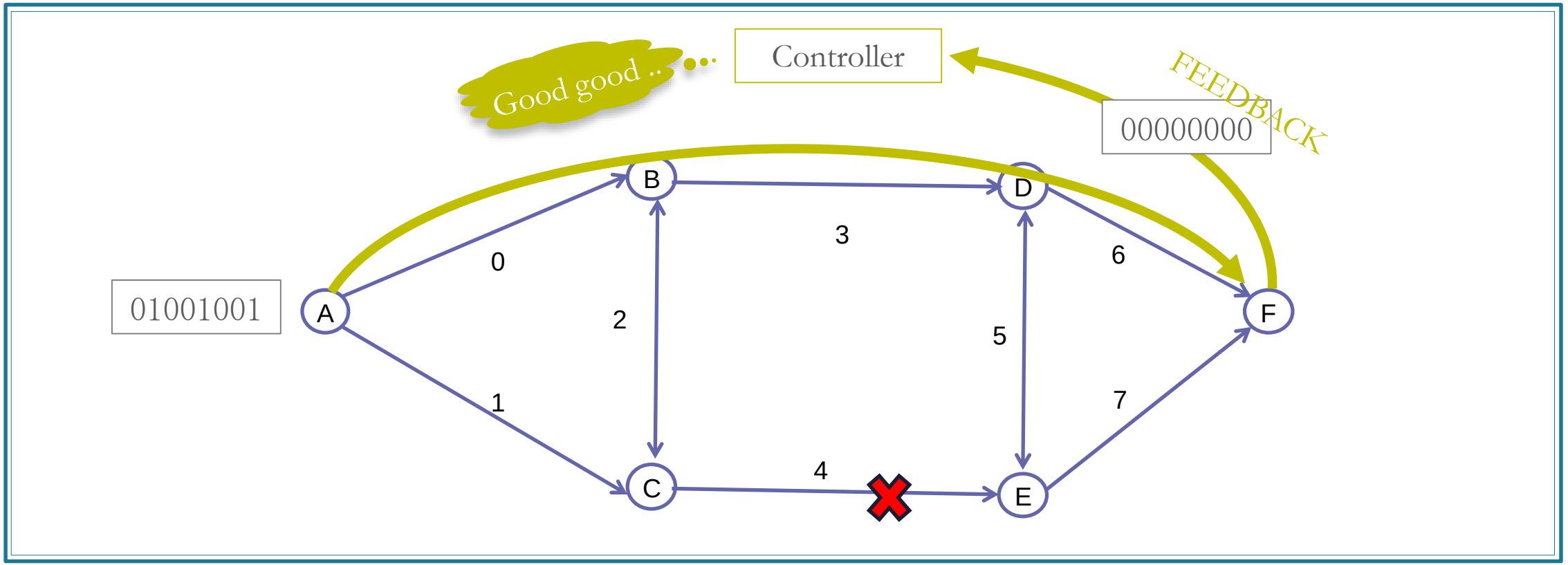
a single path with retries is used for efficient packet delivery and less power consumption, and the multipath mechanism is enabled only when necessary to provide higher reliability and eliminate burst losses.

BitString Control Loop

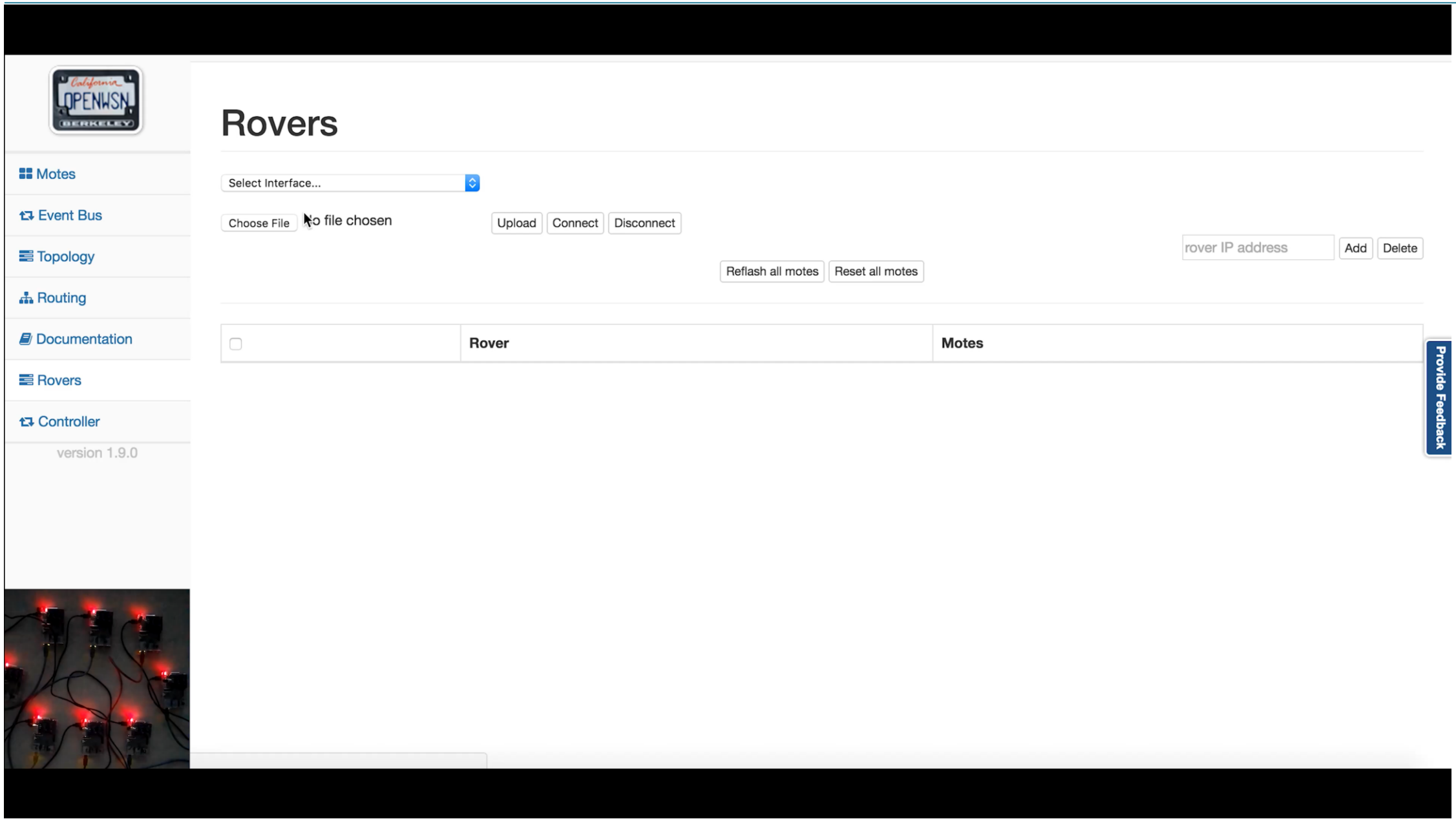


a single path with retries is used for efficient packet delivery and less power consumption, and the multipath mechanism is enabled only when necessary to provide higher reliability and eliminate burst losses.

BitString Control Loop



Dynamic Multipath Control



Control loop experiments during nighttime

Packet Loss

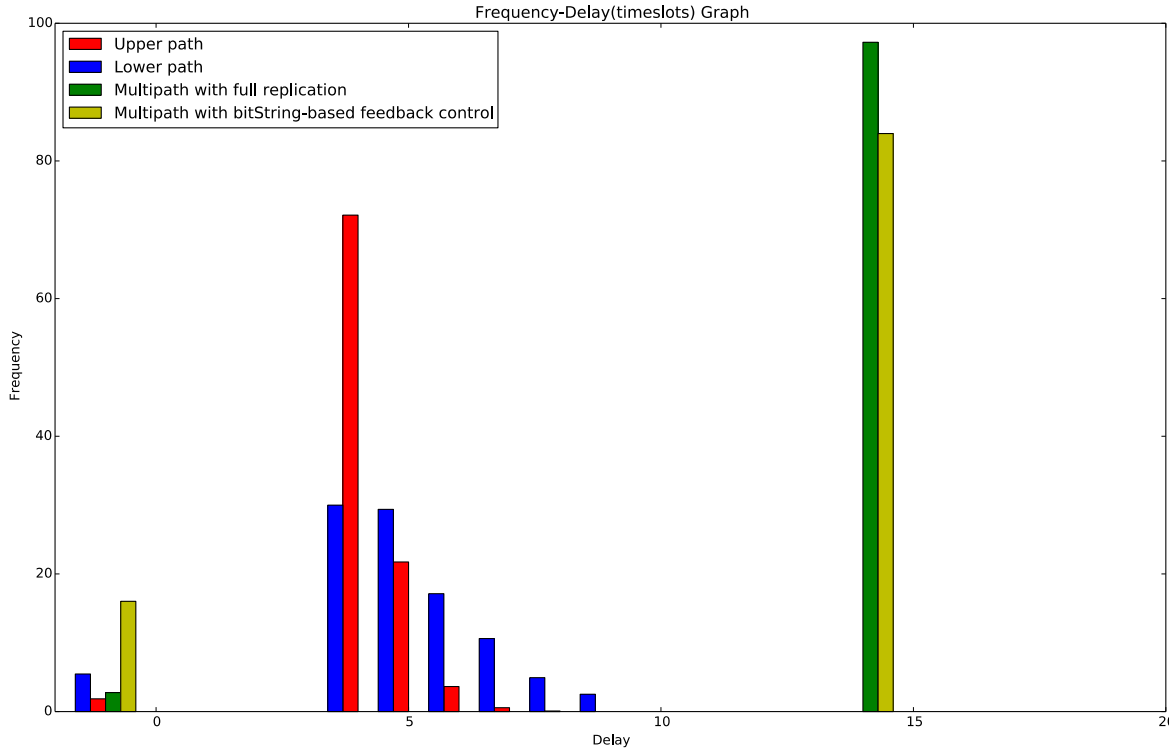


Figure 1: Receiving delays (in Time Slots)
frequency (% of messages sent)

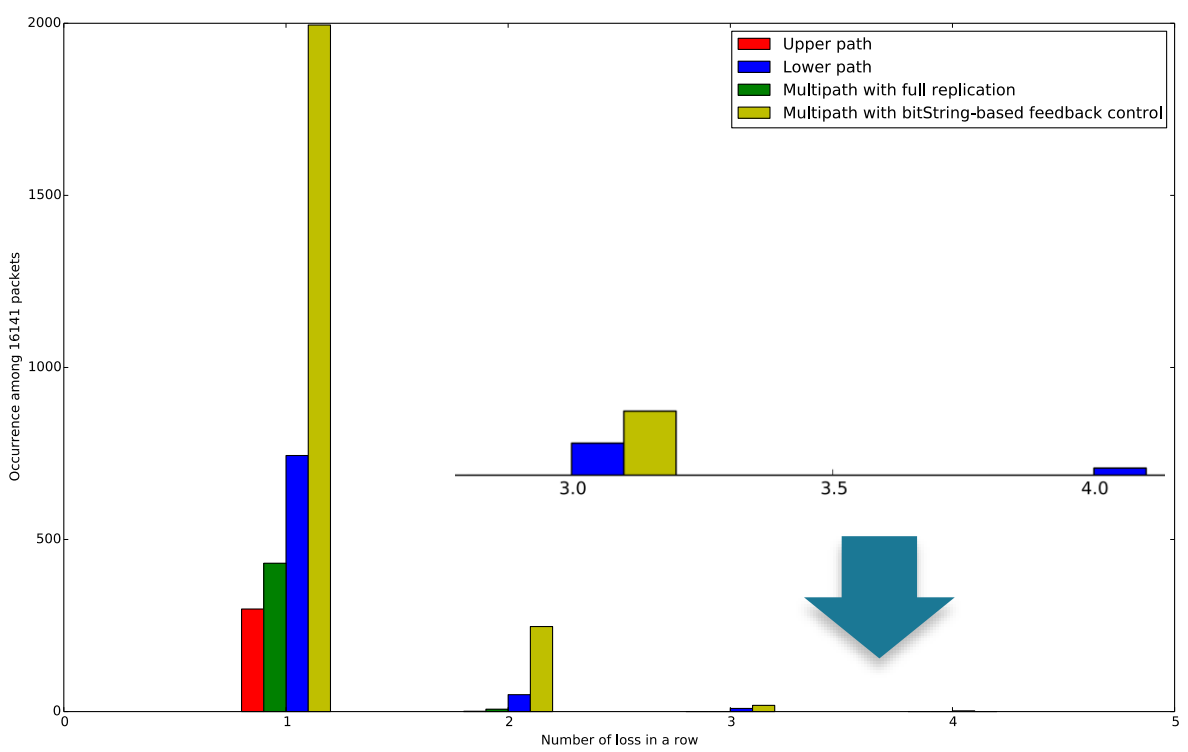


Figure 2: Burst losses with different lengths (in the number of packets)
occurrence among 16141 messages sent

Control loop experiments during nighttime

Energy Consumption

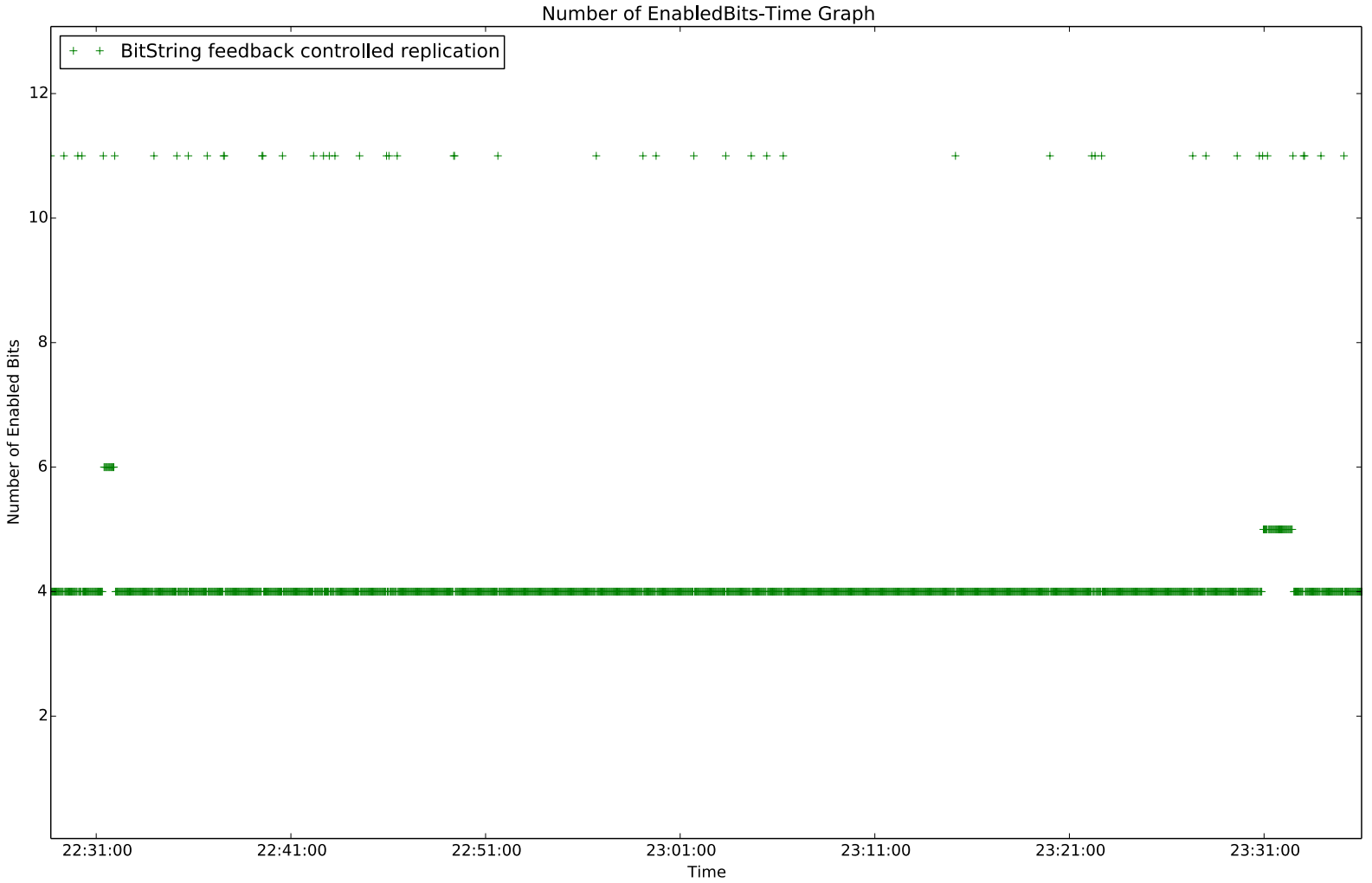
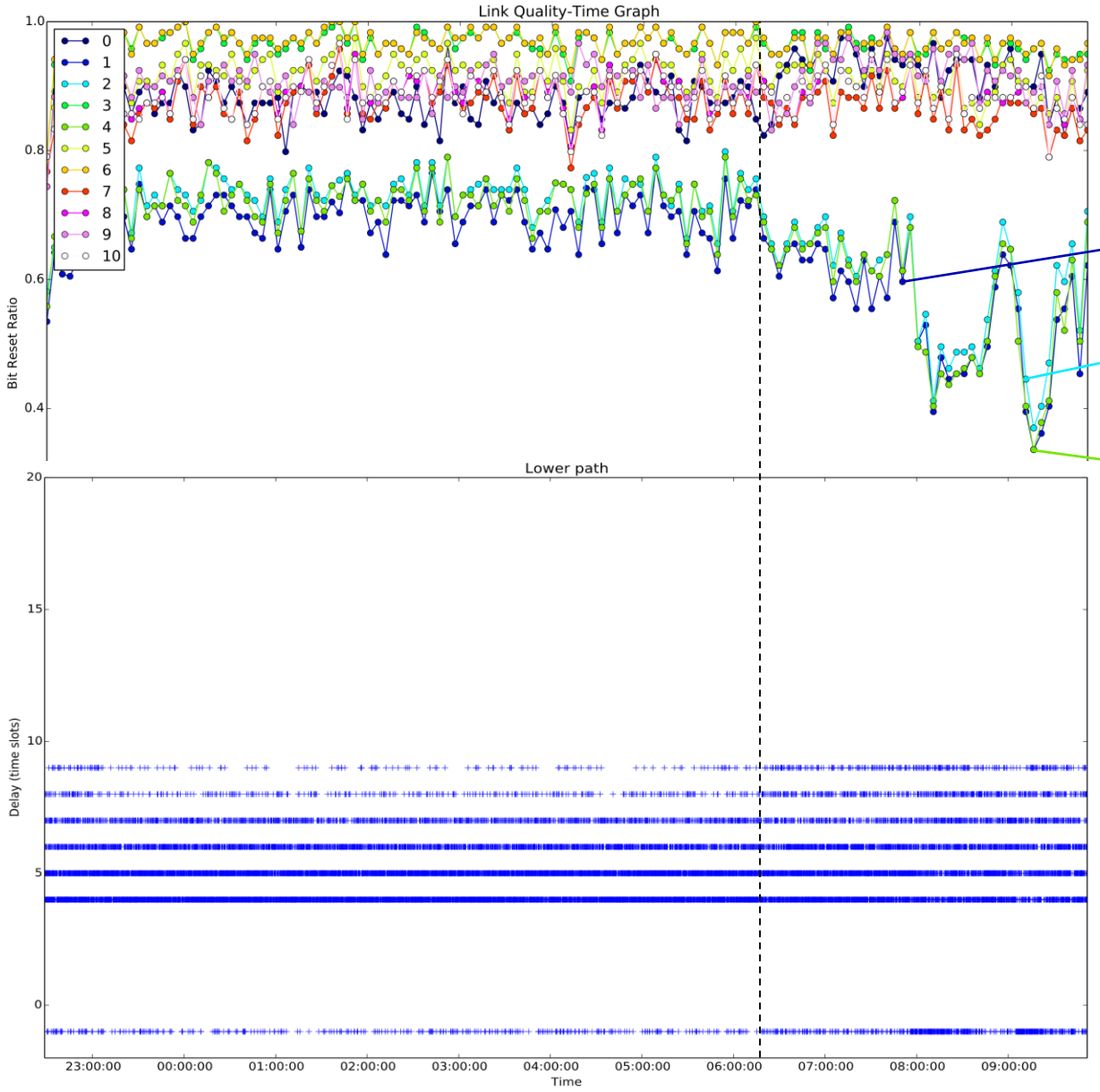
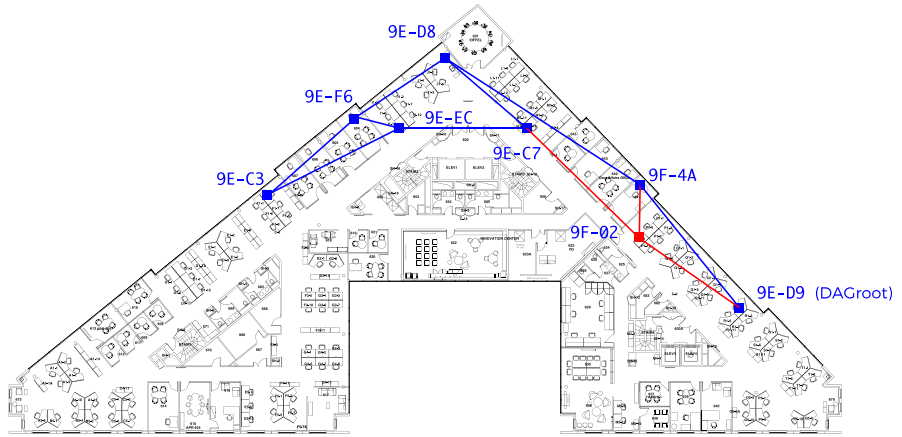


Figure 1: Number of enabled bits over time in an hour

Control loop experiments during nighttime



	15	16	17	18	19	20	21	22
0	SERIAL	SHARED	9ed9-0>9f4a					
1				9ed9-1>9f02				
2					9f4a-2>9f02			
3						9f02-2>9f4a		
4							9f4a-3>9ed8	
5								9f4a-3>9ed8
6							9f02-4>9ec7	
7								9f02-4>9ec7



Control loop experiments during daytime

Packet Loss

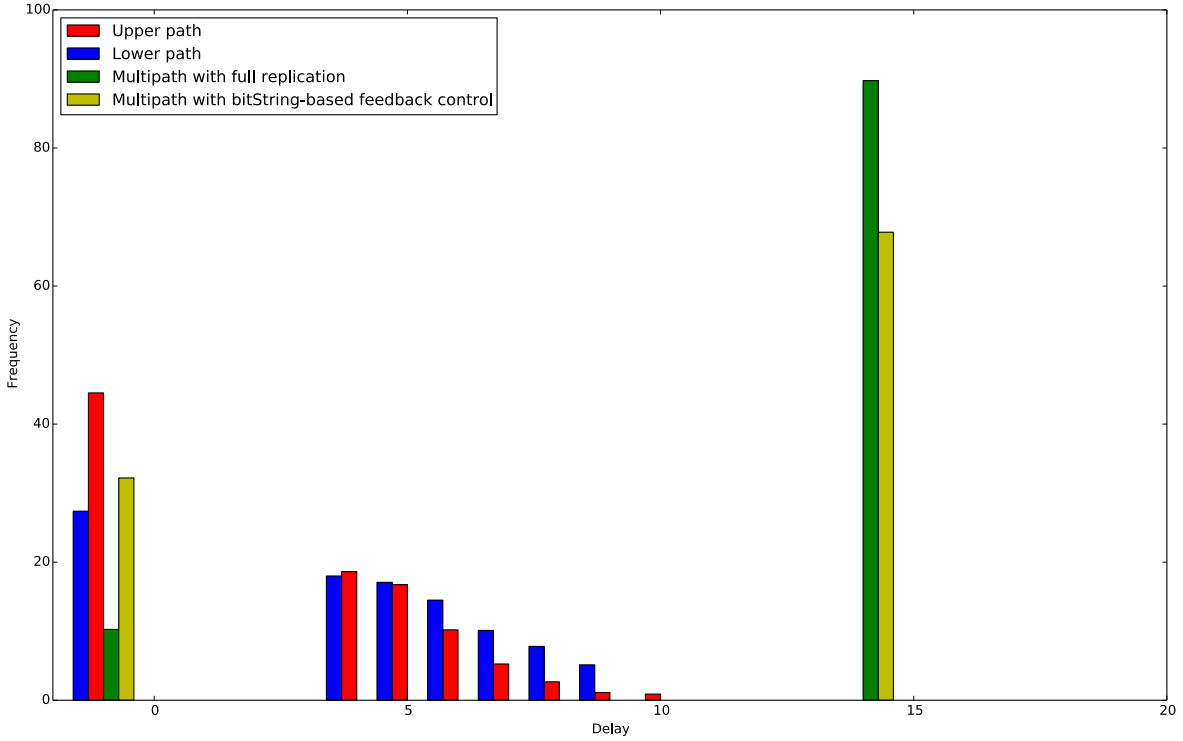


Figure 1: Receiving delays (in Time Slots)
frequency (% of messages sent)

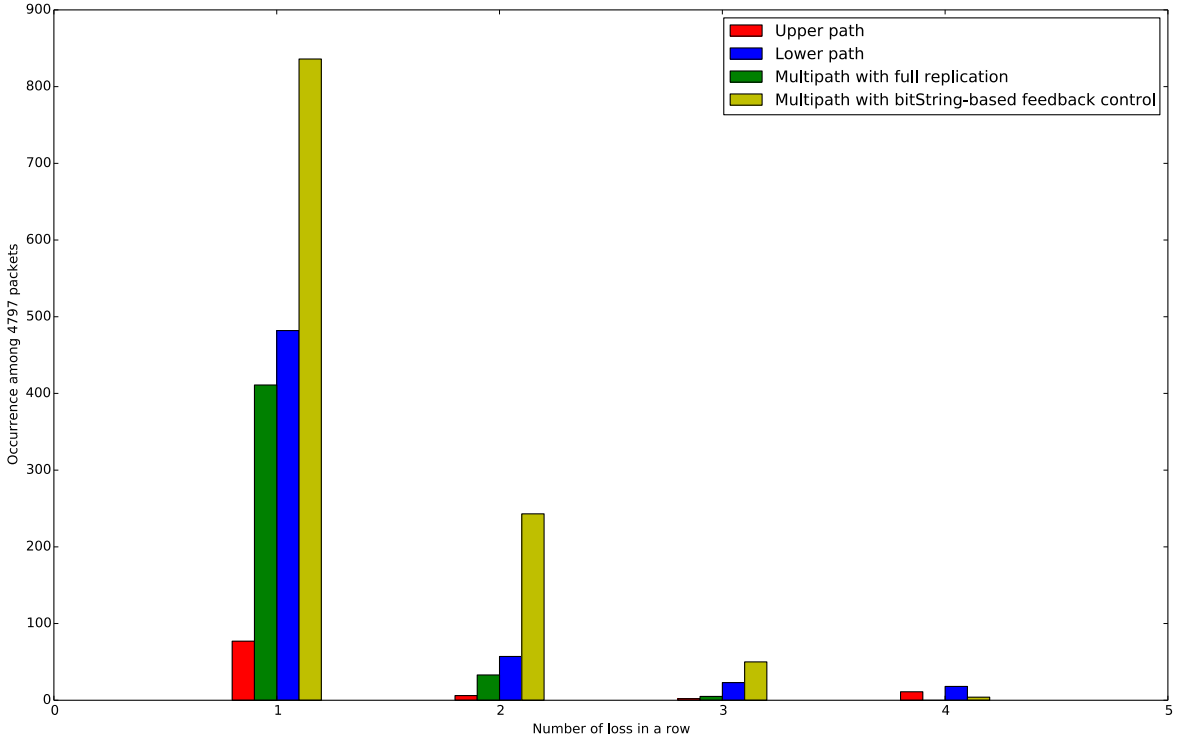


Figure 2: Burst losses with different lengths (in the number of packets)
occurrence among 4797 messages sent

Control loop experiments during daytime

Energy Consumption

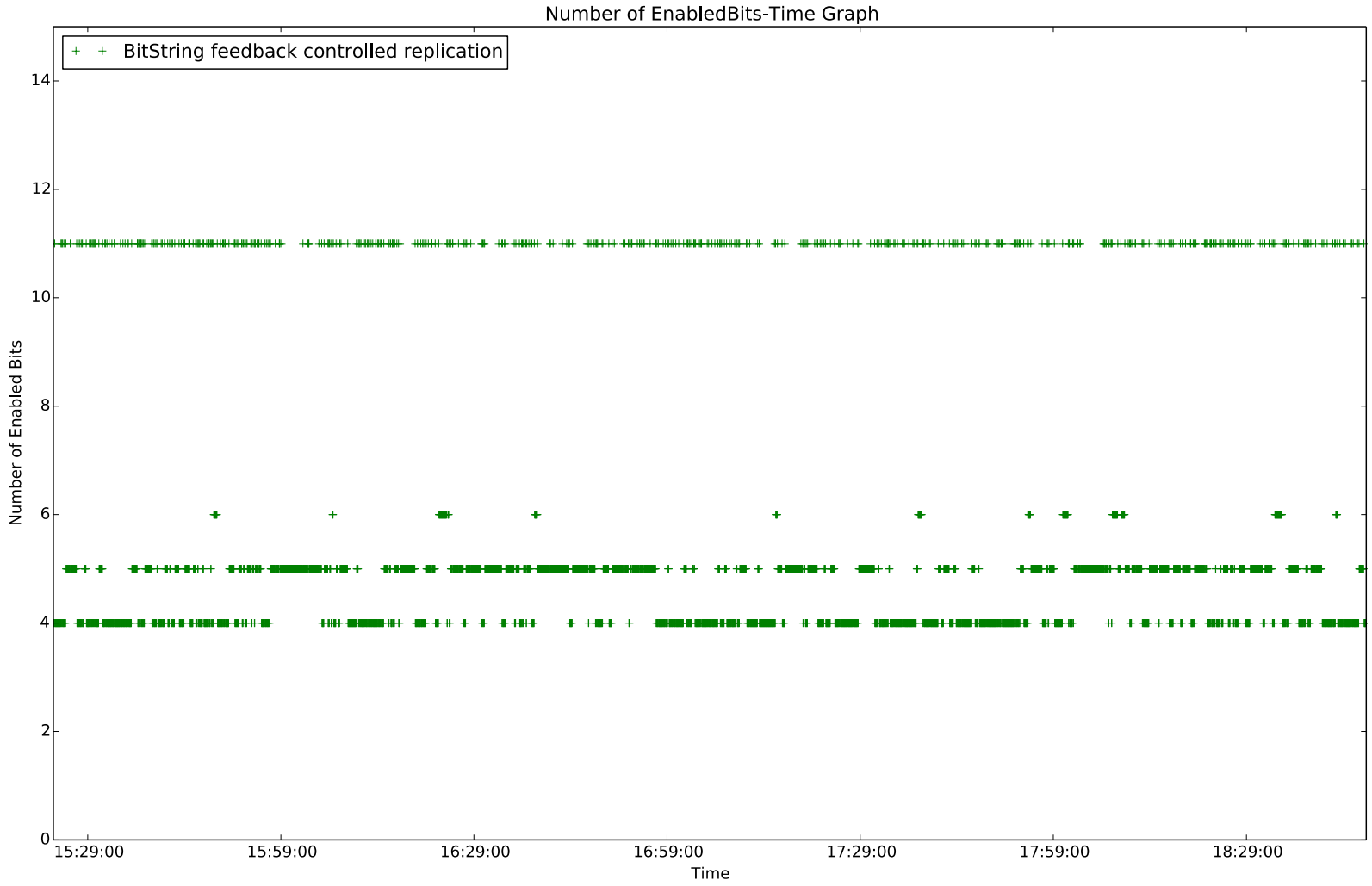
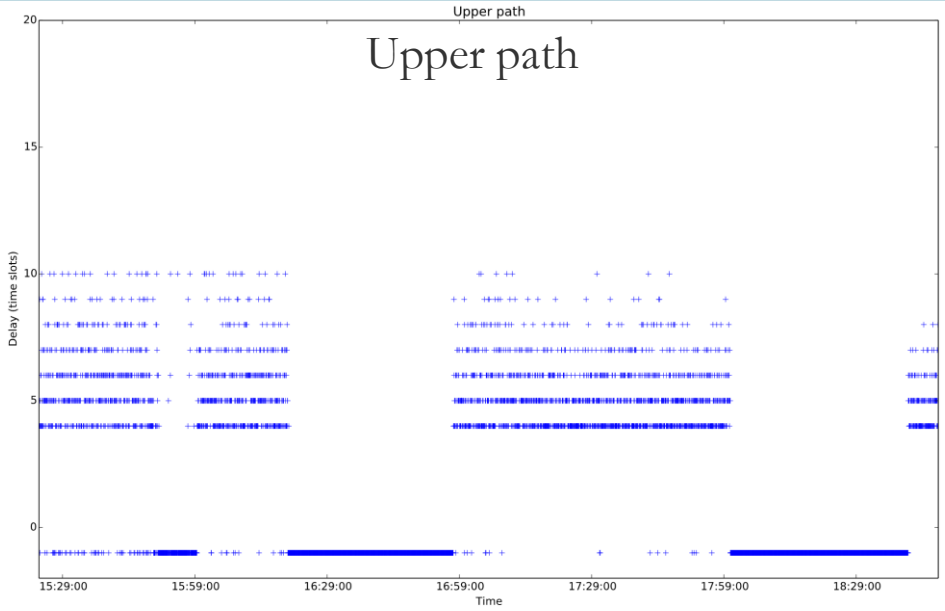
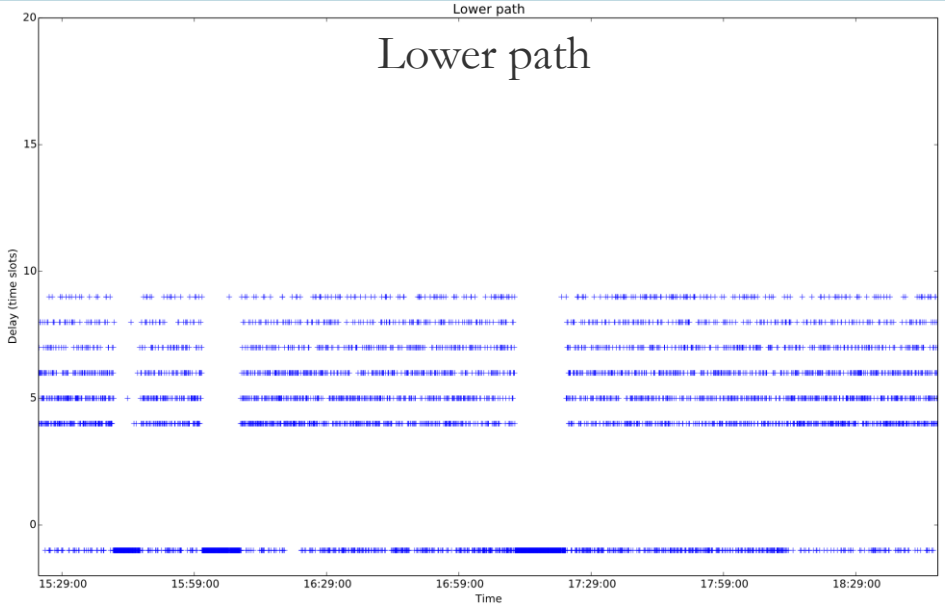
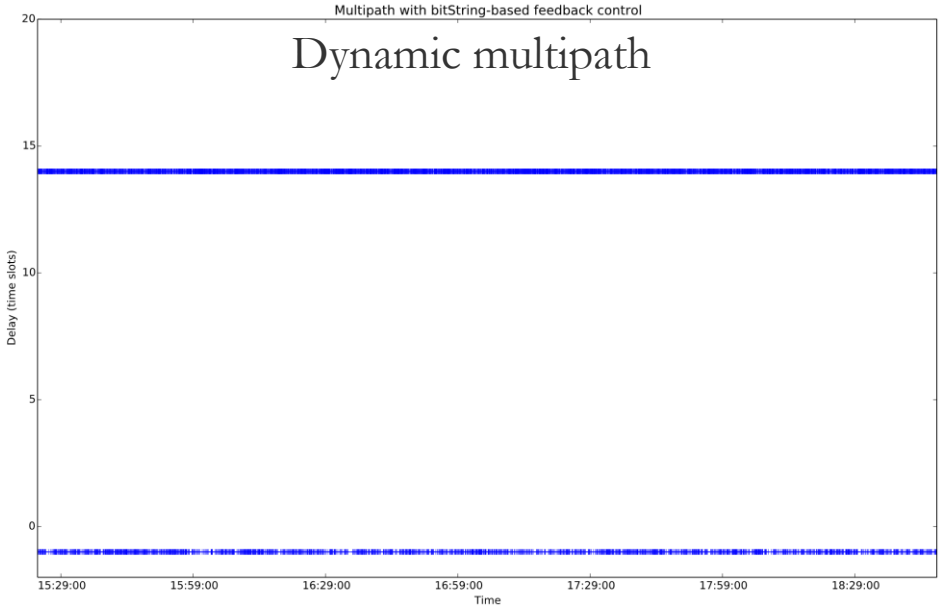


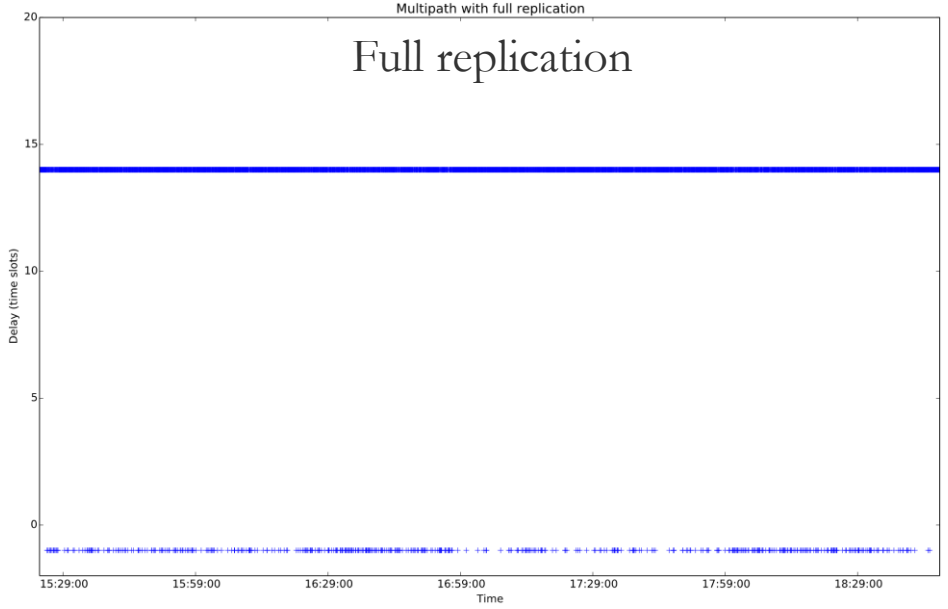
Figure 1: Number of enabled bits over time

Control loop experiments during daytime









Conclusion

Summary

- **Centralized operations**: Topology learning from RPL, Track computation and reservation based on ARC, etc.
- **Testbed setup**: OpenWSN open-source code, OpenMote hardware, Raspberry Pi 3
- **Multipath experiments**: TSCH retries vs. multipath
- **Control loop**

Constraints

- Static in-building testbed topology
- Limited number and pattern of experiments
- ARC construction order issue (to be solved)
- Control loop schedule is not optimized
- Hardly reproducible

Future Work

- Dynamic scheduling

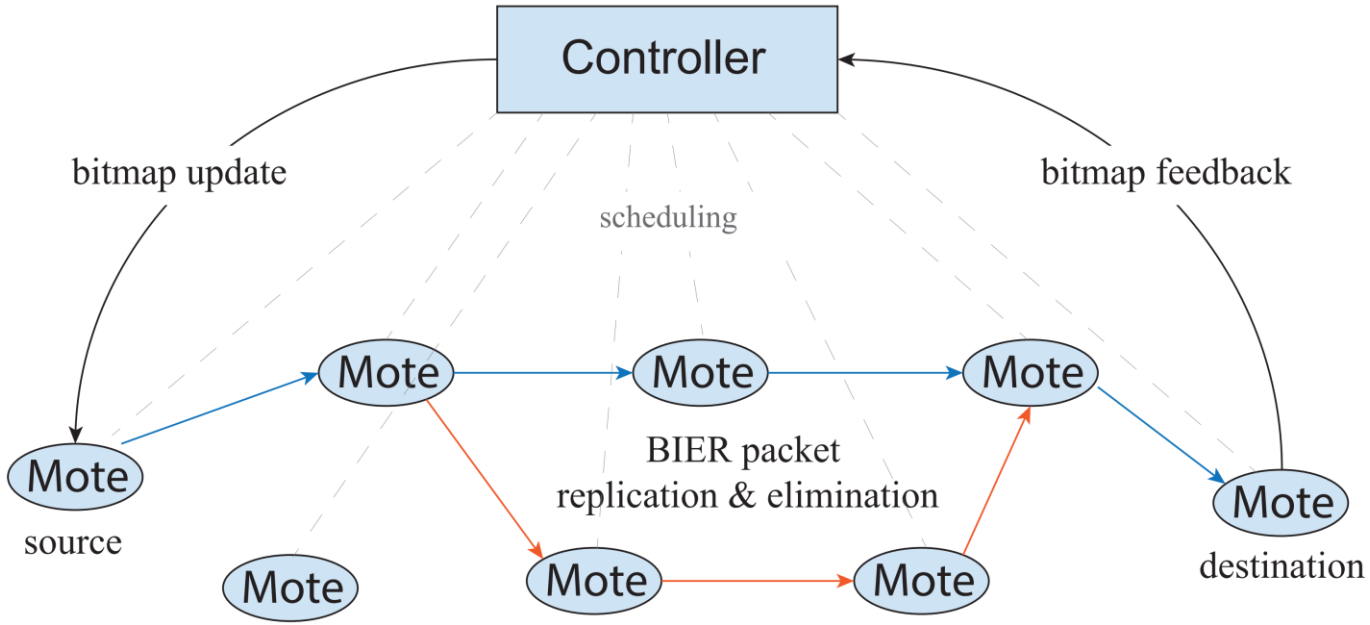


Figure 1: BIER-TE Multipath Control

Accomplishments

[1] P. Thubert, Z. Brodard, **H. Jiang**, G. Texier. IETF Draft. "A 6loRH for BitStrings". Current version: draft-thubert-6lo-bier-dispatch-01. Available online: https://tools.ietf.org/html/draft-thubert-6lo-bier-dispatch-01. June 29, 2016.

[2] P. Thubert, Z. Brodard, **H. Jiang**. "TIMESLOT SHIFTING FOR LOWER-PRIORITY PACKET IN A TIME SLOTTED NETWORK". Filed 12-Jul-2016. U.S. Patent.

[3] Z. Brodard, **H. Jiang**, T.F. Chang, T. Watteyne, X. Vilajosana, P. Thubert, Géraldine Texier. "Rover: Poor (but Elegant) Man's Testbed". 13th ACM PE-WASUN 2016. Submitted September 04, 2016.

[4] P. Thubert, Z. Brodard, **H. Jiang**, T. Watteyne, T.F. Chang, T. Matsui, G. Z. Papadopoulos, G. Texier, N. Montavont. "Leapfrog Collaboration". Work in Progress.

[5] Z. Brodard, P. Thubert, **H. Jiang**, T. Watteyne, T.F. Chang, G. Texier, G. Z. Papadopoulos, N. Montavont. "Multipath Redundancy for Wireless Time-Slotted Networks". Work in Progress.

[6] P. Thubert, Z. Brodard**, H. Jiang**. "BIER-TE-based OAM, Replication and Elimination". Current version: draft-thubert-bier-replication-elimination-00. Available online: https://tools.ietf.org/html/draft-thubert-bier-replication-elimination-00. September 14, 2016.

Source code directories

- DAO compression firmware: github.com/Heriam/openwsn-fw/tree/rpl-path-diversity
- DAO compression dissector: github.com/Heriam/dissectors/tree/rpl-dao-compression
- DAO compression OV: github.com/Heriam/openwsn-sw/tree/rpl-dao-compression
- BIER-6LoRH dissector: github.com/Heriam/dissectors/blob/BIER/packet-6lowpan.c
- BIER-TE OpenVisualizer Controller: github.com/Heriam/openwsn-sw/tree/Controller
- BIER-TE Firmware: github.com/Heriam/openwsn-fw/blob/BIER
- OpenWSN-ready Raspberry Pi Rover distribution: github.com/zach-b/openroverpi

Thank you.

The 6LoRH for BitStrings

In order to include bitStrings in BIER packets, a new 6LoWPAN Routing Header (6LoRH) [34] for bitString, BIER-6LoRH, is defined and proposed in the IETF draft [35]. Figure 26 shows the format for this new header.

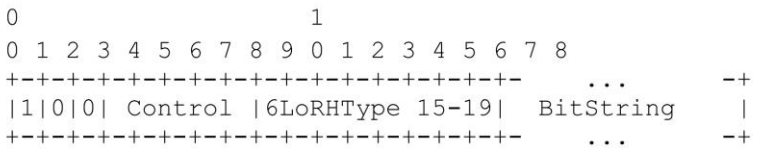


Figure 26 - The BIER-6LoRH

I have developed BIER-6LoRH dissecting features for the Wireshark dissector, which should be merged into OpenWSN project [36] soon.
The draft defines different types of the header as shown in Figure 27. The Type of a BIER-6LoRH header indicates the size of words used to build the BitString and whether the BitString is operated as an uncompressed bit-by-bit mapping, or as a Bloom filter. Note that the Group ID is used in the Control field for type 15 – 19 in order to allow shorter bitStrings by dividing a network into groups so that the BitString is locally significant to one group only.

Type	Encoding	Control field	BitString Size
15	bit-by-bit	Group ID	8 bits
16	bit-by-bit	Group ID	16 bits
17	bit-by-bit	Group ID	32 bits
18	bit-by-bit	Group ID	64 bits
19	bit-by-bit	Group ID	128 bits
20	Bloom filter	Hash function Set ID	8 bits
21	Bloom filter	Hash function Set ID	16 bits
22	Bloom filter	Hash function Set ID	32 bits
23	Bloom filter	Hash function Set ID	64 bits
24	Bloom filter	Hash function Set ID	128 bits

Figure 27 - The BIER-6LoRH Types

BIER-TE Forwarding Rules

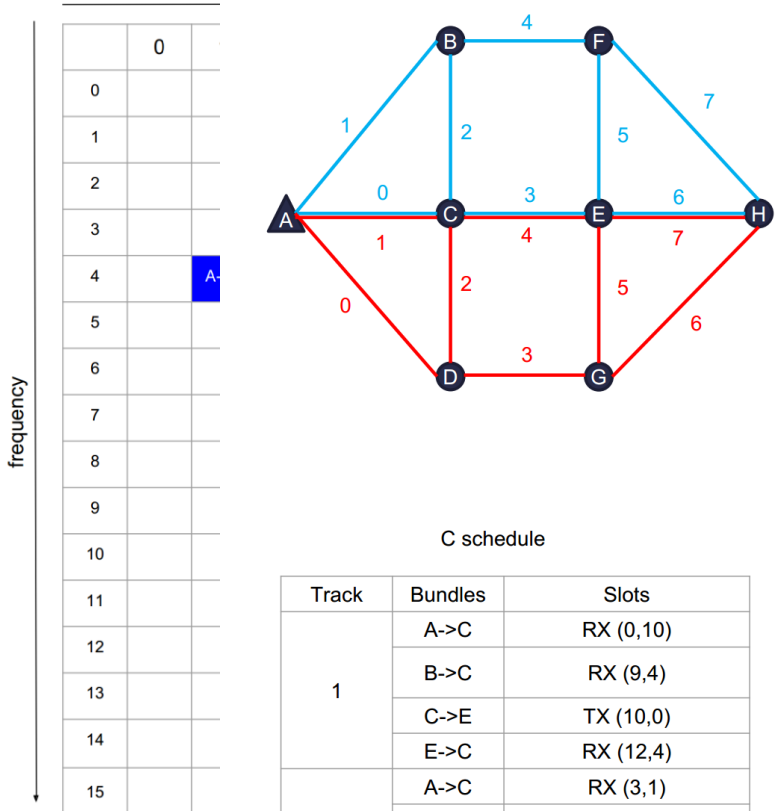
Upon the reception of a packet on a BIER-TE track RX slot, the node must:

1) Deduce the trackID of the packet from the RX slot information pre-installed by the PCE.
2) Check if a copy of this packet has already been received. If so, perform an AND operation on the bitmaps contained in the two packet headers and drop one of the packets.

On a BIER-TE track TX slot, the node does following operations:

1) If the transmission on the first slot of a TX bundle succeeds, the node would sleep on the rest slots of this bundle.
2) The node would sleep until next slot if no packet of the track associated to this slot exists in the buffer.
3) If there is a packet in buffer, check if the bitPosition associated with this slot is SET to '1' in the bitmap of the packet, if yes, duplicate the packet, reset the bit of the duplicate to '0' and transmit the copy, otherwise sleep until next slot.
4) The last slot of a TX bundle can choose not to require an ACK.

6TiSCH BIER-TE Data Structures



Blue Track : trackID 1

Bit Index	Bundles	Slots
0	A->C	(0,10)
1	A->B	(1,13)
	B->A	(5,10)
		(7,9)
2	B->C	(9,4)
3	C->E	(10,0)
	E->C	(12,4)
4	B->F	(3,2)
5	F->E	(14,1)
6	F->H	(16,3)
7	E->H	(18,5)

Red Track : trackID 2

Bit Index	Bundle	Slots
0	A->D	(2,7)
1	A->C	(3,1)
2	C->D	(4,4)
		(10,3)
	D->C	(11,7)
3	D->G	(12,8)
		(13,4)
4	C->E	(13,2)
5	G->E	(14,7)
6	G->H	(16,5)
7	E->H	(17,9)

Node C schedule

2	5	6	8
12	7	2	5
0xFFFF	0xFFFF	0xFFFF	0xFFFF
RX	RX	TX	TX
False	False	False	False
1	1	1	1
1	2	2	4
True	True	True	True

C schedule

Track	Bundles	Slots
1	A->C	RX (0,10)
	B->C	RX (9,4)
	C->E	TX (10,0)
	E->C	RX (12,4)
2	A->C	RX (3,1)
	C->D	TX (4,4)
		TX (10,3)
	D->C	RX (11,7)
	C->E	TX (13,2)

C track 1 BIFT

bitIndex	Bundle
3	C->E

C track 2 BIFT

bitIndex	Bundle
2	C->D
4	C->E

Node D schedule

slot Offset	7	8
channel Offset	10	5
Neighbor	0xFFFF	0xFFFF
Type	RX	RX
Shared	False	False
trackID	1	1
bitIndex	3	4
BIER-TE	True	True

led slot on a node

Figure 25 - an example of data structure [18]

Multipath Experiments Configurations

The two non-BIER-TE tracks are kept fixed in all the experiments. The packet in Non-BIER-TE Track 1 takes the upper path with hops A→B→D→F→H specified in its source routing header (6LoRH), while packets of Non-BIER-TE Track 2 take the lower path. For each of the two tracks:

- Each node would listen on all time slots of this track until a packet is received
- Try to forward the packet on all the followed timeslots until an ACK is received
- Each hop allows unlimited retries until the last timeslot of the track
- A total of 12 timeslots is assigned for each track

During the experiments, each sending comprises a number of test packets with exactly one packet for each track. The packets forwarded are UDP packets sent to port 1009 containing a payload of 4 bytes and the headers are compressed according to OpenWSN protocol stack (6LowPAN). Each timeslot has a duration of 15ms.

Parameter	Value
Slotframe duration in seconds	1.065
Sending interval in seconds	2
Number of packets in each sending	5
Packets sent for each track	6632
Experiment Duration	11:50 - 15:44

Table 3: Experiment Parameters

BIER-TE Track	Description
Multipath with full replication	Packets are replicated at every step (replication points) of the ladder
Low latency multipath without	Use snooping and concurrent transmissions to minimize the latency
Low latency multipath with	Provide a retry for each hop of the low latency multipath

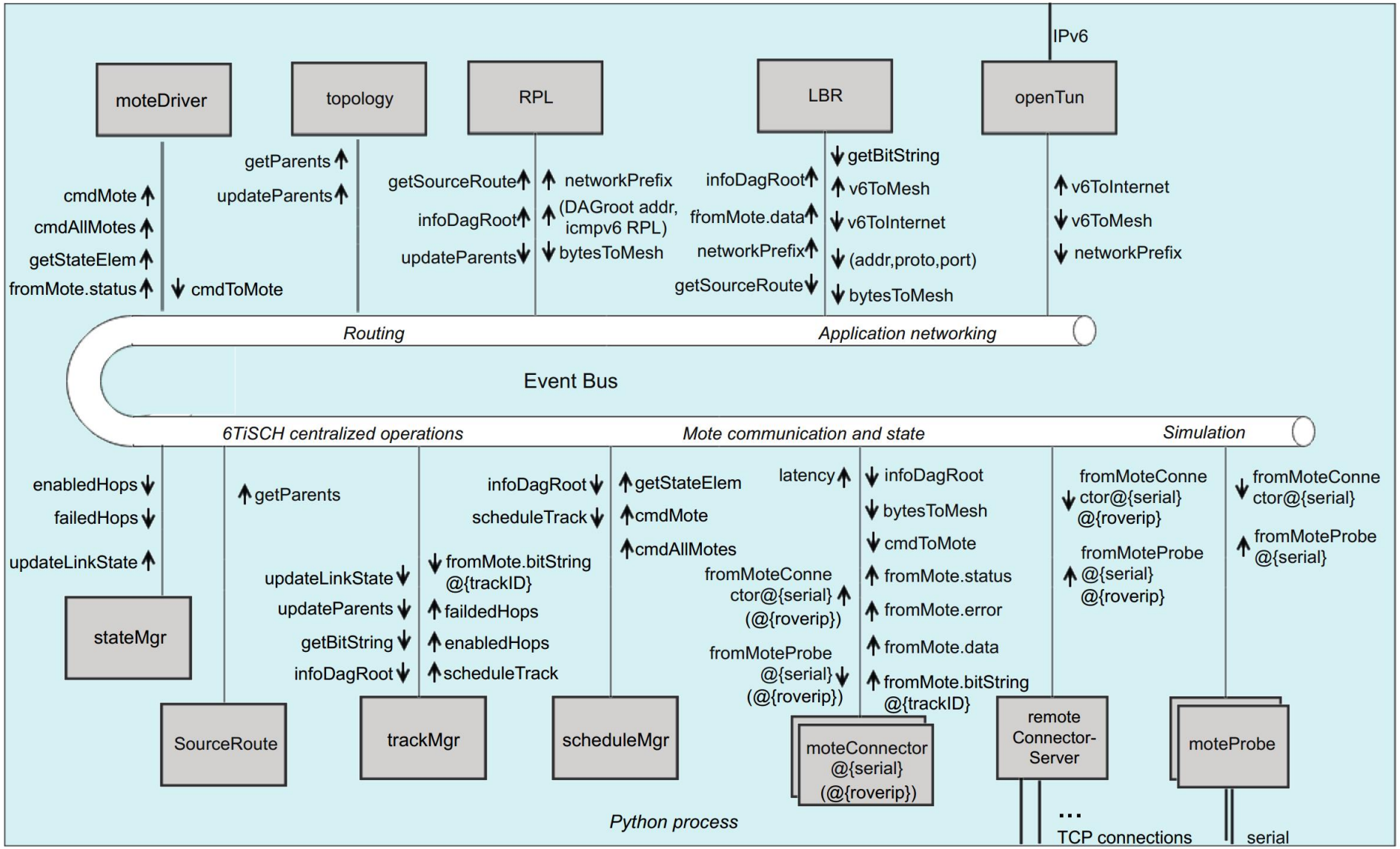
Table 1: BIER-TE Track Description

Usage	Number of Slots
Upper path	12
Lower path	12
Multipath with full replication	14
Low latency multipath without retries	5
Low latency multipath with retries	10
Shared slots for synchronization and RPL	3
Serial communication	15
Total Size	71

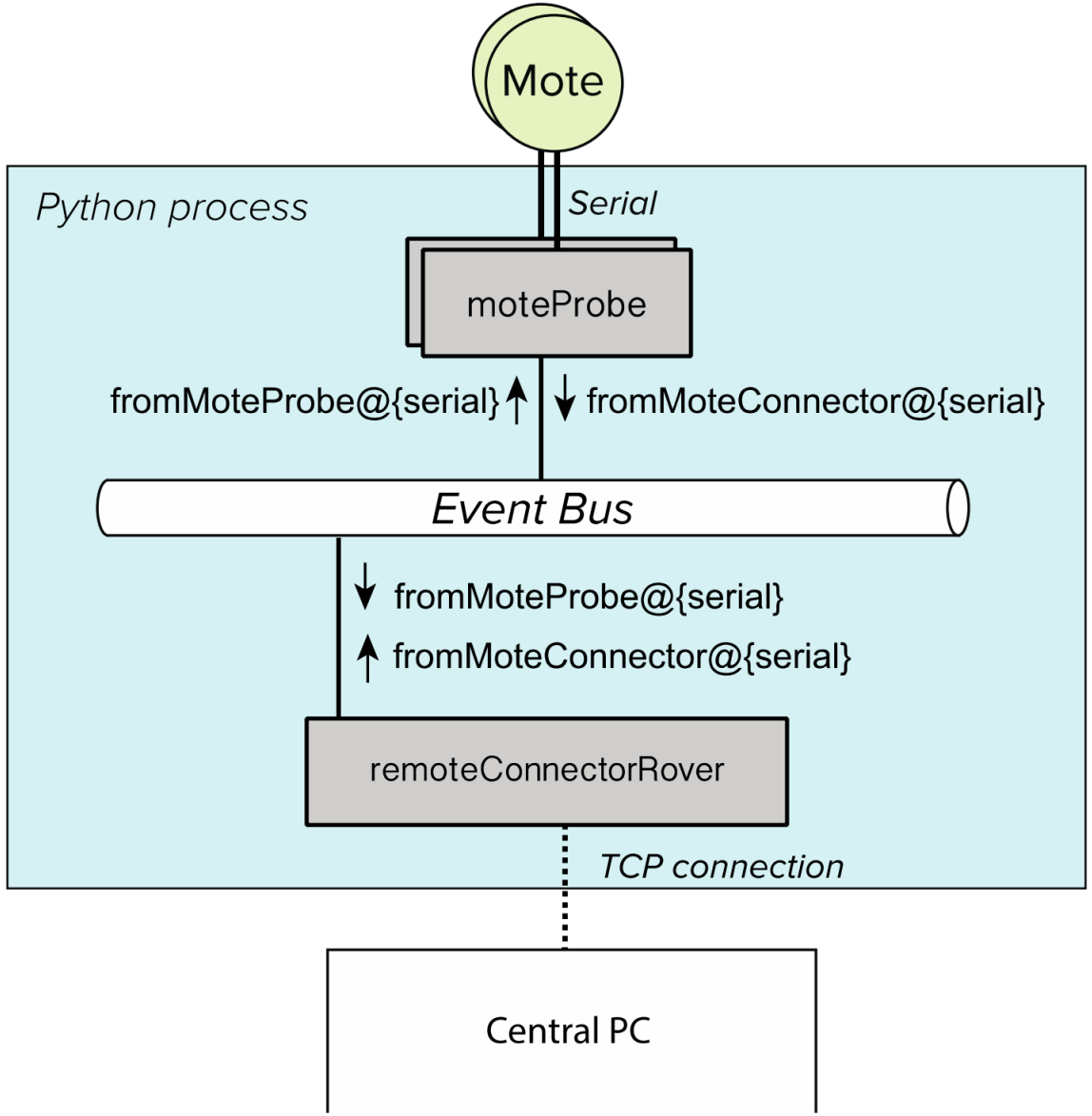
Table 2: Slot Allocation

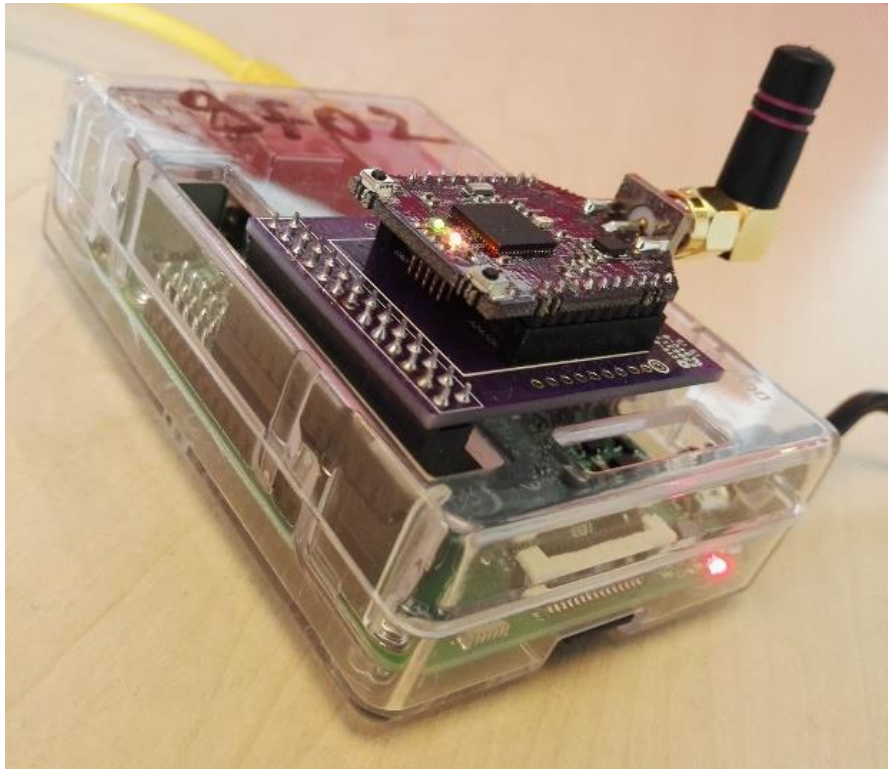
TSCH Schedule:

New Architecture of OpenVisualizer

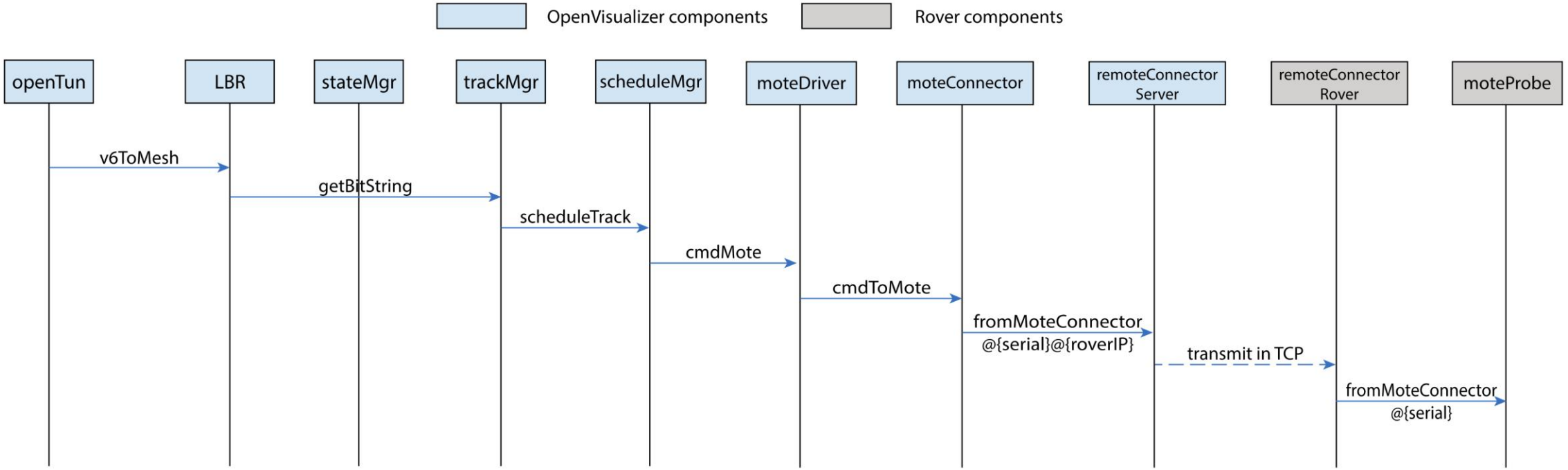


Rover Architecture

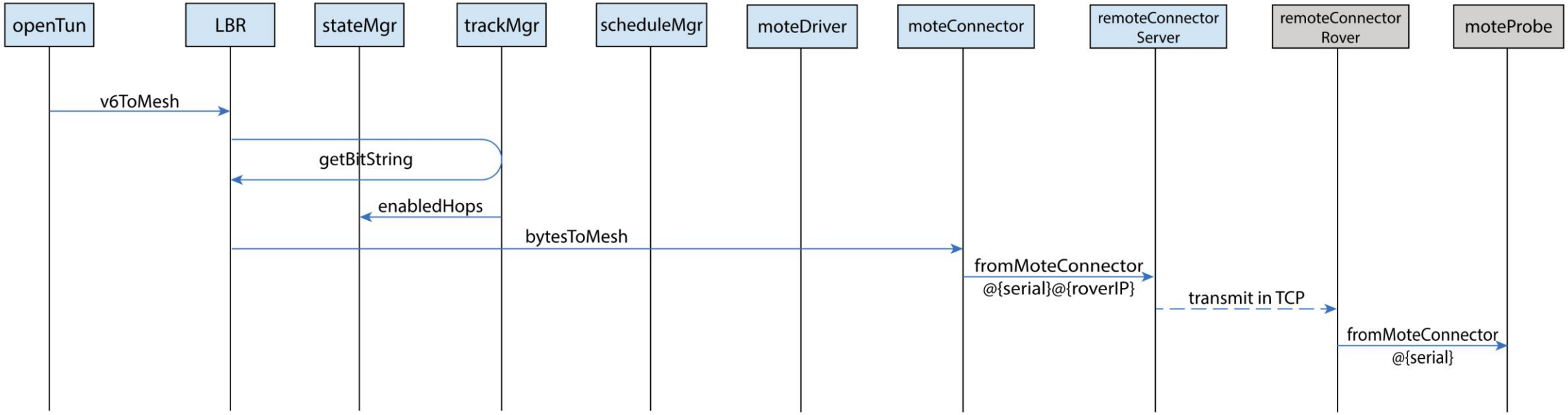




Track Reservation Interaction Diagram



Internet-to-BIER Message Handling Interaction Diagram



Deepening: Fast Resynchronization on Channel Hopping Mode

In current implementation of OpenWSN, each mote synchronizes according to the information contained in its RPL parent messages (i.e. it is its time source neighbor). Problem is that in case of a node or link failure, the child mote tends to desynchronize before it chooses a new RPL parent. It then needs to re-synchronize before it can forward packets again, which can take some time in a TSCH network.

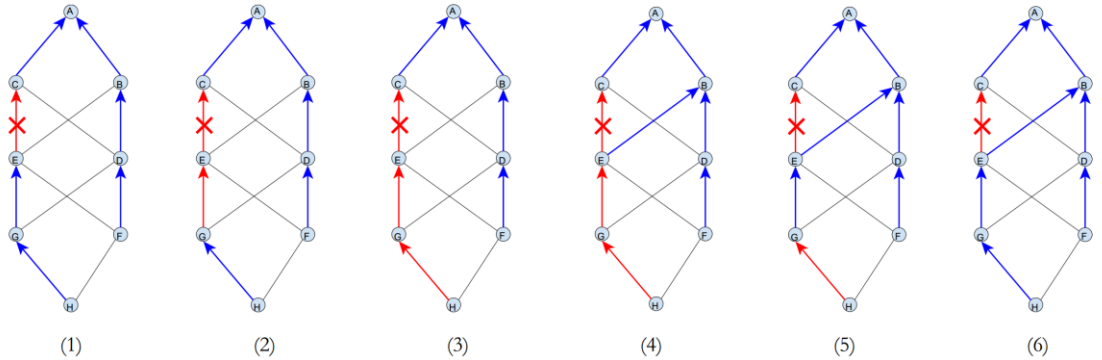


Figure 104 - an example of re-synchronization issue from RPL parent. (The red arrow means desynchronization)

The multi-parent selection algorithm introduced in the section 3.4.1 can effectively solve this issue by maintaining a list of parents with different preferences. We identified two possible solutions:

- Each mote synchronizes to its two most preferred parents at the same time
- Each mote synchronizes to its second preferred parent before it desynchronizes from its preferred parent

The first solution is easier to implement and more straightforward, but sometimes may be unstable and causes loop. The second one should be more stable but needs to define new rules in order to activate the failover. However, during the experiments, we implemented the first one because it actually worked well and we did not observe any loops in TSCH mode. We did not find additional time to research further on the second solution, which might be interesting in the future work.

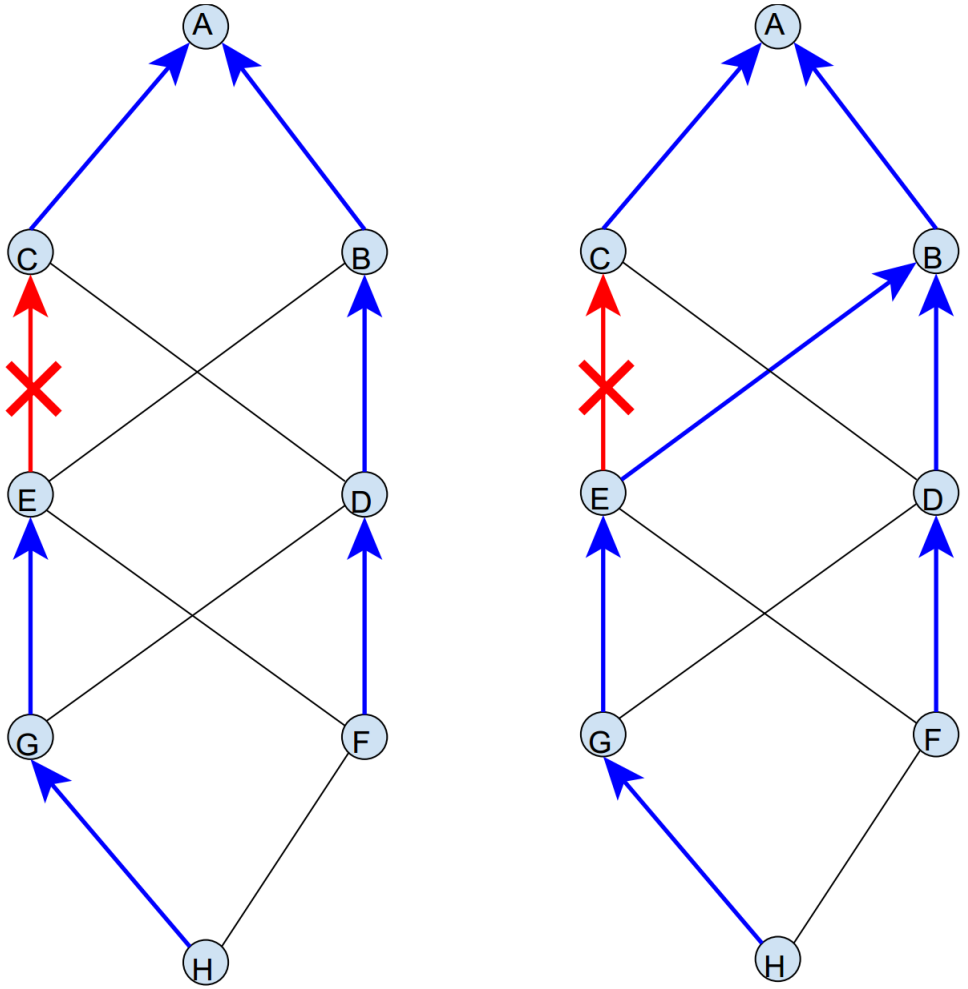
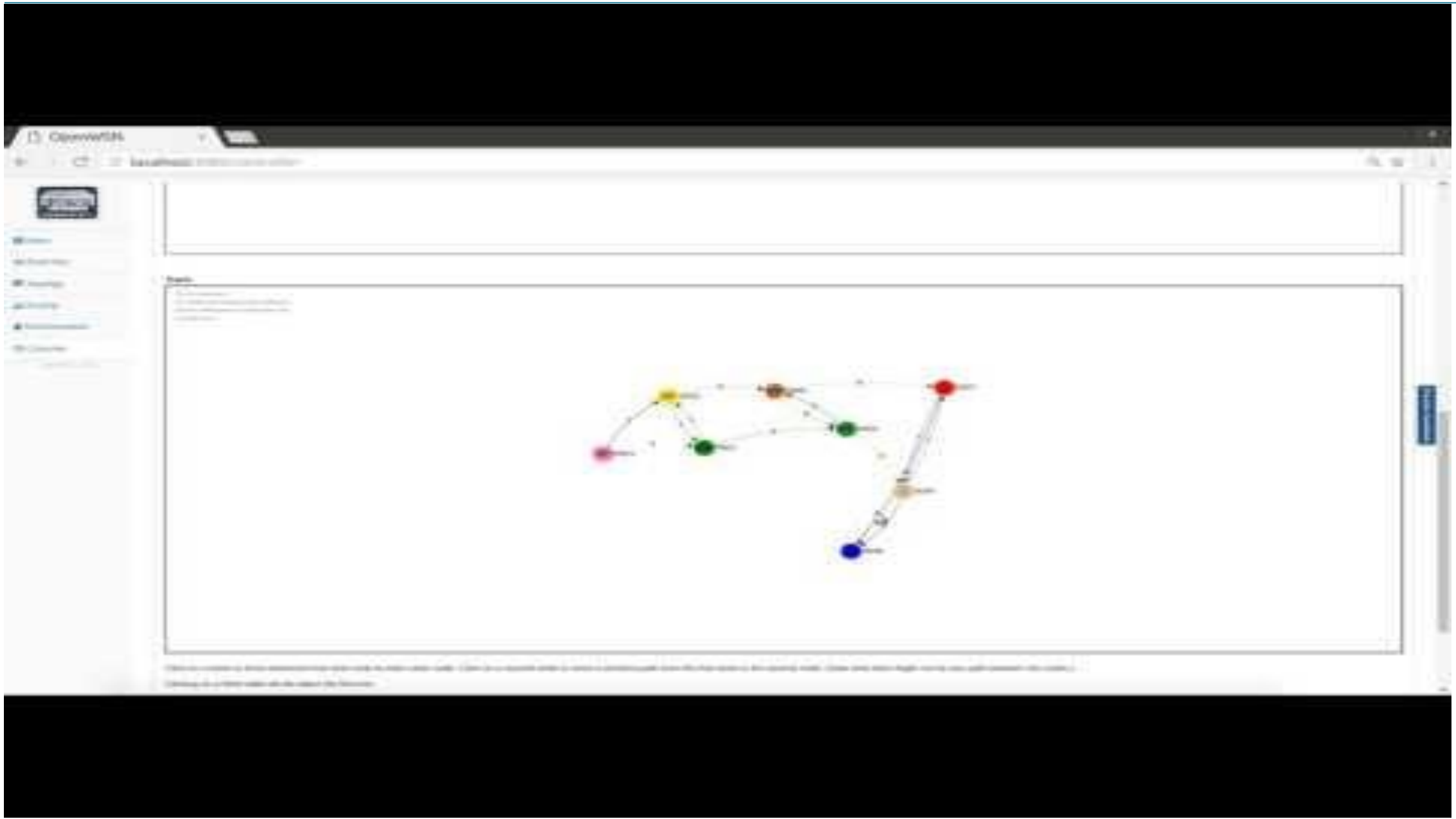


Figure 105 - fast resynchronization

Backup Path Setup Demo



Backup Control Loop Demo