6TiSCH Centralized Scheduling and Multipath Construction

INTERNSHIP REPORT

Hao JIANG

14 March 2016 - 09 September 2016

Cisco Paris Innovation and Research Lab (PIRL)

Institut Mines-Télécom / Télécom Bretagne

Industrial Mentor: Pascal THUBERT

Academic Coordinator: Géraldine TEXIER

EXECUTIVE SUMMARY

Software-defined Networking (SDN) promises to be a key enabler for future Internet of Things (IoT) by centralizing control, abstracting network devices, and providing a flexible, dynamic and automated reconfiguration of the network. Based on SDN paradigm, the IoT 6TiSCH architecture defines a centralized approach to build TSCH schedules in Low-power and Lossy Networks(LLN). Latency and reliability are critical to some use cases like industrial process control and automation. In order to improve the chance of delivery in face of a node or a link going down in these networks, the capability to forward multiple copies over parallel paths provides an additional spatial diversity to Time-Slotted Channel Hopping (TSCH), which seems valuable as an alternative to the temporal diversity with retransmission mechanisms and the spectral diversity with channel hopping.

Bit-Indexed Explicit Replication - Traffic Engineering (BIER-TE) enables this capability by setting the bits of the bitString contained in packet header to control the packet replication and elimination activities along redundant paths. BIER-TE defines a controller which keeps track of the network topology. Upon receiving a new flow, the controller will compute a complex path with a shape of a ladder, reserve timeslots on schedule for each transmission, and install a Bit-Indexed Forwarding Table (BIFT) on each node along the path forming a track from the source to the destination. The BIFT on each node specifies which bit of the bitString in a packet header corresponds to which adjacency. The associated bitString for a track is assigned by the controller to the ingress node, and would be inserted into the header of every packet belonging to the track. For each node along the complex path receiving the packet, it examines the bitString against its BIFT, and if there is a bit corresponding to one of its adjacencies is set to '1' in the bitString, the node would send to that adjacency a copy of the packet with the associated bit reset to '0'. The output bitString from the destination node thus can be used to identify transmission failures. For example, if all the enabled transmissions succeed, the bitString of the packet would be received with all bits reset to '0', while a failure of transmission will result in an output bitString with the corresponding bit still set to '1'. This information can be passed to the central controller, which accordingly can deduce the link conditions and the packet delivery ratio, and in return modify the bitStrings for next packets to dynamically control the replication and elimination activities according to the QoS requirements.

This internship designs, implements, and evaluates a multipath forwarding and control mechanism based on IPv6 over the TSCH mode of IEEE 802.15.4e (6TiSCH) with BIER-TE. The mission is to validate experimentally the value of path diversity with packet replication and elimination, analyze when and how the mechanism should be activated and incorporated with other diversity techniques such as retries and channel hopping, explore centralized operations and methods to optimize the scheduling in terms of jitter, latency, loss ratio, reliability, and energy consumption. During the first 4 months, I co-worked with another intern. Together we set up a real testbed with OpenMote hardware and OpenWSN open-source code. I mainly worked on the control plane to manage the TSCH schedule and transmission settings so as to make tests easier, replicable and automate, while the other intern worked on the data plane to support the forwarding schedules with the capability to replicate and eliminate the packets in addition to performing retries and channel hopping. We conducted several experiments to explore methods to improve the delivery ratio while lowering the energy budget along deterministic tracks.

During the last two months, I further improved and conducted more experiments to consolidate our conclusions, based on which I designed and implemented a control loop prototype integrating multipath with ARQs and channel hopping. The prototype enables fast recovery and dynamic control of packet replication and elimination activities in response to changes of the link statuses, maintaining a constant reliability while lowering the energy consumption at a minimum.

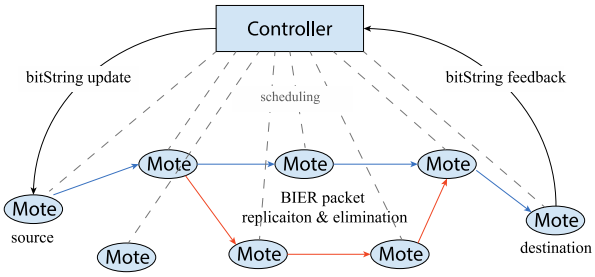


Figure 1 – multipath forwarding and control mechanism

The experiments are basically carried out as expected, and with the results we have been able to draw some conclusions: 1) multipath can increase reliability by providing additional spatial diversity for the packet delivery. It makes the occurrence of packet losses more independent, and effectively reduce burst losses. It can also overcome node failures or link failures with redundant paths. 2) Compared with retries, deterministic multipath based on BIER-TE introduces no jitter by design. It could provide a higher reliability than retries for low latency use cases. While in cases that are less sensitive to latency, retries seem to be more efficient under TSCH. 3) Scheduling replication and elimination points would provide higher reliability, more precise and more flexible path control, but also introduce delays and more power consumption. Using snooping can reach a similar reliability as replication, with less energy consumption, shorter bitString, and no additional delays, at the cost of a more complex ARQ mechanism, possibly less efficient and less precise multi-path control. 4) The control loop shows a good integration of the three diversity techniques, whereby a single path with retries can be used for efficient packet delivery and less power consumption with TSCH, and multipath mechanism is enabled only when necessary to provide higher reliability and eliminate burst losses.

There are some limitations in this experimental study. Firstly, the testbed topology is hardcoded and static during the whole experiment process, however, the reality is that the topology of a LLN is normally unstable, which causes that practically the testbed does not really represent the world. Moreover, the hardcoded neighborhood limits the scheduling patterns that we could design and implement for experiment. In addition, the performance of control loop can be potentially further improved. For example, the schedule can be optimized with more retries, the snooping mechanism can also be introduced to reduce latency, and the multipath activation threshold can be made less to increase the burst sensitivity of the mechanism. Finally, more repeated experiments could have been done in order to consolidate the conclusions. Therefore, there is still a long way to go for the evaluation of the multipath mechanisms over LLNs, and we hope this research work can be helpful for continued study in the future.

ACKNOWLEDGEMENTS

In performing this internship, I have taken the help and guidance from some respectable persons, who provided me priceless support to complete this report and deserve my greatest gratitude. I would firstly very much like to take this opportunity to express my sincere thanks to my mentor Pascal Thubert, for providing me such a wonderful subject and a great platform to conduct this internship, which would not have even been possible without his invaluable support and guidance. Special thanks also to my workmate Zacharie Brodard, who is not only a nice colleague to work with but also a great friend to have in life. I wish him all the best for his study at KTH in the year ahead and success in his future endeavors. I would also like to convey sincere gratitude to Tengfei Chang, Thomas Watteyne, Jonathan Muñoz, and all other mates in INRIA Paris for their time and effort to get me familiar to OpenWSN, and their precious advice during the whole period of my internship. Thanks to Prof. Géraldine Texier and Joëlle Le Bruno for their kind guidance and coordination from Telecom Bretagne. Thanks to Mohsin Kazmi, Clément Fischer, Victor Nguyen, Pierre Pfister for their help at Cisco. Thanks to my friends Jamal Boulmal, Chao Guo, Arunprabhu K, Tara Petric, Qian Li, Chenxi Du and Limin Hu.

Thank you.

ABSTRACT

Software-defined Networking (SDN) promises to be a key enabler for future Internet of Things (IoT) by centralizing control, abstracting network devices, and providing a flexible, dynamic and automated reconfiguration of the network. Based on SDN paradigm, the IoT 6TiSCH architecture defines a centralized approach to build TSCH schedules in Low-power and Lossy Networks(LLN). Latency and reliability are critical to some use cases like industrial process control and automation. In order to improve the chance of delivery in face of a node or a link going down in these networks, the capability to forward multiple copies over parallel paths provides an additional spatial diversity to TSCH, as an alternative to the time diversity with retransmission mechanisms and frequency diversity with channel hopping. BIER-TE enables this capability by setting the bits of controlling bitString in a BIER header. This internship is to develop SDN-like routing and resource allocation schemes for IoT deterministic networks, focusing on the design and implementation of 6TiSCH centralized scheduling and multipath mechanisms based on BIER-TE. This internship validates experimentally the value of spatial diversity introduced from multipath with packet replication and elimination, and deterministic scheduling along a track in a TSCH network. The results show that multipath increases reliability by offering additional space diversity for packet delivery, significantly reduces burst losses, and effectively overcomes link failures with redundant paths.

RÉSUMÉ

SDN promet d'être un facteur clé pour un futur IoT en centralisant le contrôle, en abstrayant les périphériques réseau, et en fournissant une reconfiguration souple, dynamique et automatisée du réseau. Basé sur le paradigme SDN, l'architecture IoT de 6TiSCH définit une approche centralisée pour construire des ordonnancements du MAC TSCH sur des réseaux de capteurs à basse puissance. La latence et la fiabilité sont critiques pour certains cas d'utilisation, comme le contrôle des processus industriels et l'automatisation. Afin d'améliorer les chances de livraison en cas de perte d'un nœud ou d'un lien dans ces réseaux, la capacité de transmettre des copies multiples sur des chemins parallèles fournit une diversité spatiale supplémentaire pour TSCH, comme une alternative à la diversité de temps avec les mécanismes de retransmission et à la diversité de fréquence avec saut de fréquence. BIER-TE permet d'activer cette capacité en réglant les bits de bitString de contrôle dans l'en-tête du BIER. L'objet de ce stage est de développer des schèmas de routage et d'allocation des ressources par SDN pour les réseaux déterministes du IoT, avec une concentration sur la conception et la mise en œuvre des mécanismes d'ordonnancement et de multi-trajets centralisés sous 6TiSCH basé sur BIER-TE. Ce stage valide expérimentalement l'intérêt d'un contrôleur pour le calcul centralisé du chemin complexe avec la réplication et l'élimination de paquets, et pour le l'ordonnancement déterministe le long d'un chemin complexe dans un réseau de TSCH. Les résultats montrent que multi-trajets augmente la fiabilité en offrant la diversité spatiale supplémentaire pour la livraison de paquets, réduit de manière significative les pertes en rafale, et surmonte efficacement les échecs de liens avec des chemins redondants.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Internship Overview

This internship was conducted in Cisco Paris Innovation and Research Lab (PIRL) for graduation of my master study at Telecom Bretagne, under the guidance of Pascal Thubert. It was started on 14th March 2016 and finished on 09th September 2016. During the first four months, I co-worked with another intern. Together we set up a real testbed with OpenMote hardware and OpenWSN source code, based on that we designed and implemented a set of experiments in order to validate the value of spatial diversity provided by multipath with packet replication and elimination over 6TiSCH Low-power and Lossy Networks (LLNs). During the last two months I further improved and conducted more new experiments to consolidate our conclusions, based on which I designed and implemented a control loop prototype integrating multipath with other diversity techniques such as ARQs and channel hopping. The prototype enables fast recovery and dynamic control of packet replication and elimination activities in response to changes of the link statuses, maintaining a constant reliability while lowering the energy consumption at a minimum.

1.2 About Cisco PIRL

The Paris Innovation and Research Lab(PIRL), which is officially opened in 2015, is one of nine Cisco innovation centers in the world. It serves as a hub for open innovations, where start-ups, universities, developers, researchers and ecosystem partners come together, with a goal of driving groundbreaking technologies for Internet of Everything(IoE) and a focus on smart city innovation. The PIRL team is fostering technological innovation and driving architecture in domain such as IPv6, Information Centric Networking, Big Data Analytics, AI, Virtualization, Cloud and Media. They are also supporting the Country Digitization Acceleration program in France, which goal is to enable customers, partners and start-up ecosystem to capture the value generated by the Internet of Everything, including Smart Cities, Cyber Security and IoT infrastructures & services. Their projects are developed and deployed on an open and connected Immersive Lab and Data Center. The picture below shows the main features of the Immersive Lab.

Figure 2 - Cisco PIRL Immersive Lab Features

1.3 State of the Art

1.3.1 Deterministic Networking (DetNet)

In the last 20 years, voice, data and video networks have converged to digital over IP. However, many industrial applications are still using isolated and dedicated wired network technologies (which is also referred to as Operational Technology, OT) for their operations because of the lack of "determinism" in IP-based technologies (also refers to Information Technologies, IT). Determinism in a network means the capability to carry unicast or multicast data flows for real-time applications with extremely low data loss rates, timely delivery and bounded delay variation [1]. In a deterministic network, each hop along the path of a deterministic flow is predictable in terms of time and frequency, regardless the load of the network. Determinism is critical to some use cases such as power grid, pro audio and video, public transportation, building automation systems, and industrial process control, etc. as described in [2]. The DetNet WG [3] defines three techniques to achieve the requirements required by deterministic networks:

- Congestion protection operates by reserving resources along the path of a DetNet flow
- Explicit routes which can be seen as a form of source routing or a pre-reserved path
- Packet replication and elimination along non-congruent explicit routes to the destination(s)

1.3.2 IPv6 over the TSCH mode of IEEE802.15.4e (6TiSCH)

6TiSCH is a key to enable the further adoption of IPv6 in industrial standards and the convergence of OT with IT [4], as envisioned as the Industrial Internet. The art of 6TiSCH puts together a set of open standards to achieve quasi-deterministic performance in terms of jitter, latency, scalability, and reliability for industrial operations along with an effective coexistence of stochastic traffic over IEEE802.15.4e TSCH [5]. Below are main techniques related to this internship:

- Software-defined Networking (SDN) architecture has defined a control plane in which a centralized controller owns global visibility on network resources and capabilities, making it quite easy and efficient to compute, establish and manage the explicit routes in network.
- IEEE802.15.4 Time-Slotted Channel Hopping (TSCH) provides a high reliability with diversities in time and frequency, allowing all communications to be scheduled and network resources reserved in a deterministic fashion over a synchronized wireless network.
- Traffic engineering for Bit-Indexed Explicit Replication (BIER-TE) protocol allows for traffic engineering with explicit hop-by-hop forwarding with replication and elimination, by setting bits of the bitmap in the packet header.
- IPv6 Routing Protocol for Low-power and Lossy Networks (RPL) provides a mechanism whereby multipoint-to-point traffic from devices inside the LLN towards a central control point as well as point-to-multipoint traffic from the central control point to the devices inside the LLN are supported [6].
- Available Routing Construct (ARC) designed by [7] depicts a new routing algorithm which can significantly improve the network utilization in respect with its fault-tolerance intrinsic properties and also fast rerouting, load balancing and multipath features.

1.3.3 Software-defined Networking (SDN)

The Software Defined Network (SDN) paradigm is a promising enabling technology for future Internet of Things(IoT) networks. Aware of that, DetNet and 6TiSCH working groups at IETF aim to develop controller-based routing and resource allocation schemes for IPv6-based IoT deterministic networks, with a protocol suite that allows supporting different classes of traffic, with different QoS requirements, on the same network, at the same time, by allocating resources in a centralized and efficient way [8]. SDN architecture is dynamic, manageable, cost-effective and adaptable, making it ideal for the today's IoT applications. It decouples the network control and forwarding functions, enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services [9].

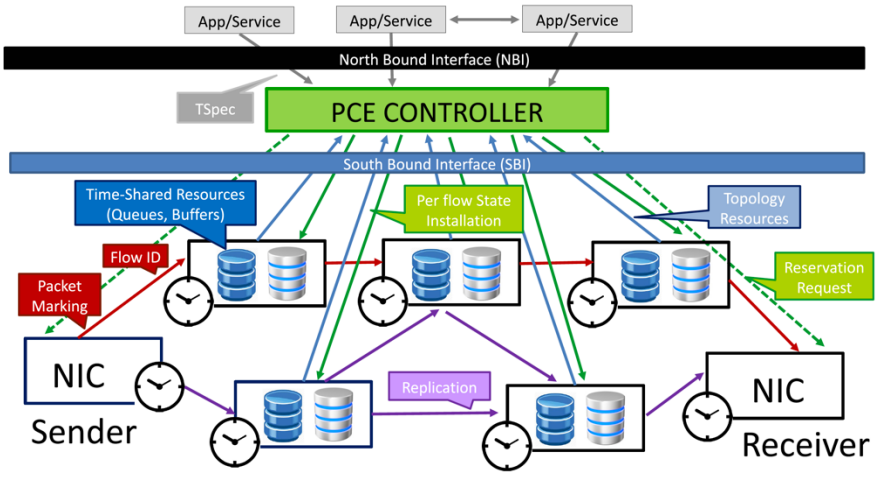


Figure 3 - SDN model for deterministic networks proposed by DetNet [8]

The DetNet WG defines an overall SDN model for deterministic networks as shown in Figure 3. And they are currently working on the signaling elements to be used for centralized path setup and resource reservation, which are also partly covered in this internship:

- the definition of data models to report topology and capabilities of devices to the controller
- the protocol elements to request a path setup for a given flow and configure the Network Interface Card (NIC) in the end nodes
- algorithms and data models to establish complex (and explicit) routes with path diversity that supports packet replication, retry, and elimination mechanisms for the given flow
- the time-based reservation of physical resources in the forwarding devices along the path
- support for life cycle management for the path [10] (instantiate/modify/update/delete)
- the definition of the tagging elements (Flow ID, packet marking) to be used to identify the flow to be forwarded along a pre-installed path

1.3.4 Time-Slotted Channel Hopping (TSCH)

6TiSCH provides a statistically multiplexed network layer based on IPv6 over the highly predictable TSCH MAC layer in IoT low power and lossy networks. A Low-Power and Lossy Network (LLN) is composed of devices with constrained resources, which typically have low processing power, memory and energy and operate on lossy wireless links based on IEEE802.15.4, Bluetooth, low-power WiFi, or Power Line (PLC) [11]. These wireless links are naturally unstable due to external interference and multi-path fading, resulting in a much poorer performance in terms of packet loss rate, latency and reliability compared with point-to-point wired links. However, the IEEE802.15.4 TSCH mode, which is also used in the industrial solutions such as WirelessHART and ISA100.11a, addresses these problems by adopting time-synchronized communication and channel hopping to provide network robustness with spectral and temporal redundancy. Time-slotted communication increases potential throughput by minimizing unwanted collisions that might lead to catastrophic failure and provides deterministic yet flexible bandwidth for a wide variety of applications. Channel hopping extends the effective range of communications by mitigating the effects of multipath fading and interference [12], which generally affects limited spectrums of the 16 channels available in IEEE802.15.4 for 2.4GHz band, as is also supported by the experimental results from [13]:

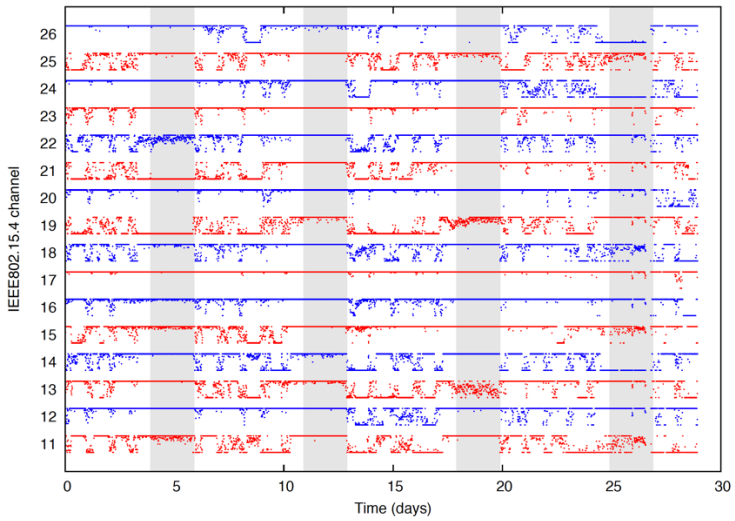


Figure 4 - Packet Delivery Ratio evolving over time on a link (The grayed out time intervals represent week-ends)

In TSCH, each node has its own schedule which repeats itself over time, while at the network scale a schedule can be represented as a grid matrix with a vertical axis of up to 16 channels and a lateral axis of slotted time. Each transmission can be scheduled in a cell with a duration of 10 or 15 ms indicating not only a specific timeslot but also a particular channel where the transmission happens. The latter is translated by both communicating motes into a frequency using function [14]:

$$frequency = F(channelOffset + ASN) \bmod nFreq$$

in which *nFreq* is the number of available frequencies, and ASN means the Absolute Slot Number. Thus, it is very important that all the nodes in the network have a synchronized sense of time and channel hopping, so that the motes can send or listen on the right channel at the right time for each communication along the path, and keep slept to save battery on other time slots. As is shown below in Figure 5, a slot can be scheduled either in transmission (TX) or reception (RX).

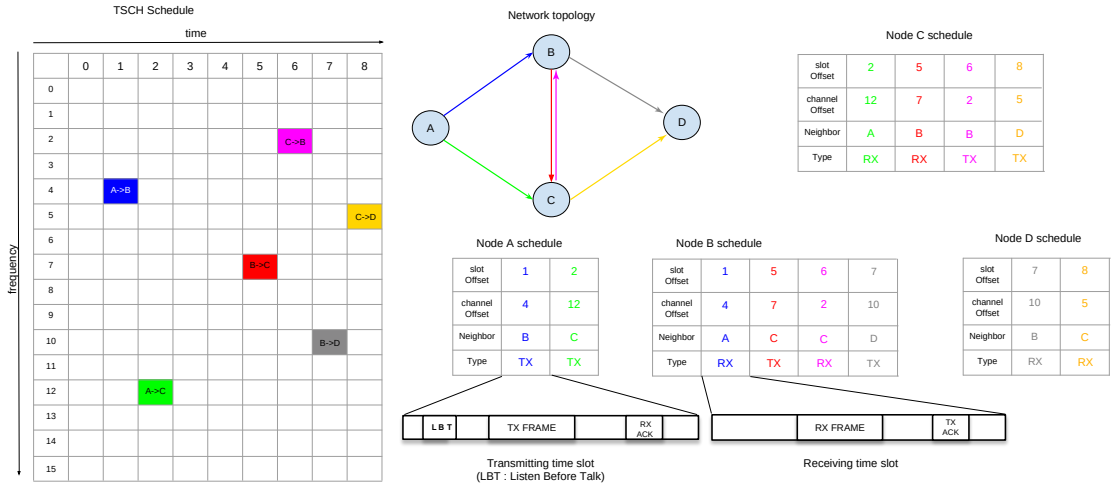


Figure 5 - Simple example of an IEEE802.15.4e schedule

1.3.5 Bit-Indexed Explicit Replication – Traffic Engineering (BIER-TE)

BIER (Bit Indexed Explicit Replication) is a new multicast forwarding protocol which is defined in the Internet draft [15]. When a multicast data packet enters a "BIER domain", the ingress router determines a set of egress routers to which the packet needs to be sent, and then encapsulates the packet in a "BIER header". The BIER header contains a bitString in which each bit represents exactly one egress router in the domain. To send a packet to a particular set of egress nodes, the ingress node sets the bits for each of those egress nodes, and clears other bits in the bitString [16]. Each packet can then be forwarded along the unicast shortest path tree from the ingress node to the egress nodes based on an IGP such as OSPF or ISIS.

BIER-TE is a Traffic Engineering protocol inspired from BIER and is being standardized in [17]. It forwards and replicates packets like BIER based on a bitString in the packet header but it does not require an IGP. The key differences over BIER are:

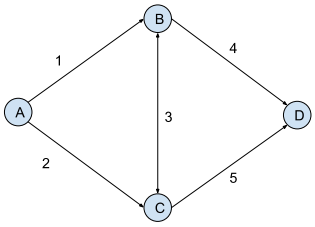
- BIER-TE replaces in-network, autonomous, and IGP-based path calculation by explicit paths calculated off-path by a controller.
- In BIER-TE every bitPosition of the bitString of a BIER-TE packet indicates one or more adjacencies - instead of a set of destination nodes as in BIER.
- In BIER-TE each node has no routing table but only a BIER-TE Forwarding Table (BIFT) specifying how the node should replicate packets to its adjacencies according to the bitmap (sometimes called "bitString") contained in the BIER header of the packets.

This way, BIER-TE supports traffic engineering by explicit hop-by-hop forwarding and loose hop forwarding of packets, and enables to activate the packet replication and elimination functions in a manner that is abstract to the data plane forwarding information, that is, an adjacency represented by a bit in the BIER header can correspond to an Ethernet hop, a routing segment, etc.

As is forementioned, BIER-TE relies on a controller to setup and maintain explicit paths and the BIFT on each node. To be more specific, a BIER-TE controller should be able to:

- Keep track of the network topology
- Compute explicit paths and assign bitPositions for required adjacencies
- Push the bitString/flowID mappings to the ingress node, allowing it to know what flows are mapped to what BitStrings and insert the right bitStrings into the packets to send
- Push the bitPosition/adjacency mappings to the BIFT of the intermediate nodes, allowing them to know how to forward a packet depending on the bitmap in the packet's header
- Maintain the paths and the BIFT of the nodes during day-to-day operations and topology changes

In the Internet draft [1], BIER-TE is proposed as an alternative DetNet data plane protocol, which can be used to enable path diversity in a deterministic networking perspective by controlling the process of the replication and elimination of a packet. A simple example can be taken as blow in Figure 6, in which we assume that the schedules and BIFTs of the nodes are already constructed by the controller, and the schedules are constructed as shown in Figure 5.



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 6 - A simple network of 4 nodes and their BIFTs (constructed by an external controller) [18]

If A want to send a packet to D using all the adjacencies corresponding to bitIndexes (also named bitPositions) 1, 2, 3, 4, 5 respectively, it would insert a BIER header containing the bitString '11111' with all bits set to '1' into the packet in order to enable transmissions on all the adjacencies. Then the packet would be forwarded according the BIFT at each hop:

- A checks bit 1 and 2 of the bitmap, which are both SET to '1', sends a copy of the packet to B and C with corresponding bit of bitString RESET to '0'. ('01111' to B, '10111' to C)
- B receives the packet and checks bit 1, 3, and 4. As only bit 3 and 4 are SET, it sends only to C and D with bit 3 and 4 of the bitString RESET. ('01011' to C, and '01101' to D)
- C has then received a copy of the packet from A with a bitString '10111' and one from B with a bitString '01011'. It makes an 'AND' operation of the two copies which results in a bitString '00011', and then checks bit 2, 3, 5 of the bitString. As only bit 5 is SET, C would thus only RESET bit 5 and send the copy with bitString '00010' to D.

- D has finally received a copy from B with a bitString '01101' and one from C with bitString '00010'. It makes an 'AND' operation of the two copies which results in a bitString '00000', and forwards only one copy to the upper layer.

Figure 7 illustrates how a BIER packet is replicated and eliminated based on the previous example.

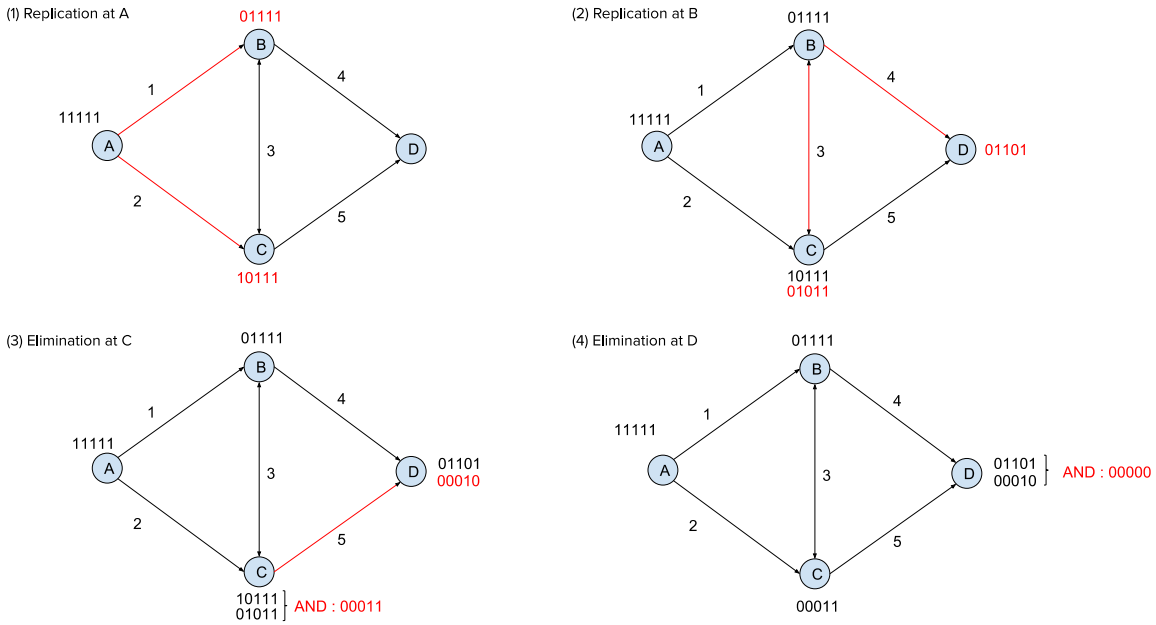


Figure 7 - BIER-TE multi-path with replication and elimination

It is noteworthy that:

- The order in which the transmissions take place is important, as an example, the controller should not schedule the transmission from C to D before the one from B to C.
- Bit 3 is assigned for both directions for the link between B and C, so that the transmission happens only once between B and C if both of them have the packet.
- In this example A enables all the bits and provides a relatively higher reliability, however it depends on the needs and the link status to decide what bitString to use. As an example, a single path with bitString '10010' can be taken into account if the links A→B, B→D are satisfactory and the energy consumption is supposed to be as low as possible; or a bitString '10111' is also feasible if the link B→D becomes unstable and needs to be "protected".
- The output bitString from the destination node thus can be used to identify transmission failures. For example, if all the enabled transmissions succeed, the final output of bitString from D would be '00000'; a failure from A to B would result in a bitString of '10000'; while a failure from C to D would result in a bitString of '01101'. This information can be passed to the central controller, which in turn can modify the bitString for the next packets.

1.3.6 IPv6 Routing Protocol for Low-power and Lossy Networks (RPL)

RPL is a proposal as an IPv6 routing protocol for LLNs by the ROLL working group in IETF [6]. It is incorporated in the 6TiSCH protocol stack for distributed operations [19]. The objective of RPL is to be the IPv6 routing protocol for networks which "comprise up to thousands of nodes", where the majority of the nodes have very constrained resources, where the network to a large degree is managed by single or few central "supernodes" called DAGroot, and where handling mobility is not an explicit design target [20]. RPL involves many concepts which make it a flexible protocol, but also complex. This section gives a simple introduction to RPL in order to make the general context of the internship can be fully understood. For further reading please refer to the Internet draft [6].

Topology Formation. A network running RPL may maintain one or more RPL instances with each defines a topology that is built using a unique metric or constraint within the network. Each RPL instance consists of multiple Destination Oriented Directed Acyclic Graphs (DODAGs). Each DODAG has a different DAGroot. A node may join multiple RPL instances, but must only belong to one DAG within each instance [21], which can be illustrated as below:

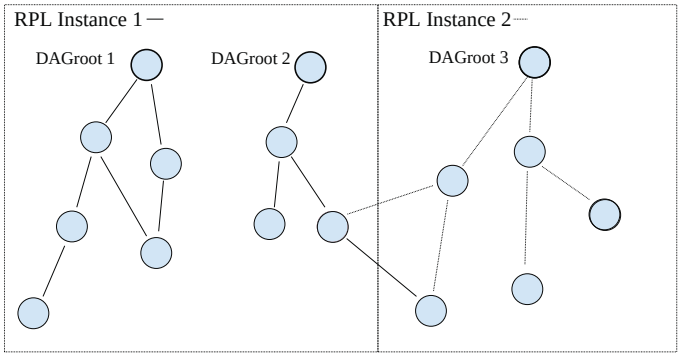


Figure 8 - A network with three DAGs in two RPL instances

DIO Message. A DAG Information Object (DIO) carries information that allows new nodes to discover and join an RPL instance. Each node of a DODAG will emit DIO messages containing its *rank* in the DODAG (relative distance to the DAGroot), and when receiving a DIO message, a node will compute its own *rank* to make it larger than its parents, select a preferred parent and start emitting its own DIO messages. The DODAG formation starts at DAGroot and spreads to cover the whole LLN gradually through DIO messages.

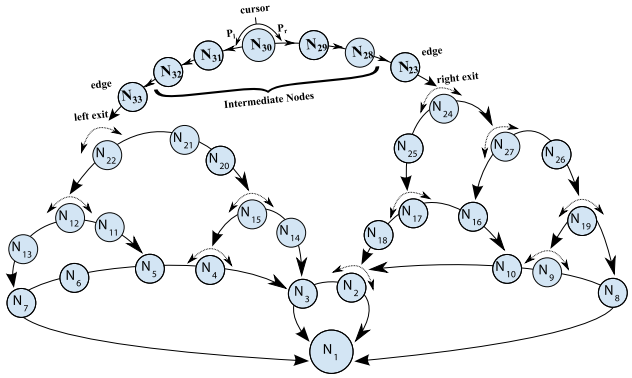
DAO Message. A Destination Advertisement Object (DAO) propagates destination information (their own address prefix) to their neighborhood using multicasts, or back to the DAGroot using unicasts after joining a DODAG.

Route Construction. "Upward routes" towards nodes (parents) of decreasing *rank* are provided by DIO messages; "downward routes" towards nodes (children) of increasing *rank* are enabled by DAO messages.

1.3.7 Available Routing Construct (ARC)

During the internship, a variation of ARC is used to compute the complex paths to support packet replication and elimination. The ARC technology was introduced with an IETF Internet draft [7] and the paper [22] evaluates it with a simulation result showing that ARC over-performs classical multi-path routing algorithms such as the Flooding algorithm, the modified Dijkstra's algorithm (MDA), and the modified Bellman-Ford algorithm (MBFA). An innovative concept in ARC is that each routing construct is made of a sequence of nodes and links with two outgoing edges (towards which the traffic is forwarded). The main advantages of this are better spatial diversity introduced by multipath and faster recovery from network failure without additional computation.

Definitions. As shown in Figure 9, 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 of nodes {Left Exit} {Intermediate Nodes} {Right Exit}. The cursor of an ARC is a virtual high point which can be located on a node or a link between 2 nodes. The traffic in an ARC flows away from its cursor and can be distributed on both directions. In fact, except the exit links which are directed, the links between intermediate nodes are reversible.



ARC Topology

ARC	{Left Exit} {Intermediate Nodes} {Right Exit}
1	{ N_1 } { N_3 N_2 } { N_1 }
2	{ N_1 } { N_7 N_6 N_5 N_4 } { N_3 }
3	{ N_2 } { N_{10} N_9 N_8 } { N_1 }
4	{ N_7 } { N_{13} N_{12} N_{11} } { N_5 }
5	{ N_4 } { N_{15} N_{14} } { N_3 }
6	{ N_2 } { N_{18} N_{17} N_{16} } { N_{10} }
7	{ N_9 } { N_{19} } { N_8 }
8	{ N_{12} } { N_{22} N_{21} N_{20} } { N_{15} }
9	{ N_{22} } { N_{33} N_{32} N_{31} N_{30} N_{29} N_{28} N_{23} } { N_{24} }
10	{ N_{17} } { N_{25} N_{24} } { N_{27} }
11	{ N_{16} } { N_{27} N_{26} } { N_{19} }

ARC Set Description

Figure 9 - Example of an ARC topology in which N_1 is the root

ARC Algorithm. The computation of the ARCs is done according to the Open Lowest ARC First (OLAF) algorithm, which is a variation of Shortest Path First (SPF) algorithm [23], [24] that creates ARCs by connecting SPF paths. The set of ARCs includes the SPF tree. The algorithm identifies the mono-connected zones and provides redundancy inside such zones. OLAF is based on the Dijkstra's algorithm for SPF [25]: "the reverse SPF tree towards a destination is conserved and preferred for forwarding along the resulting ARC set". Omega(Ω) is the destination to be reached with an ARC set, which in the case of the previous example is the node N_1. By design, both edges of each ARC cannot terminate in the same node, except for the first formed ARC that terminates in Omega. In order to make the algorithm well understood, we use the following convention [26]:

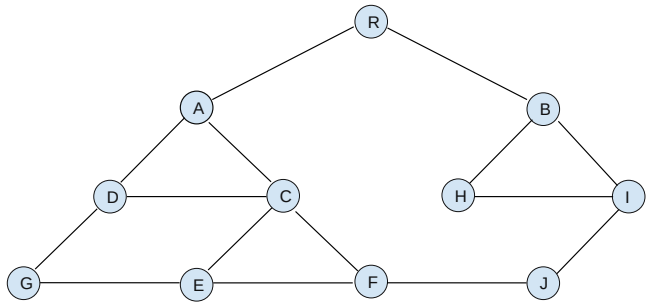
- a node is considered safe if there is no single point of failure - apart from the node itself - on its way to Omega. From this definition, a safe node has at least two non-congruent paths to two other different Safe Nodes, finally at least two paths to Omega.

- a node N is S-dependent (denoted as '? -S') if S is the last single point of failure along N's shortest path to Omega. DSet(S) is the set of nodes that is S-dependent.

The algorithm starts at Omega, and spread over the network as follows:

- For each of the node H_i that has a direct link with Omega (which is called Heir Link), we create DSet(H_i) in which we place H_i.
- Select the node in the set of nodes that is closest to Omega using the cost towards Omega like in traditional reverse SPF tree construction, and place the node in the same dependent set as its parent (for example DSet(H_j)) in the reverse SPF tree.
- Each sub-tree grows separately inside DSet(H_k) where H_k is the virtual safe node (the root of the sub-tree).
- Once a selected node *S* is placed in a DSet, all its neighbors are considered one by one, and if at least one of the neighbors is already in a different DSet with this node, we select the neighbor that provides the shortest alternate path to Omega for node *S*.
- As a result, node *S* has two paths to Omega which are isolated, as the two sets are different and have no intersection. And in turn node *S* becomes safe.
- The ARC is built by adjoining the two non-congruent paths, and the farthest node from Omega, which is node *S*, becomes the cursor. All the intermediate links becomes reversible and the nodes in this ARC becomes safe. Note that the node which has at least 2 outgoing links to 2 different safe nodes forms an ARC itself with the two outgoing edges.
- Finally, the forwarding path between a source and a destination follows a cascade of ARCs.

Figure 11 pictures different steps of the ARC algorithm applied to the network topology presented in Figure 10(a). In order to distinguish the state of each link, we use a black segment to simply represent a link, a green arrow "C → A" to show that A is the SPF successor of node C towards Omega, and a red arrow "I → B" to show that B is the non-shortest path successor of node I. A safe node is colored in blue. Finally, a total of five ARCs are constructed as listed in Figure 10(b).



(a) Network Topology

ARC	{Left Exit} {Intermediate Nodes} {Right Exit}
1	{ R } { A C F J I B } { R }
2	{ A } { D } { C }
3	{ C } { E } { F }
4	{ D } { G } { E }
5	{ B } { H } { I }

(b) Computed ARCs

Figure 10 - Example of a simple network

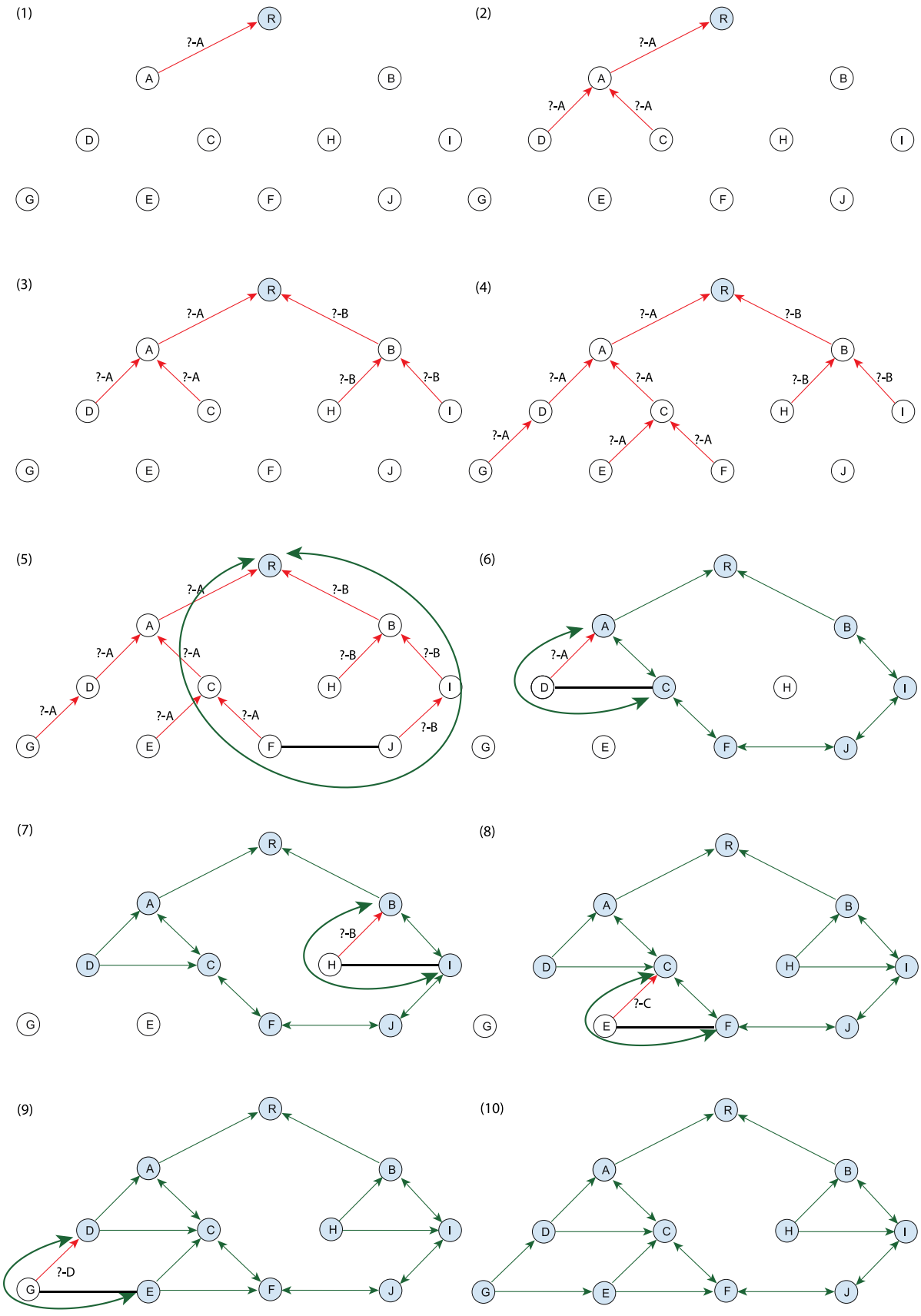


Figure 11 – Different steps of ARC algorithm applied to a simple network

1.3.8 The Art of 6TiSCH

Protocol Stack. The art of 6TiSCH incorporates 6LoWPAN, which is a standard way to support IPv6 over LLNs, RPL for distributed, stochastic operations, a PCE for centralized, deterministic routing, and a suite of IETF open standards allowing to support different classes of traffic over IEEE802.15.4 TSCH. Figure 12 gives the protocol stack of 6TiSCH. In particular, the 6TiSCH operation sublayer (6top) [27] is a sublayer of a Logical Link Control (LLC) that provides the abstraction of an IP link over the TSCH MAC, in which the 6top Protocol (6P) enables distributed scheduling allowing neighbor nodes to add/delete TSCH cells to one another. On the other hand, BIER-TE, which is briefly introduced in section 1.3.5, provides the capability for packet replication and elimination, and deterministic centralized scheduling with a PCE. The latter one is of interest and implemented in this internship.

<table>
<tr><td>(COMI)
CoAP / DTLS</td><td>(PANA)</td><td>6LoWPAN
ND</td><td>RPL</td><td></td></tr>
<tr><td>UDP</td><td colspan="3">ICMP</td><td></td></tr>
<tr><td colspan="5">IPv6</td></tr>
<tr><td colspan="5">6LoWPAN adaptation and compression (HC)</td></tr>
<tr><td colspan="2">6top</td><td colspan="3">BIER-TE</td></tr>
<tr><td colspan="5">IEEE802.15.4e TSCH</td></tr>
<tr><td colspan="5">Radio</td></tr>
</table>

Figure 12 - 6TiSCH protocol stack

6TiSCH architecture, as defined in the Internet draft [19], is composed of a high speed powered backbone router and a set of IEEE802.15.4 TSCH low-power wireless networks attached and synchronized by Backbone Routers (6BBRs). Figure 13 gives an example of the architecture.

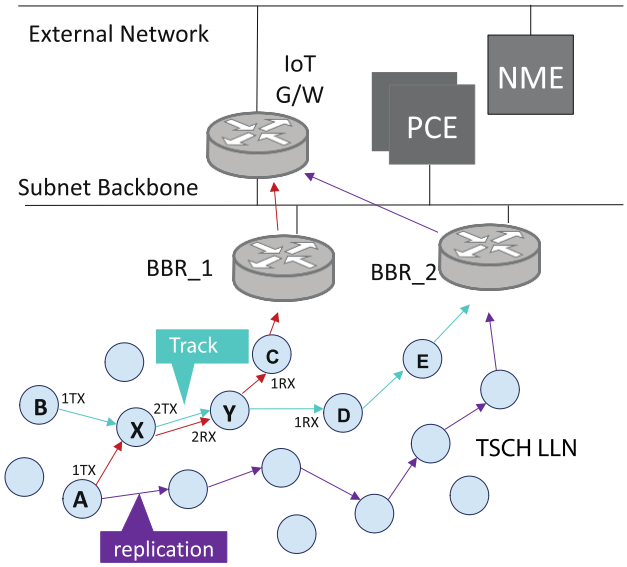


Figure 13 - Example of a 6TiSCH architecture

6TiSCH elements. 6TiSCH [28] defines a number of new terminology elements in addition to the existing ones inherited from its protocol stack. Based on TSCH, 6TiSCH defines a new global concept that is called Channel Distribution/Usage (CDU) matrix. A **CDU** is a matrix of cells with each identified by a **slotOffset** and a **channelOffset**, and it can be seen as a network-scale schedule which specifies the global characteristics (e.g., channels used, timeslot duration and the number of timeslots per iteration) and operations (at which time on which frequency which mote sends data and which motes listen).

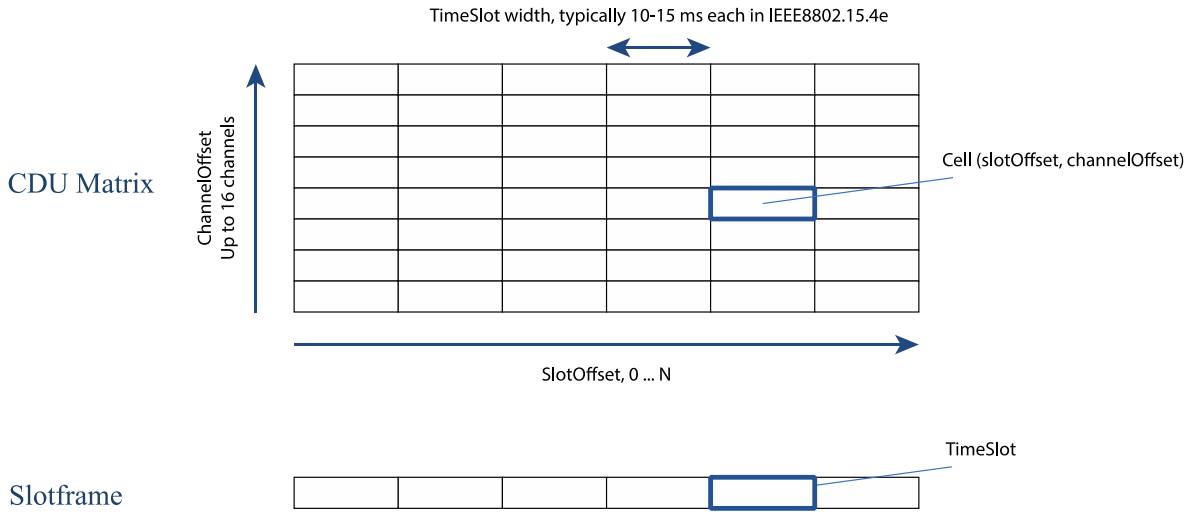


Figure 14 - Channel Distribution Usage matrix and slotframe

All the nodes in TSCH network are synchronized in a slotframe, and communicate by following a common global schedule. A **slotframe**, as presented in Figure 14, is a MAC-level abstraction that is internal but common to all nodes and contains a series of timeslots of equal length and priority. Note that several slotframes can also coexist in a node's schedule but with different priorities [29]. In addition, 6TiSCH also defines the following concepts:

- a **track** is a determined sequence of reserved cells along a multi-hop path which is uniquely identified by a globally significant trackID. In the global schedule a track can be represented as a unidirectional path between a source and a destination. For the example in Figure 13, a track is established from a field device in a 6TiSCH network to an IoT gateway that is located on a deterministic backbone. [8]
- a **bundle** is a peer-wise concept, which is a group of equivalent scheduled cells and needed for the communication between adjacent nodes, i.e., a bundle is a set of cells identified by different [slot Offset, channel Offset], which are scheduled for a same purpose, with the same neighbor, with the same flags, and the same slotframe. Ultimately a bundle represents a half-duplex link between nodes with a bandwidth that amount to the sum of the timeslots in the bundle. [28] A bundle is identified by (source MAC, destination MAC, trackID).

1.4 Problem Statement

A primary goal of deterministic networking is to support critical industrial data traffics that {

- need absolute guarantees of maximum and minimum latency, sometimes also a controlled jitter in the worst case
- require packet loss to be eliminated much lower compared to normal routing and switching
- cannot suffer throttling, congestion feedback, or any network-imposed transmission delay
- can absorb in total more than half of the network's available bandwidth

} regardless of the load composed by other traditional best-effort IP flows in a same LLN network.

By **scheduling** of the network, 6TiSCH aims to achieve this with the following efforts:

- Packet Delivery Ratio. TSCH allows to defeat external interference and multipath fading, and schedule all communications to eliminate congestion loss over a synchronized network.
- Jitter and Latency. Fully time-controlled operations as each transmission and reception along a path is explicitly scheduled and highly predictable.
- Fault Tolerance and Reliability. Introduce multipath with packet replication, retries and elimination to eliminate single failure losses and enable fast rerouting.
- Energy Consumption. Put peer devices in deep sleep between scheduled transmission; therefore, the waste of power spent in idle listening and long preambles is eliminated.

However, "a real-world network cannot be mathematically deterministic since there is no way to guarantee a perfect delivery ratio, and radios are worse than wires due to high rates of losses in transmission." [8] What can be made deterministic is the bounded worst case latency, within which the delivery ratios can only be optimized, generally by exploiting different forms of **diversity**, such as temporal diversity provided by ARQs, spectral diversity provided by channel hopping, spatial diversity provided by multi-antenna, or MIMO (Multiple Input Multiple Output) [30], etc.

1.5 Mission

This study explores another form of spatial diversity provided by multi-path redundancy, whereby each packet can be replicated into two or more copies and delivered to the destination along non-congruent routes.

The mission of this internship is to validate experimentally the value of path diversity with packet replication and elimination over 6TiSCH LLNs, analyze when and how the mechanism should be activated and incorporated with other diversity techniques such as ARQs and channel hopping, explore centralized operations and methods to optimize the scheduling and finally to minimize the jitter, latency, loss ratio, and energy consumption while maintaining the reliability at a maximum.

1.6 Goals and Methods

The primary goal of this internship is to develop a deterministic networking prototype over IEEE 802.15.4e TSCH based low-power and lossy network by exploiting spatial diversity from multipath in addition to time and frequency diversities provided by retry and channel hopping mechanisms. The work is thus mainly divided into the following parts.

1.6.1 Control Plane Operations

1.Topology Discovery

Design and implement RPL path diversity mechanism in a centralized manner to expose the network topology to the PCE.

2.Multipath Computation

Design and implement an ARC-based multipath computation mechanism to provide path redundancy.

3.Track Reservation

Design and implement the data structures to support centralized scheduling and resource reservation for the computed path which forms a track.

4.Bitmap Control

Design and implement a simple bitString feedback control prototype to enable fast recovery and dynamical control of the replication and elimination activities in response to changes of the link statuses to maintain a constant reliability while lowering the energy consumption, as was forementioned in section 1.3.5.

1.6.2 Data Plane Operations

Design and implement BIER-TE protocol to support multipath forwarding with the control of packet replication and elimination activities.

1.6.3 Testbed Development

Design and deploy a testbed to validate experimentally the value of path diversity with packet replication and elimination in terms of latency, loss ratio, reliability, and energy consumption.

1.6.4 Experiments and Results

Design and conduct experiments to validate the value of path diversity, analyze the results and explore methods to improve the overall performance.

1.7 Team and Divisions

During this internship at PIRL, I co-worked with Zacharie Brodard, who is another intern from École Polytechnique. We set up the testbed with OpenMote hardware [31] and OpenWSN open-source code [32] where I mainly worked on the control plane (1.6.1) to manage the device schedule and transmission settings so as to make tests easier, replicable and automate, while Zacharie mainly worked on the data plane (1.6.2) to support forwarding schedules with the capability to replicate and eliminate the packets in addition to performing retries and also channel hopping. Together we conducted various experiments to explore methods to improve the delivery ratio while lowering the energy budget along deterministic tracks.

1.8 Contributions

This project studies 6TiSCH schedule management and multipath with following contributions:

- An open-source contribution to the OpenWSN project.
- An evaluation of path diversity over IEEE802.15.4e on a real hardware platform.
- An exploration for the centralized operations in deterministic networking.
- Design and implementation of BIER-TE protocol for 6TiSCH track forwarding.
- Implementation of centralized RPL path diversity and DAO message compression.
- Design and implementation of ARC-based track computation algorithm.

1.9 Report Structure

The rest of this report is structured as below:

- Chapter 2 illustrates techniques that are used to enable 6TiSCH centralized scheduling and multipath forwarding operations involved during this internship (1.6.1 and 1.6.2).
- Chapter 3 describes in detail the design and deployment of the testbed at Cisco PIRL, and how the forementioned enabling techniques are implemented in code (1.6.3).
- Chapter 4 presents various experiments conducted to explore the value of spatial diversity and the way for it to cooperate with other kinds of diversities (e.g. temporal and spectral) in order to improve the performance while keeping the power consumption at a low level (1.6.4).
- Chapter 5 deepens the study with a discussion of some critical parts during the internship.
- Chapter 6 concludes the report.

2. 6TISCH CENTRALIZED SCHEDULING AND MULTIPATH

2.1 Overview

In most cases, it would be too complex to do operations like complex path computation with replication and elimination, or precise scheduling along a path in a distributed manner, therefore, an SDN-like centralized approach would be more effective and preferred. As is introduced in the section 1.3.3, 6TiSCH architecture [19] defines a remote monitoring and scheduling management paradigm and a Track Forwarding (TF) model for TSCH LLNs adopting a centralized architecture, under the control of a PCE. This internship works on the major operations for 6TiSCH centralized scheduling with multipath forwarding capabilities, which include but not are not limited to:

- Discovering and exposing the 6TiSCH topology to the PCE
- Complex path computation and Traffic Engineering-like explicit routes setup
- Multipath forwarding with packet replication, elimination and retries based TSCH
- Detecting and rerouting around interferences

This chapter presents mechanisms designed and implemented in this internship for the operations as forementioned, and is structured as follows. Section 2.2 illustrates a centralized approach for RPL to enable path diversity by making each mote announce multiple parents in its DAO messages to the DAGroot on non-storing mode. Section 2.3 introduces a variation of ARC, which is used to compute the complex path with a shape of a ladder to support packet replication and elimination operations. Section 2.4 explains the design of BIER-TE protocol to enable multipath forwarding with a capability of controlling packet replication and elimination activities. Section 2.5 discusses how the schedule is centrally managed and how the resources are reserved for a track along the complex path from the source to the destination based on BIER-TE. Section 2.6 analyses how the feedback bitString can be used to detect transmission failures, and the PCE accordingly adjusts replication and elimination activities for the ensuing packets by updating the input bitString.

2.2 Centralized Path Diversity for RPL

In many cases there is a need to maintain a preferred parent and several candidate parents for each mote to provide path diversity and fast recovery mechanisms. This could be achieved by either a centralized approach or a distributed, while the centralized one is normally considered to be easier and more efficient. Thus in this internship, we have designed and implemented a centralized path diversity mechanism for RPL.

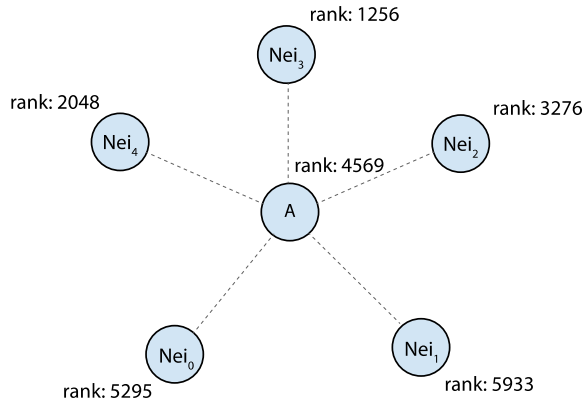
2.2.1 Parent Selection

RPL uses *rank* to represent a node's individual position relative to other nodes with respect to a DODAG root. The *rank* strictly increases in the down direction and decreases in the up direction. The RFC 6550 [6] and RFC 6552 [33] defines general behaviors that a mote should follow to select an ordered list of parents based on the *rank*, however, the exact *rank* calculation is implementation

dependent. In this section we only discuss how the alternate parents are selected out from the rest of neighbors in addition to the preferred parent with a calculated *rank* associated to each. The rules are quite simple:

- A neighbor cannot be both preferred parent and alternate parent at the same time.
- The neighbor with a smaller *rank* would be more preferred.
- Neighbors that are not in the same DODAG are ignored.
- Neighbors that are of equal or greater Rank than the node are ignored.

As shown in Figure 15, the mote A maintains a table of neighbors with some additional parameters in the same DODAG. According to the rules, it would select neighbor 3 as a preferred parent with the highest preference, and neighbor 4, neighbor 2 as its candidate parents with less preferences. Neighbor 0, neighbor 1 would not be selected as their *ranks* are larger than mote A.



Neighbor Index	DAG rank	Parent Preference	...
0	5295	0	
1	5933	0	
2	3276	4	
3	1256	5	
4	2048	3	

Figure 15 - Multi-parent selection example

2.2.2 Destination Advertisement

As introduced in section 1.3.6, the Destination Advertisement Object (DAO) is used to propagate destination information upward along the DODAG. The RFC 6550 defines two modes that RPL can operate at. In Storing Mode, the DAO message is unicast by the child to the selected parent(s), and nodes store downward routing tables for their sub-DODAG. RPL routes messages downward by the IPv6 destination address. Each hop on a downward route in a storing network examines its routing table to decide on the next hop. In Non-Storing mode, a node uses DAOs to report its DAO parents to the DODAG root by unicast, and nodes do not store downward routing tables. Downward packets are routed with source routes populated by the DODAG root. The centralized path diversity mechanism depends on the Non-Storing mode to propagate parentship information of each mote to the DAGroot, which is normally directly connected with the PCE. Through this way, the PCE should be able to have a global view to support the path redundancy, and the nodes in network should maintain much less information compared with approaches with Storing mode.

To make it clear, all the operations described hereafter are based on Non-Storing mode, as Storing mode has a separate set of rules and is not what we are caring about. In Non-Storing mode, the Transit Information option in DAO message can be used for a node to indicate its parents to the DAGroot, which is collecting the DODAG routing information for the purpose of constructing source routes. The format of the option is as below:

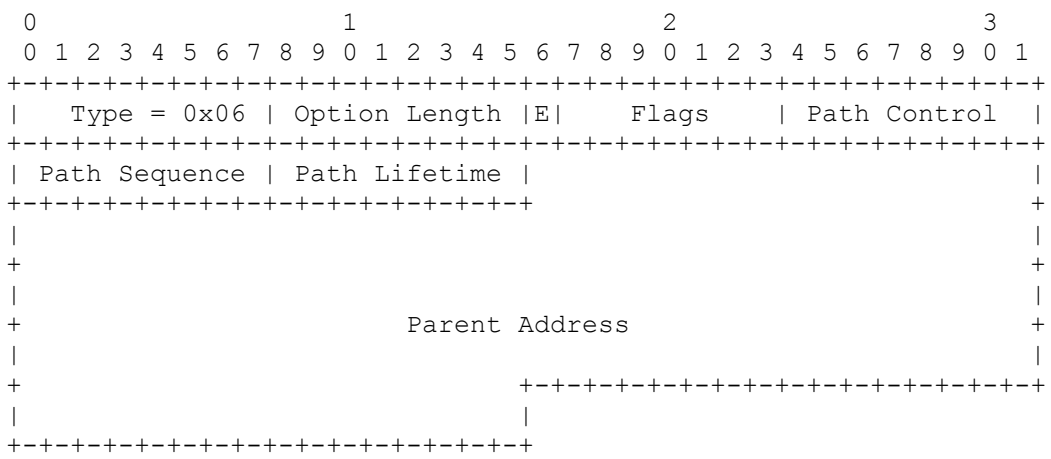


Figure 16 - Format of the Transit Information Option

For each node with multiple parents, it may include a Transit Information option for each parent in a DAO message. The Path Control field can be used to signal a preference among the parents, which may influence the decision of DAGroot when selecting among the alternate parents/paths for constructing downward routes. One or more Transit Information options must be preceded by one or more RPL Target options. In this way, the RPL Target option indicates the child node, and the Transit Information option(s) enumerates the DODAG parents.

However, since IEEE802.15.4 specifies a Maximum Transmission Unit (MTU) of 127 bytes, each DAO message can actually contain only one transit information option, which brings the need for the work on DAO compression.

2.2.3 Topology Exposed by RPL DAO Messages

With all the nodes reporting their parentship information to the DAGroot, the DAGroot should be able to calculate the source routes for the downward packets. Multipath diversity can be enabled by many cases, such as when a node has at least two parents sharing a common grandparent, when a node shares one or more common parents with its another parent, and when a node shares a common parent with one of its grand parent, etc. For example, as shown in Figure 17, the arrows with a same color indicate the parentship information for a node being reported to the DAGroot, and a bold arrow means a preferred parent. With these information, the DAGroot can have the knowledge of all possible paths to a node which forms a general shape of a ladder with crosses in between the two sides.

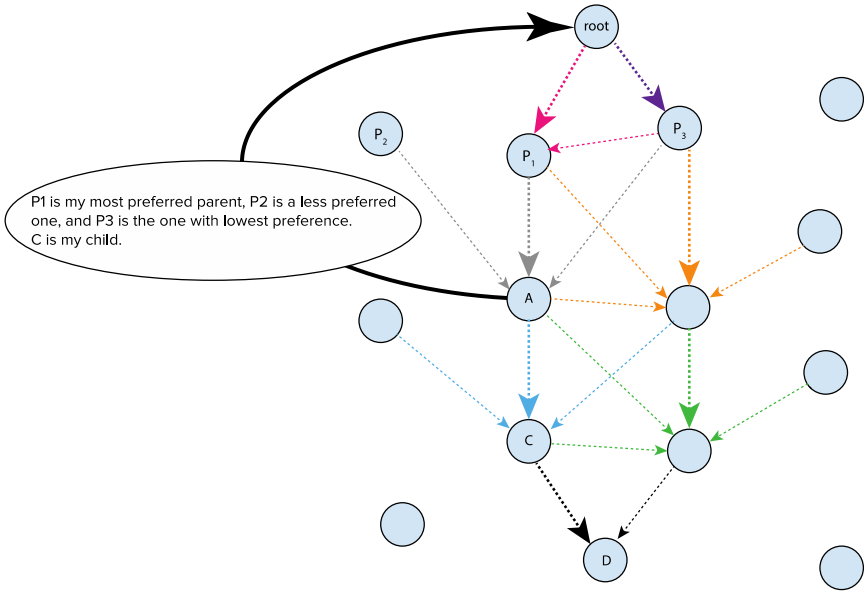


Figure 17 - RPL path diversity

2.3 Centralized Multipath Construction with ARC

In the section 1.3.7 we introduced the ARC algorithm which can be used to build the forwarding path between a source and a destination follows a cascade of ARCs. In this section we present a variation of ARC that can be leveraged to compute a complex path, which we call an ARC chain shaped like a ladder with two parallel paths between a source and a destination, and then steps that connect the two paths along the way to support packet replication and elimination.

The ARC technique is applied to build the ladder in a way as illustrated below with an example. Assume that we have a topology as shown in Figure 18, and the goal is to protect the wireless path from A to E with a secondary path in parallel and as many connection links as possible in between to avoid single point of failure. Thus we protect the path hop-by-hop starting from the destination.

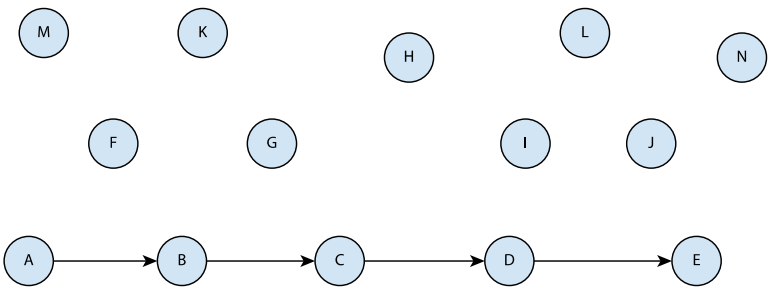


Figure 18 - an example for ARC chain computation

In order to illustrate better, we use the same convention as used in the section 1.3.7 for diagrams.

Step 0. Create a safe set, add the destination node into the safe set.

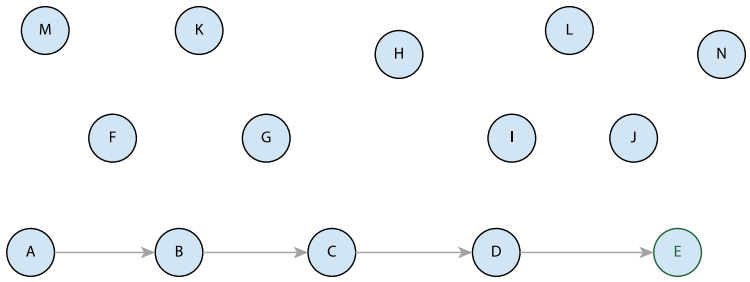


Figure 19 - Step 0. Create a safe set

Step 1. Select the first safe node in path, which is E, and the last unsafe node in path, which is D, and find a secondary SPF path from D to E, which in this case is D→I→J→E.

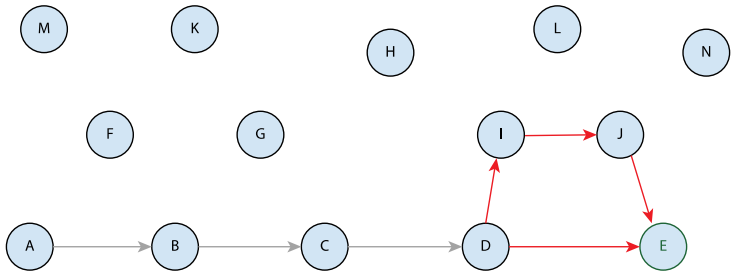


Figure 20 - Step 1. Protecting the last hop

Select the last unsafe node in path, which D, and the last unsafe node off path, which is J, and make the links in between reversible. This way the first ARC is constructed, which is {E}-{D-I-J}-{E}, and all the intermediate nodes in this ARC become safe with each having two outgoing links to the destination node.

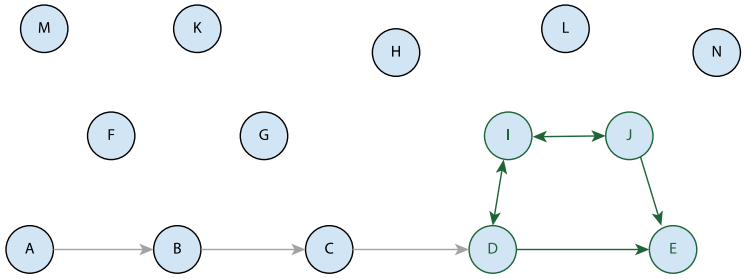


Figure 21 - Step 1. Building the first ARC

Step 2. Similar to Step 1., select the first safe node in path, which is D, and the last unsafe node in path, which is C, and find a secondary SPF path from C to D, with the last hop being safe and off path, which in this case is C→G→H→I→D. And same as before, select the last unsafe node in path, which is C, and the last unsafe node off path, which is H, and make the intermediate links reversible. The second ARC is then formed as {D}-{C-G-H}-{I} and the nodes C, G, H become safe.

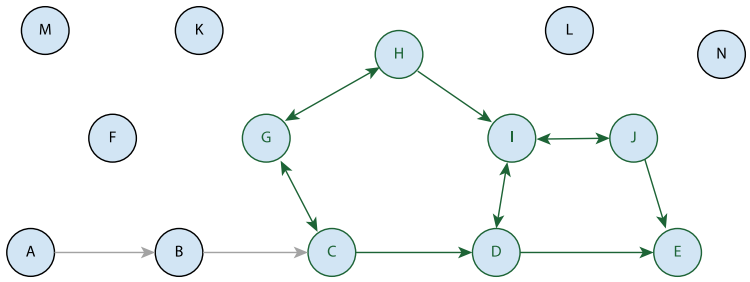


Figure 22 - Step 2. Building the second ARC

Step 3. Same as Step 2., select B and C, find the secondary SPF path B→G→C with the last off-path hop G being safe, and make the ARC {C}-{B}-{G}. Node B becomes safe.

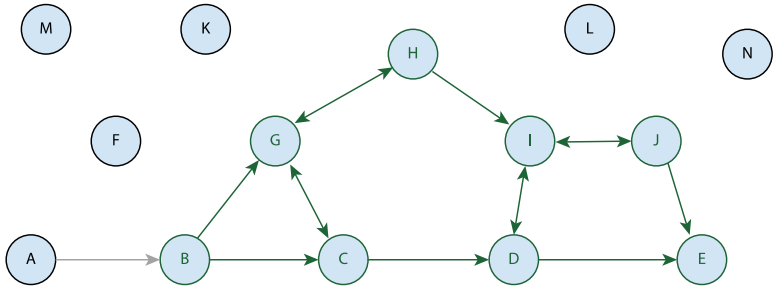


Figure 23 - Building the third ARC

Step 4. Again we find secondary path from A to B as A→F→G→B, and make the last ARC as {B}-{A-F}-{G}. Finally, we have formed an ARC chain:

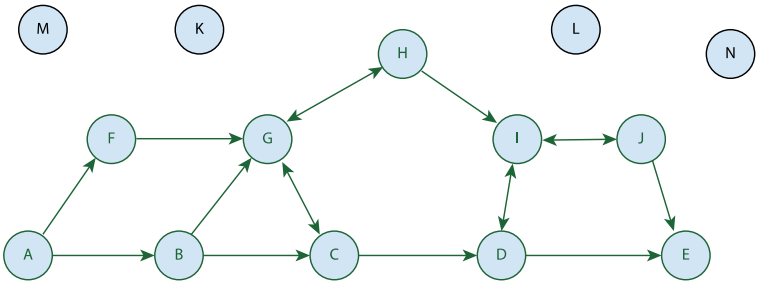


Figure 24 - Formation of the ARC chain

Please note that, if there is no secondary path with last hop being safe and off-path can be found, we may want to use the shortest secondary path to protect the link, otherwise more retries for this hop, but in this way, the receiving node of this hop still becomes the single point of failure along the path.

2.4 Packet Replication and Elimination with BIER-TE

As introduced in section 1.3.5, after the centralized controller computes the complex paths based on ARC, it reserves timeslots for each transmission on the TSCH schedule, as show in Figure 5, and installs the BIFT, as shown in Figure 6, on each node along the computed path. In this way, packets on this explicit path are transmitted according to the schedule, replicated and eliminated according to the BIFT at each hop in network. Before we discuss further how the path is set up by the controller, in order to make the whole picture of this internship can be fully understood, in this section we present the BIER-TE protocol, which is the key enabler of packet replication and elimination activities in forwarding plane. The mechanism is mainly designed and implemented by Zacharie Brodard and more details can be found in his report [18].

2.4.1 Data Structures

Timeslot: a timeslot is a cell in the CDU matrix as presented in the Figure 14, which is uniquely identified by the tuple (slotOffset, channelOffset).

Bundle: same as the bundle concept introduced in the section 1.3.2. Since each bit in a BIFT of a node is associated with an adjacency, which is represented by a bundle for a half-duplex link or a pair of bundles for a full-duplex link, thus each bit in a bitmap is actually associated to one bundle in case of unidirectional adjacency or a pair of bundles in case of bidirectional adjacency.

Track: In our case a track is a merge of concepts of 6TiSCH track [28] and BIER subdomain [15]. As a WSN can potentially consist of hundreds or thousands of nodes, assigning a bitIndex to each of the adjacencies would lead to a huge bitmap size. to overcome this, we rely on a controller to compute and set up the BIER-TE tracks, which are directional paths between the sources and the destinations. For each BIER-TE track, an associated bitmap is used to control the forwarding of the packets on this track with packet replication and elimination capabilities. Non-BIER-TE tracks can co-exist with BIER-TE tracks, which means a node can be on several BIER-TE tracks and on several non-BIER-TE racks.

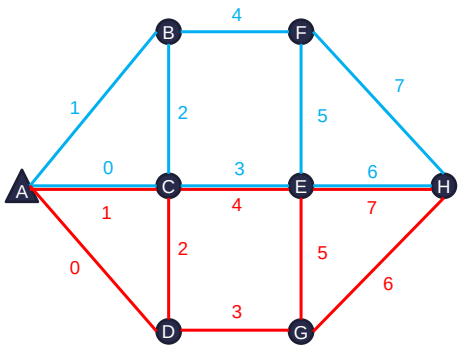
2.4.2 BIER-TE Track Forwarding

Track Forwarding is a forwarding model corresponding to the remote monitoring and scheduling management paradigm with a PCE. This model can effectively be seen as a Generalized Multi-protocol Label Switching (G-MPLS) operation, in that the information used to switch a frame is not an explicit label, but rather related to other properties of the way the packet was received, a particular cell in the case of 6TiSCH [29]. This is particularly elegant, as the track associated to a transmitted data can be deduced from the timeslot it was received within, making it unnecessary to add the trackID information in the message.

Each track is associated with a succession of paired bundles, a receiving bundle from the previous hop and a transmitting bundle to the next hop, dedicated for each transmission along the track. A packet that is forwarded along a track normally has a destination MAC address of broadcast, and will be only forwarded within slots belonging to this track. Only one packet with its copies can be sent on a track per slotframe iteration.

A BIER-TE track normally takes a complex path with redundant routes from the source to the destination, and associates each bundle in the track with a specific bitPosition of the BIER bitmap. In this way, the path a BIER-TE packet actually take to the destination is composed of a succession of links represented by bundles whose bits are SET in the bitmap of the packet.

Figure 25 gives an example showing the data structure. Please note that a bundle normally contains multiple slots for retries with the same neighbor, and if the transmission fails in the first slot with no acknowledgement received, next slots of the same bundle can be used to retransmit the packet.



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)

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

Figure 25 - an example of data structure [18]

2.4.3 Forwarding Rules

In order to illustrate better, the operations inside a node on different timeslots are briefly presented as below. More details are described in Zacharie's report [18].

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.

Note that for each BIER-TE track, the controller reserves 2 buffer spaces in each node on the track for the storage of a packet and its copy. Also note that as a bitPosition is associate with a pair of bundles for transmissions in both directions, a packet received from a bundle would never be sent out on the paired bundle to the same neighbor. For example, in the case of Figure 25, track 1 has a pair of bundles between C and E with a common bitIndex 3, if E receives a packet from C on slot 10, as the bit 3 of this packet has already been RESET to '0' by C, it thus would not send the packet on slot 12.

2.4.4 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.

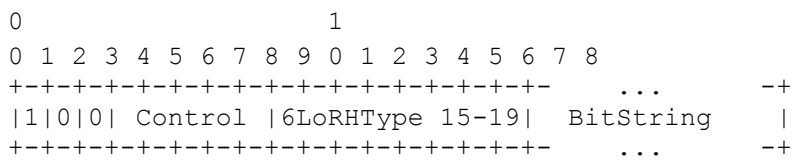


Figure 26 - The BIER-6LoRH

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

2.5 Centralized Scheduling and Track Reservation

In this section we discuss how the controller manages the global schedule, and how it signals the nodes in order to perform track reservation and other operations. We assume that the topology is exposed by RPL multipath mechanism, and a complex path has already been computed in a form of an ARC chain.

2.5.1 Data Structures

On the node side, a scheduled cell is normally associated with a set of information as shown below.

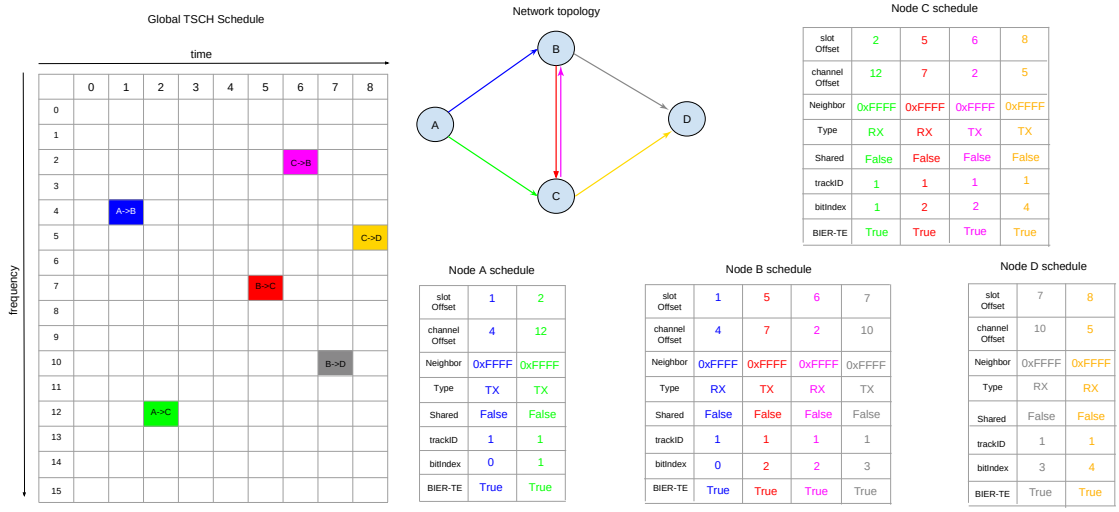


Figure 28 – an example for the information associated with each scheduled slot on a node

Note that the BIER-TE Boolean is used to distinguish if the slot belongs to a BIER-TE track.
On the controller side, a scheduled cell is associated with the information set in a slightly different format, which is shown as below in Table 1.

slotOffset	channelOffset	txNode	rxNodeList	shared	trackID	bitIndex	BIER
1	4	A	B	False	1	0	True
2	12	A	C	False	1	1	True
5	7	B	C	False	1	2	True
6	2	C	B	False	1	2	True
7	10	B	D	False	1	3	True
8	5	C	D	False	1	4	True

Table 1 - an example for the information associated with each scheduled slot on the controller

In order to schedule a transmission, the controller should at least reserve a TX cell on one transmit node and a RX cell at same (slotOffset, channelOffset) on one or more receive nodes by command. The command used to manage schedule on each node should follow a common format with some specific fields, which can be generally presented as below:

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 2 - example commands to node A and B for the transmission A to B

The Node filed is used to indicate the node that a given command is targeted to.
The Type field indicates the command format, so that it can be correctly parsed on the node side.
The FrameID specifies the targeted slotframe, as there can be multiple schedules in the network.
The Action field indicates the type of operation that is to be taken on that schedule.
The Cell Information field gives parameters about the cell in detail for the operation.
As all packets are sent with a broadcast destination in their mac header in our case, thus we choose not to specify the neighbor for each slot.

2.5.2 6TiSCH Centralized Scheduling

We define following operations that a controller is supposed to take in order to centrally manage the network schedule.

Schedule monitoring. The controller should monitor the state of schedules running on each node, and accordingly maintain an exact global schedule for the network. The mechanism should be able to ensure the state of the global schedule on the controller are synchronized with local schedules on each node, and be able to detect and correct any failure of local-to-global cell mapping.

Schedule editing. The controller should be able to manipulate the schedules on each node with a granularity of cell at real time, or operating with bundles can also be possible. This capability is a must to perform Track Reservation.

Schedule recovery. The controller should be able to correct any unexpected errors of the schedules on the nodes. For example, an unexpected reboot of a node would cause all the previous schedules lost, and it is the responsibility of the controller to recover all the scheduled cells.

2.5.3 Track Reservation with ARC and BIER-TE

This section discusses how a track is reserved based on the ARC chain. Figure 29 gives an ARC {I}-{H-G-C}-{D} with directed links computed by the ARC algorithm. Each link is tagged with a number for a better illustration.

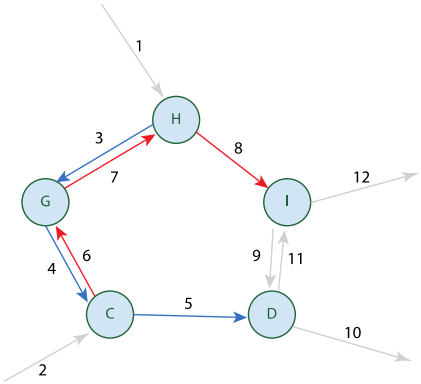


Figure 29 - an example for track reservation

The scheduling rules are generally as below:

1) Unlike the order of construction of the ARCs, the scheduling should start from the source to the destination, from higher ARCs to lower ARCs. We normally should make the first transmission inside an ARC happen only after all transmissions of the higher ARC finish if the higher ARC is rooted only at this ARC. For example, the transmission 1 and 2 should be scheduled before all the transmissions in the lower ARC. The bitIndexes should also be assigned along with the scheduling, starting from the source to the destination.
2) For the transmissions inside an ARC, there can be two scheduling solutions. We can make the two outgoing transmissions, 5 and 8 in our case, happen only after all the intermediate transmissions finish, which result in a scheduling order like {3, 4, 6, 7, 5, 8}. We may want a packet copy reach the next ARC as soon as possible, in this case we can also choose to just take care of the transmissions in a same color as shown in the figure. In this way the scheduling order can also be {3, 4, 5, 6, 7, 8}. Note that the assignment of the bitIndexes inside an ARC should generally also follow the transmission order, and only one bitIndex is assigned for the paired bundles. There is also a proposal on that only one bitIndex is necessary for bundles 3, 4, 6 and 7, as in this case node G does not need to do any switching.

3) The scheduling order of the transmission direction for the intermediate bundles should be interleaving among ARCs. For example, the intermediate bundles are scheduled with an order {3, 4, 6, 7}, which is downward direction first and upward later in this ARC, then for the next ARC, the intermediate bundles should be scheduled with an order with upward direction first and downward later, which in this case should be {11, 9}. This is discussed in detail in the section 5.1 for deepening.

After the track is reserved with a complex path from the source to the destination, each node along the path is scheduled with a set of bundles reserved for the track, of which the first is a RX bundle from the previous hop and the last is a TX bundle to the next hop. There may be intermixed RX and TX bundles, though having all the RX bundle first is preferable.

2.6 Reactive BitString Feedback Control

The controller should maintain information for each track somehow in order to manipulate the bitStrings, while of course all the information can be deduced from the global schedule.

After the track is set up, a bitString is assigned for this track to the source node. For each packet that belongs to this track, the source node would insert the associated bitString into the BIER-6LoRH of the packet, and send it on the reserved TX bundle of this track to next hops.

2.6.1 The Construction of BitStrings

The bitString represents the actual path that the controller planed for a packet along a given track. The exact way it is constructed depends on the QoS policy and also the conditions of the network. For example, we might want the packets are forwarded along a single path when the link conditions are satisfying to save energy. However, when the packet loss ratio goes higher, we might then want to enable packet replication on some additional links.

Each ARC can be seen as an individual independent safe domain. Assume that the original single path goes through the links 2, 5, 10 in Figure 29, and if now the link 5 goes bad, then we might want to enable the bits associated with links 6, 7, 8, 9 to protect the link 5, so that the packets will be replicated at node C with a copy to D and a copy to G, and later eliminated at node D.

2.6.2 The feedback of BitStrings

As is illustrated in the Figure 7 in the section 1.3.5, the output bitString can be the feedback to the controller to deduce the link condition and the packet delivery ratio, so that the controller can thus modify the bitString accordingly to dynamically adjust packet replication and elimination activities based on the QoS requirements. A simple control loop is implemented during the internship which will be presented in the next Chapter. Please note that a transmission failure at any hop along a single path would make no bitString can be received at the destination.

3. TESTBED IMPLEMENTATION

3.1 OpenMote and OpenWSN

OpenMote [31] is the hardware used during this internship. It is an open-hardware prototyping ecosystem designed to accelerate the development of the Industrial Internet of Things (IIoT). The OpenMote-CC2538, shown in Figure 30, is the core of the ecosystem which provides computation and communication capabilities. Its main component is Texas Instrument CC2538: a SoC with a Cortex-M3 microprocessor and a radio transceiver operating at 2.4GHz band [18].

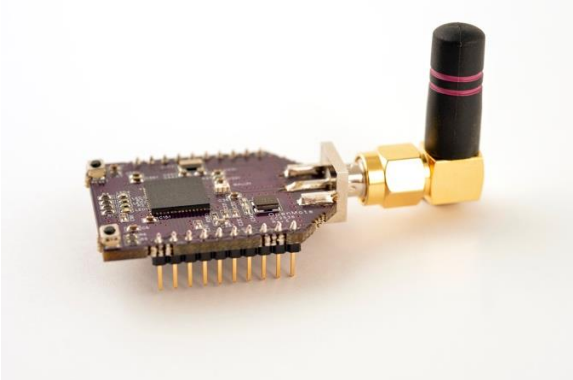


Figure 30 - OpenMote-CC2538 Hardware (from OpenMote website)

OpenWSN [37] project is an open-source implementation of protocol stacks that based on IoT standards, using a variety of hardware and software platforms. With a suite of free and open-source debugging and integration tools, it aims to help academia and industry verify the applicability of these standards to the Internet of Things, for those networks to become truly ubiquitous. It is the first open-source implementation of the IEEE802.15.4e standard [32] and also fully support the OpenMote-CC2538. Figure 31 depicts the protocol stack implemented in the OpenWSN firmware with pure-C code running on the motes.

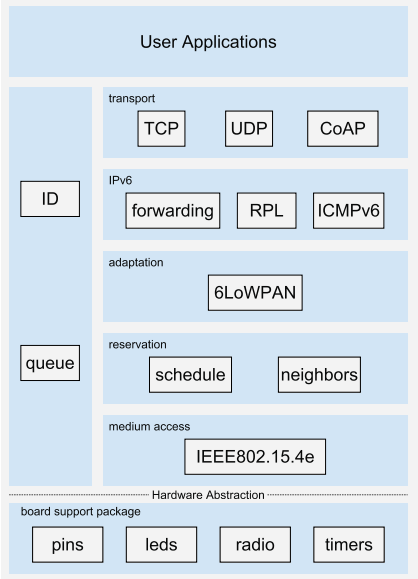


Figure 31 - OpenWSN protocol stack

OpenVisualizer, a software offered by OpenWSN, is the main part that this internship works on. It is a Python-based debugging and visualization program that runs on a PC and interacts with the motes connected to it over the serial ports. OpenVisualizer contains the following capabilities [37]:

- Connects your OpenWSN network to the Internet over a virtual interface.
- Shows internal state (neighbor table, scheduling table, queue, multi-hop routing graph, etc.) of each mote physically connected to the OpenVisualizer, as is shown in Figure 32.
- Runs simulation via OpenSim, which allows coexistence of real and simulated motes.
- Sends commands to the motes directly connected (e.g., toggle the DAGroot) and receives debugging information sent from motes.

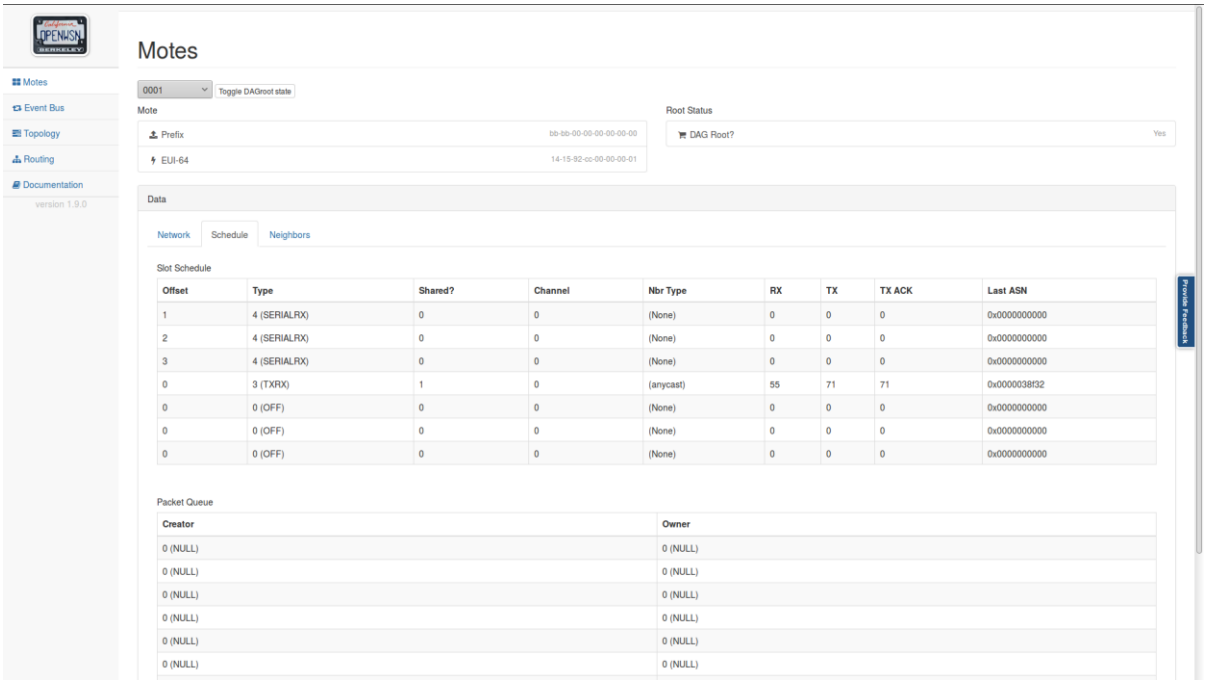


Figure 32 - OpenVisualizer Web UI

OpenVisualizer is also used to facilitate IPv6 functionality, by allowing users to connect to an LBR. Aside from providing a visualization framework, OpenVisualizer is composed of a modularized Python framework, which can be used easily to write powerful client-side applications interfacing with the network [32]. Figure 33 shows the general architecture of OpenVisualizer, in which:

- **openTun** creates and uses a local IPv6 TUN interface to communicate with an application on the same computer as the Event Bus, and implements mesh-to-Internet and Internet-to-mesh message handling.
- **LBR** provides border router translation between IPv6 packets on the external network and 6LoWPAN packets on the LLN. It generates (IP, proto, ...) events for ICMP messages from the LLN DAG root, which then may be handled by the RPL component.

- **RPL** manages 6LoWPAN routing. It reads DAO messages from the LLN and passes 'updateParents' messages to the path state components to build a local representation of the network topology. RPL also generates the DIO messages for the LLN.
- Each **moteConnector** is a primary bus interface for each of individual motes. It generates the 'fromMote.{evtType}' events out from the motes and onto the bus. If the mote represents a DODAG root, it forwards 'bytesToMesh' events out to the motes. Each moteConnector communicates with its serially attached mote via a **moteProbe**.
- Each **moteState** provides a local cache of the current state of each attached mote, via the 'fromMote.status' events. It also provides 'cmdToMote' to send an out of band command to the mote.

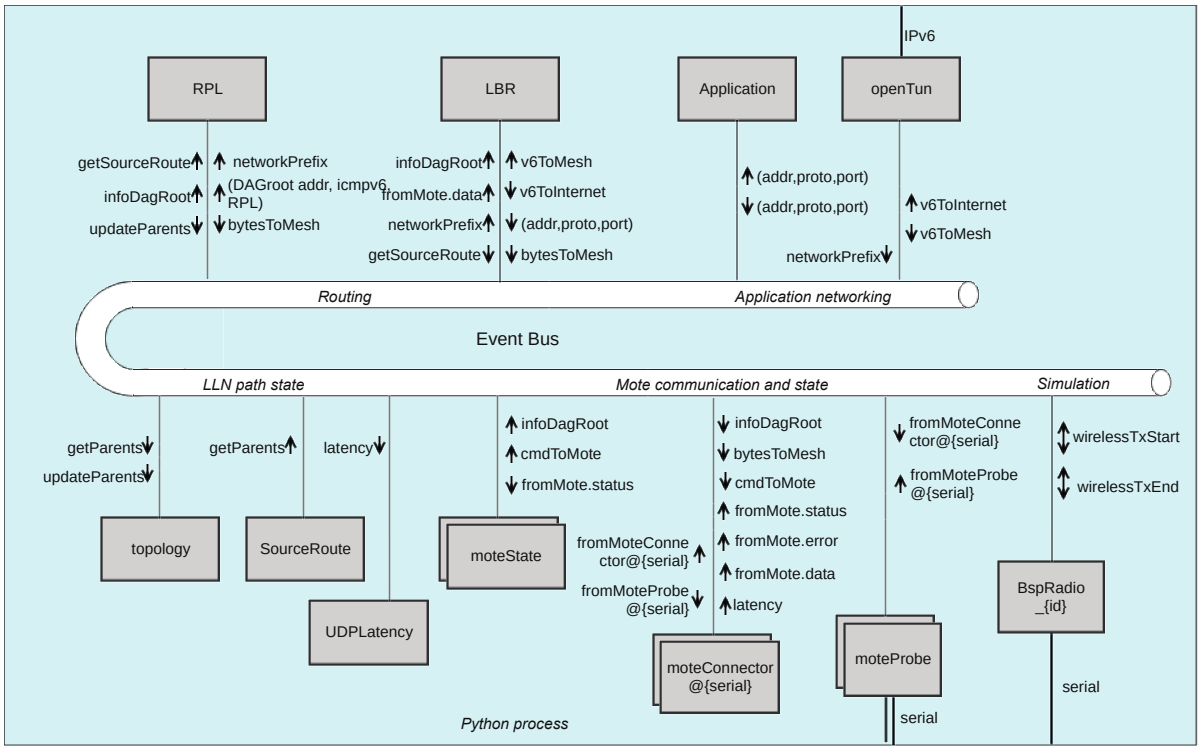


Figure 33 - OpenVisualizer architecture

A detailed description about OpenVisualizer and its architecture can be found on the OpenWSN website [37].

3.2 Design Goals

In order to implement and evaluate the techniques mentioned in Chapter 2, and to further conduct multipath experiments, a testbed is deployed at PIRL with a LLN composed of eight OpenMotes running OpenWSN firmware code in the data plane and a central PC running OpenVisualizer in the control plane.

3.2.1 Requirements

The testbed is supposed to satisfy the following requirements:

- Have a data plane with a low-power and lossy network composed of the motes dispatched in the whole building.
- Have a central PC running OpenVisualizer which can remotely monitor and manage the network as well as each individual mote without locally connected through serial.
- The network should be as 'real' as possible with some 'good' links and 'bad' links randomly, and the motes should be 'far' away from each other enough to enable multi-hop forwarding, while also be 'close' to each other enough to support multi-path for packet replication and elimination.
- The central PC should be able to perform 6TiSCH minimal centralized operations such as topology learning, path computation, resource reservation and release, bitString feedback control and path lifecycle management.
- The data plane should support BIER-TE forwarding with the capability to replicate and eliminate packets in addition to performing retries and channel hopping.

3.2.2 Design Pattern

As is shown in Figure 34, OpenVisualizer keeps track of the network topology in real time. Upon receiving a new flow, it computes a complex path, reserves timeslots, and installs BIFT into nodes for each segment forming a track in schedule from the source to the destination. An associated bitmap for this flow is then assigned by OpenVisualizer to the ingress node, and inserted into the BIER header of each packet. For each node along the path receiving the packet, it examines bitmap against its BIFT to perform packet forwarding, replication or elimination operations. When the destination node receives the packet, it forwards the payload data to the upper layer and feeds the bitmap back to OpenVisualizer. The OpenVisualizer then examines the output bitmaps, deduces transmission failures, and accordingly adjust the bitStrings for next packets.

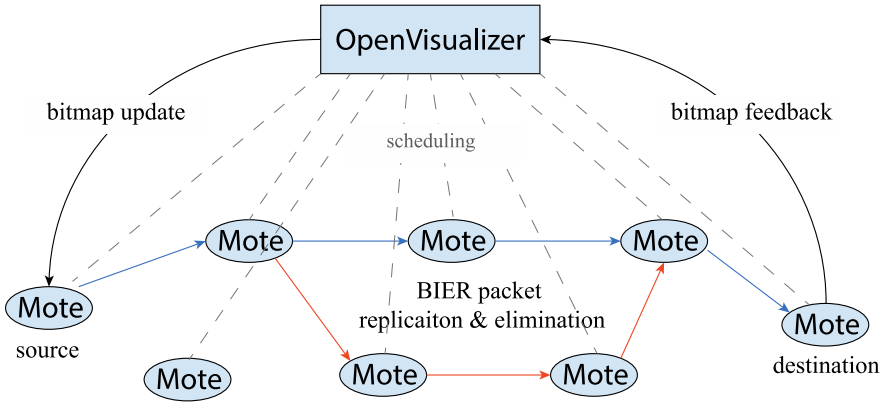


Figure 34 – General work pattern of the testbed

3.3 Architectural Components

3.3.1 Testbed Architecture

Figure 35 depicts the architecture of the testbed, in which each OpenMote is directly connected to a Raspberry Pi. OpenVisualizer runs on the Raspberry Pi that directly connects the DAGroot. ('OV' means OpenVisualizer, 'Rover' means the program running on the Raspberry Pi that bridges data from the serial port to the central OV and vice versa through TCP connections). Each mote-RaspberryPi pair is dispatched and plugged in the company's network. All the controlling messages from motes are transferred to the OV and all the commands from OV are transferred to the motes by the Rovers.

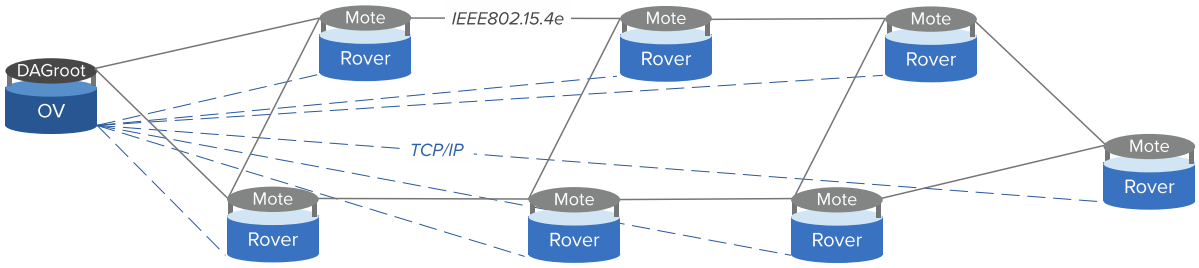


Figure 35 - Testbed architecture

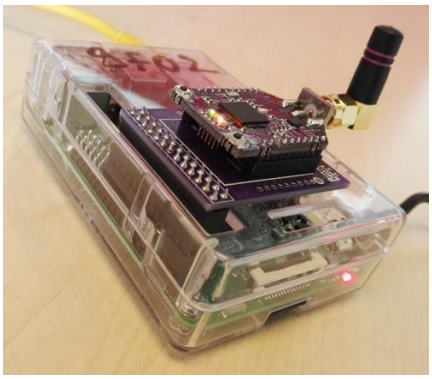


Figure 36 - an OpenMote-RaspberryPi pair

3.3.2 Main Components

remoteConnectorServer and remoteConnectorRover. As OpenVisualizer is originally designed to monitor only the motes that locally connected through serial ports, the first task would thus be to enable remote monitoring and management capabilities, therefore, we decide to run the part of OpenVisualizer that handles the serial communication with the motes on each of the Pies, and develop new components in OpenVisualizer to enable communications with those remote parts. We add a remoteConnectorServer component for OpenVisualizer and a remoteConnectorRover component for the Rover to bridge all signaling data between the motes and the OV, so that all OpenVisualizer functionalities work exactly as if the devices are connected locally with the serial links.

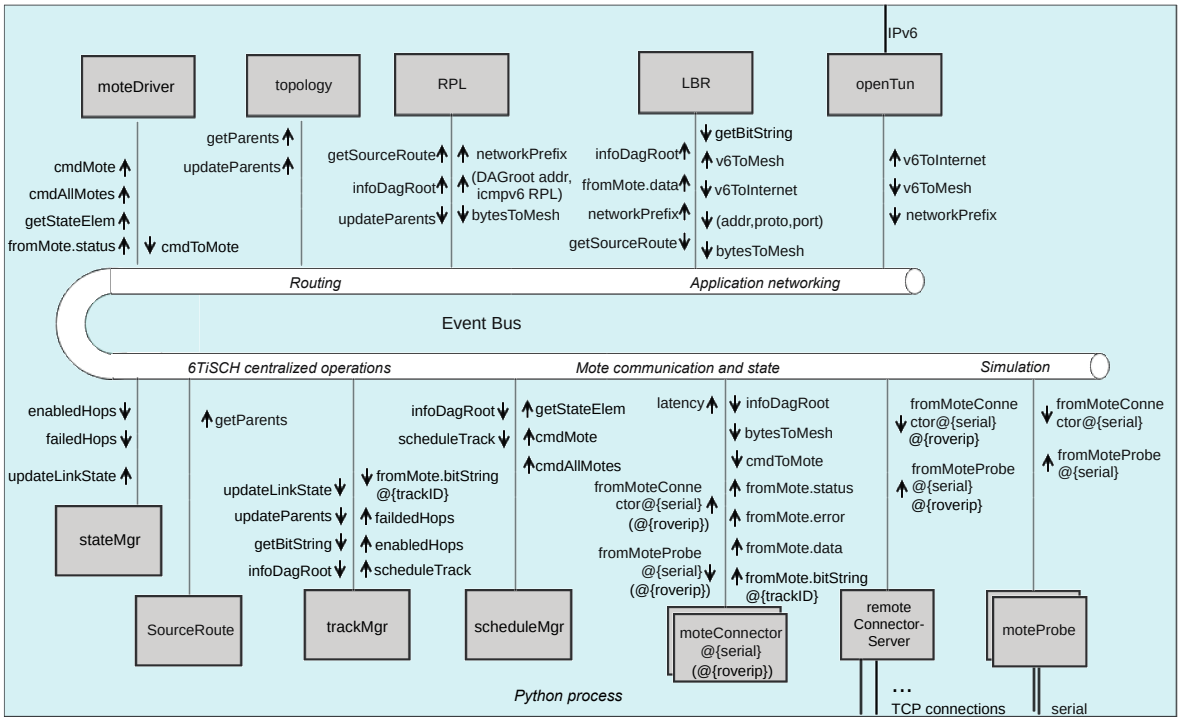


Figure 37 - New architecture of OpenVisualizer

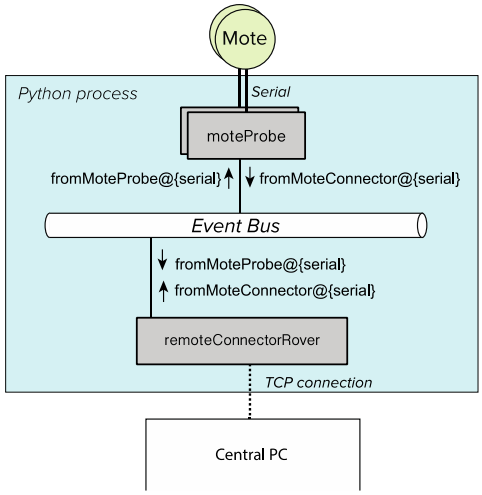


Figure 38 - Rover architecture

As is shown in Figure 37, the remoteConnectorServer uses rover IP and serial port to distinguish motes from different Rovers. It uses ZeroMQ to publish 'fromMoteConnector@*' events on its 50000 port and subscribe to the 50000 port of each Rover. In Figure 38, the remoteConnector-Rover uses ZeroMQ to publish 'fromMoteProbe@{serial}@{roverIP}' events on its 50000 port and subscribe to the 50000 port of the central PC.

This feature has been fully integrated into the main branch of OpenWSN, a detailed description can be found at their website [38]. Figure 39 shows the WebUI which is used to establish ZeroMQ connections to the Rovers with two steps:

- Select local IP address that is used to publish events.
- Add/upload a list of rover IP addresses to subscribe to.

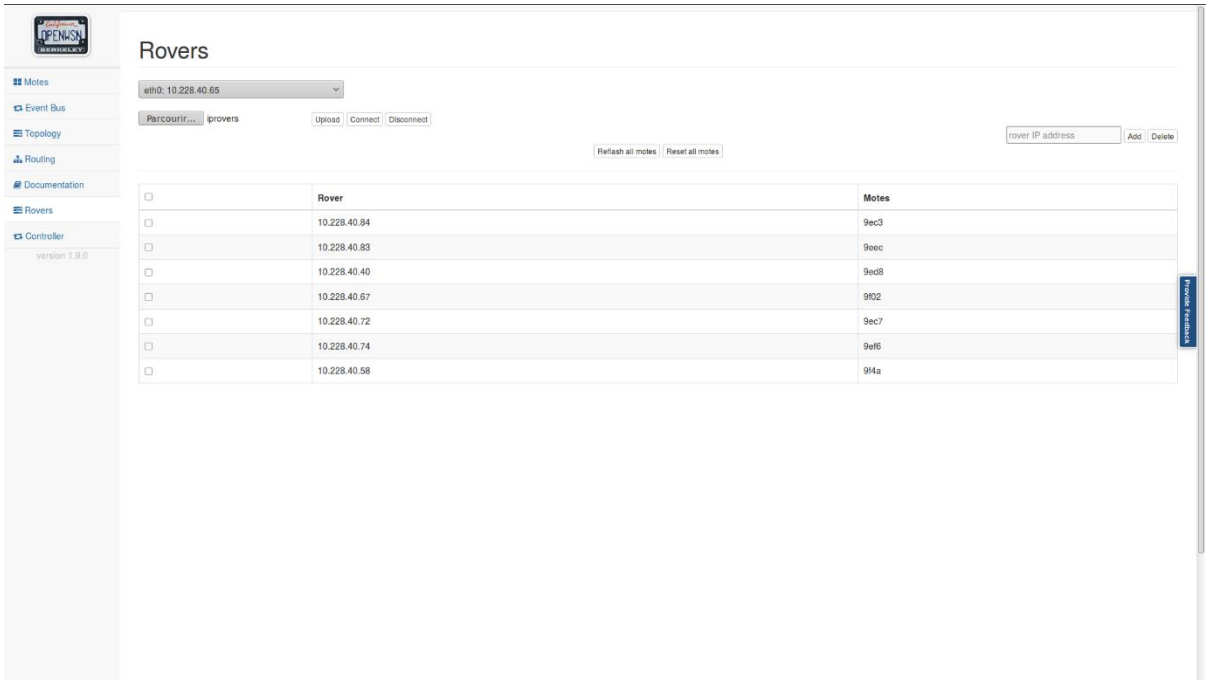


Figure 39 - Rover WebUI

trackMgr. It learns the topology of network from centralized path diversity from RPL, computes and maintains multi-path tracks for different flows. It responses to the LBR component with the bitmap used to insert into the packet's BIER header, and interprets the feedback bitmap reported by the destination mote into real-time link status. The ARC algorithm is implemented here.

scheduleMgr. It maintains information about the global schedule, and receives commands from the trackMgr component to perform track reservation operations. It also provides WebUI through which the global schedule (including tracks, BIFTs, etc.) can be displayed and configured in real time.

stateMgr maintains the link status information for the bitmap control. It monitors the conditions of each link deduced from feedbacks of bitmap.

moteDriver contains all the moteState instances and maintains a local cache of the current state of each mote from the 'fromMote.status' events. It also provides 'cmdToMote' to send commands to the motes.

3.3.3 Interaction Diagrams

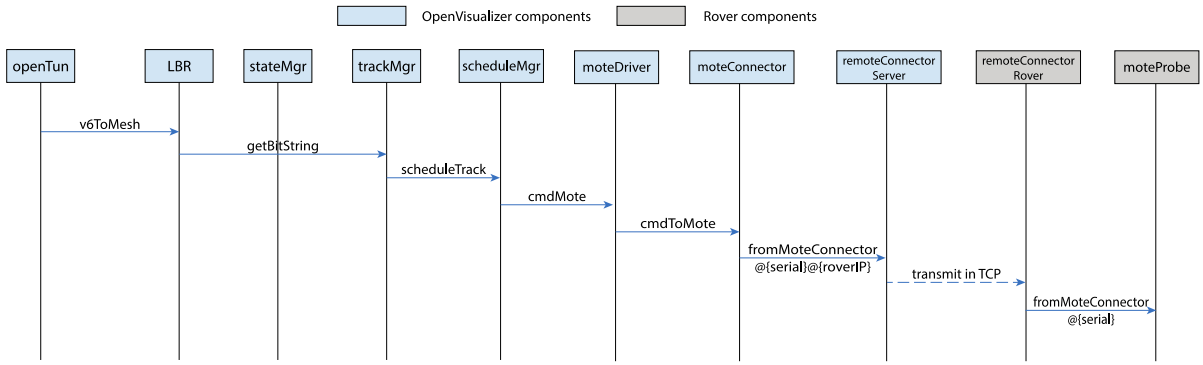


Figure 40 - Track reservation interaction diagram

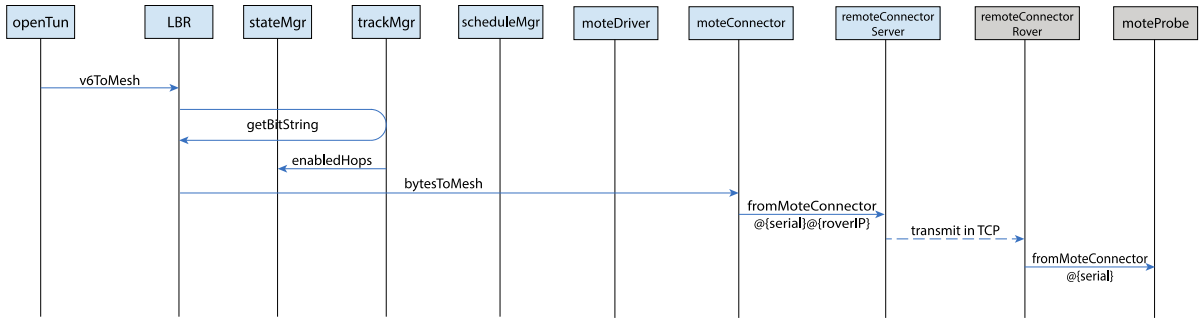


Figure 41 - Internet-to-mesh BIER forwarding message handling

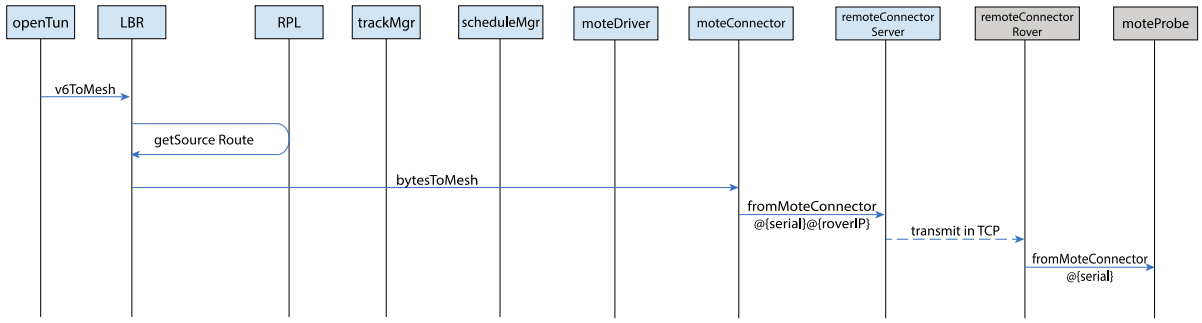


Figure 42 - Internet-to-mesh RPL source routing message handling

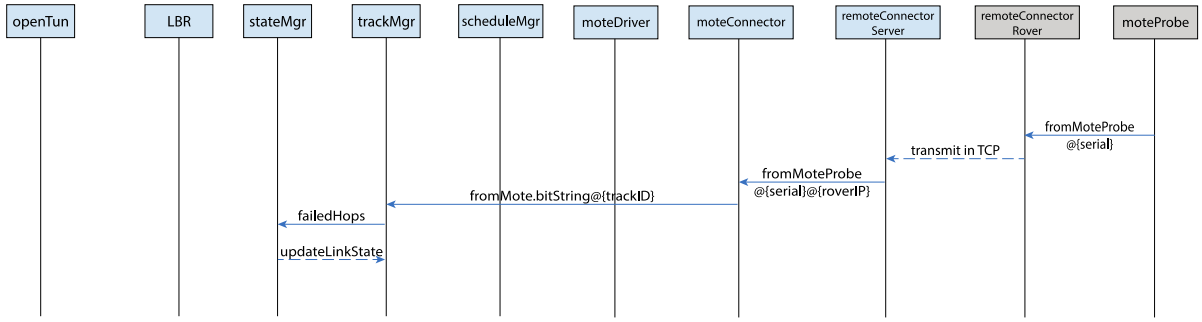


Figure 43 - BitString feedback

3.4 Implementations

3.4.1 Centralized Path Diversity for RPL

RPL Multi-Parent Selection Algorithm. Originally in OpenWSN, a mote selects and maintains only one preferred parent, and announces this single parent in its DAO messages to the DAGroot in Non-storing mode [6]. In order to make a mote select one preferred parent while maintaining multiple candidate parents, a new algorithm is designed and implemented in C as below:

```
uint8_t neighbors_getPreferredParentEui64(uint8_t neiIdxArray[MAXPREFERENCE]) {
   uint8_t    i,j,k;
   uint8_t    foundParentInUse,foundCandidate;
   dagrank_t  minRankValArr[MAXPREFERENCE];
   uint8_t    minRankIdxArr[MAXPREFERENCE];
   foundParentInUse     = 0;
   foundCandidate       = 0;

   for (i=0;i<MAXPREFERENCE;i++){
      minRankValArr[i]=neighbors_vars.myDAGrank;
      neiIdxArray[i]=MAXNUMNEIGHBORS+1;
      minRankIdxArr[i]=MAXNUMNEIGHBORS+1;
   }
   //===== step 1. Try to find parent set
   for (i=0; i<MAXNUMNEIGHBORS; i++) {
      if (neighbors_vars.neighbors[i].used==TRUE){
         if (neighbors_vars.neighbors[i].parentPreference >0) {
            neiIdxArray[MAXPREFERENCE-neighbors_vars.neighbors[i].parentPreference] = i;
            foundParentInUse++;
         }
         if (neighbors_vars.neighbors[i].DAGrank < minRankValArr[MAXPREFERENCE-1]) {
            //Found parent candidate with neighbor index i
            for (j=MAXPREFERENCE-1;j>=0;j--){
               if (j == 0 || neighbors_vars.neighbors[i].DAGrank > minRankValArr[j-1]){
                  //if candidate rank larger than parentIdx j-1
                  for (k=MAXPREFERENCE-1;k>j;k--){
                     //Move the parent minRankIdxArr[k-1] from index k-1 to index k
                     minRankValArr[k] = minRankValArr[k-1];
                     minRankIdxArr[k] = minRankIdxArr[k-1];
                  }
                  //Insert nei i at index j, break
                  minRankValArr[j] = neighbors_vars.neighbors[i].DAGrank;
                  minRankIdxArr[j] = i;
                  if (foundCandidate<MAXPREFERENCE){foundCandidate++;}
                  break;
               }
            }
         }
      }
   }
   //===== step 2. (backup) Update parent set if any changes
   if (foundParentInUse < foundCandidate){
      // reset preference for old parents
      for (i=0;i<MAXNUMNEIGHBORS;i++) {
         neighbors_vars.neighbors[neiIdxArray[i]].parentPreference = 0;
      }
      // assign new preference for new parents
      for (j=0;j<foundCandidate;j++){
         neighbors_vars.neighbors[minRankIdxArr[j]].parentPreference = MAXPREFERENCE - j;
         neighbors_vars.neighbors[minRankIdxArr[j]].stableNeighbor         = TRUE;
         neighbors_vars.neighbors[minRankIdxArr[j]].switchStabilityCounter = 0;
      }
      // return an array of parents with preference in descending order
      memcpy(neiIdxArray,minRankIdxArr,sizeof(minRankIdxArr));
      foundParentInUse = foundCandidate;
   }
   return foundParentInUse;
}
```

This function writes a list of updated parents into the array passed to it in descending order of the preference, which means the first parent at index 0 is the preferred parent, while the others are the candidate parents. It selects parents and decides the preference of each by comparing the DAG rank, and only the neighbor with lower rank than itself can be a candidate parent, and the one with the lowest rank be the preferred parent.

RPL DAO Compression. Another issue is for a mote to convey the information of its multiple parents to the DAGroot via transit information options of DAO messages in non-storing mode. Since IEEE802.15.4 specifies a Maximum Transmission Unit (MTU) of 127 bytes, each DAO message can have only one transit information option actually for just one parent. Using one DAO message for each parent or using fragmentation are obviously not a wise choice, while trying to compress DAO seems to be a good way to solve the problem. Therefore, the DAO message is finally compressed as follows:

- Reserved field is elided;
- Flag Octet in option headers is elided;
- Parent prefix is elided (8 bytes address instead of 16);
- DODAGID prefix is elided (8 bytes EUI64 instead of 16);
- RPL instance ID is integrated in the DAO Flag Octet;
- Option length field is elided;
- Prefix length field in Target option is elided;
- Path lifetime field in Transit option is elided;

And the compressed DAO message can convey 1 Target Option and up to 5 Transit Information Options with the parent preference specified in the Path Control field.

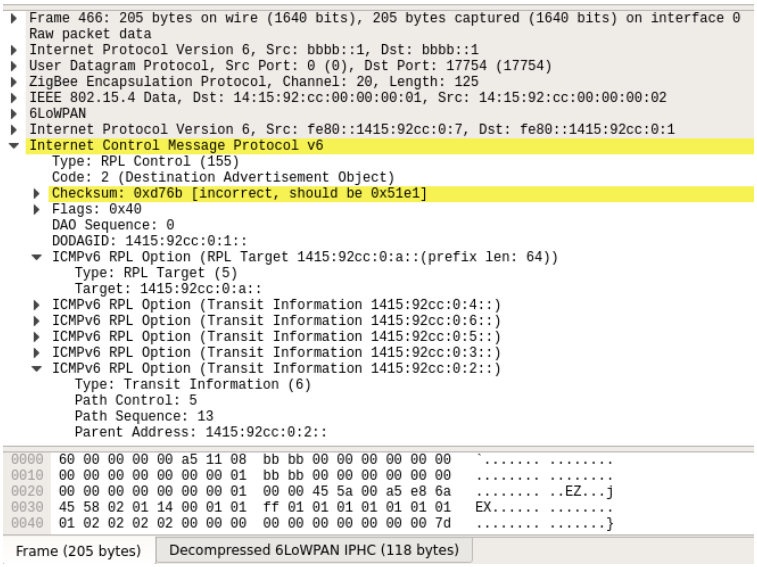


Figure 44 - Compressed DAO message

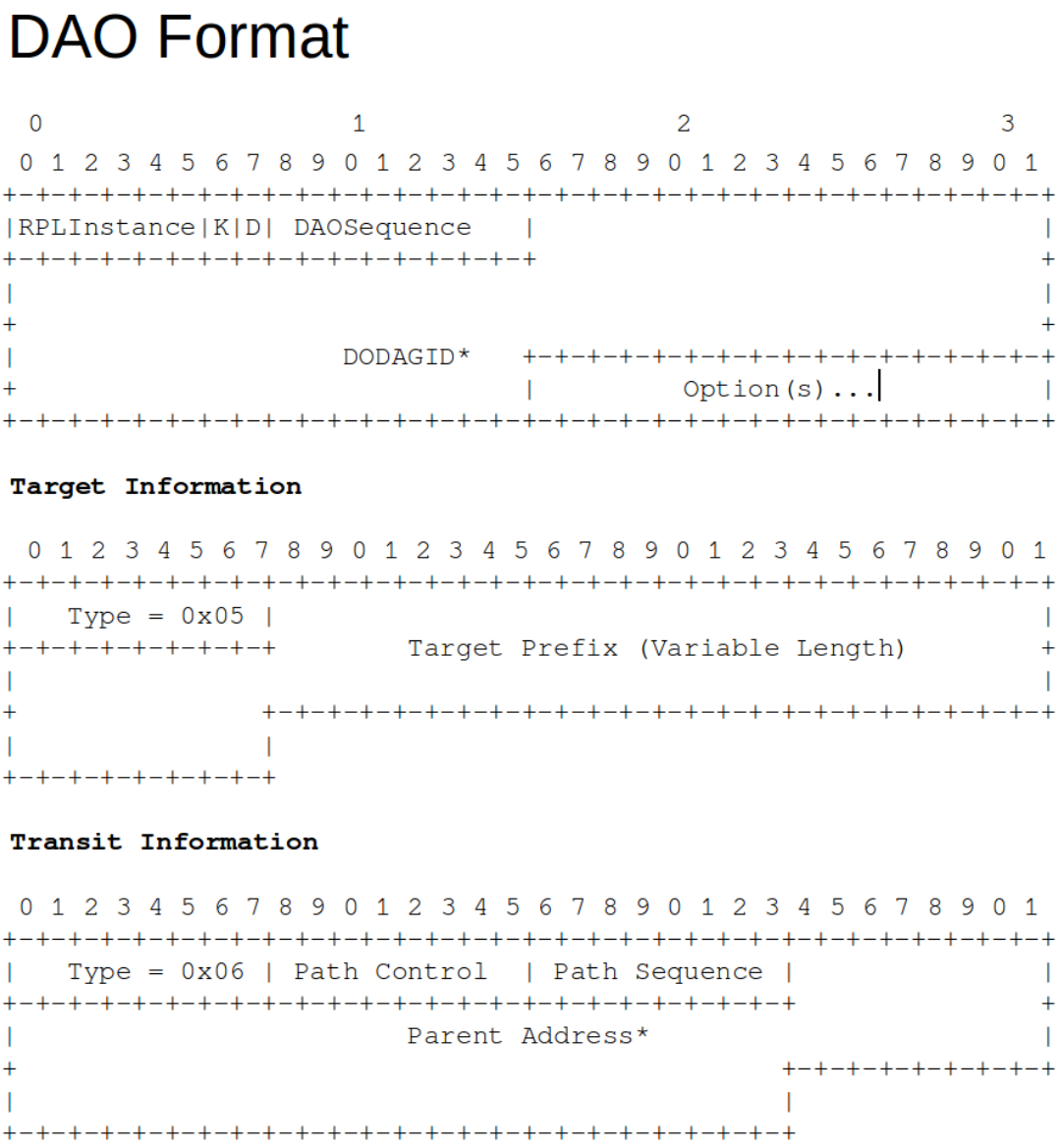


Figure 45 - Compressed DAO format

Figure 45 gives the format of compressed DAO message. Please note the compression is not fully considered and is only used for the proof of concept of centralized RPL path diversity.

OpenVisualizer Topology Learning. As is forementioned, the OpenVisualizer RPL component reads DAO messages from the LLN, and compute source routes for the downward (Internet-to-mesh) packets based on the parenthood information. The right figure below shows printed DAO messages received from a LLN with the topology shown in the left. The value behind each parent is the preference for the parent, and in our case the maximum preference is set to 5, which means a preferred parent. The captures are from simulation for a better illustration, however please note that the mechanism has been implemented and worked on the testbed with real motes as well.

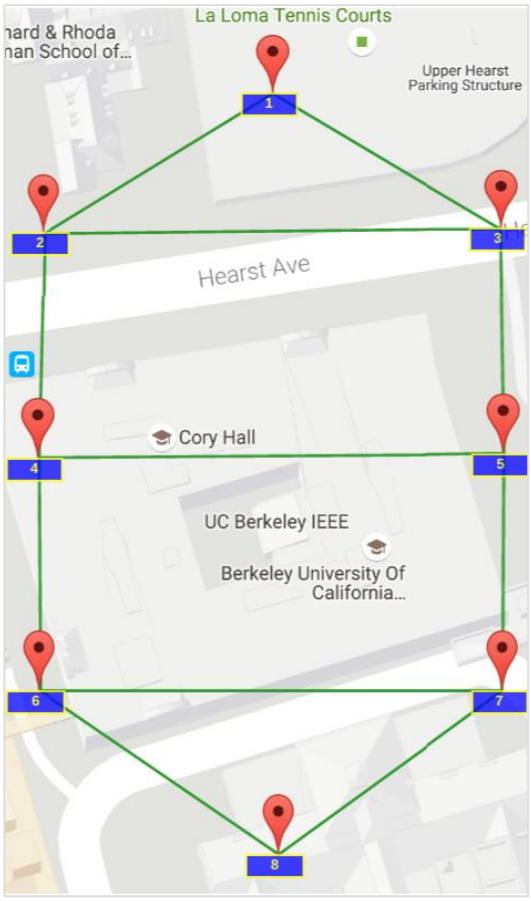


Figure 46 - Example topology

```
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 47 - DAO messages

And the WebUI displays the OpenVisualizer's view of network topology deduced from DAOs:

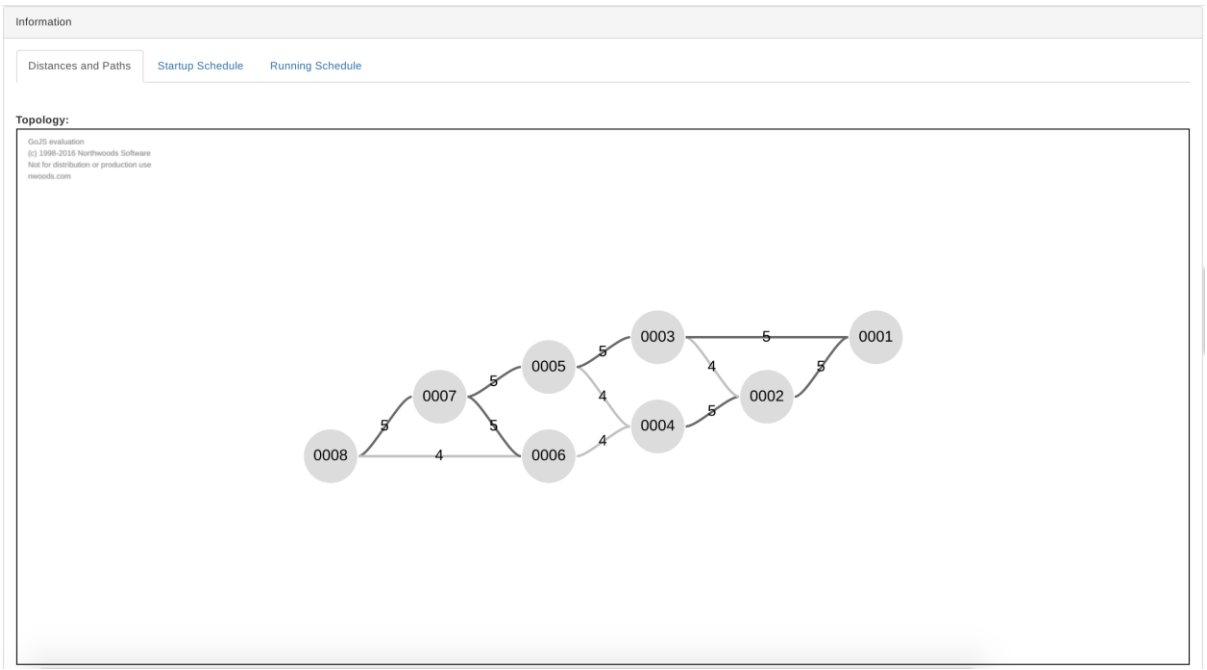


Figure 48 - OpenVisualizer's view of topology

3.4.2 ARC based Multipath Construction

Code Implementation. In order to be able to set up complex paths to avoid single point of failure and support packet replication and elimination, the ARC algorithm is implemented in Python as below. This function recursively invokes itself and builds an ARC at a time to protect each hop along its shortest path to Omega. It finally generates an ARC chain forming a track, and dispatches a command to scheduleMgr to reserve timeslots on the schedule of each mote along the track.

```
def _buildTrack(self, tracker):
    '''
    returns a tracker class
    '''
    # if the first hop is safe, complete
    if tracker.srcRoute[0] in tracker.track:
        # track reservation
        self.dispatch('scheduleTrack', (tracker.trackId, tracker.arcs))
        tracker.postInit()
        # return a tracker instance to manage this track
        return tracker
    # find the last single point of failure node and first safe node along its path to Omega
    _0hop = [node for node in tracker.srcRoute if node in tracker.track][0]
    _1hop = [node for node in tracker.srcRoute if node not in tracker.track][-1]

    ARC   = namedtuple('ARC', 'bits edges arcPath hop')
    # find a list of candidate paths between this two nodes
    safeNode1 = _0hop
    altPaths = list(nx.shortest_simple_paths(self.topo, _1hop, _0hop))[1:]
    arcPath = []
    arcBits = []
    arcEdges= []

    # find a secondary path whose last hop is already safe but not in the original route
    for altPath in altPaths:
        if altPath[-2] in tracker.track and altPath[-2] not in tracker.srcRoute:
            arcPath = altPath
            break
    # choose the shortest secondary path if not find such a path;
    if not arcPath:
        arcPath = altPaths[0] if altPaths else [_1hop, _0hop]
    # identify intermediate nodes and edge nodes
    medNodes = [node for node in arcPath if node not in tracker.track]
    safeNode2 = arcPath[arcPath.index(medNodes[-1]) + 1]
    medNodes.reverse()
    # construct ARC and assign bit Index
    preHop = safeNode2
    for nexHop in medNodes:
        arcEdges.append((nexHop, preHop, {'bit': tracker.bitOffset}))
        tracker.bitOffset += 1
        arcBits.append(tracker.bitOffset)
        preHop = nexHop
    # reverse the intermediate nodes
    medNodes.reverse()
    bits = arcBits[:]

    preHop = safeNode1
    if _1hop != tracker.srcRoute[0]:
        for nexHop in medNodes:
            arcEdges.append((nexHop, preHop, {'bit': bits.pop()}))
            preHop = nexHop
    else:
        arcEdges.append((medNodes[0], preHop, {'bit': bits.pop()}))

    tracker.bitOffset += 1
    tracker.track.add_edges_from(arcEdges)
    # add this ARC to the ARC chain
    tracker.arcs.append(ARC(bits=arcBits,edges=arcEdges,arcPath=arcPath,hop=(_1hop,_0hop)))
    # recursive invocation
    return self._buildTrack(tracker)
```

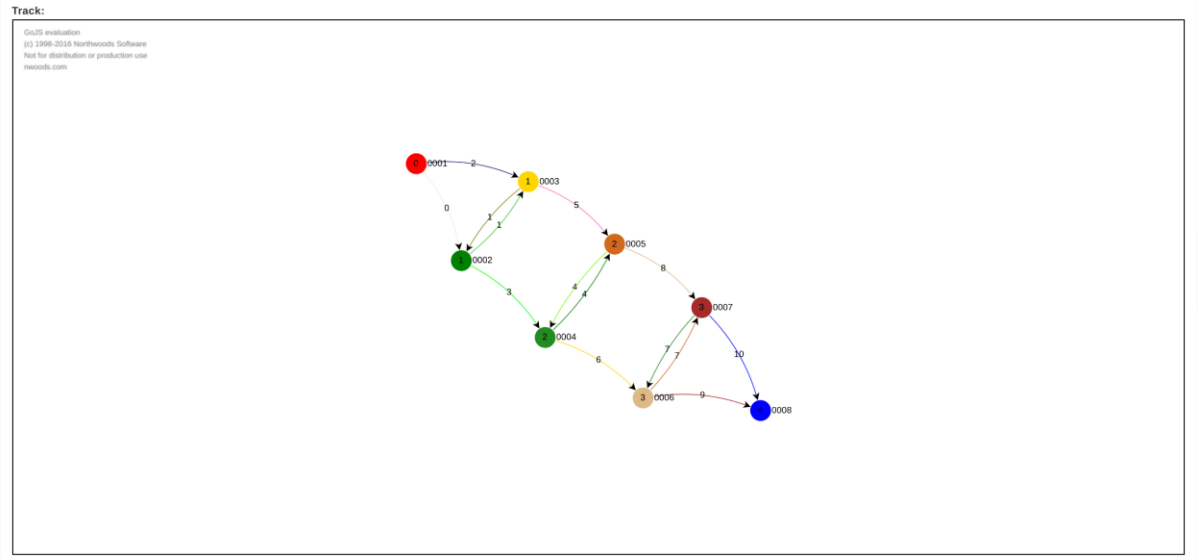
In the example below, the DAGroot is trying to send ping messages to the mote '0008'. (For the information about internal interactions of OV components please refer to the section 3.3.3). As is shown in Figure 49, the first packet is lost due to the delay for track reservation, while subsequent pings are encapsulated in BIER packets and successfully forwarded.

```
hao@pc:~$ ping6 -s 10 -c 5 bbbb::1415:92cc:0:8
PING bbbb::1415:92cc:0:8(bbbb::1415:92cc:0:8) 10 data bytes
18 bytes from bbbb::1415:92cc:0:8: icmp_seq=2 ttl=64
18 bytes from bbbb::1415:92cc:0:8: icmp_seq=3 ttl=64
18 bytes from bbbb::1415:92cc:0:8: icmp_seq=4 ttl=64
18 bytes from bbbb::1415:92cc:0:8: icmp_seq=5 ttl=64

--- bbbb::1415:92cc:0:8 ping statistics ---
5 packets transmitted, 4 received, 20% packet loss, time 4006ms
```

Figure 49 - pings to the mote '0008'

Figure 50 shows the captures from WebUI. The topology of network learned from RPL is already shown in Figure 48, where the number on the link is the parent preference. Figure 50(a) shows the track taken by the ping messages. An arrow means a scheduled transmission, and the number on a link means the assigned bitPosition for the transmission in the sender's BIFT. Figure 50 (b) is the global schedule in which the track is reserved from slotOffset 4 until 17.



(a)

Information

Distances and Paths | Startup Schedule | Running Schedule

Running Schedule: **DAGroot:** 0001 **FirstFreeSlot:** 18 / 20

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	SHARED	SERIALRX	SERIALRX	SERIALRX	0001-0>0002	0002-1>0003	0001-2>0003	0003-1>0002	0002-3>0004	0004-4>0005	0003-5>0005	0005-4>0004	0004-6>0006	0006-7>0007	0005-8>0007
1															
2															
3															
4															
5															
6															
7															
8															
9															
10															
11															
12															
13															
14															
15															

	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	0007-7>0006	0006-9>0008	0007-10>0008												

(b)

Figure 50 - Track reservation WebUI captures

3.4.3 Centralized Scheduling and Track Reservation

We have defined 7 actions that the controller can use to manage the schedule on each node:

- ADD: reserve a slot, return Unavailable Error if the specified slot is already scheduled.
- OVERWRITE: overwrite a scheduled slot for other use with new information. Return Unscheduled Error if the specified slot is originally unscheduled.
- REMAP: move a scheduled slot to a new (slotOffset, channelOffset). Return Unscheduled Error if the slot to be moved is not scheduled. Return Unavailable Error if the new slot is already scheduled for other use.
- DELETE: release a slot from current use, making it unscheduled and available for other reservation. Return Unscheduled Error if the slot is already unscheduled.
- LIST: get information about all the scheduled cells on a node.
- CLEAR: clear all the slots belonging to a BIER-TE track on a node. Return Unscheduled Error if no slot is scheduled for the given track.
- SETFRAMELENGTH: configure the frame length of the schedule before the initiation.

And the Python code to configure slots with these actions:

```
def configSlot(self, operation, slotList):
    '''
    configs slots on slot frame

    '''
    with self.frameLock:
        slotFrame = self.slotFrame[:]
    for slotInfo in slotList:
        shared = slotInfo[self.PARAMS_SHARED]
        occupied = slotFrame[slotInfo[self.PARAMS_SLOTOFF]][slotInfo[self.PARAMS_CHANNELOFF]]
        if shared:
            if operation== self.OPT_ADD and occupied:
                continue
            slotInfo[self.PARAMS_TYPE] = self.TYPE_TXRX
            self._cmdAllMotes(['schedule',
                               self.frameID,
                               operation,
                               slotInfo])

        else:
            txMote = slotInfo[self.PARAMS_TXMOTEID]
            rxMoteList = slotInfo[self.PARAMS_RXMOTELIST]
            if txMote in rxMoteList:
                rxMoteList.remove(txMote)
            if txMote:
                slotInfo[self.PARAMS_TYPE] = self.TYPE_TX
                self._cmdMote([txMote],
                              ['schedule',
                               self.frameID,
                               operation,
                               slotInfo])

            if rxMoteList:
                slotInfo[self.PARAMS_TYPE] = self.TYPE_RX
                self._cmdMote(rxMoteList,
                              ['schedule',
                               self.frameID,
                               operation,
                               slotInfo])
```

The Python code to configure the parameters, e.g., frame length, of the schedule:

```
def configFrame(self, operation, slotFrameInfo=None, moteList="all"):
    '''
    configures a slotFrame

    :param: slotFrameInfo: an element of stored slotFrames

    '''

    if moteList == 'all':
        self._cmdAllMotes(['schedule',
                           self.frameID, operation,
                           slotFrameInfo])
    else:
        self._cmdMote(moteList,
                      ['schedule',
                       self.frameID, operation,
                       slotFrameInfo])
```

The Python code to perform track reservation with the ARCs built as presented in the last section:

```
def installTrack(self, trackId, arcs):
    '''
    installs a track on slot frame

    '''
    self._update()
    with self.frameLock:
        slotFrame = self.slotFrame[:]

    slotList = []
    for arc in arcs:
        for (rxMote, txMote, bitDict) in arc.edges:
            if [None]*self.CHANNELS in slotFrame:
                slotOff = slotFrame.index([None]*self.CHANNELS)
            else:
                log.debug('Warning! No enough available slots')
                return
            slotList.append({
                self.PARAMS_TXMOTEID: ''.join(['%02x' % b for b in txMote[6:]]),
                self.PARAMS_RXMOTELIST: [''.join(['%02x' % b for b in rxMote[6:]])],
                self.PARAMS_BITINDEX: bitDict['bit'],
                self.PARAMS_TRACKID: trackId,
                self.PARAMS_SHARED: False,
                self.PARAMS_CHANNELOFF: self.CHANNELOFF_DEFAULT,
                self.PARAMS_SLOTOFF: slotOff,
                self.PARAMS_BIER: True
            })
            slotFrame[slotOff] = [slotList[-1]]
    self.configSlot(self.OPT_ADD, slotList)
```

A demo showing RPL-based Topology Learning, ARC-based Track Computation and Reservation can be found at Youtube: https://www.youtube.com/watch?v=3LpKqtJL0Pg. Note that all slots of each track are reserved on a single channel with this function. In addition, it always reserves the first available slot one after one. In this way, if retries need to be scheduled for one hop in future, we would need to reschedule all the followed transmissions, or schedule retries for all the followed related transmissions.

For example, assume that we want to reserve the track as shown in Figure 28 on the global schedule which originally looks like as below:

Figure 51 - an example schedule before track reservation

Based on the scheduling function given above, the track would be reserved as below:

Figure 52 - an example schedule after track reservation

If a retry is needed for the transmission C→B, then we would need to reschedule the transmissions B→D and C→D a slot back:

Figure 53 - Slot rescheduling for retries

Otherwise we would need to schedule additional retries for B→D also at the end:

Figure 54 - additional retries are needed for followed transmissions

Or maybe we can leave several unscheduled slots between each transmission for future use when performing the track reservation, however it also introduces latency:

Figure 55 - Leave vacant slots between each hop along a track

However of course, this may easily be solved with TSCH.

Please note that this internship is not to design and implement a complete 6TiSCH protocol stack for centralize operations. All the controlling messages like OV commands and bitString feedbacks are transmitted through serial via TCP connection. And also the topology learning from RPL path diversity feature is also a workaround. As is discussed in [2], "6TiSCH expects the PCE commands will be issued directly as CoAP requests or be mapped back and forth into CoAP by a gateway function at the edge of the 6TiSCH network." On the other hand, "6TiSCH expects DetNet to define a protocol to proactively push the neighborhood information to a PCE. The protocol should operate over CoAP, and can carry multiple metrics, in particular the same metrics as used for RPL operations." These mechanisms are out of the scope of this internship.

3.4.4 BIER-TE Forwarding Plane

The BIER-TE forwarding protocol is fully implemented by Zacharie in the OpenWSN firmware code running on the OpenMote-CC2538 during the experiments.

Key points of the BIER-TE implementation from Zacharie's report [18]:

- Packets are never copied in memory even when a node performs a replication in order to save buffer space (and power and time).
- It allows a mote to forward both BIER-TE deterministic flows on BIER-TE tracks (that have to be set up by a controller), and stochastic flows using the already implemented RPL protocol, as shown below.

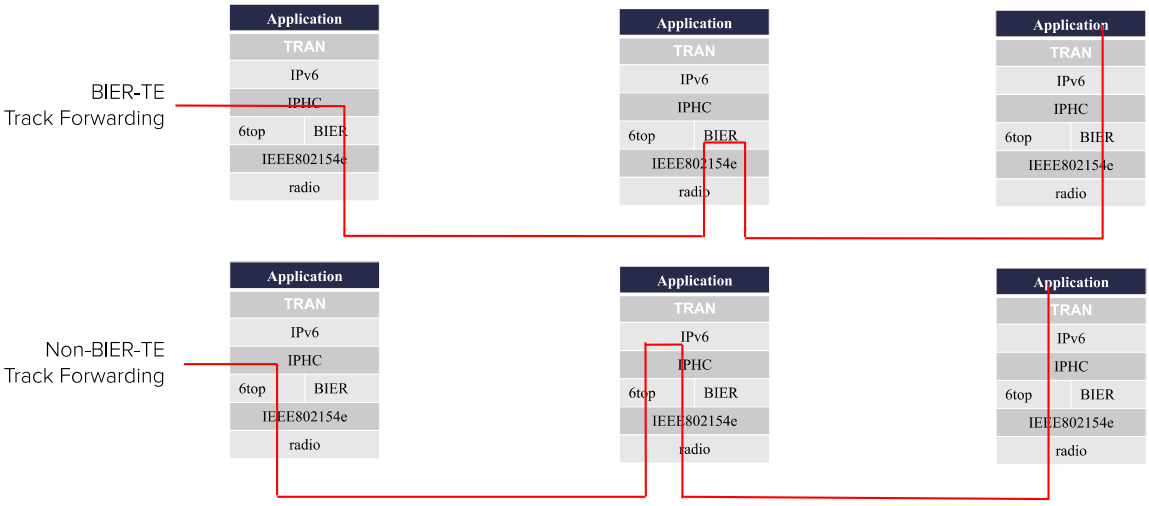


Figure 56 - BIER-TE and non-BIER-TE track forwarding

3.4.5 BitString Feedback

There can be two possible solutions on how the output bitStrings are used to deduce the network conditions. The first is that the destination node, which normally should be the DAGroot, sends the output bitString to the controller every time when it receives a BIER-TE packet, and it is the responsibility of the controller to deduce the link statuses and packet losses. Another possible way is that the destination node, or the DAGroot, do the calculation locally, and report the information to the controller periodically or upon request.

This internship implements the first solution as it is simple and efficient, and the destination node in our case has no global view of the network topology and tracks. Therefore, we define the format of a bitString feedback message as below:

Figure 57 - General format of a bitString feedback message

3.5 Deployment

The mote-rover pairs are deployed as is shown in Figure 58 in the sixth floor at PIRL. The caption aside each node is the last two bytes of the mote's EUI-64. The mote '9ED9' is the DAGroot and it is directly connected to the Raspberry Pi that is running OpenVisualizer instance, while the other Pis are running Rovers, as is shown in Figure 35. The topology is hardcoded in a general shape of a ladder, and each mote check at layer 2 if each received packet comes from one of their neighbors, if not the packet is simply dropped. The testbed has finally the following features for OV:

- Establish multiple ZeroMQ sessions concurrently;
- Remote reflash and reset the motes (developed by Zach) without touching the device;
- Topology learning, track computation, schedule management, bitString feedback;

For data plane:

- Allowing OV to change a mote's schedule in real time;
- Support packet replication and elimination using BIER-TE protocol (developed by Zach);
- Feed the output bitString back to OV from the destination mote;

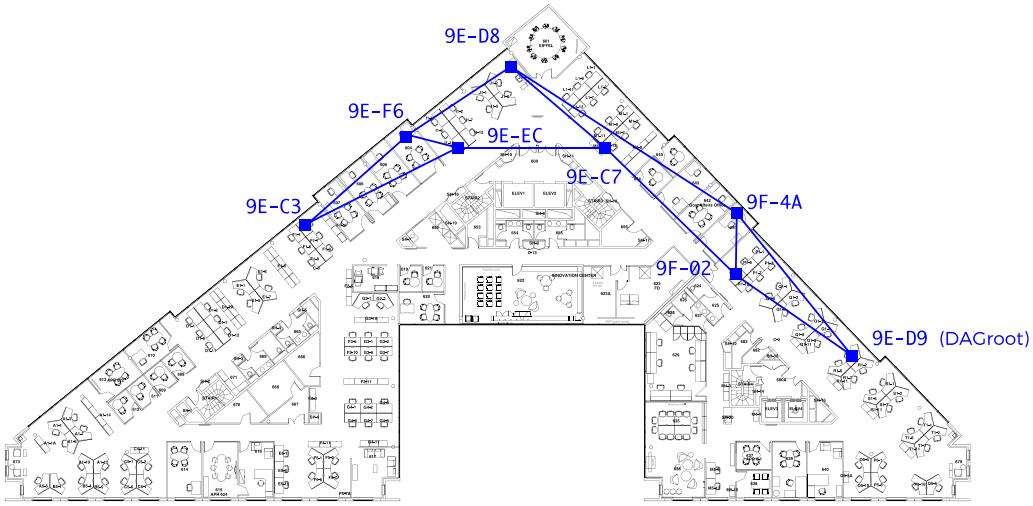


Figure 58 - Motes disposition

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

4. EXPERIMENTS

4.1 Objectives

According to the mission of this internship as introduced in the section 1.5, a set of experiments were conducted with following objectives:

- Validate the value of multipath forwarding with packet replication compared with single-path forwarding with retries;
- Compare different multipath scheduling patterns, analyze when the mechanism should be activated and how it should be incorporated with other diversity techniques such as ARQs and channel hopping.
- Explore centralized operations to optimize the scheduling, in order to minimize the jitter, latency, loss ratio, and energy consumption while maintaining the reliability at a maximum.

4.2 Principle

Therefore, we implemented a ladder-shape topology as shown in Figure 59.

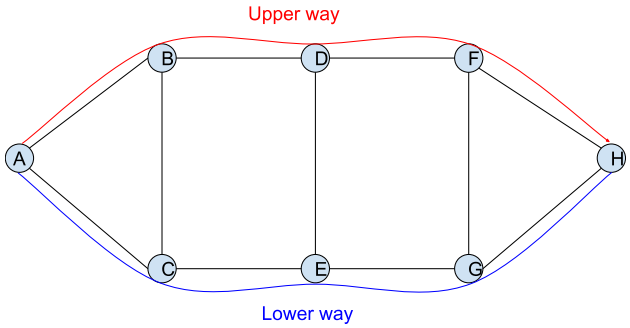


Figure 59 - Ladder-shape topology

Packets are sent from A to H. For each experiment, two non-BIER-TE tracks, and one or more BIER-TE tracks with different scheduling patterns are put in a same slot frame, as shown below:

Figure 60 - General structure of an experimental schedule

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

The BIER-TE tracks are varied during different experiments with different scheduling patterns. By scheduling pattern, we mean how the timeslots of a track are reserved, i.e., how does the track looks like on the global schedule. The packets in BIER-TE tracks are forwarded along a complex path with packet replication and elimination according to the bitString contained in their BIER-6LoRH.

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.

For each experiment, we measure packet delivery ratio, delay, jitter and care about reliability and energy consumption. As Zacharie mentioned in his report [18], one important value of this study is the evaluations on a real hardware platform, in contrary to most of the work in literature which usually resort to simulations based on analytical models. In many case, simplifying hypothesis are needed and prevent the model from reflecting the real connectivity condition of IEEE802.15.4e links, in particular they may elide the influence of external interference or multipath fading that account for the two major causes of link failures in LLNs. However of course, such kind of experiments are also subject to some limitations. One is that the results of experiments are hardly reproducible, since the quality of the links may vary significantly due to a slight change in the environment. Therefore, during each experiment we evaluate multiple BIER-TE tracks with different scheduling patterns along with non-BIER-TE tracks with retries concurrently.

Since the motes of the testbed are fixed inside the building, the environment of the experiments is sometimes too perfect to represent the world. To make the experiments closer to the reality, we would sometimes randomly interfere a mote with a 'metal box' as below:

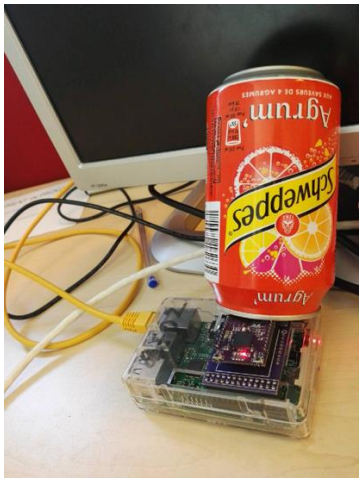


Figure 61 – Randomly interfere a mote

4.3 Multipath Scheduling Experiments

For each experiment of this section, a fixed bitString with all bits SET to '1' is used for all BIER-TE packets, while different BIER-TE multi-path scheduling patterns are compared between each other and with non-BIER-TE ARQ-based single-path forwarding. A concept of 'snooping' is also implemented in these experiments, which means a common neighbor of the sender and receiver of a transmission can optionally overhear the transmission if it has not yet get the copy. This can improve the chance of packet delivery and reduce latency for multi-path mechanisms. Of course, only one receiver of the two should be required to ACK for the frame to avoid collision. During the internship, we don't have time to design and implement collision-avoidance ACK mechanisms for snooping, however in our implementation, the sender does not require an ACK if it is the last slot of a TX bundle, thus we shall still able to explore the value of snooping for multipath to some degree.

4.3.1 Multipath on a Single Channel

The first experiment conducted is with a schedule displayed as below without channel hopping.

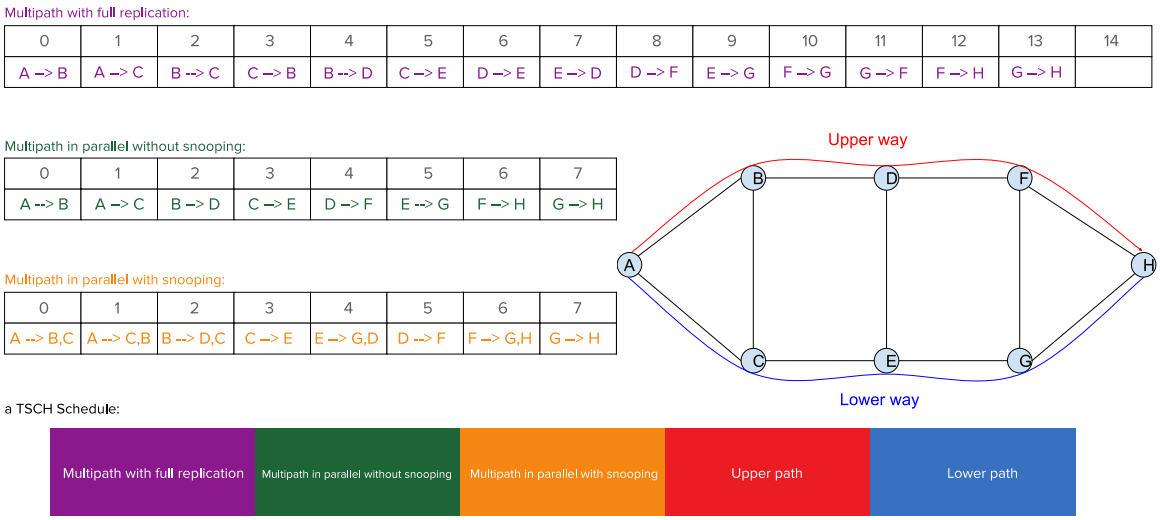


Figure 62 - Schedule (CDU) for multipath experiment on a single channel

The BIER-TE tracks are illustrated as below:

BIER-TE Track	Description
Multipath with full replication	Packets are replicated at every step (replication points) of the ladder
Multipath in parallel without snooping	Packets are simply forwarded along two parallel paths
Multipath in parallel with snooping	Packets are forwarded along two parallel paths with snooping in between

Table 3 – Descriptions of BIER-TE tracks

The slots are allocated as below:

Usage	Number of Slots
Upper path	12
Lower path	12
Multipath with full replication	14
Multipath in parallel with snooping	8
Multipath in parallel without snooping	8
Shared slots for synchronization and RPL	5
Serial communication	15
Total Size	74

Table 4 - Slot allocation of the schedule

And some experimental parameters:

Parameter	Value
Slotframe duration in seconds	1.11
Sending interval in seconds	2
Number of packets in each sending	5
Packets sent for each track	1917
Experiment Duration	16:51 - 18:04

Table 5 - Experimental parameters

Thus a test packet can be sent on each track every 2 seconds.

Results Figure 63 displays the delays and loss.

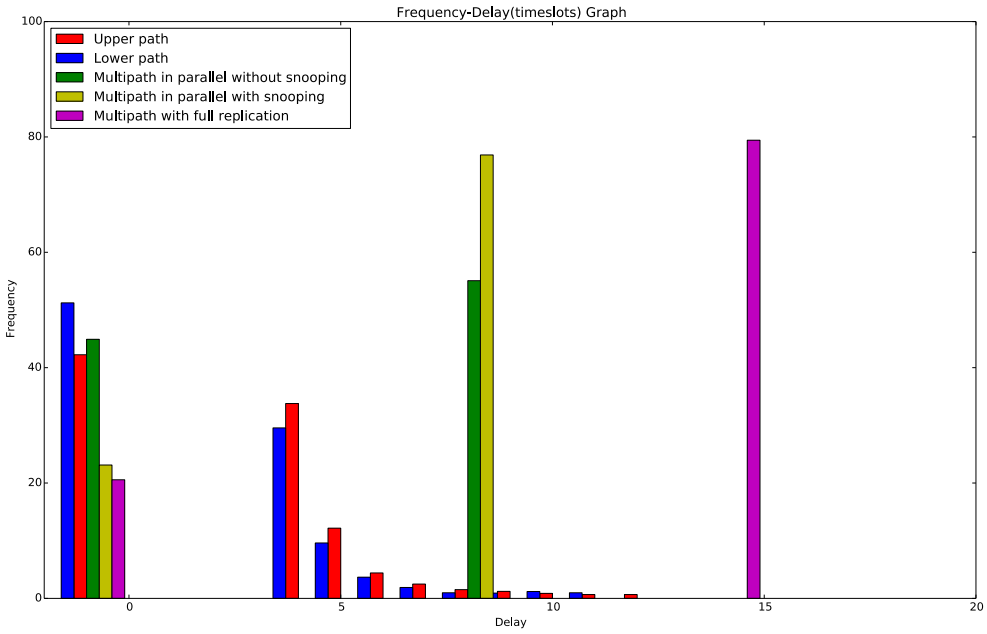


Figure 63 - Receiving delays (in Time Slots) frequency (% of messages sent)

Jitter. It shows that the BIER-TE tracks have no jitter by the protocol design, while using retries delays the packets and introduces jitters, as we can see there are still packets received after 10 timeslots for the upper path and lower path.

Delay. The average delay of using retries seems to be less than multipath on a single channel, as each directed adjacency takes exactly one timeslot, so that the more replication points it has, the more delay it would result in. As shown in the picture, the parallel paths without replications points have a delay of only 8 slots, which is nearly half of the delay of multipath with full replication.

Packet loss. The packet loss of using retries is generally higher than using multipath, especially much higher than multipath with snooping or with replication. This is because on a single channel, retries are more likely to fail without channel hopping, and the more retries fail, the less chances that the next retry will succeed. It is the same for the end-to-end packet delivery, the more packets are not received, the less chances that the next will succeed. From the figure we can also see that, snooping improves packet delivery ratio at a value similar to full replication without introducing additional delays.

As we can see, on a single channel, multipath with replication or snooping provides better reliability than using retries, but also consumes more energy with more transmissions. Using full replication provides highest reliability. Using snooping can provide a reliability similar to using full replication while lowering the energy consumption.

Figure 64 shows the occurrence of successive packet losses during the experiment.

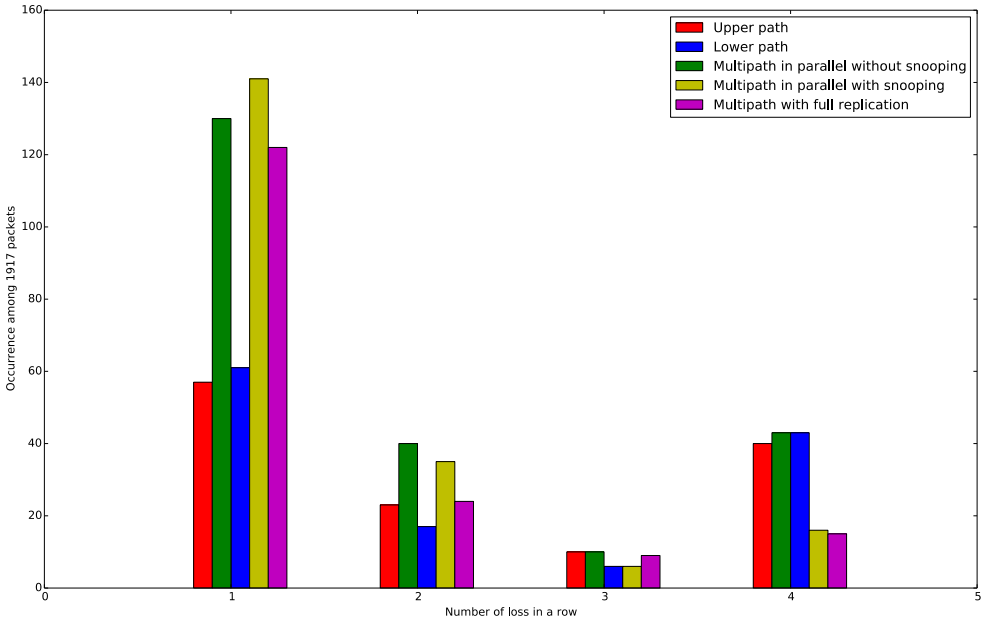


Figure 64 – Occurrence of successive packet losses

The value of occurrence for each case is counted as below. Note that case 4 and 5 are the same in our case which both are counted as one occurrence of a burst loss of 4 or more packets.

sequence	case	description	count
1		a loss of 1 packet	+1
2		a burst loss of 2 packets	+1
3		a burst loss of 3 packets	+1
4		a burst loss of 4 or more	+1
5		a burst loss of 4 or more	+1

Table 6 - Occurrence counting of successive packet losses (green: received a packet, red: lost a packet)

A burst loss of 4 or more packets are not acceptable in industrial process control systems. From Figure 64, we can see that the single paths with retries present very different patterns than multipath with replications. Packet losses in multipath cases are more scattered over time, and this is because a link failure or external interference along the path would very likely cause a successive packet loss using retries on a single channel, while with multipath, the second copy of the packet over the other branch can still reach the destination in due time, which makes the success rate of each packet more independent from the previous transmissions. This can also be confirmed from the figures presented below.

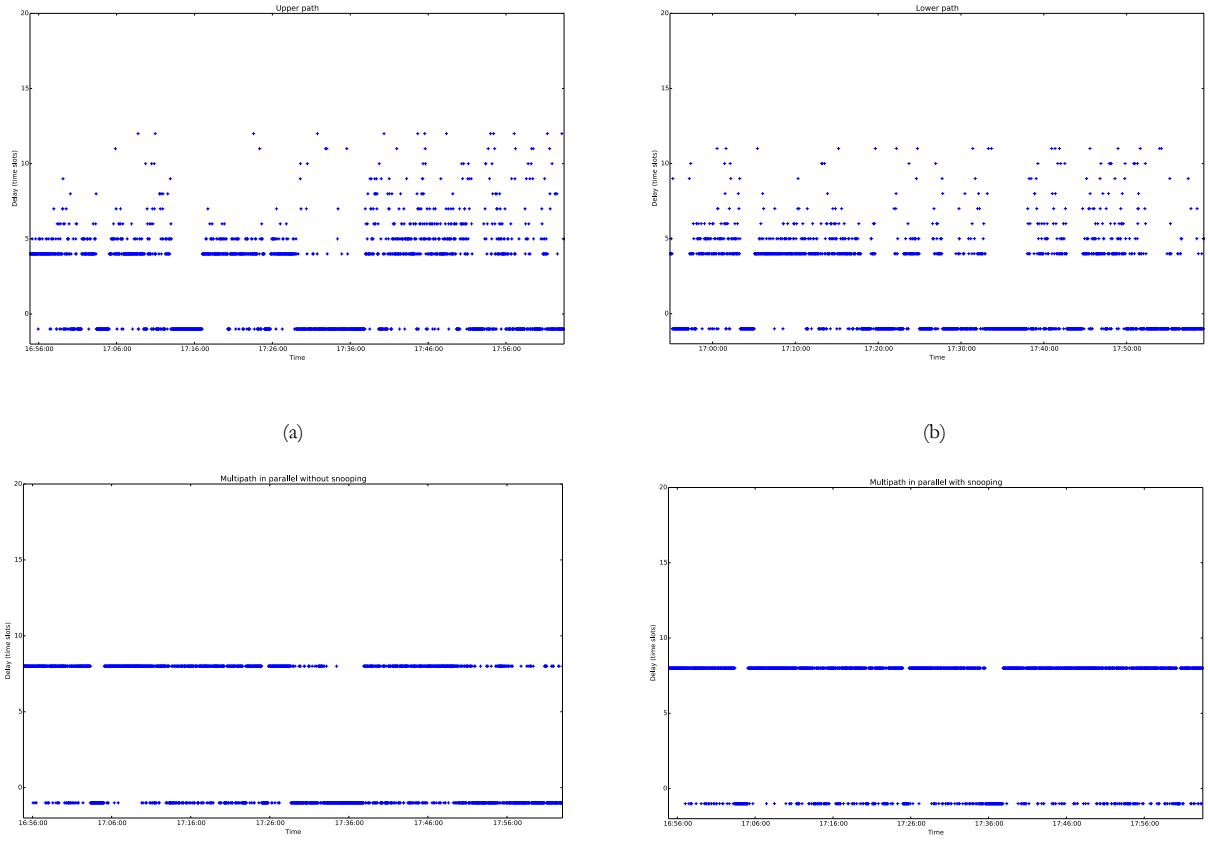
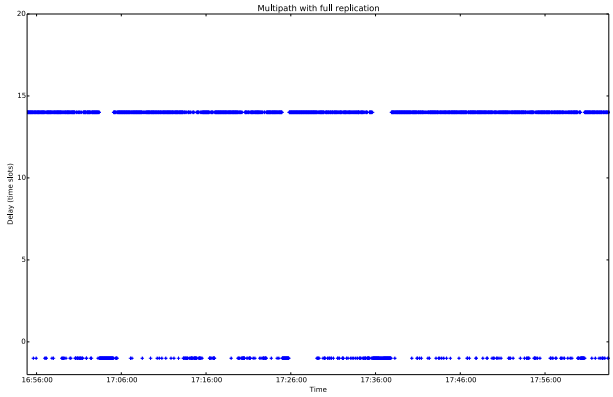


Figure 65 - Latency (unit : time slots) function of time. One packet is sent every 2 seconds. Latency of -1 means that no packet was received. (a) upper path, (b) lower path, (c) multipath in parallel without snooping, (d) multipath in parallel with snooping, (e) multipath with full replication



(e)

Figure 65 allows us to have a better understanding of what was discussed before. As we can see, the link quality changes over time, and the single paths with retries have more successive packet losses and less scattered ones, while it is on the contrary for the multipath with replications. There is no strong spatial dependency of packet loss can be identified from the figures, where the packet losses of upper path and lower path seem to be independent from each other. Therefore, multipath allows to compensate for link failures between these two paths. This was also proved in Zacharie's experiment [18]: High temporal dependency explains why retries do not work so well. Lower space dependency explains why multipath works better in this case.

We can also see the value of packet replication and snooping along the way by comparing with the forwarding pattern on two isolated parallel paths. Multipath makes the packet losses more 'isolated' and quasi-identically distributed. The worse the link quality is, the more valuable to have replication or snooping in multipath.

4.3.2 Multipath with Channel Hopping

The second experiment conducted is with a schedule displayed as below with channel hopping.

Multipath with Full Replication

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	A --> B														
2		A --> C													
3			B --> C												
4				C --> B											
5					B --> D										
6						C --> E									
7							D --> E								
8								E --> D							
9									D --> F						
10										E --> G					
11											F --> G				
12												G --> F			
13													F --> H		
14														G --> H	
15															

(a)

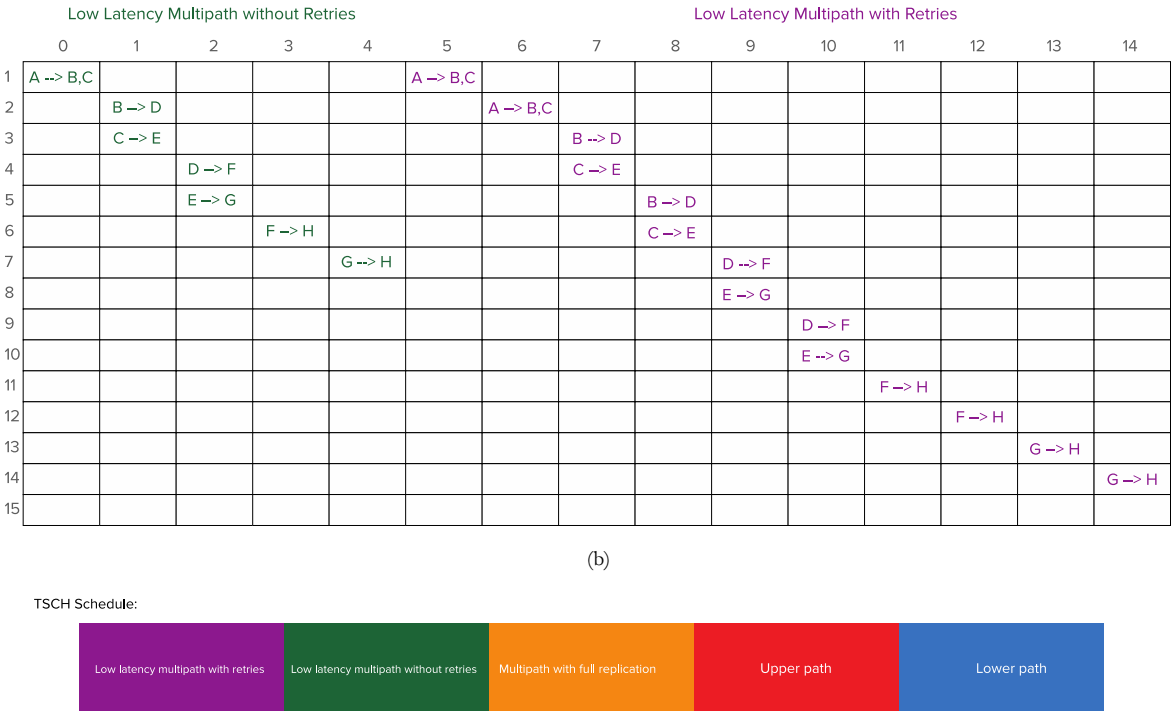


Figure 66 - Schedule (CDU) for multipath experiment with channel hopping. (a)BIER-TE track with full replication(orange) (b)BIER-TE low latency track with(purple)/without(green) retries, (c)general structure of the experiment schedule

The BIER-TE tracks are illustrated as below:

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

Table 7 - Descriptions of BIER-TE tracks

The slots are allocated as below:

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 8 - Slot allocation of the schedule

And the experimental parameters:

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 9 - Experimental parameters

As concurrent transmissions are enabled for the parallel paths, snooping cannot be adopted in this case except for the first hop A→ B,C. Note that for the 'purple' track, the snooped transmission A→B,C is enabled twice with two different bitIndexes, in this case, the second transmission will happen regardless the result of the first, and neither would require ACK. This is a workaround to avoid collisions of ACK while doing a 'retry'.

Results From Figure 67, we can witness a better link condition compared with the previous experiment. The retries have become more efficient on the upper and lower paths with channel hopping. Concurrent forwarding on parallel paths with retries shows a better packet delivery ratio and lower latency than forwarding with full replication which again confirms the value of retries on different channels.

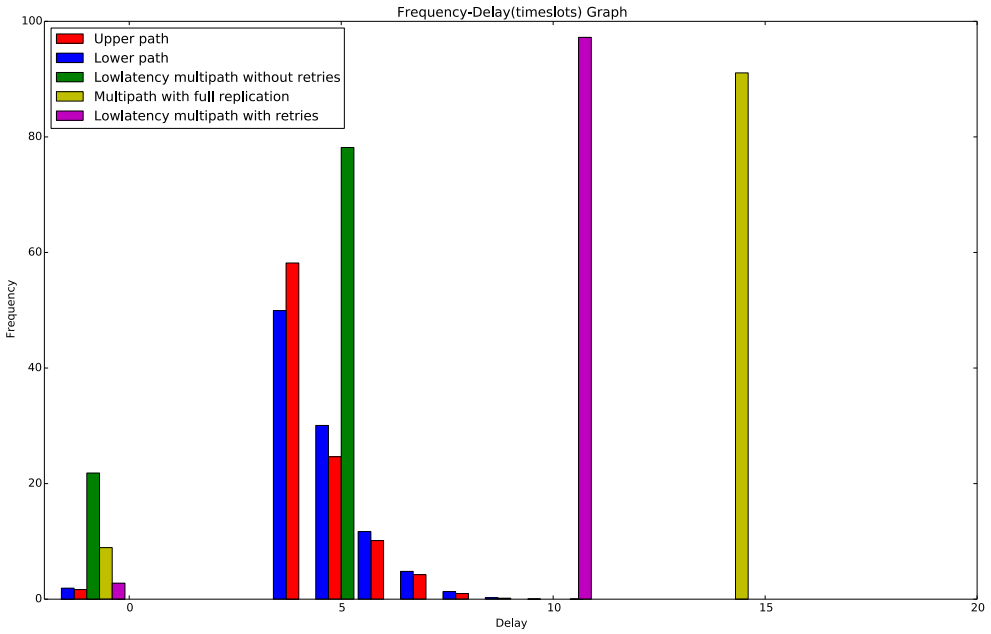


Figure 67 - Receiving delays (in Time Slots) frequency (% of messages sent)

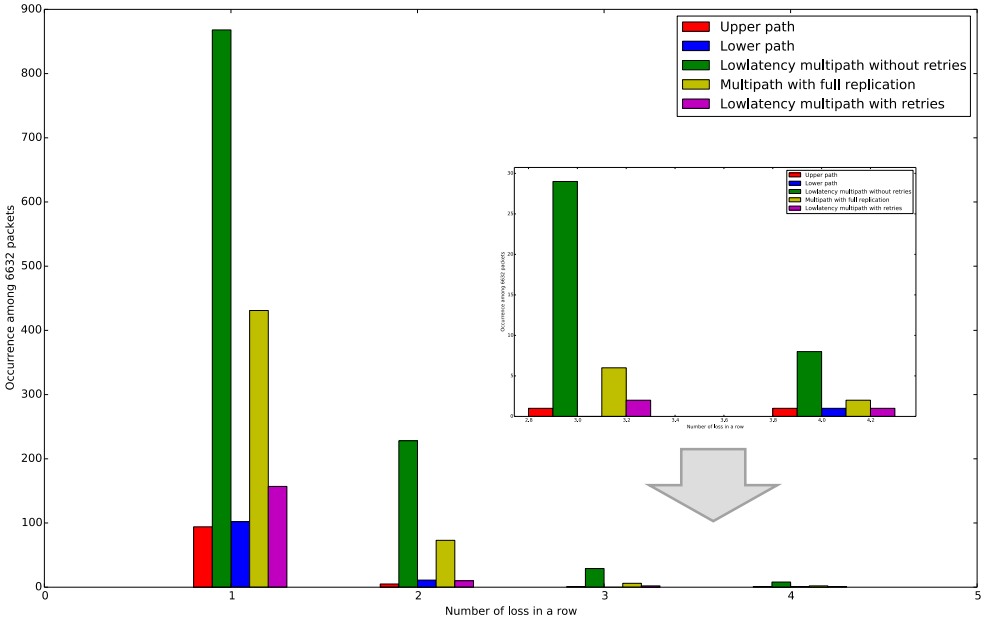
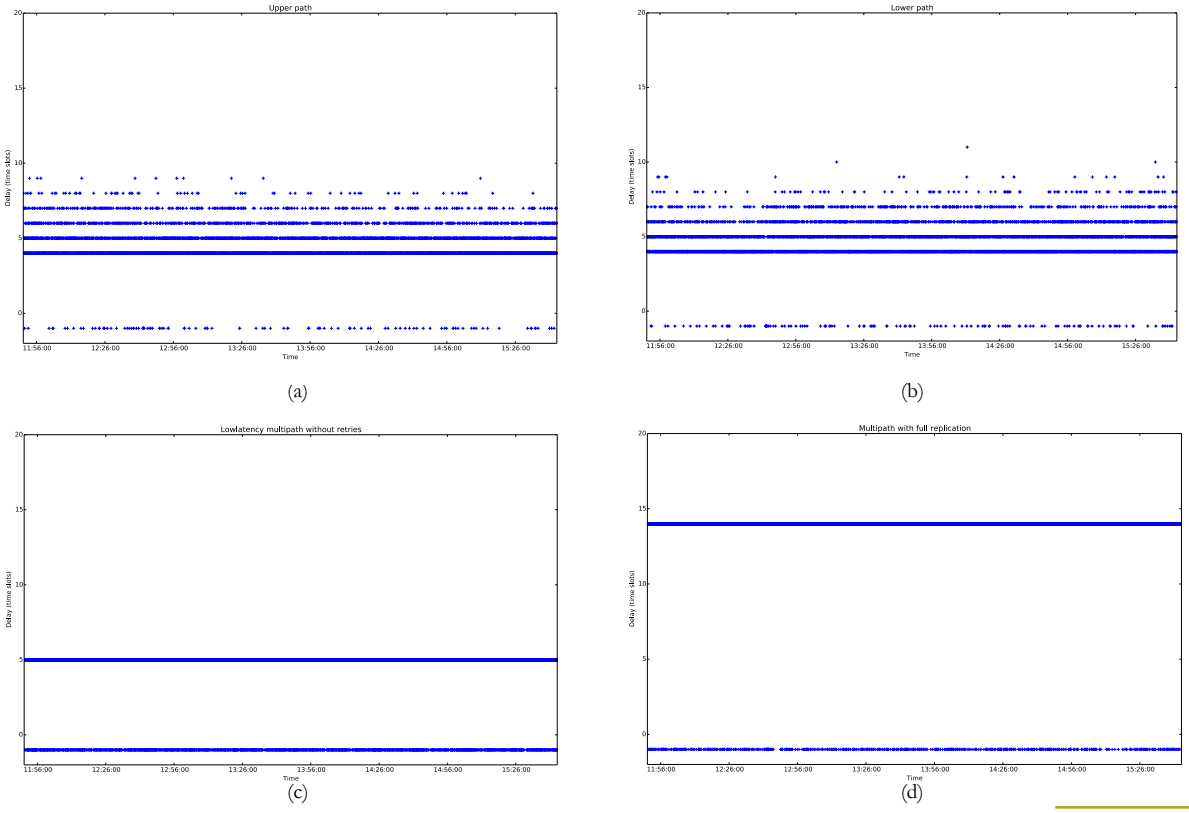
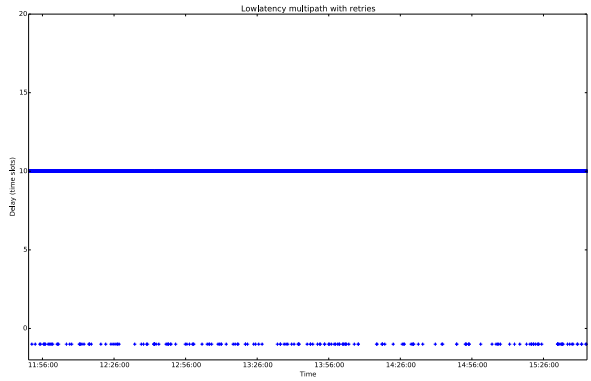


Figure 68 – Occurrence of successive packet losses

The packet delivery ratio of multipath in this case is generally lower than single path with retries. However, due to less dependency between packet losses, most of the packet losses of multipath are still scattered. Multipath tracks still maintain a low occurrence of a successive loss of more than 3 packets, which is especially the case for multipath with full replication and multipath with retries. We have to admit that under this case multipath underperforms compared to TSCH-based retries.





(e)

Figure 69 - Latency (unit : time slots) function of time. One packet is sent every two seconds. Latency of -1 means that no packet was received. (a) upper path, (b) lower path, (c) low latency multipath without retries, (d) multipath with full replication, (e) low latency with retries

Figure 69 confirms that time diversity seems to work better than spatial diversity with a constant good link quality under channel hopping. We can also directly see that the distribution of packet losses is scattered over time for all the tracks. However, in order to explore further the reliability of these tracks, we reran a same experiment to explore how these tracks could suffer a link failure. The new experiment uses same parameters as before, except the ones specified as below:

Parameter	Value
Packets sent for each track	8799
Experiment Duration	16:30 - 21:37

Table 10 - Experimental parameters

During the new experiment, we create a link failure on the lower path for about 20 minutes. Then we get the results as below:

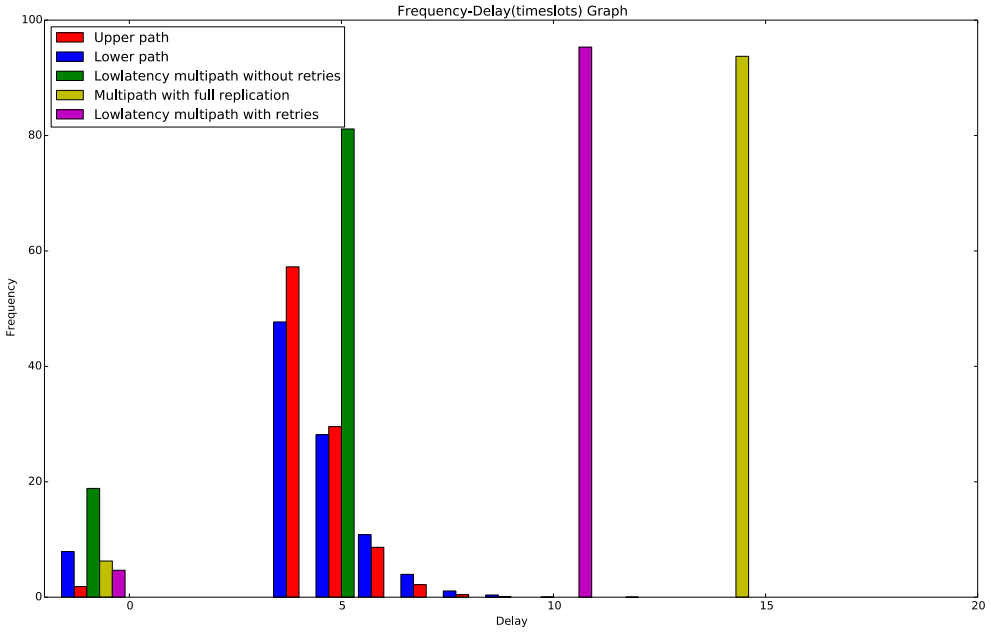


Figure 70 - Receiving delays (in Time Slots) frequency (% of messages sent)

The lower path has experienced a much higher number of packet losses than both before and the upper path, while other tracks almost maintain a same value as before in the last experiment. The figure below would allow us to have a direct understanding.

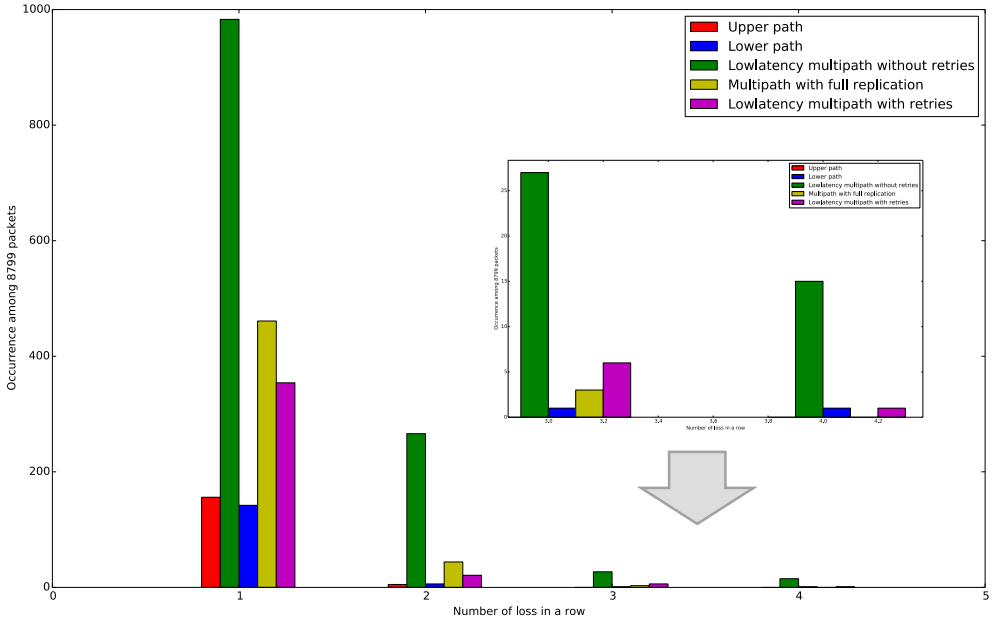
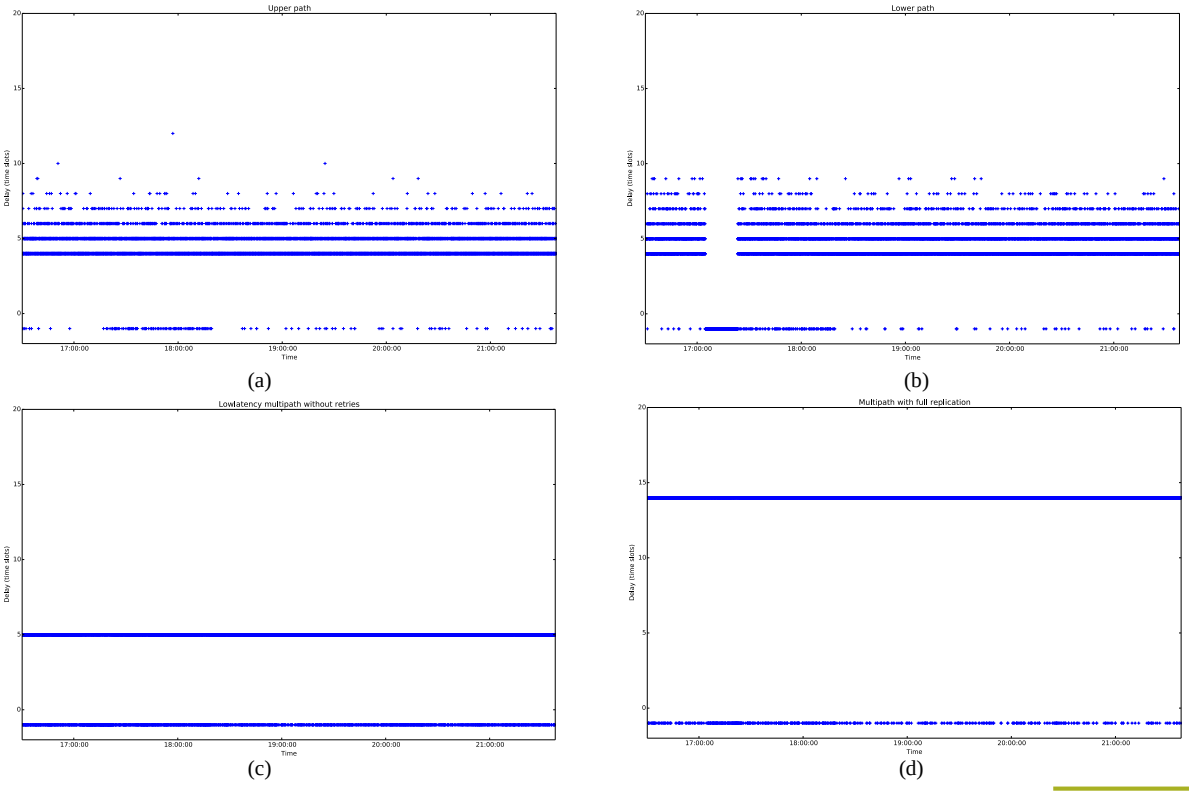
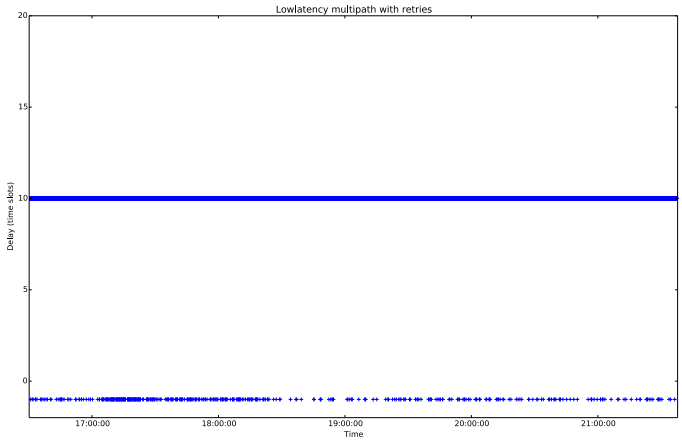


Figure 71 – Occurrence of successive packet losses

As we can see in *Figure 71*, the only two tracks that overcome the link failure is upper path track and the full replication track, which in return confirms the value of spatial diversity.





(e)

Figure 72 - Latency (unit : time slots) function of time. One packet is sent every two seconds. Latency of -1 means that no packet was received. (a) upper path, (b) lower path, (c) low latency multipath without retries, (d) multipath with full replication, (e) low latency with retries

Although there are more packet losses in multipath tracks during the link failure than normal cases, all of them are still scattered as we only observe increases on the occurrence of 1 packet loss in a row, and the loss increase of the purple track (multipath with retries) is much larger than that of the yellow track (multipath with full replication). It is then again proved that multipath has a better node/link-failure-tolerance than retries.

4.3.3 Low Latency Concurrent Multipath

The goal is to improve reliability on a network where a very low latency is required. In such a case, scheduling retries is often not an option. The idea is to forward concurrently a copy of the packet on each of the two non-congruent paths, and implement as more snooping as possible along the track to improve reliability. As replication on each step of the ladder introduce latencies, therefore, there would be no replication points. The schedule is designed as below.

Both have a latency of 5 slots

	Low latency multipath without snooping, without retries					Low latency multipath with snooping, without retries									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	10	11	12	13	14
1	A --> B,C					A --> B,C									
2		B —> D					B --> D,C								
3		C —> E						C —> E							
4			D —> F					D --> F							
5			E --> G						E —> G						
6				F --> H					F —> H						
7					G —> H					G --> H					
8															
9															
10															
11															
12															
13															
14															
15															

(a)

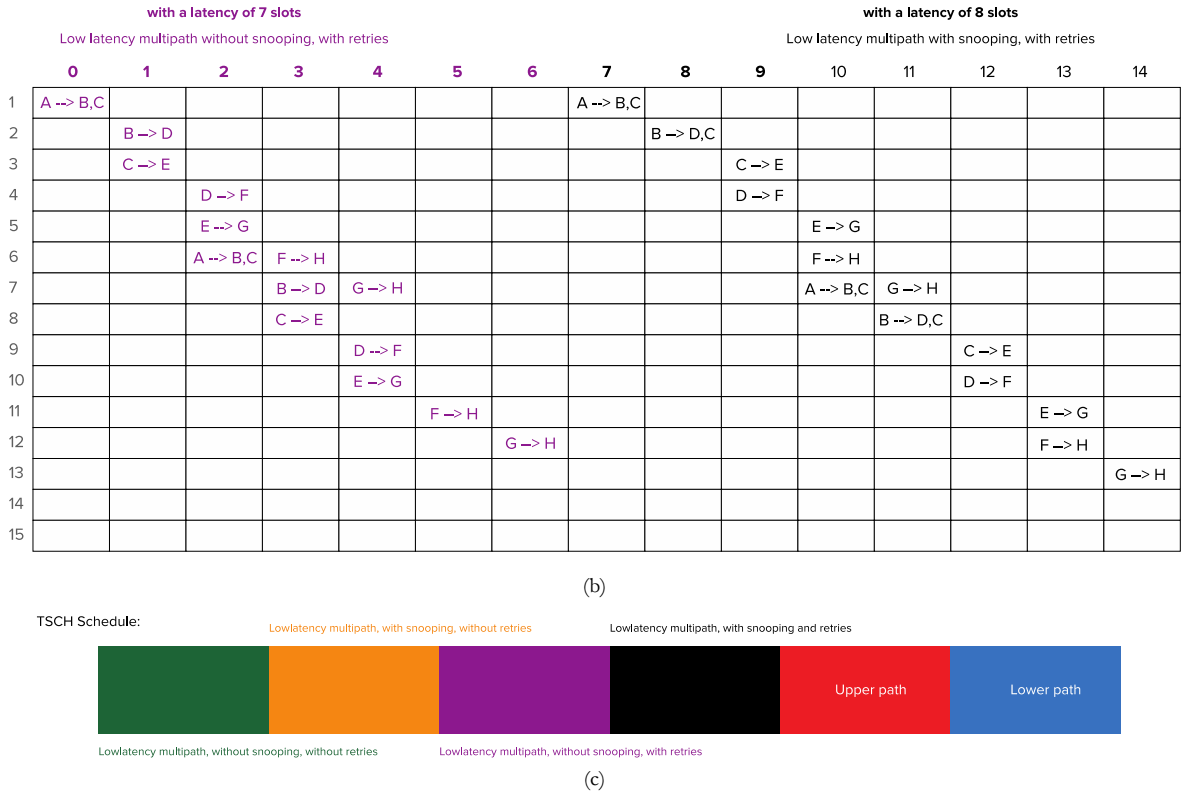


Figure 73 - Schedule (CDU) for low latency concurrent multipath experiment. (a) low latency multipath, without snooping, without retries (green), low latency multipath, with snooping, without retries (yellow) (b) low latency multipath, without snooping, with retries (purple), low latency multipath, with snooping and retries (black) (c)general structure of the schedule

We bound the maximum latency to be 5 timeslots. However, in order to explore further the value of concurrent multipath with retries, we also added two additional BIER-TE tracks as presented in Figure 73 (b). The slots of the experiment schedule are allocated as below:

Usage	Number of Slots
Upper path	12
Lower path	12
Low latency multipath, without snooping, without retries	5
Low latency multipath, with snooping, without retries	5
Low latency multipath, without snooping, with retries	7
Low latency multipath, with snooping and retries	8
Shared slots for synchronization and RPL	3
Serial communication	15
Total Size	67

Table 11 - Slot allocation of the schedule

Please note that, we do not count the first transmission A-->B,C as snooping here, as neither B or C has any other additional slots for receiving the frame from A. This one-to-many transmission is only used in order to reduce latency, instead of increase reliability like other snooping scheduled.

The BIER-TE tracks are illustrated as below:

BIER-TE Track	Description
Low latency multipath, without snooping, without retries	Use concurrent transmissions to minimize the latency
Low latency multipath, with snooping, without retries	Use snooping to increase reliability of low latency multipath
Low latency multipath, without snooping, with retries	Use concurrent retries for the first BIER-TE track
Low latency multipath, with snooping and retries	Use concurrent retries for the second BIER-TE track

Table 12 - Descriptions of BIER-TE tracks

And the experimental parameters:

Parameter	Value
Slotframe duration in seconds	1.005
Sending interval in seconds	2.4
Number of packets in each sending	6
Packets sent for each track	3674
Experiment Duration	14:26 - 17:04

Table 13 - Experimental parameters

Results The results are shown as below. By comparing the tracks with a latency bounded in 5 timeslots, we can see that forwarding each copy on two isolated parallel paths (the green BIER-TE track) does not really improves packet delivery ratio, as was also witnessed in Zacharie's experiment. However, with snooping enabled just for B --> D,C, we can observe an apparent increase on packet delivery ratio on the yellow BIER-TE track. This again confirms that snooping contributes reliability without introducing additional latencies. In addition, The figure also shows how concurrent retries can improve packet delivery ratio with just a few additional 2~3 slots.

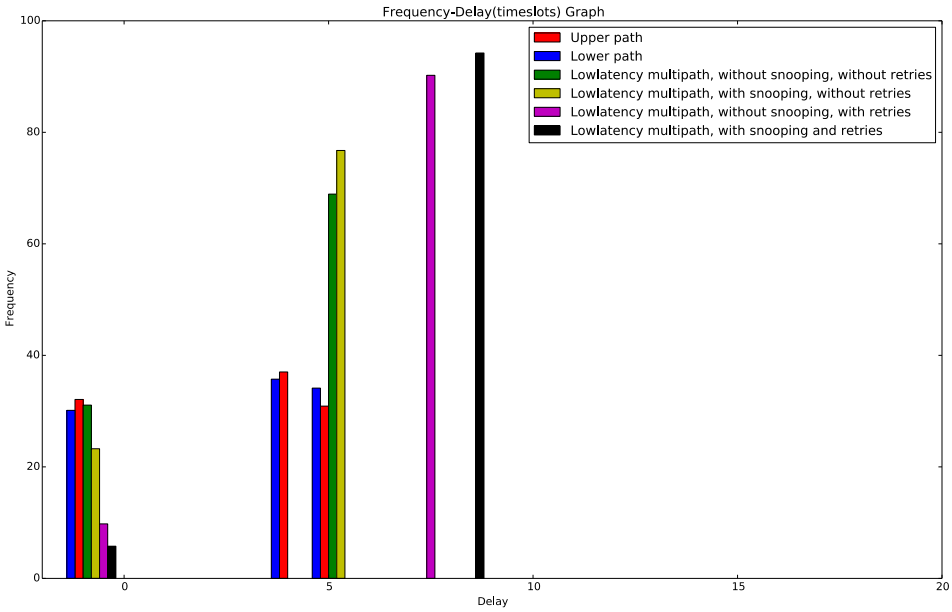


Figure 74 - Receiving delays (in Time Slots) frequency (% of messages sent)

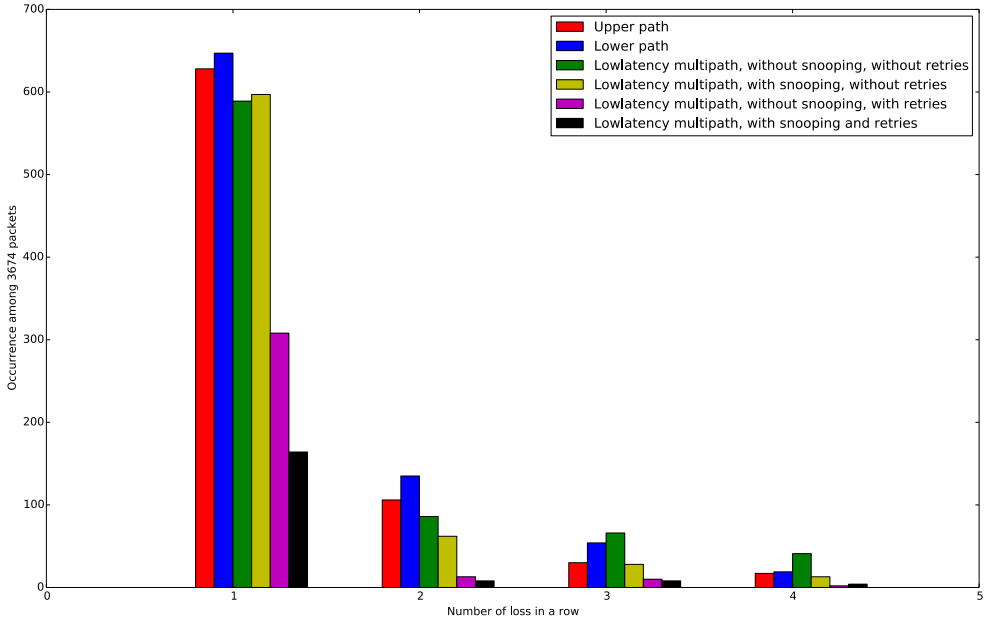
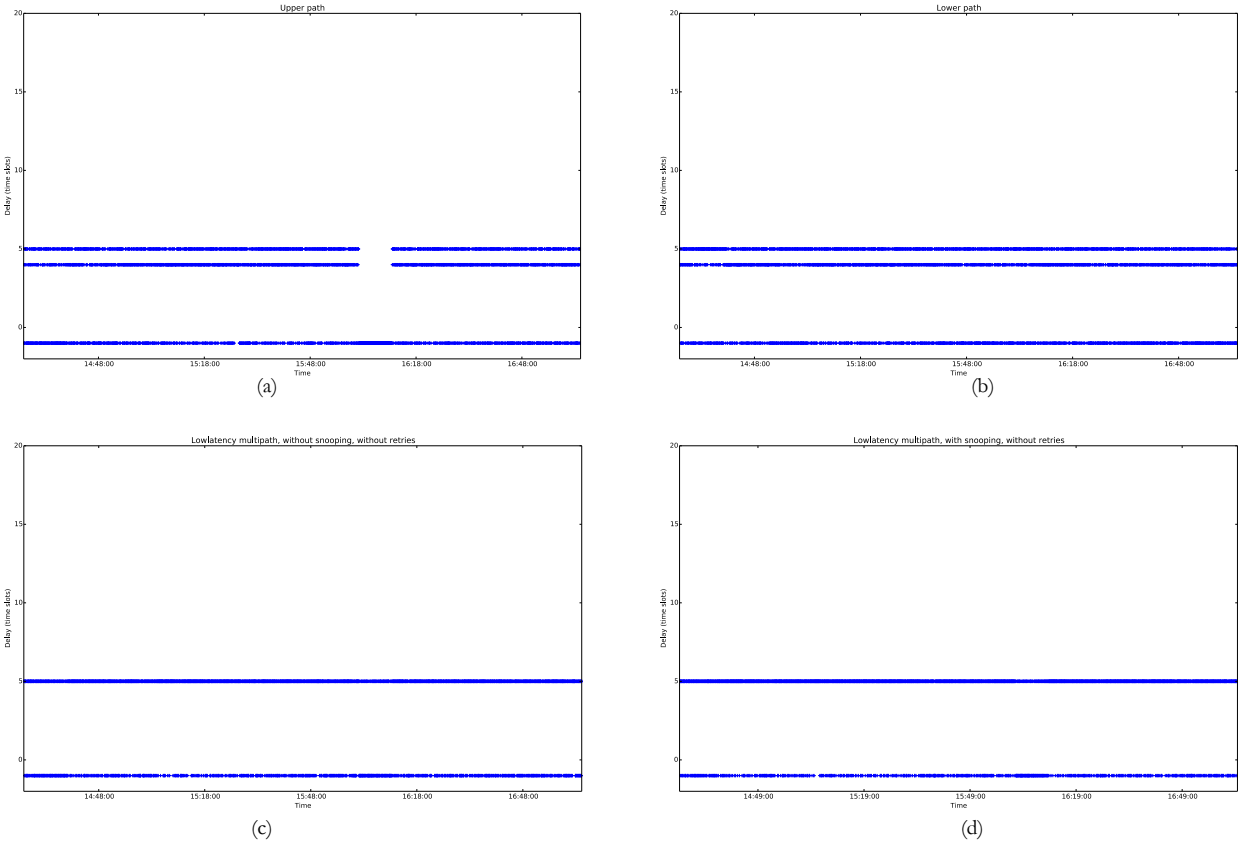
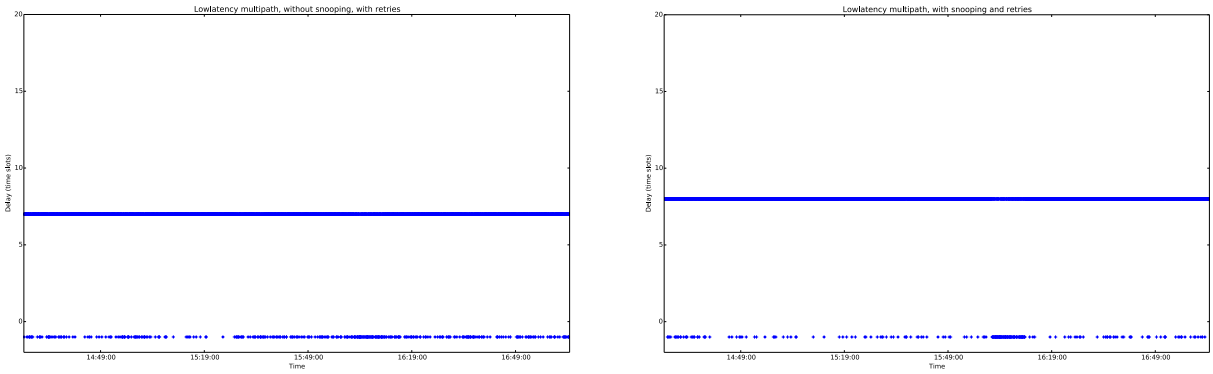


Figure 75 – Occurrence of successive packet losses

Figure 75 shows that the low latency multipath with snooping loses less packets with a lower number of occurrence of successive packet losses.





(e) (f)

Figure 76 - Latency (unit : time slots) function of time. One packet is sent every 2.4 seconds. Latency of -1 means that no packet was received. (a) upper path, (b) lower path, (c) low latency multipath without snooping, without retries, (d) low latency multipath with snooping, without retries, (e) low latency multipath without snooping, with retries, (f) low latency multipath with snooping, with retries

The 'black' BIER-TE track combines snooping on concurrent multipath with retries and channel hopping, and reaches the highest value of packet delivery ratio among the experiment tracks with only a latency of 8 timeslots. There is a link failure on the upper path, during which we can observe more packet losses happen than other time in Figure 76 (f). However, from Figure 75, we can see that the losses are scattered because of multipath as discussed before.

4.3.4 Concurrent Multipath based on Channel Hopping

As we can see before, with channel hopping, we can enable concurrent transmission of a packet copy along each of the two parallel paths without affecting each other. In order to examine this, we designed a simple experiment with a schedule as below:

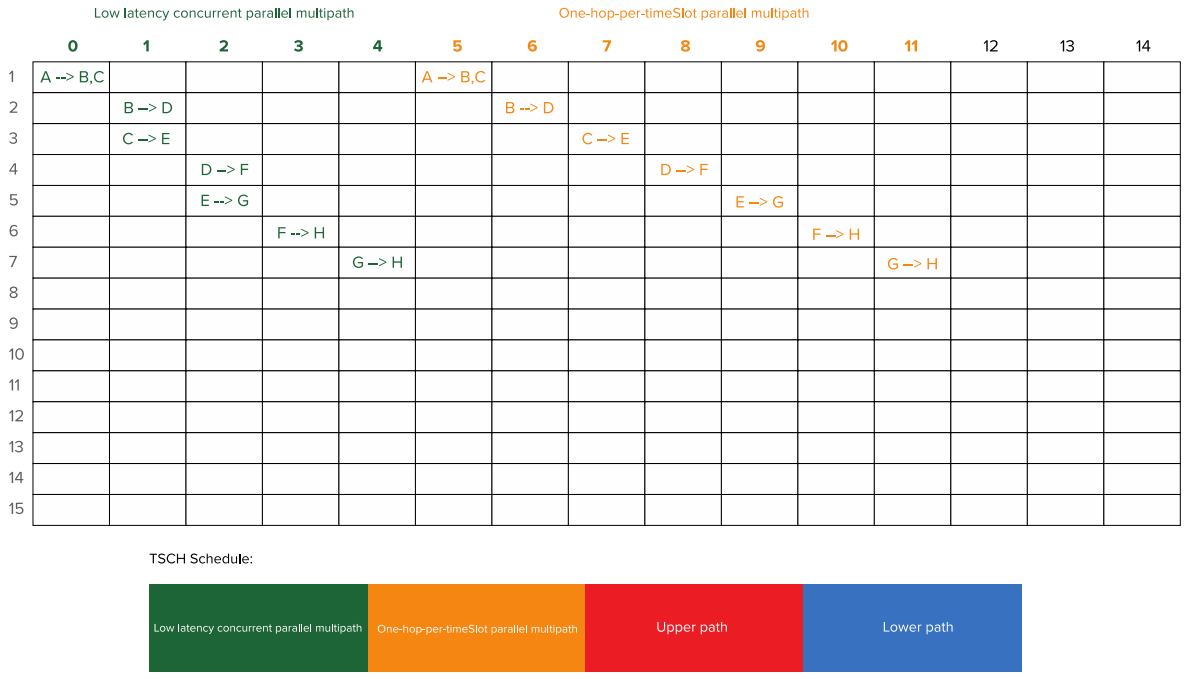


Figure 77 - Schedule (CDU) for concurrent multipath examination

We expect that the two multipath tracks should have very similar results based on our assumption. We run the schedule with slots assigned as below:

Usage	Number of Slots
Upper path	12
Lower path	12
Low latency concurrent parallel multipath	5
One-hop-per-timeslot parallel multipath	7
Shared slots for synchronization and RPL	3
Serial communication	15
Total Size	54

Table 14 - Slot allocation of the schedule

with following parameters:

Parameter	Value
Slotframe duration in seconds	0.81
Sending interval in seconds	1.6
Number of packets in each sending	4
Packets sent for each track	6245
Experiment Duration	15:09 - 18:02

Table 15 - Experimental parameters

Results However, we see distinct difference between the results of these two scheduling patterns. The one without concurrent transmissions shows a better packet delivery ratio than the other.

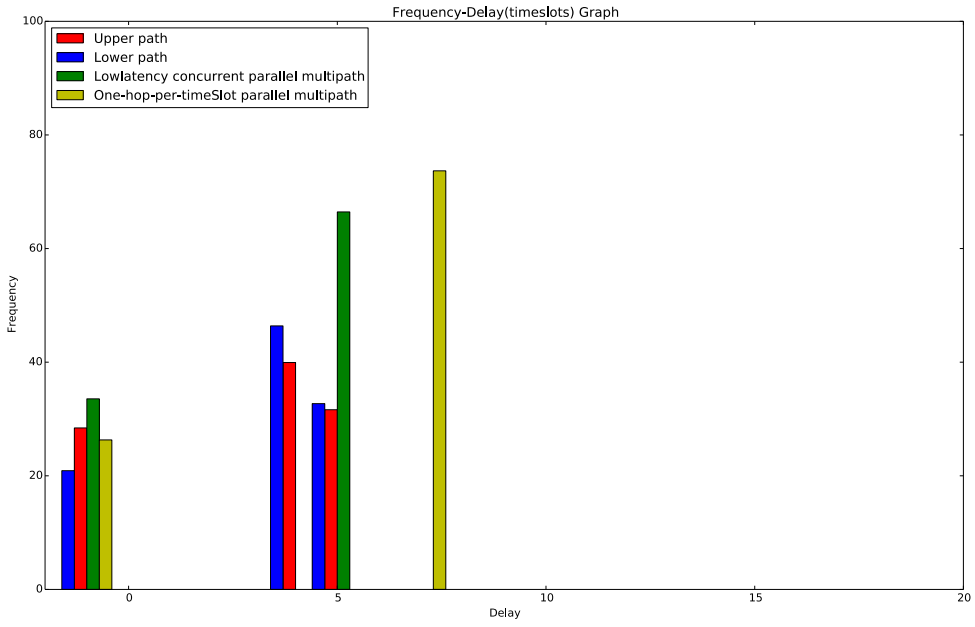


Figure 78 - Receiving delays (in Time Slots) frequency (% of messages sent)

This may be because it spreads more in time domain. Or maybe the concurrent transmissions still affect each other somehow to some extent. The figure below shows that the One-hop-per-timeslot parallel multipath forwarding outperforms concurrent parallel multipath at every segment.

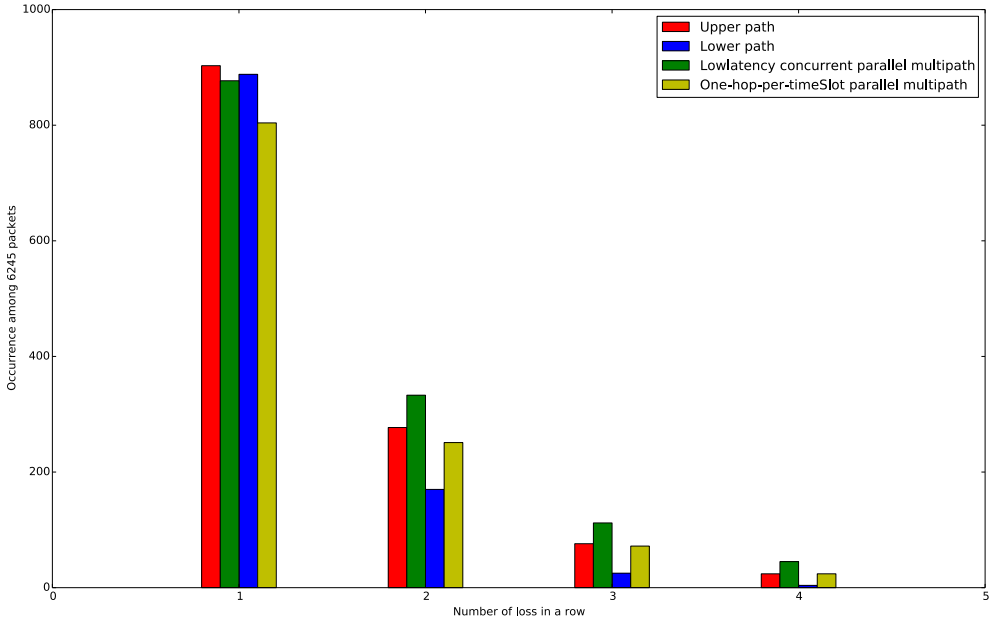


Figure 79 – Occurrence of successive packet losses

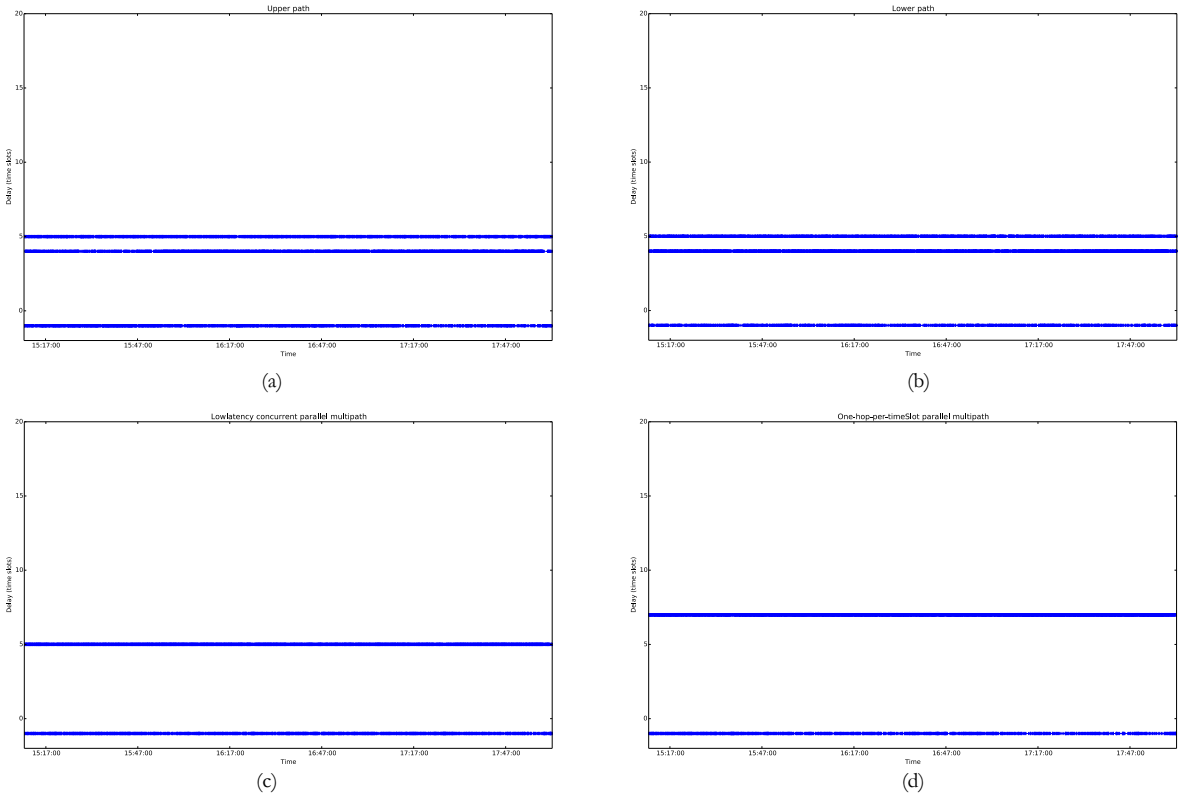


Figure 80 - Latency (unit : time slots) function of time. One packet is sent every 1.6 seconds. Latency of -1 means that no packet was received. (a) upper path, (b) lower path, (c) low latency concurrent parallel paths, (d) one-hop-per-slot parallel paths

We have to admit that the distinct difference between the results of the two patterns is out of our expectation. It is a pity that we are not able to find additional time to conduct more experiments on it. However, we can keep this in mind and try to research on it in the future work.

4.3.5 Enhanced Spatial Diversity with X-shaped Multipath

As shown in Figure 17, the topology exposed by RPL is actually shaped in a ladder with additional crosses inside each grid. In addition to ARC, we should be able to find other algorithms that can form a more complex path with increased spatial diversity, for example, something like below.

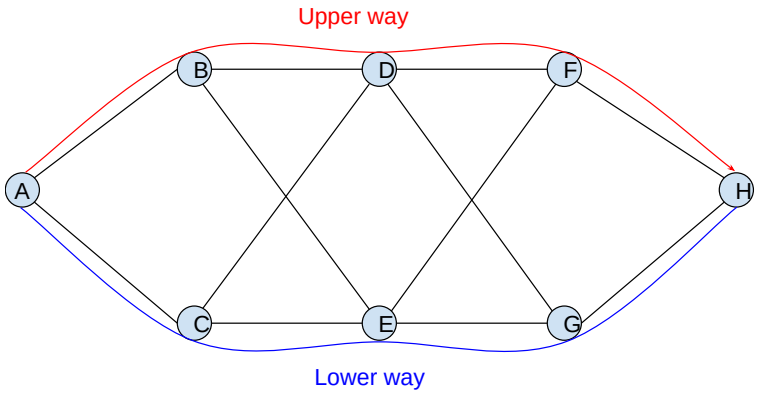


Figure 81 - X-shape multipath topology

This topology is perfect for implementing snooping, which can provide a high reliability within a very short timeslots (latency). In order to examine the value of this, we implemented the topology as below on the testbed at Cisco PIRL.

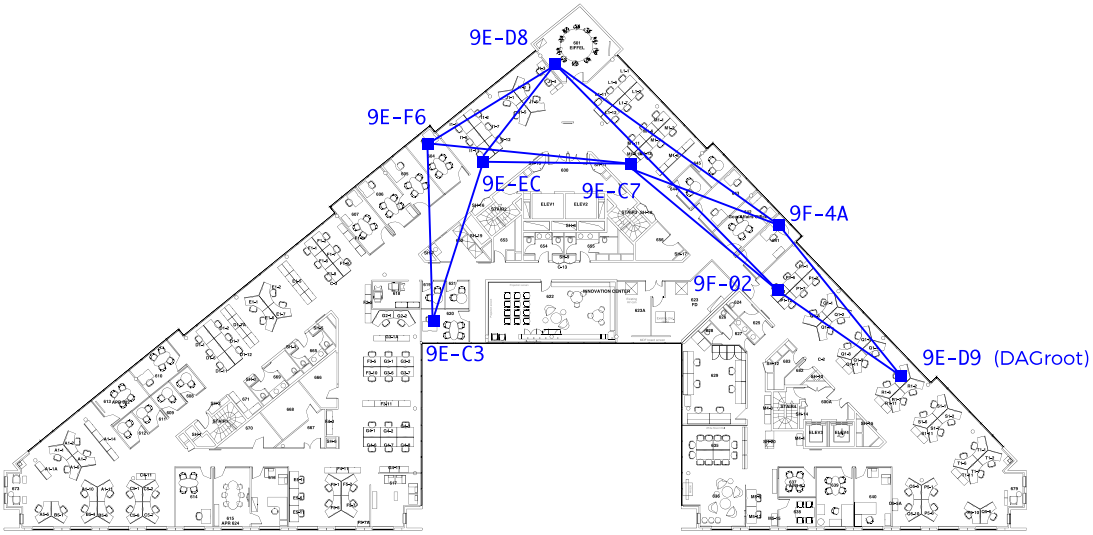


Figure 82 - Testbed topology

Multipath with full replication:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A --> B	A --> C	B --> E	B --> D	C --> E	C --> D	D --> G	D --> F	E --> G	E --> F	F --> H	G --> H			

Multipath in parallel with snooping:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A --> B,C	A --> C,B	B --> E,D	C --> E,D	D --> G,F	E --> G,F	F --> H	G --> H							

Multipath in parallel without snooping:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A --> B	A --> C	B --> D	C --> E	D --> F	E --> G	F --> H	G --> H							

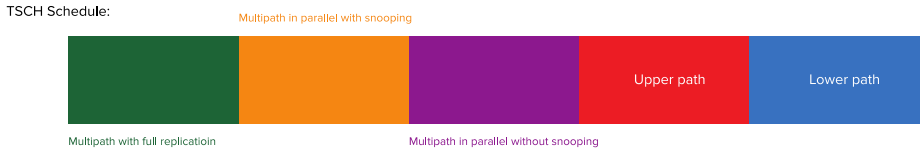


Figure 83 - Schedule (CDU) for X-shape multipath

Figure 83 shows the schedule for the experiment on a single channel, and the slots are allocated as below:

Usage	Number of Slots
Upper path	12
Lower path	12
Multipath with full replication	12
Multipath in parallel with snooping	8
Multipath in parallel without snooping	8
Shared slots for synchronization and RPL	5
Serial communication	15
Total Size	72

Table 16 - Slot allocation of the schedule

Table 17 introduces each BIER-TE track.

BIER-TE Track	Description
Multipath with full replication	Reserve a time slot for each adjacency
Multipath in parallel with snooping	Enable all possible snooping between the two parallel paths
Multipath in parallel without snooping	Reserve slots simply for two isolated parallel paths

Table 17 - Descriptions of BIER-TE tracks

Experiment parameters:

Parameter	Value
Slotframe duration in seconds	1.08
Sending interval in seconds	2
Number of packets in each sending	5
Packets sent for each track	4070
Experiment Duration	14:17 - 16:37

Table 18 - Experimental parameters

Results As expected, snooping in this case performs as good as full replication in terms of packet delivery ratio, and consumes less timeslots and of course energy. While retries again become much less efficient on a single channel.

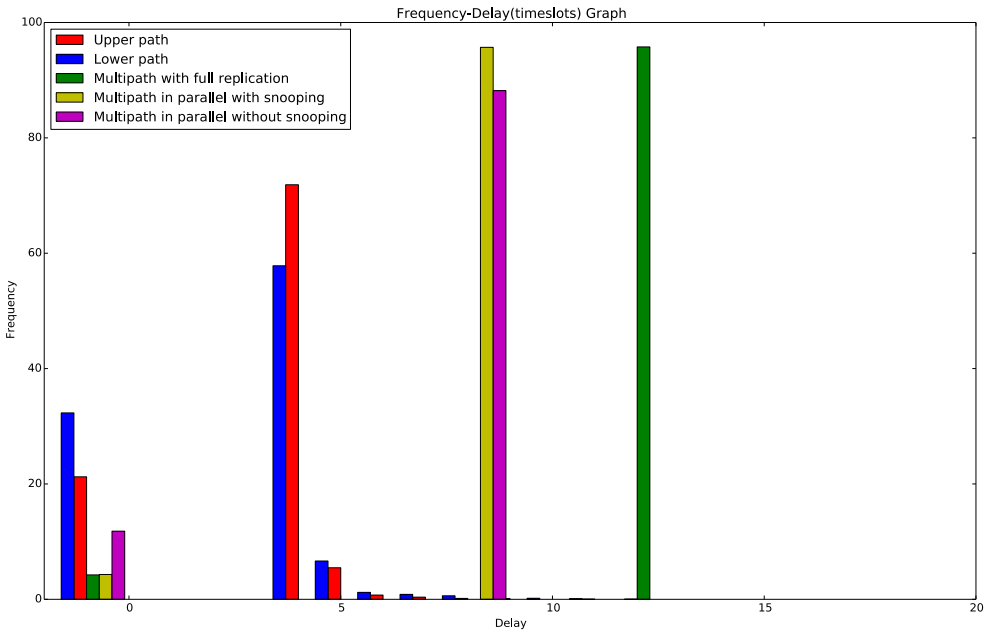


Figure 84 - Receiving delays (in Time Slots) frequency (% of messages sent)

The performance regarding successive packet losses of multipath with snooping is also very close to that of multipath with full replication. Most of the packet losses in multipath tracks are scattered, which is not the case with retries on a single path. By comparing the 'yellow' track with the 'purple' track, we can again confirm that snooping improves reliability at lower cost of energy consumption than full replication.

However, there are also drawbacks for enabling snooping. Firstly, it makes the ARQ mechanisms more complex. Secondly, although it shortens the size of bitString, it would also possibly make the bitString-based path control and feedback mechanisms less precise and less efficient.

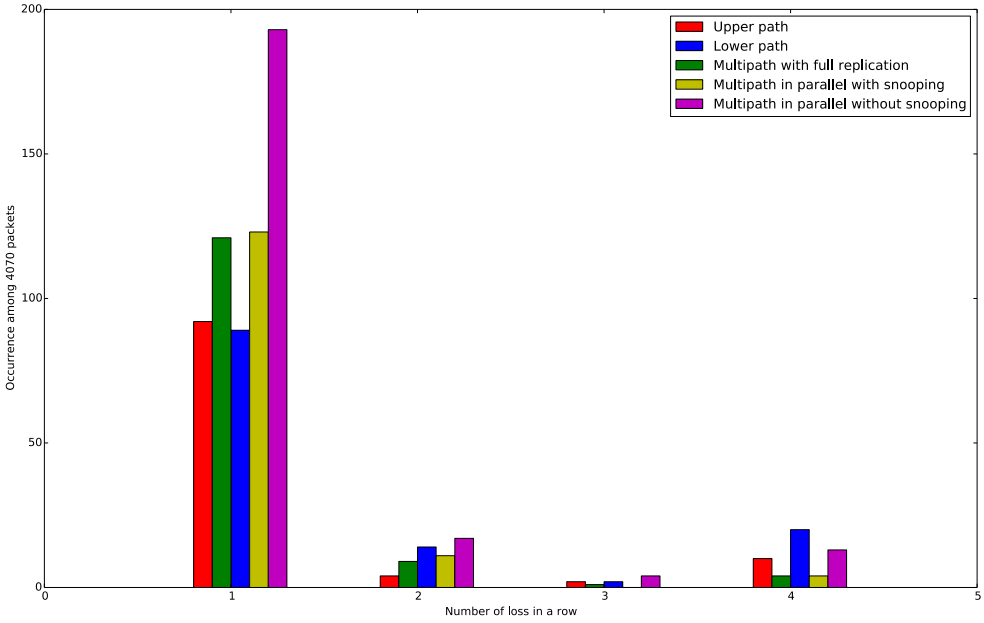
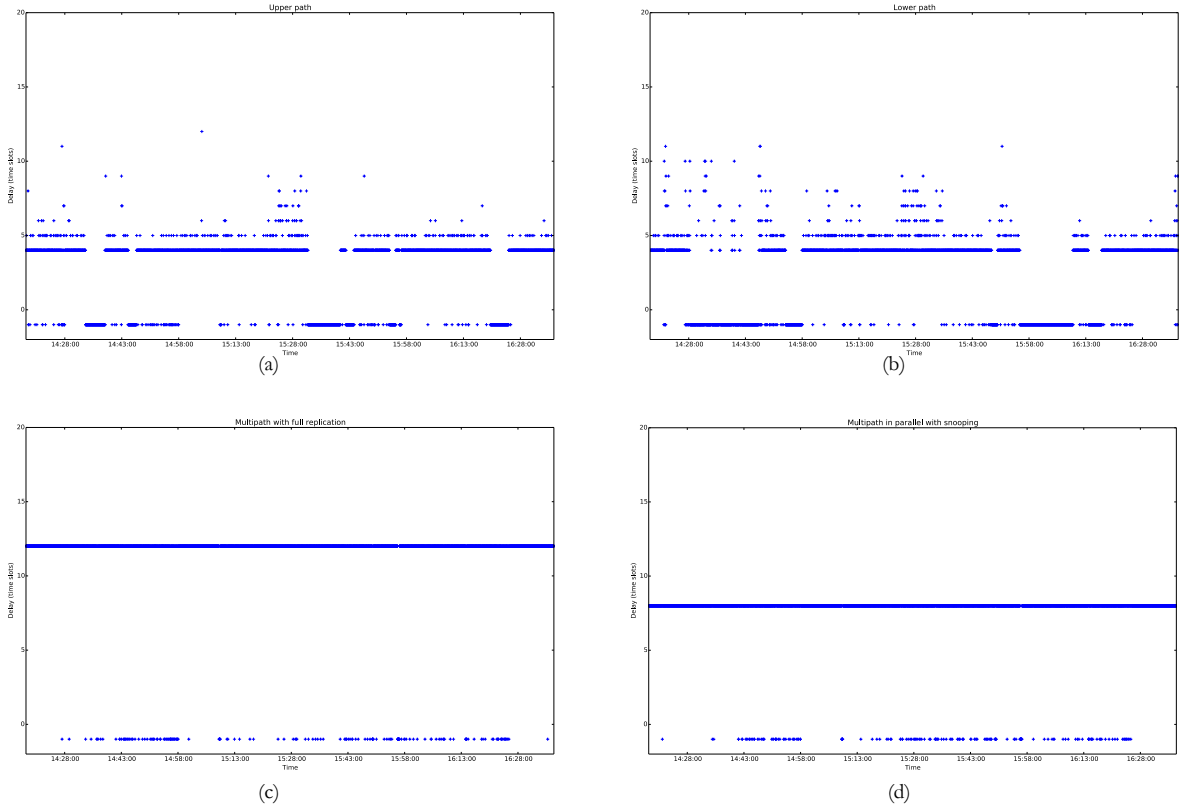
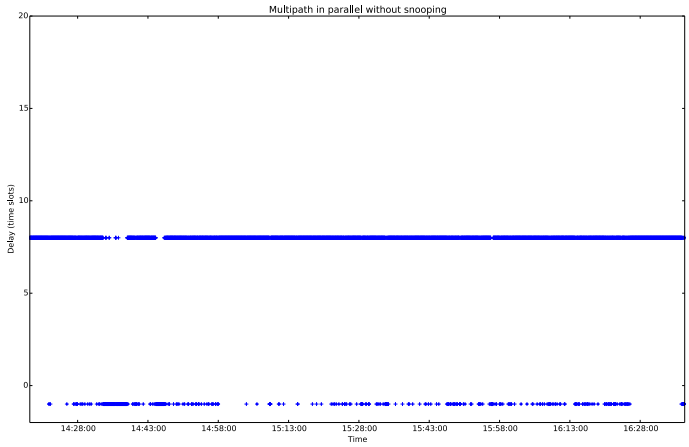


Figure 85 – Occurrence of successive packet losses

The figures below allow us to have a better understanding how the enhanced spatial diversity could improve the packet delivery ratio to a very high level without channel hopping, by compensating link failures of the two paths with one another.





(e)

Figure 86 - Latency (unit : time slots) function of time. One packet is sent every 2 seconds. Latency of -1 means that no packet was received. (a) upper path, (b) lower path, (c) multipath with full replication, (d) multipath in parallel with snooping, (e) multipath in parallel without snooping

4.3.6 Key points

The results allow us to summarize with the following key points:

Jitter Retries introduces jitters, while deterministic protocol schedules all transmissions with no jitter.

Delay Packet replication introduces delays, however, it can be overcome by using snooping along the concurrent parallel paths to improve the reliability as packet replication while not introducing additional delays.

Loss ratio Packet loss ratio depends on a variety of factors, and different diversity techniques can contribute to packet delivery ratio in different aspects. For example, TSCH overcomes more efficiently external interference and multipath fading, while path diversity shows a better tolerance on node or link failures with less burst losses than single path.

Energy Multipath forwarding naturally consumes more energy with more transmissions. This can be improved by using snooping with less transmissions and shorter bitStrings, and BIER-TE protocol can provide a control loop which allows to save energy by enabling replication only when necessary.

Reliability The process of replication and elimination at each step of the ladder improves overall reliability in addition to TSCH. Unlike ARQs, adopting concurrent multipath increases reliability without introducing additional delays while avoiding single point of failures.

However, as shown by the results of 4.3.4, concurrent transmissions may introduce interferences to each other even with channel hopping, which needs to validate further in the future work.

4.4 BitString Feedback Control Experiments

In this section, we design and implement a BIER-TE bitString feedback control loop based on the ladder topology. The goal is to exploit all possible diversity techniques to improve reliability, while maintaining the energy consumption as low as possible. The idea is that packets take a single path with each hop having limited slots for retries using TSCH, and the destination node feeds output bitStrings of received packets back to the openVisualizer (normally the bitString should be all '0'). As both retries and channel hopping are enabled, it provides already high packet delivery ratio in normal conditions as shown in the result of the experiment 4.3.2, therefore, when openVisualizer detects a successive loss of 2 or more packets, it is very likely that the link qualities are going really bad, or possibly going down. OpenVisualizer would thus enable multipath mechanisms with full replication for next packets in order to improve success rate and meanwhile diagnose link failures. Based on the output bitStrings of the full replicated packets received from the destination node, openVisualizer identifies the link failure, and selects a secondary single path using a new bitString with limited number of bits enabled for next packets. In this way, the control loop saves energy by enabling packet replications only when necessary, and combines high packet delivery capability using TSCH-based ARQs with high link failure-tolerance using multipath. A demo can be found at youtube with link https://www.youtube.com/watch?v=Xt56_rdWzho.

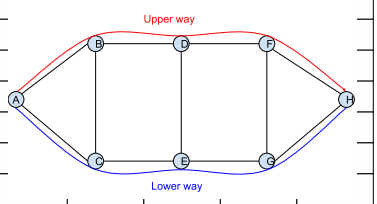
4.4.1 Experiment Setup

We follow the same pattern as the previous experiments, for which we have implemented multiple deterministic tracks with different scheduling patterns and 2 stochastic tracks in a same schedule. This time we implement two deterministic tracks with same scheduling pattern, but one uses full replication all the time with a fixed bitString '1111111111', and the other implements the control loop with bitString adjusted by openVisualizer according to the link conditions as illustrated above.

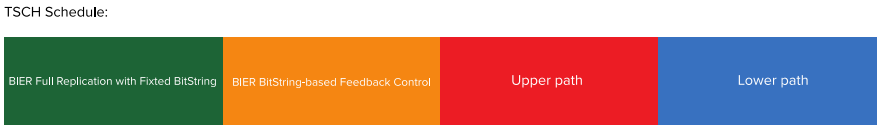
Figure 87 gives the schedule for the two deterministic tracks. We did not enable retries for each hop as it introduces two much delay. Only retries for concurrent transmissions are scheduled.

Both use the same schedule:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	A --> B														
2		A --> C													
3			B --> C												
4				C --> B											
5					B --> D	B --> D									
6					C --> E	C --> E									
7							D --> E								
8								E --> D							
9									D --> F	D --> F					
10									E --> G	E --> G					
11											F --> G				
12												G --> F			
13													F --> H		
14														G --> H	
15															



(a)



(b)

Figure 87 - Schedule (CDU) for bitString feedback control experiment

We did not enable snooping as it potentially makes the bitString based path control more complex and less precise. Of course, both the schedule and the mechanism can be further optimized. The experiments designed hereafter are just for a proof of concept.

The slots are allocated as below:

Usage	Number of Slots
Upper path	12
Lower path	12
BIER-TE full replication with a fixed bitString	14
BIER-TE bitString based feedback control	14
Shared slots for synchronization and RPL	2
Serial communication	15
Total Size	69

Table 19 - Slot allocation of the schedule

We conducted this experiment with same configurations twice, one during the night with a low level of external interference, one during the daytime with a strong external interference, which would be presented in the next two sections.

4.4.2 BitString Feedback Control Experiment during Nighttime

Experiment parameters:

Parameter	Value
Slotframe duration in seconds	1.035
Sending interval in seconds	2.5
Number of packets in each sending	4
Packets sent for each track	16141
Experiment Duration	22:28 - 09:51
BitString based link quality measurement interval	5 min
Multipath enablement threshold in burst loss length	2

Table 20 - Experimental parameters

Results Similar to the experiment 4.3.2, retries again show a very low ratio of packet loss. The track with bitString feedback control (colored in yellow) still has the highest packet loss among these tracks. This is because it takes a cost of one packet loss each time to enable multipath mechanisms.

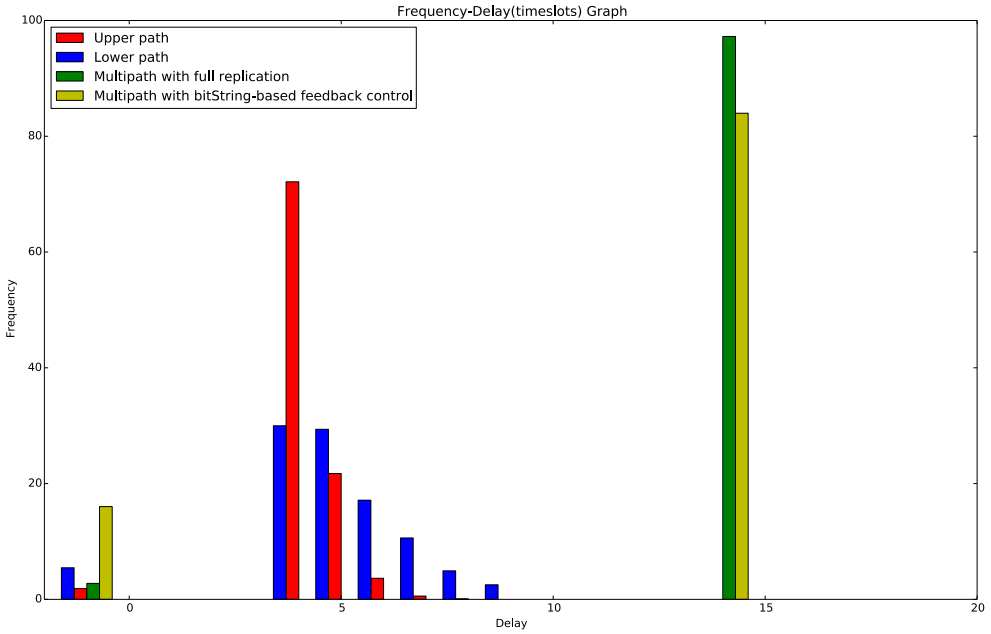


Figure 88 - Receiving delays (in Time Slots) frequency (% of messages sent)

However, when we look at the distribution of successive packet losses, we can see that most of the packet losses of the track (in yellow) are scattered, and only a few are successive losses in two, few are in three, and never in four. As shown in Table 21, after a successive loss of two packets, the multipath mechanism improves the reliability and makes the delivery has much more chance to succeed. This is not the case for single paths with retries, of which the packet losses are more 'bursty' as mentioned before, this is why we can still see a burst length of 4 on the blue track.

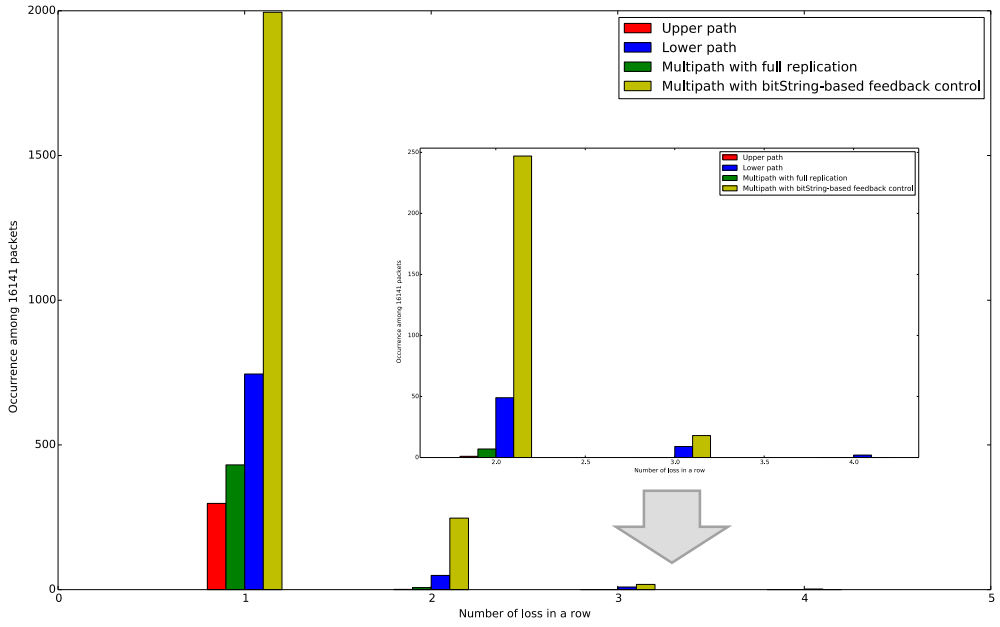


Figure 89 – Occurrence of successive packet losses

	Full Replication	BitString Control	Upper Path	Lower Path
Lose a packet	0.0271	0.1398	0.0185	0.0498
Lose 1 more	0.0160	0.1173	0.0033	0.0745
Lose 1 more	0	0.0679	0	0.1833
Lose 1 more	0	0	0	0.1818

Table 21 - Possibility of a packet loss if the previous one is lost

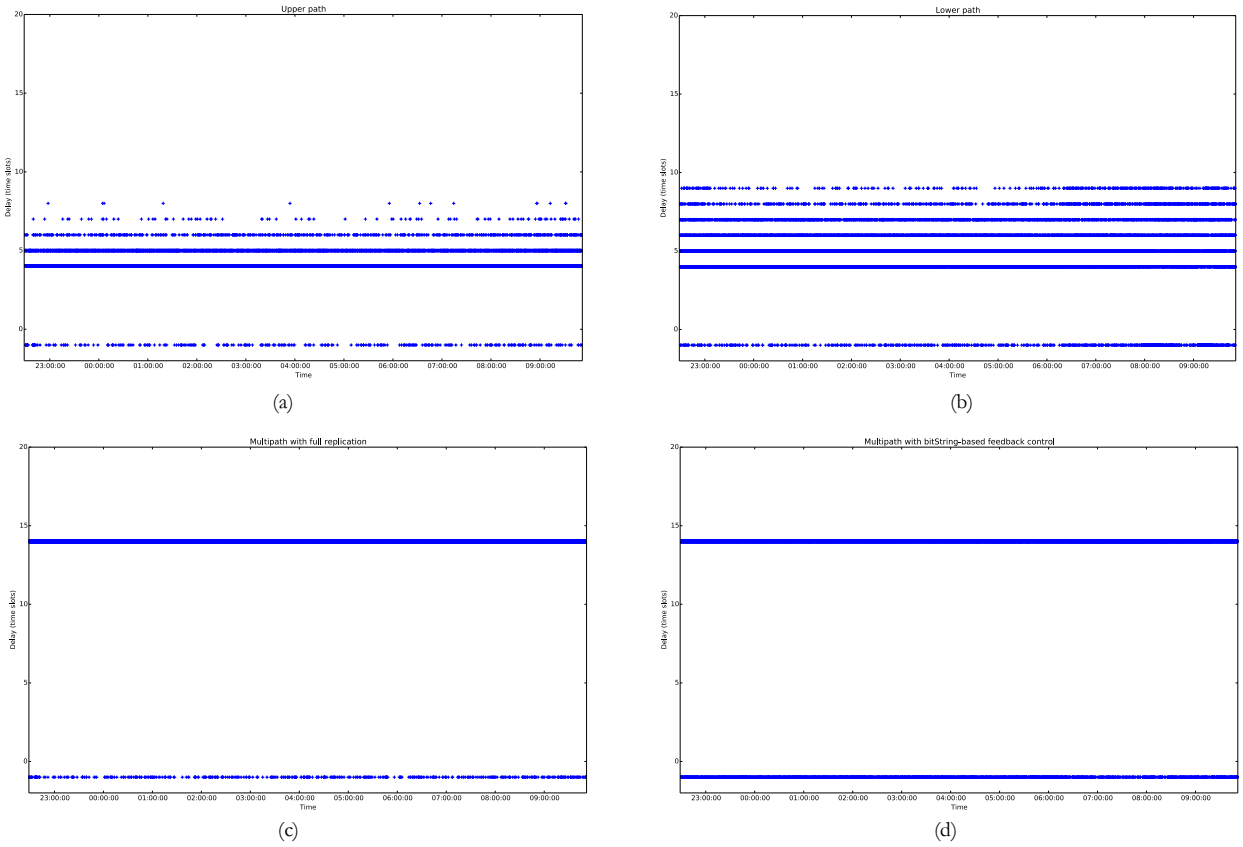


Figure 90 - Latency (unit : time slots) function of time. One packet is sent on every 2.5 seconds. Latency of -1 means that no packet was received. (a) upper path, (b) lower path, (c) multipath with full replication, (d) multipath with bitString control

From Figure 90 we can see the link quality at night is generally stable, although there is difference between the upper path and the lower path. Figure 91 shows the number of bits enabled over time for the bitString controlled track. As a bit enabled corresponds to a transmission in a bitString, the number of bits enabled for a track means the number of transmissions used to deliver a packet over network, which for sure has an impact on the overall energy consumption. As we can see during the hour from 22:31:00 to 23:31:00, most of time only 4 bits is enabled which corresponds to a single path, while the few crosses distributed over time at the value of 11 means the enablement of full packet replication after a successive loss of 2 packets. We can also notice that there are some periods during which the packets take more complex paths with 5 or 6 bits enabled in the bitString. Therefore, with bitString feedback, we can dynamically control packet replication activities to leverage the goods of multipath while keeping the energy consumption at a level similar to that of a single path.

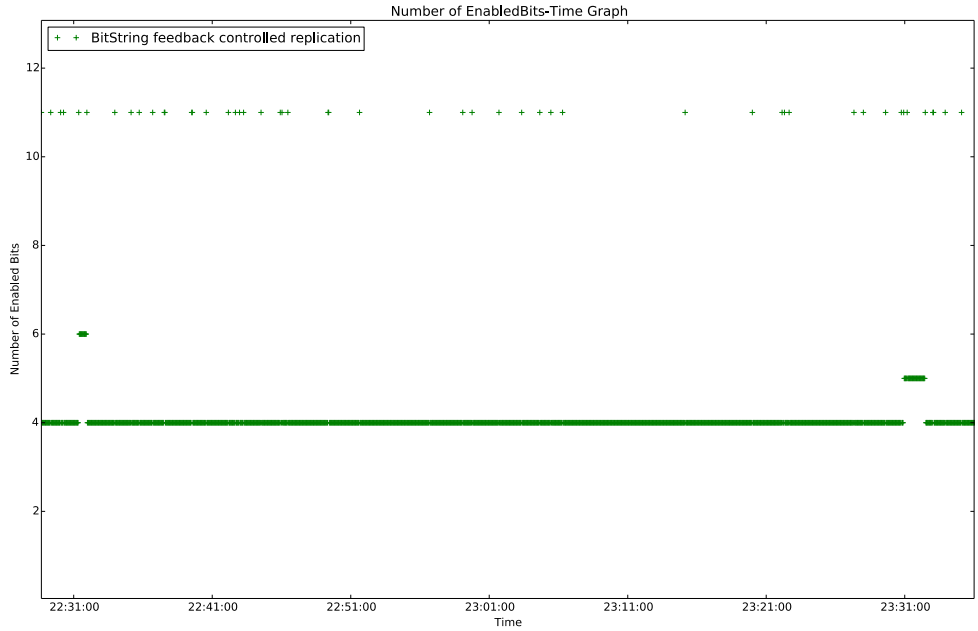


Figure 91 - Number of bits enabled over time for the bitString controlled track

Figure 92 - Link quality deduced by BIER bitStrings over time

Figure 92 shows the general quality of each link represented by bitIndex deduced from bitString feedback of the fixed full replication track. The Bit Reset Ratio (BRR) for each bit is computed as:

$$BRR = \frac{N_{reset}}{N_{feedback}}$$

	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

Figure 93 – webUI shows the BitIndex assignment of the experiment

BRR is computed every 5 minutes. It represents the contribution of a link to the overall reliability added by multipath mechanisms, potentially also to the packet delivery ratio. A bitStirng feedback means a succeeded packet delivery, while the bits reset in a bitString feedback mean the succeeded transmissions for a finished packet delivery. In this case, the links with bits reset contribute to this succeeded delivery, the ones without do not. The BBR of a bitIndex is the ratio that the link with the bit contributes among all the succeeded packets. We suppose this to some extent can infer the link quality. From the feature, we can find that links with bit 1, bit 2, and bit 4 are generally worse than other links, especially after 06:00:00, during which the links become more lossy and unstable.

Referring to the bitIndex assignment shown in Figure 93, we can find the links with lower qualities are the three links colored in red as below:

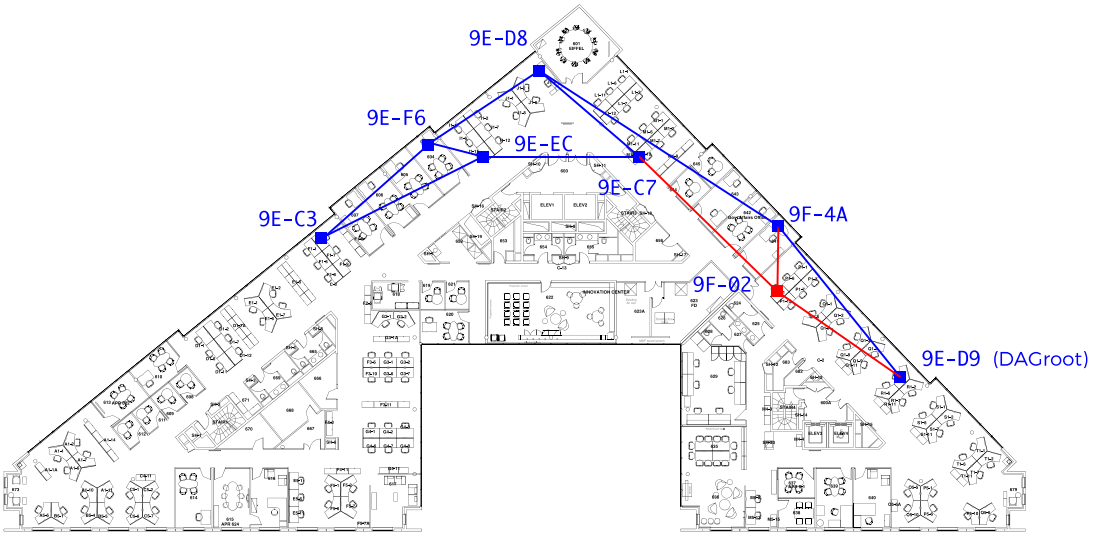


Figure 94 - Testbed topology

This explains the underperformance of the lower path as shown in *Figure 88 Figure 89 Figure 90*, and correspondingly we can see an increase of packet losses after 06:00:00 on lower path in *Figure 90*(b).

4.4.3 BitString Feedback Control Experiment during Daytime

Experiment parameters:

Parameter	Value
Slotframe duration in seconds	1.035
Sending interval in seconds	2.5
Number of packets in each sending	4
Packets sent for each track	4797
Experiment Duration	15:23 - 18:47
BitString based link quality measurement interval	5 min
Multipath enablement threshold in burst loss length	2

Table 22 - Experimental parameters

During the daytime we expect stronger external interferences and worse link conditions, and aim to evaluating the mechanisms under more unstable environments. We also use a metal box in order to interfere the modes randomly.

Results Multipath mechanism again shows a better performance in unstable environments. The upper path track becomes the one with highest packet losses. However, we can still see the lower path track has less packet loss than BitString controlled track.

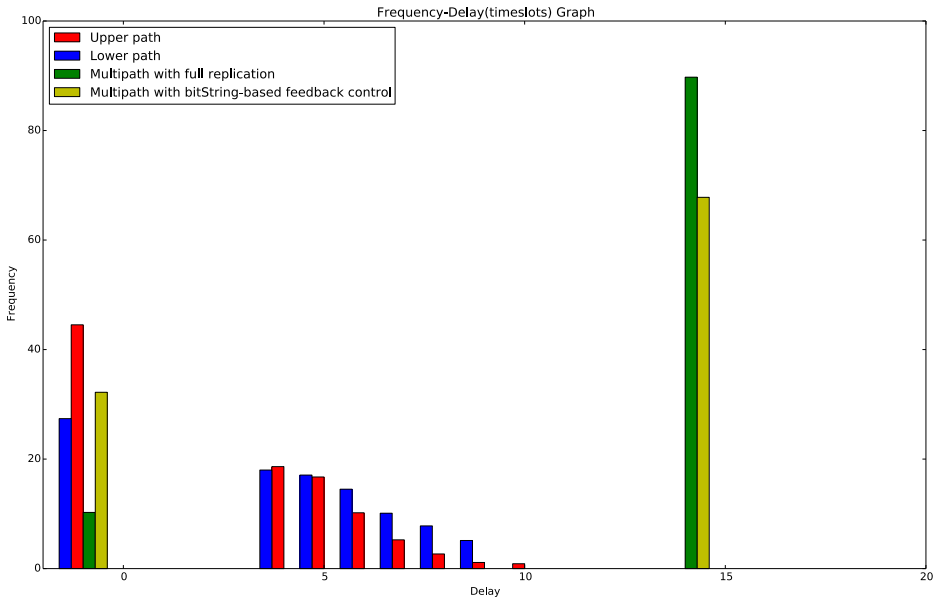


Figure 95 - Receiving delays (in Time Slots) frequency (% of messages sent)

When we look at the occurrence of successive packet losses of each track, multipath tracks have less burst losses with a length of four than single-path tracks because of the spatial diversity as discussed before. The bitString controlled track shows a very close performance with the full replication track after a burst loss of 2 packets as shown in Table 23, and it could be more close if we set the multipath enablement threshold to be 1. We can also see strong dependencies between packet losses in upper and lower tracks.

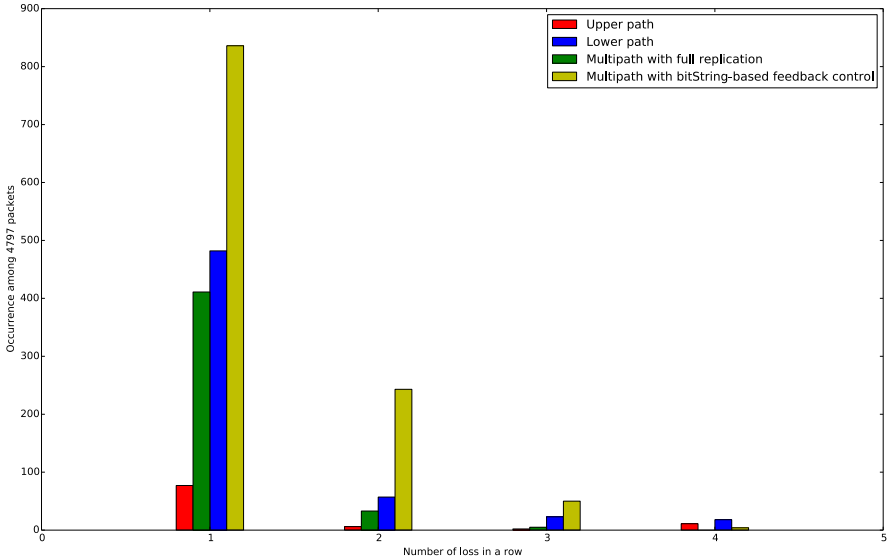


Figure 96 – Occurrence of successive packet losses

	Full Replication	BitString Control	Upper Path	Lower Path
Lose a packet	0.0936	0.2362	0.1044	0.1453
Lose 1 more	0.0846	0.2621	0.8443	0.3042
Lose 1 more	0.1316	0.1819	0.9811	0.7170
Lose 1 more	0	0.0741	0.9928	0.8487

Table 23 - Possibility of a packet loss if the previous one is lost

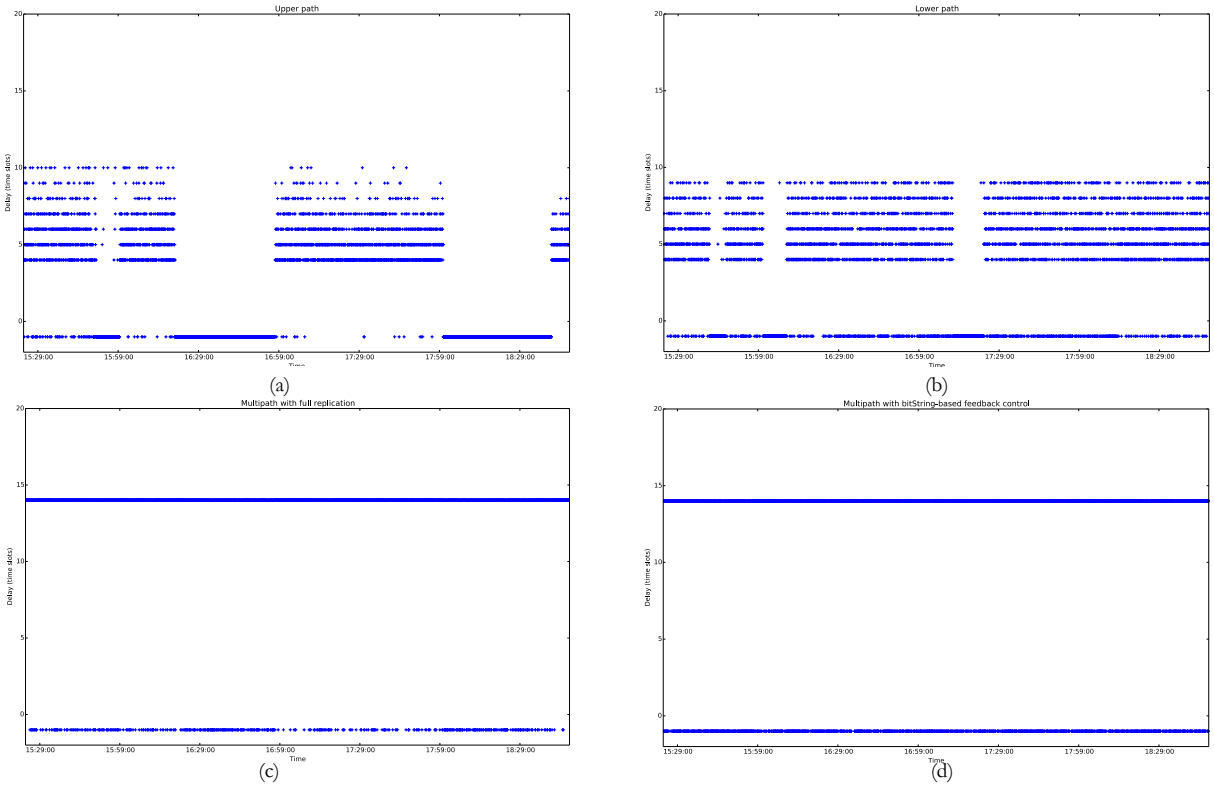


Figure 97 - Latency (unit : time slots) function of time. One packet is sent on every 2.5 seconds. Latency of -1 means that no packet was received. (a) upper path, (b) lower path, (c) multipath with full replication, (d) multipath with bitString control

Figure 97 gives the packet delivery conditions over time of each track, from which we can see that with the control loop, the BIER-TE track can bypass the link failures dynamically without keeping full replication all the time. Figure 98 shows the number of bits enabled over time for the bitString controlled track during the three hours. The bitString changes more often than during the night since the link quality become more unstable. However most of the time the number of enabled bits are maintained at a level of 4 or 5. Figure 99 shows the Bit Reset Ratio of the links.

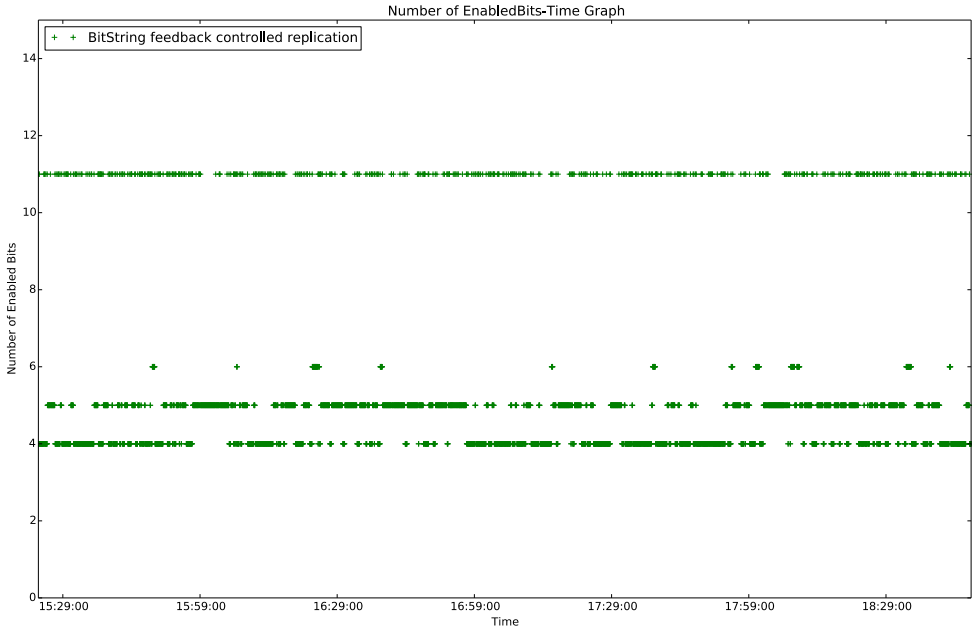


Figure 98 - Number of bits enabled over time for the bitString controlled track

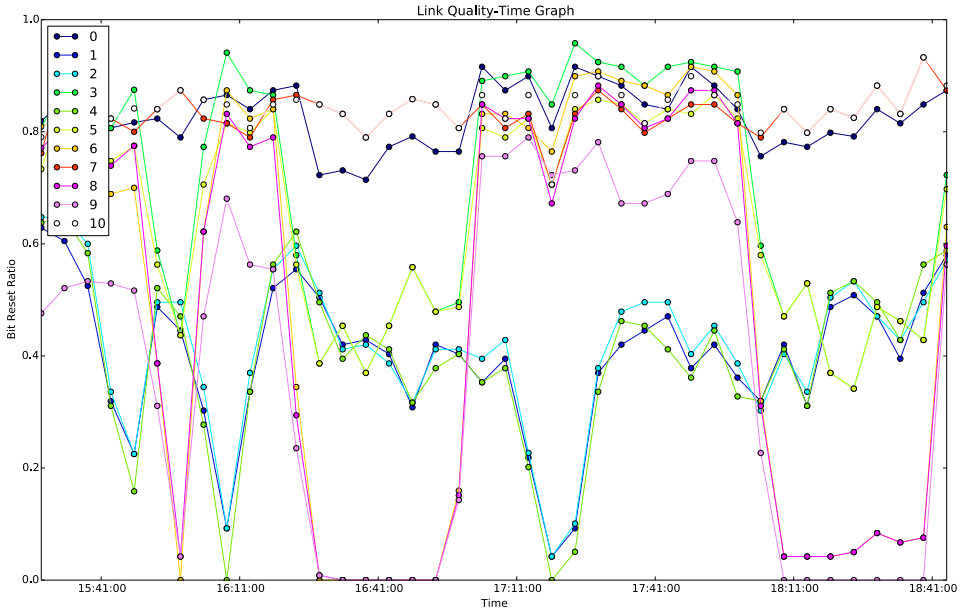


Figure 99 - Link quality deduced by BIER bitStrings over time

4.5 Conclusion

This chapter presents a set of experiments to validate the value of multipath based on BIER-TE protocol over 6TiSCH network. The results can be concluded as below:

- Multipath increases reliability by providing additional spatial diversity for a packet delivery. It makes the occurrence of packet losses more independent, and effectively reduce burst losses. It can also overcome node failures or link failures with redundant paths.
- Compared with retries, deterministic multipath based on BIER-TE introduces no jitter by design. By adopting overhearing (snooping) between parallel paths, multipath can provide a higher reliability than retries for low latency use cases. While in cases that are less sensitive to latency, retries seem to be more efficient under TSCH.
- Scheduling replication and elimination points would provide higher reliability, more precise and flexible path control, but also introduce delays and more power consumption. Using snooping can reach a similar reliability as replication, with less energy consumption, shorter bitString, and no additional delays, at the cost of more complex ARQ mechanism, possibly less efficient and less precise bitString based multipath control.
- Multipath and TSCH retries can be incorporated with a control loop based on BIER-TE, where normally single path with retries are used for efficient packet delivery and less power consumption, while multipath mechanism can be used to eliminate burst losses of packets. This concept has been implemented and validated in the last two experiments.

Of course, there are some flaws and confusions during the experiments, and also improvements that could have been done but we did not find additional time to conduct.

- The testbed topology is hardcoded and static during the whole experiment process, which is with either a shape of a ladder or a shape of two sides with crosses in between. However, the reality is that the topology of a LLN is often unstable, which causes that practically the testbed does not really represent the world. Moreover, the hardcoded neighborhood limits the scheduling patterns that we can design and implement for experiment, for example, we could have more snooping patterns such as when A sends to C, E can also listen.
- The metal box seems to interfere too much that almost kills the mote by putting it inside. A gentler alternative could have been just putting the metal box aside the antenna, which would make the experiments more practical.
- There are some experiments that could be designed and implemented further. Such as: 1) more snooping patterns can be tried as forementioned. 2) Review the experiment 4.3.4 and conduct further tests to examine the performance of concurrent transmissions with TSCH. 3) More experiments can be conducted with the topology of 4.3.5, for example with TSCH.
- The performance of control loop can be potentially further improved. For example, the schedule can be optimized with more retries, the snooping mechanisms can be introduced to reduce latency, and the multipath activation threshold can be made less (in our case one).
- More repeated experiments should have been done in order to consolidate the conclusions.

5. DEEPENINGS

5.1 Bi-directional Bit Issue for Link Failure Detection

In the ladder topology, the intermediate paired bundles are not symmetrical on schedule, as the transmission on the latterly scheduled bundle would occur only when the formerly scheduled did not. This may cause the bitString feedback sometimes confusing and inefficient for detecting link failures. For example, assume that we have a topology, bit assignment, and schedule as below:

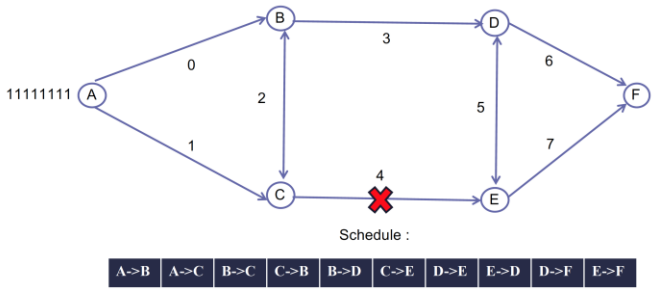
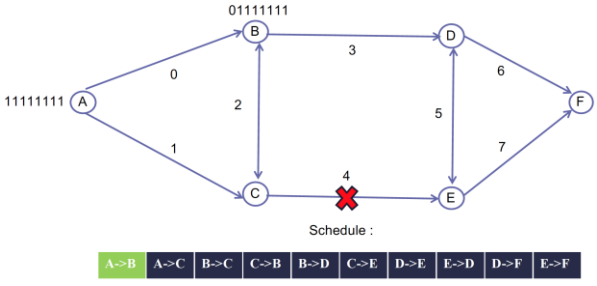
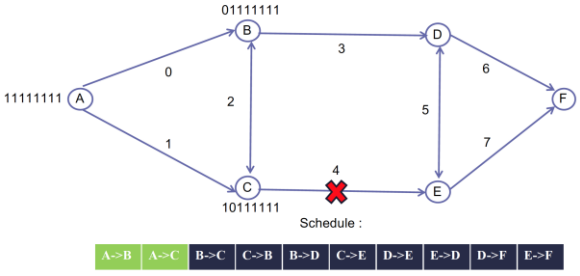


Figure 100 - an example for bi-directional bit issue

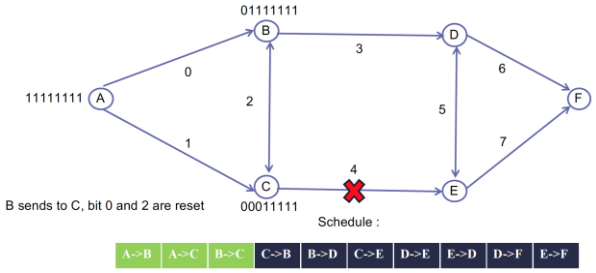
Now if there is a link failure happening on the time slot C→E, the multipath mechanism with full replication would be activated in order to detect the failure. One or more packets with a bitString '11111111' would traverse the topology link by link according to the BIFTs on each node:



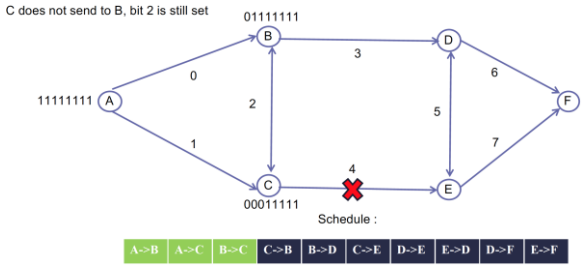
(1)



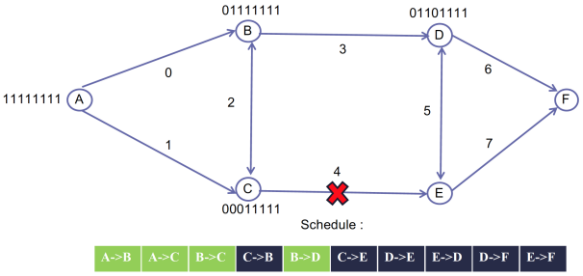
(2)



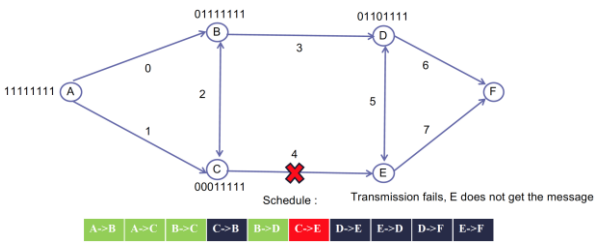
(3)



(4)



(5)



(6)

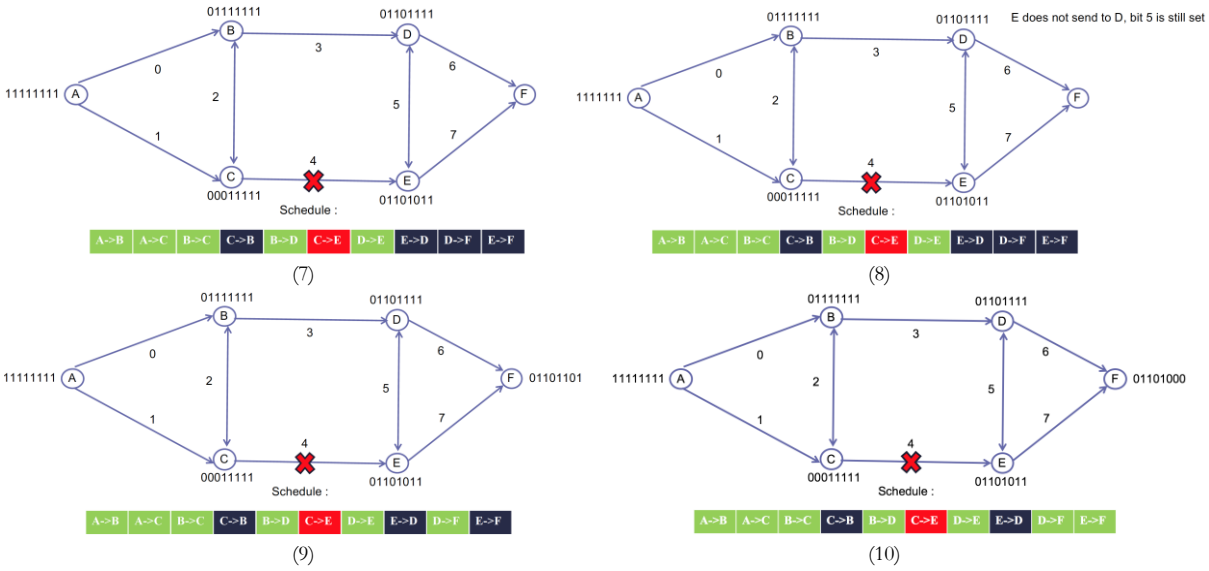


Figure 101 - illustration of the bi-directional bit issue

As we can see, the controller would receive an output bitString '01101000', based on which the controller could not determine which of those is true:

- transmission C->E failed
- transmissions A->C and B->C failed
- transmissions A->C and C->E failed
- transmissions B->C and C->E failed
- transmissions A->C, B->C and C->E failed

Zacharie proposed two solutions for this:

- use one bit for B->C and another one for C->B
- keep a single bit, send only the bitmap back to the other direction

The first one offers more precise path control with longer bitString and more power consumption, while the latter prevents longer bitStrings with new complex rules. Here we continue the discussion on the case with a breakage at the transmission E ->F:

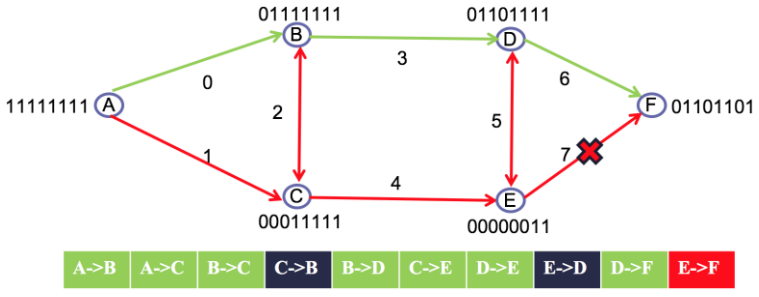


Figure 102 - a breakage at the link E->F

With the same schedule, a breakage at link E->F causes a more severe confusion to the controller. Based on the current implementation, interleaving the scheduling order of bi-directional bundles can be used to limit the number of irresolvable bits to be equal or less than three.

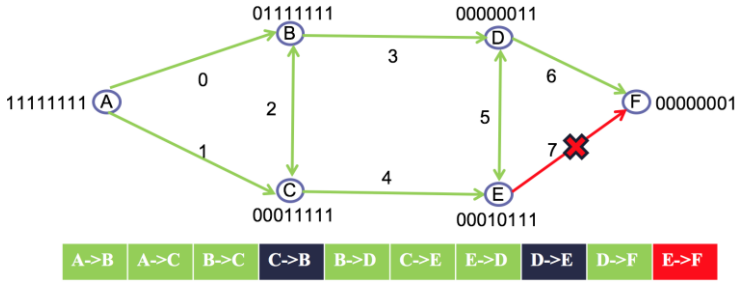


Figure 103 - exchange the order of slots E->D and D->E on schedule

If the former bi-directional bundles are scheduled with an order B->C first and then C->B, then the latter should be scheduled reversely with E->D first and then D->E. However of course, using one bit per direction is the most straight forward method for precise path control.

5.2 Fast Resynchronization

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.

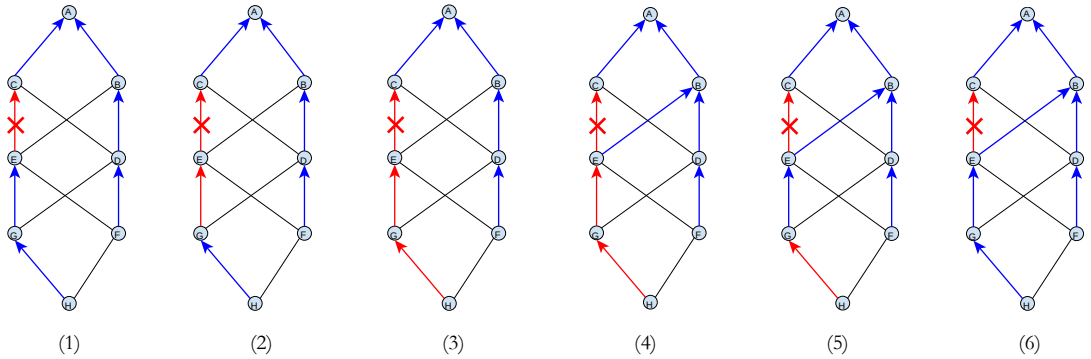


Figure 104 - an example of re-synchronization issue from RPL parent. (The red arrow means desynchronization)

Figure 104 gives an example showing the typical process of the resynchronization of a sequence of motes after one of their ancestors desynchronizes in a TSCH network. The expected solution is to make a mote choose a new RPL parent to synchronize to before it desynchronizes in case of a node or link failure, as shown below.

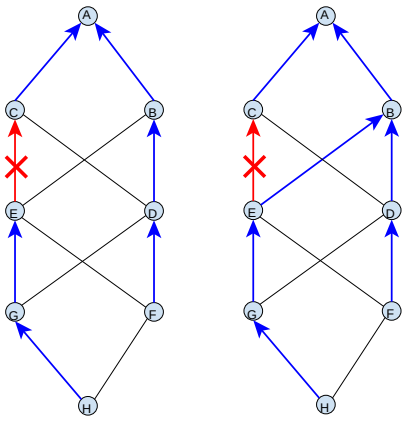


Figure 105 - fast resynchronization

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.

5.3 Reversible Links of the ARC Chain

In current implementation for the computation of ARC chain, the intermediate links in each ARC would be made reversible, which means the transmissions on both directions would be enabled. This makes unnecessary slots are scheduled in some cases which would results in additional delays and energy consumption. For example, current implementation would compute an ARC chain as Figure 106. However, the intermediate links colored in red can optionally be made unidirectional, i.e., only slots for the communications G->H and I->J are needed in order to reduce latency and energy consumption while maintaining path diversity.

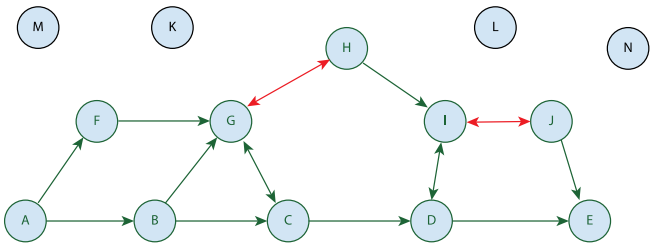


Figure 106 - An example ARC chain computed by current implementation

Apparently in current BIER-TE implementation, a better solution might be only making the links reversible between the nodes on which the higher ARC is rooted, i.e., making only links between sibling nodes reversible. This issue was noticed at the last moment of this internship, therefore, it is a pity that I could not manage to solve this. However, enabling all the intermediate links of each ARC to be reversible migth provide more possibilities and flexibilities for new mechanisms. For example, OR operations can be then enabled between transmissions H->I and H->G, i.e., if the transmission H->I fails, H can try to enable a secondary path by sending the packet to G. The mechanism could be done within a slot frame and increases reliability for each packet, unlike the bitString feedback based multipath mechanism which each time consumes one or more packet losses and slot frames to detect link failures and then modify bitStrings.

6. CONCLUSION

This internship validates experimentally the value of multipath mechanisms with packet replication and elimination over 6TiSCH LLNs, analyzes when and how the mechanisms should be activated and incorporated with other diversity techniques such as ARQs and channel hopping, and further explores centralized operations and methods to optimize the scheduling in terms of jitter, latency, loss ratio, reliability and energy consumption. The experiments are basically carried out as expected based on a real hardware testbed, and with the results we have been able to draw some conclusions: 1) multipath can increase reliability by providing additional spatial diversity for the packet delivery. It makes the occurrence of packet losses more independent, and effectively reduce burst losses. It can also overcome node failures or link failures with redundant paths. 2) Compared with retries, deterministic multipath based on BIER-TE introduces no jitter by design. It could provide a higher reliability than retries for low latency use cases. While in cases that are less sensitive to latency, retries seem to be more efficient under TSCH. 3) Scheduling replication and elimination points would provide higher reliability, more precise and more flexible path control, but also introduce delays and more power consumption. Using snooping can reach a similar reliability as replication, with less energy consumption, shorter bitString, and no additional delays, at the cost of a more complex ARQ mechanism, possibly less efficient and less precise multipath control. 4) The control loop experiment validates the concepts by combining dynamic multipath with retries and channel hopping, which shows a good integration whereby a single path with TSCH-based 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.

However of course, there are some limitations in this experimental study. More improvements and repeated tests could have been further implemented to consolidate the conclusions. Some interesting issues could also have been further researched on but we did not manage to find time to do. Such as, the testbed topology is hardcoded and static during the whole experiment process, which is with either a shape of a ladder or a shape of two sides with crosses in between. However, the reality is that the topology of a LLN is often unstable, which causes that practically the testbed does not really represent the world. Moreover, the hardcoded neighborhood limits the scheduling patterns that we can design and implement for experiment. In addition, the performance of control loop can be potentially further improved. For example, the schedule can be optimized with more retries, the snooping mechanisms can be introduced to reduce latency, and the multipath activation threshold can be made less. Finally, more repeated experiments could have been done in order to consolidate the conclusions. Therefore, there is still a long way to go for evaluation of multipath mechanisms over LLNs, and we hope this research work can be helpful for the continued study in the future.

BIBLIOGRAPHY

[1] J. Korhonen, J. Farkas, G. Mirsky, P. Thubert, Y. Zhuang and L. Berger, "DetNet Data Plane Protocol and Solution Alternatives," 04 August 2016. [Online]. Available: https://www.ietf.org/id/draft-dt-detnet-dp-alt-02.txt.

[2] E. Grossman, C. Gunther, P. Thubert, P. Wetterwald, J. Raymond, J. Korhonen and Y. Kaneko, "Deterministic Networking Use Cases," 04 July 2016. [Online]. Available: https://www.ietf.org/id/draft-ietf-detnet-use-cases-10.txt.

[3] N. Finn, P. Thubert and M. Johas Teener, "Deterministic Networking Architecture," 25 July 2016. [Online]. Available: https://www.ietf.org/id/draft-finn-detnet-architecture-07.txt.

[4] 6TiSCH WG Charter, "IETF Datatracker," [Online]. Available: https://datatracker.ietf.org/wg/6tisch/charter/. [Accessed 11 August 2016].

[5] D. Dujovne, T. Watteyne, X. Vilajosana and P. Thubert, "6TiSCH: deterministic IP-enabled industrial internet (of things)," *IEEE Communications Magazine,* pp. 36-41, 2014.

[6] T. Winter, P. Thubert, A. Brandt, S. Designs, J. Hui, R. Kelsey and P. Levis, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," March 2012. [Online]. Available: https://tools.ietf.org/html/rfc6550.html.

[7] P. Thubert and P. Bellagamba, "Available Routing Constructs," 22 January 2015. [Online]. Available: https://tools.ietf.org/html/draft-thubert-rtgwg-arc-03#section-1.

[8] P. Thubert, M. Rita Palattella and T. Engel, "6TiSCH Centralized Scheduling: when SDN Meet IoT," in *Proceedings of Standards for Communications and Networking (CSCN)*, 2015.

[9] Open Networking Foundation, ONF, "Software-Defined Networking (SDN) Definition," [Online]. Available: https://www.opennetworking.org/sdn-resources/sdn-definition. [Accessed 12 August 2016].

[10] N. Finn and P. Thubert, "Deterministic Networking Problem Statement," 4 April 2016. [Online]. Available: https://www.ietf.org/id/draft-ietf-detnet-problem-statement-00.txt.

[11] J. Vasseur, "Terms Used in Routing for Low-Power and Lossy Networks," January 2014. [Online]. Available: https://tools.ietf.org/html/rfc7102.

[12] IEEE Std 802.15.4e™, "Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)," New York, 2012.

[13] B. Kerkez, T. Watteyne, M. Magliocco, S. Glaser and K. Pister, "Feasibility analysis of controller design for adaptive channel hopping," in *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, 2009.

[14] T. Watteyne, M. Palattella and L. Grieco, "Using IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the Internet of Things (IoT): Problem Statement," May 2015. [Online]. Available: https://tools.ietf.org/html/rfc7554.

[15] I. Wijnands, E. Rosen, A. Dolganow, T. Przygienda and S. Aldrin, "Multicast using Bit Index Explicit Replication," 18 June 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-bier-architecture-04.

[16] Bit Indexed Explicit Replication (bier) Working Group, "Charter for Working Group," [Online]. Available: https://datatracker.ietf.org/wg/bier/charter/. [Accessed 13 August 2016].

[17] T. Eckert, G. Cauchie, W. Braun and M. Menth, "Traffic Engineering for Bit Index Explicit Replication BIER-TE," 8 July 2016. [Online]. Available: https://www.ietf.org/id/draft-eckert-bier-te-arch-04.txt.

[18] Z. Brodard, "Multipath redundancy for wireless Time-Slotted networks," 2016.

[19] P. Thubert, "An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4," 10 June 2016. [Online]. Available: https://www.ietf.org/id/draft-ietf-6tisch-architecture-10.txt.

[20] T. Clausen, U. Herberg and M. Philipp, "A Critical Evaluation of the "IPv6 Routing Protocol for Low Power and Lossy Networks" (RPL)," 2011.

[21] N. Tsiftes, J. Eriksson and A. Dunkels, "Low-Power Wireless IPv6 Routing with ContikiRPL," in *Proceedings of 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, 2010.

[22] F. Melakessou, M. Palattella and T. Engel, "Towards a New Way of Reliable Routing: Multiple Paths over ARCs," in *Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference)*, 2014.

[23] M. Pioro, A. Szentesi, J. Harmatos, A. Jttner, P. Gajowniczek and S. Kozdrowski, "On open shortest path first related network optimization problems," in *proceedings of Performance Evaluation 48, An International Journal*, 2002.

[24] D. Sidhu, T. Fu, S. Abdallah and R. Nair, "Open Shortest Path First (OSPF) Routing Protocol Simulation," in *proceedings of SIGCOMM*, San Francisco, 1993.

[25] E. Dijkstra, "A note on two problems in connexion with graphs," in *proceedings of Numerische Mathematik*, 1959.

[26] F. Melakessou, T. Cholez, J. François and M. Palattella, "Smart Routing Mechanisms Design," 2013.

[27] Q. Wang and X. Vilajosana, "6top Protocol (6P)," 25 July 2016. [Online]. Available: https://www.ietf.org/id/draft-ietf-6tisch-6top-protocol-02.txt.

[28] M. Palattella, P. Thubert, T. Watteyne and Q. Wang, "Terminology in IPv6 over the TSCH mode of IEEE 802.15.4e," 21 March 2016. [Online]. Available: https://www.ietf.org/id/draft-ietf-6tisch-terminology-07.txt.

[29] P. Thubert, "6TiSCH requirements for DetNet," 11 June 2015. [Online]. Available: https://tools.ietf.org/html/draft-thubert-6tisch-4detnet-01.

[30] S. Cui, A. J. Goldsmith and A. Bahai, "Energy-efficiency of MIMO and cooperative MIMO techniques in sensor networks," *IEEE Journal on selected areas in communications,* pp. 1089-1098, 2004.

[31] X. Vilajosana, P. Tuset, T. Watteyne and K. Pister, "OpenMote: Open-Source Prototyping Platform for the Industrial IoT," *International Conference on Ad Hoc Networks,* pp. 211-222, September 2015.

[32] T. Watteyne, V. Xavier, K. Branko, C. Fabien, W. Kevin, W. Qin, G. Steven and P. Kris, "OpenWSN: a standards-based low-power wireless development environment," *Transactions on Emerging Telecommunications Technologies,* pp. 480-493, 2012.

[33] P. Thubert, "Objective Function Zero for the Routing Protocol for Low-Power and Lossy Networks (RPL)," March 2012. [Online]. Available: https://tools.ietf.org/html/rfc6552#section-4. [Accessed August 2016].

[34] P. Thubert, C. Bormann, L. Toutain and R. Cragie, "6LoWPAN Routing Header," June 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-roll-routing-dispatch-00.

[35] P. Thubert, Z. Brodard, H. Jiang and G. Texier, "A 6loRH for BitStrings," 28 June 2016. [Online]. Available: https://tools.ietf.org/html/draft-thubert-6lo-bier-dispatch-00.

[36] openwsn-berkeley, "6TiSCH/IEEE802.15.4e and 6LoWPAN Wireshark dissectors," [Online]. Available: https://github.com/openwsn-berkeley/dissectors. [Accessed 1 September 2016].

[37] OpenWSN, "Home page," [Online]. Available: https://openwsn.atlassian.net/wiki/display/OW/Home. [Accessed 18 08 2016].

[38] OpenWSN, "Remote control your mote through raspberry pi," [Online]. Available: https://openwsn.atlassian.net/wiki/display/OW/Remote+control+your+mote+through+raspberry+pi. [Accessed 20 August 2016].

[39] R. Sudhaakar and P. Zand, "6TiSCH Resource Management and Interaction using CoAP," 9 March 2015. [Online]. Available: https://tools.ietf.org/html/draft-ietf-6tisch-coap-03.

GLOSSARY

6TiSCH IPv6 over the Timeslotted Channel Hopping mode of IEEE802.15.4e. It defines (i)the 6top sublayer; (ii) a set of protocols for setting up a TSCH schedule with a centralized and/or distributed approach, for managing the allocation of resources; and (iii) the architecture to bind them together, for use in IPv6 TSCH based networks.

ASN Absolute Slot Number, the total number of timeslots that has elapsed since the PAN coordinator has started the TSCH network. It is incremented by one at each timeslot. It is wide enough to not roll over in practice.

Bundle A group of equivalent scheduled cells, i.e. cells identified by different [slotOffset, channelOffset], which are scheduled for a same purpose, with the same neighbor, with the same flags, and the same slotframe. The size of the bundle refers to the number of cells it contains. Given the length of the slotframe, the size of the bundle translates directly into bandwidth. A bundle represents a half-duplex link between nodes, one transmitter and one or more receivers, with a bandwidth that amount to the sum of the cells in the bundle. A bundle is globally identified by (source MAC, destination MAC, TrackID).

Cell A single element in a TSCH schedule which is identified by a slotOffset, a channelOffset, a slotframeHandle. A cell can be scheduled or unscheduled.

ChannelOffset Identifies a row in a TSCH schedule. The number of available channelOffsets is equal to the number of available frequencies. The channelOffset translates into a frequency when the communication takes place, resulting in channel hopping.

CDU matrix Channel Distribution/Usage (CDU) matrix. Matrix of cells (i,j) representing the spectrum (channel) distribution among the different nodes in the 6TiSCH network. The CDU matrix has width in timeslots, equal to the period of the network scheduling operation, and height equal to the number of available channels. Every cell (i,j) in the CDU, identified by (slotOffset, channelOffset), belongs to a specific chunk. It has to be noticed that such a matrix which includes all the cells grouped in chunks, belonging to different slotframes, is different from the TSCH schedule.

GMPLS Generalized Multi-Protocol Label Switching, a 2.5 layer service that is used to forward packets based on the concept of generalized labels.

PCE Path Computation Element, the entity in the network which is responsible for building and maintaining the TSCH schedule when centralized scheduling is used.

Link	A communication facility or medium over which nodes can communicate at the link layer, i.e., the layer immediately below IP. Thus, the IETF parlance for the term "Link" is adopted, as opposed to the IEEE802.15.4e term. In the context of the 6TiSCH architecture, which applies to Low Power Lossy Networks (LLNs), an IPv6 subnet is usually not congruent to a single link and techniques such as IPv6 Neighbor Discovery Proxying are used to achieve reachability within the multilink subnet. A link is distinct from a track. Finally, from the Layer 3 perspective (where the inner complexities of TSCH operations are hidden to enable classical IP routing and forwarding), a single radio interface may be seen as a number of Links with different capabilities for unicast or multicast services.
SlotOffset	Identifies a column in the TSCH schedule, i.e., the number of timeslots since the beginning of the current iteration of the slotframe.
Slotframe	A MAC-level abstraction that is internal to the node and contains a series of timeslots of equal length and priority. It is characterized by a slotframe_ID, and a slotframe_size. Multiple slotframes can coexist in a node's schedule, i.e., a node can have multiple activities scheduled in different slotframes, based on the priority of its packets/traffic flows. The timeslots in the Slotframe are indexed by the SlotOffset; the first timeslot is at SlotOffset 0.
TF	Track Forwarding. It is the simplest and fastest forwarding model supported by 6TiSCH. It is a GMPLS-like forwarding model. The incoming bundle (and thus, the input cell) characterizes the flow and indicates the outgoing bundle (and output cell).
Timeslot	A basic communication unit in TSCH which allows a transmitter node to send a frame to a receiver neighbor, and that receiver neighbor to optionally send back an acknowledgment.
Track	A determined sequence of cells along a multi-hop path. It is typically the result of a track reservation. The node that initializes the process for establishing a track is the owner of the track. The latter assigns a unique identifier to the track, called TrackID.
TSCH	Time Slotted Channel Hopping, a medium access mode of the IEEE802154e standard that uses time synchronization to achieve ultra low-power operation and channel hopping to enable high reliability.
ARC	a new routing algorithm which can significantly improve network utilization with fault-tolerance, fast rerouting, load balancing and multipath features.

Schedule	A matrix of cells, each cell indexed by a slotOffset and a channelOffset. The TSCH schedule contains all the scheduled cells from all slotframes and is sufficient to qualify the communication in the TSCH network. The "width of the matrix is equal to the number of scheduled timeslots in all the concurrent active slotframes. The number of channelOffset values (the "height" of the matrix) is equal to the number of available frequencies.
RPL	IPv6 Routing Protocol for Low-power and Lossy Networks (RPL) provides a mechanism whereby multipoint-to-point traffic from devices inside the LLN towards a central control point as well as point-to-multipoint traffic from the central control point to the devices inside the LLN are supported.
DODAG root	A DODAG root is the DAG root of a DODAG. The DODAG root may act as a border router for the DODAG; in particular, it may aggregate routes in DODAG and may redistribute DODAG routes to other routing protocols.
DODAG rank	A node's Rank defines the node's individual position relative to other nodes with respect to a DODAG root. Rank strictly increases in the Down direction and strictly decreases in the Up direction. The exact way Rank is computed depends on the DAG's Objective Function (OF). The Rank may analogously track a simple topological distance, may be calculated as a function of link metrics, and may consider other properties such as constraints.
DODAG parent	A parent of a node within a DODAG is one of the immediate successors of the node on a path towards the DODAG root. A DODAG parent's Rank is lower than the node's.
SDN	Software-Defined Networking (SDN) is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. This architecture decouples network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services.
BIER	Bit Indexed Explicit Replication is a new multicast forwarding protocol which is defined in the Internet draft. When a multicast data packet enters a “BIER domain”, the ingress router determines a set of egress routers to which the packet needs to be sent, and then encapsulates the packet in a “BIER header”. The BIER header contains a bitString in which each bit represents exactly one egress router in the domain. To send a packet to a particular set of egress nodes, the ingress node sets the bits for each of those egress nodes, and clears other bits in the bitString.

BIER-TE a Traffic Engineering protocol inspired from BIER and is being standardized in. It forwards and replicates packets like BIER based on a bitString in the packet header but it does not require an IGP. BIER-TE supports traffic engineering by explicit hop-by-hop forwarding and loose hop forwarding of packets, and enables to activate packet replication and elimination functions in a manner abstract to the data plane forwarding information, that is, an adjacency represented by a bit in the BIER header can correspond to an Ethernet hop, a routing segment, etc.

OpenWSN serves as a repository for open-source implementations of protocol stacks based on Internet of Things standards, using a variety of hardware and software platforms.

LLNs Low-Power and Lossy Networks (LLNs) are a class of network in which both the routers and their interconnect are constrained. LLN routers typically operate with constraints on processing power, memory, and energy (battery power). Their interconnects are characterized by high loss rates, low data rates, and instability. LLNs are comprised of anything from a few dozen to thousands of routers.

ARQ Automatic Repeat reQuest (ARQ), also known as Automatic Repeat Query, is an error-control method for data transmission that uses acknowledgements (messages sent by the receiver indicating that it has correctly received a data frame or packet) and timeouts (specified periods of time allowed to elapse before an acknowledgment is to be received) to achieve reliable data transmission over an unreliable service. If the sender does not receive an acknowledgment before the timeout, it usually re-transmits the frame/packet until the sender receives an acknowledgment or exceeds a predefined number of re-transmissions.

DetNet Deterministic Networking provides a capability to carry specified unicast or multicast data flows for real-time applications with extremely low data loss rates and bounded latency. Techniques used include: 1) reserving data plane resources for individual DetNet flows in some or all of the intermediate nodes (e.g. bridges or routers) along the path of the flow; 2) providing explicit routes for DetNet flows that do not rapidly change with the network topology; and 3) distributing data from DetNet flow packets over time and/or space to ensure delivery of each packet's data' in spite of the loss of a path.

BitString A BitString indicates a continuous sequence of bits indexed by an offset in the sequence. The leftmost bit is bit 0 and corresponds to the value 0x80 of the leftmost octet in the BitString.

TABLE OF FIGURES

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. "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.