University of South Carolina
Scholar Commons

Theses and Dissertations

Fall 2020

Providing Predictable Performance during Network Contingencies

Phani Krishna Penumarthi

Follow this and additional works at: https://scholarcommons.sc.edu/etd

 Part of the Computer Sciences Commons

Recommended Citation

Penumarthi, P. K.(2020). *Providing Predictable Performance during Network Contingencies.* (Doctoral dissertation). Retrieved from https://scholarcommons.sc.edu/etd/6171

This Open Access Dissertation is brought to you by Scholar Commons. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact digres@mailbox.sc.edu.

Providing Predictable Performance during Network Contingencies

by

Phani Krishna Penumarthi

Bachelor of Technology
Acharya Nagarjuna University, 2007

Master of Science by Research
Indian Institute of Technology Madras, 2013

Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy in

Computer Science

College of Engineering and Computing

University of South Carolina

2020

Accepted by:

Srihari Nelakuditi, Major Professor

Jason M O'Kane, Committee Member

Sanjib Sur, Committee Member

Ioannis Rekleitis, Committee Member

Guoan Wang, Committee Member

Cheryl L. Addy, Vice Provost and Dean of the Graduate School

© Copyright by Phani Krishna Penumarthi, 2020
All Rights Reserved.

Dedication

Dedicated to the innumerous teachers that ever taught me,

including my parents, wife (Soumya), and daughter (Arya)

Acknowledgments

Let Noble Thoughts Come to You from All Directions – RigVeda

Firstly, I thank my dissertation advisor, Prof. Srihari Nelakuditi, for believing in me and nurturing my intellectual capabilities towards a realistic destination. I am forever indebted to him, for all the technical discussions with him only made me stronger, holistically. I strongly admire his commitment to research and work-life balance skills, which are hard to achieve in pair.

Writing this dissertation is possible only with the help of my collaborators. I would like to thank Jason O'Kane, Ioannis Rekleitis, and Alberto Quattrini Li: for providing me a chance to learn how robots can be moved to achieve a functionality. Many thanks to Alberto for being very patient and guiding me through out the process. I would like to thank Zafar Qazi, Vyas Sekar, and Samir R Das: for providing me a great opportunity to work on re-configuring the LTE architecture.

I would like to thank Sanjib Sur and Guoan Wang for agreeing to serve on my thesis committee. Their suggestions were always helpful and motivated me to achieve better results.

Throughout my doctoral studies, I have been supported by the University of South Carolina as a Graduate Instructor. I thank each faculty of the Department of Computer Science, for providing a conducive environment to conduct research. I also thank each staff member of the university for their co-operation, whenever required.

I would like to thank my wife, Soumya, for supporting me through all the ups and downs of my doctoral studies. She is the first person to read/listen to all my manuscripts/presentations and provide an honest, constructive and detailed feedback.

This thesis would not have been possible without her support, help, and encouragement. I am forever indebted to her. I would like to thank my parents and in-laws for letting me follow my passion. I am really grateful for the values and wisdom they imparted on me and for their belief in me. I am deeply indebted to each and every one of them.

Studying in a different country is always challenging. I thank all the people that helped me stay focused and encouraged me to complete my dissertation.

Abstract

In IP backbone networks, packets may get dropped due to: i) lack of viable next hops when a link/router fails, ii) forwarding loops during network convergence, and iii) buffer overflows in case of congestion. Similarly, packets may be lost in wireless networks due to variations in signal strength between a pair of mobile nodes. This dissertation explores the possibility of providing a predictable performance during such network contingencies in wired backbone networks and robotic wireless networks.

First, we study the feasibility of developing a combination of local reroute and global update mechanisms that can achieve loop-free convergence, while performing disruption-free forwarding around a failed link/router, without carrying any additional information in the IP datagrams and with out needing any coordination between routers. We show that order of updates rarely matters for loop-free convergence when failure inference based fast reroute (FIFR) scheme with interface-specific forwarding is employed for dealing with link or router failures. In the rare cases where order matters, it can be coupled with progressive link metric increments to ensure loop-freedom with unordered updates of forwarding tables. We also demonstrate that, apart from providing protection against failures, FIFR can also be utilized to mitigate packet drops due to network congestion caused by micro traffic bursts.

Second, we address the problem of constructing a communication map, which encodes information on whether two robots at given locations can communicate using a wireless network. Unlike previous offline approaches that do not utilize data measured by robots, we propose an online method, utilizing Gaussian Processes, to efficiently build a communication map with multiple robots, by exploiting prior com-

munication models that can be derived from the physical map of the environment. Our evaluation, using a team of TurtleBot 2 platforms, confirms that the proposed method requires robots to take fewer signal strength measurements and travel less distance, and yet obtain similar accuracy as methods that consider all the locations in the environment.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The fundamental service provided by a wired or a wireless network is to deliver packets to their destinations. Packet delivery can however be adversely affected by some network and traffic conditions. In IP backbone networks, packets may get dropped due to: i) lack of viable next hops when an adjacent link or router fails, ii) transient forwarding loops during network convergence, and iii) buffer overflows at routers in case of sudden micro-bursts of traffic. Similarly, for a team of robots communicating with each other to perform search and rescue operations, packets could be lost before reaching the destination, due to bit errors caused by the variations in signal strength between a pair of robots. This dissertation, which consists of two parts, explores the possibility of providing a predictable performance during such network contingencies in both wired backbone networks and robotic wireless networks.

1.1 *Part One*: Traditional Wired Networks

The key objectives of an intra-domain routing protocol such as OSPF in a traditional network with a distributed control plane are *reachability*, i.e., forward packets to their destinations, and *optimality*, i.e., along the shortest paths. To ensure optimality, any changes in the link state need to be propagated across the network, so that all routers recompute new shortest paths and install corresponding forwarding entries. On the other hand, to respond to failures quickly, due to relatively long convergence delay associated with a link state update, optimality is often traded-off for reachability, by having routers adjacent to failures perform local rerouting, without

invoking the control plane. However, when a failure lasts beyond a certain duration, it is desirable to trigger a link state update so that all routers recompute the optimal routes in the new topology. But a straightforward link state update can cause transient forwarding loops during the convergence process unless routers coordinate to install their forwarding entries in a particular order. This dissertation studies the feasibility of developing a combination of *local reroute and global update mechanisms that can achieve loop-free convergence, while performing disruption-free forwarding around a failed link, without carrying any additional information in the IP datagram and without requiring any signaling between routers.*

Among all the IP fast reroute schemes that provide full protection against any single non-partitioning failure, we consider failure inference based fast route (FIFR), as it is an approach that does not need any changes to the IP datagram. We observe that FIFR, due to the presence of back hop entries, mitigates the problem of forwarding loops during convergence with unordered updates. Specifically, we find that *by decoupling the installation of interface-independent forwarding entries and interface-specific back hop entries, convergence can be made loop-free and order-agnostic, except in rare cases.* In those rare cases, we propose to employ the progressive increment method and show that the resulting link metric sequence length is short. We extend this approach to handle link up events and also ensure that convergence is loop-free even for router down and up events. Overall, the proposed approach, referred to as FIFR++, has four key features that make it attractive for deployment. 1) It guarantees loop-free fast rerouting around a failure to any reachable destination. 2) It provides loop-free fast convergence to optimal forwarding after a failure. 3) It does not require any changes to the IP datagram. 4) It does not require any coordination between routers.

Apart from link or router failures, packets drops may also be caused by network congestion [4, 56]. There have been several works on handling persistent congestion

through traffic engineering and network-wide redistribution of traffic. However, these approaches are not suitable for dealing with sudden and short spikes of traffic, referred to as micro bursts, that are fairly common. The time scale of network-wide reaction is too slow for dissipating micro bursts that last for a few microseconds [66, 59]. This dissertation explored the feasibility of employing FIFR for performing local rerouting when a buffer at an outgoing interface is full. We demonstrate that, apart from providing protection against link/router failures, FIFR can also be utilized to mitigate packet drops due to network congestion caused by micro traffic bursts.

1.2 *Part Two*: Robotic Wireless Networks

This dissertation also addresses the problem of building a communication map of an environment using multiple robots. A communication map encodes information on whether two robots can communicate using a wireless network when they are at two arbitrary locations and plays a fundamental role in a multi-robot system deployment to reliably and effectively achieve a variety of tasks, such as environmental monitoring and exploration. Previous work considered only scenarios with a fixed base station and designed offline methods, which did not exploit data collected online by the robots. This dissertation proposes Gaussian Process-based online methods to efficiently build a communication map with multiple robots. Such robots form a mesh network, where there is no fixed base station. Specifically, we provide two leader-follower online sensing strategies to coordinate and guide the robots while collecting data. The number of RSSI measurements used to update the communication map, and the number of candidate locations where robots should go are reduced, by exploiting prior communication models that can be built from the physical map of the environment. Our evaluation, using a team of TurtleBot 2 platforms, confirms that the proposed method requires robots to take fewer measurements and travel less

distance, and yet get similar accuracy as methods that consider all the locations in the environment.

1.3 Contributions

This dissertation makes the following research contributions.

1. It proposes FIFR++, a combination of local reroute and global update mechanisms, that can achieve loop-free convergence, while performing disruption-free forwarding around a failure, without carrying any additional information in the IP datagram and without requiring any signaling between routers. To the best of our knowledge, FIFR++ is the only scheme that has these desirable features.

2. It makes a creative reuse of interface-specific forwarding entries, meant for fast reroute in case of a failure, to extend the FIFR++ approach to provide loop-free fast convergence for link up events also. Further, it demonstrates that the proposed approach works as well for handling router failure and restoration too.

3. It demonstrates that, apart from providing protection against link and router failures, FIFR can also be employed for rerouting packets that encounter full buffers at routers to mitigate packet drops due to micro traffic bursts.

4. It develops an online method for efficiently building a communication map with multiple robots, leveraging prior communication models derived from the physical map of the environment. It shows that the proposed method requires robots to take fewer measurements and travel less distance, and yet obtain similar accuracy as methods that consider all the locations in the environment.

Part I

Traditional Wired Networks

Chapter 2

Background on IP Fast Reroute and Network Convergence

The key objectives of an intra-domain routing protocol such as OSPF in a traditional network with a distributed control plane are *reachability*, i.e., forward packets to their destinations, and *optimality*, i.e., along the shortest paths. To ensure optimality, any changes in the link state need to be propagated across the network, so that all routers recompute new shortest paths and install corresponding forwarding entries. On the other hand, to respond to failures quickly, due to relatively long convergence delay associated with a link state update, optimality is often traded-off for reachability, by having routers adjacent to failures perform local rerouting, without invoking the control plane. However, when a failure lasts beyond a certain duration, it is desirable to trigger a link state update so that all routers recompute the optimal routes in the new topology. But a straightforward link state update can cause transient forwarding loops during the convergence process unless routers coordinate to install their forwarding entries in a particular order. This dissertation studies the feasibility of developing a combination of *local reroute and global update mechanisms that can achieve loop-free convergence, while performing disruption-free forwarding around a failed link, without carrying any additional information in the IP datagram and without requiring any signaling between routers.*

There have been many proposals for performing fast reroute in traditional IP networks with a distributed control plane [8, 1, 63, 54, 9]. Many of these schemes

require encapsulation [8] or modification to the IP header to carry additional bits of information [1]. Among all the IP fast reroute schemes that provide full protection against any single non-partitioning failure, we consider *failure inference based fast route* (FIFR) [45], as it is an approach that does not need any changes to the IP datagram. Under FIFR, routers adjacent to a failed link or router perform local rerouting around the failure, without notifying non-adjacent routers about the failure. The non-adjacent routers utilize *back hop* entries, which are associated with each interface along the reverse shortest path and are precomputed based on potential inferred failures that could cause a packet for a given destination to arrive at that interface, to ensure loop-free forwarding to the destination. While FIFR is suitable for short-lived failures, forwarding packets to the point of failure and then rerouting them is undesirable for longer-lasting failures. In such cases, it is preferable to trigger a link state update and initiate re-convergence to optimal routes.

During the convergence period, i.e., between the time a link state update is initiated and the time all routers install new forwarding entries, packets may be forwarded by some routers based on the new topology and others based on the old topology, potentially leading to forwarding loops. To prevent loops, [19] proposed a scheme that imposes a certain order between the FIB updates of different routers. While it guarantees loop-free convergence, it requires some form of coordination among routers to order their updates. As an alternative, a progressive link metric increment method has been proposed [20], which sends a sequence of updates such that each update is loop-free, regardless of the order of updates. But the downside of this incremental update method is that it takes longer to converge to the target state.

This dissertation explores how we can achieve fast loop-free convergence while performing fast reroute with FIFR, without necessitating coordination between routers. Before we discuss the proposed approach, in this chapter, we present some background material about the schemes upon which the proposed approach is built.

2.1 Failure Inference based Fast Reroute

In this section, we explain the core idea behind the FIFR approach, using a simple example. We refer the reader to [63, 45] for full details. Consider the topology shown in Fig. 2.1, where each link is labeled with its cost. Suppose the link between routers B and E is down, and only B and E are aware of the failure. Imagine forwarding a packet from source A to destination F. Based on the link costs, the shortest path from A to F is A→B→E→F. So, router A will forward the packet to its next hop B. Router B, being adjacent to the failed link B–E, initiates local rerouting. Since the shortest path to destination F, without the B–E link, is B→A→C→E→F, it forwards the packet back to A. Normally, this would cause the packet destined for F to go back and forth between routers A and B, resulting in a forwarding loop.

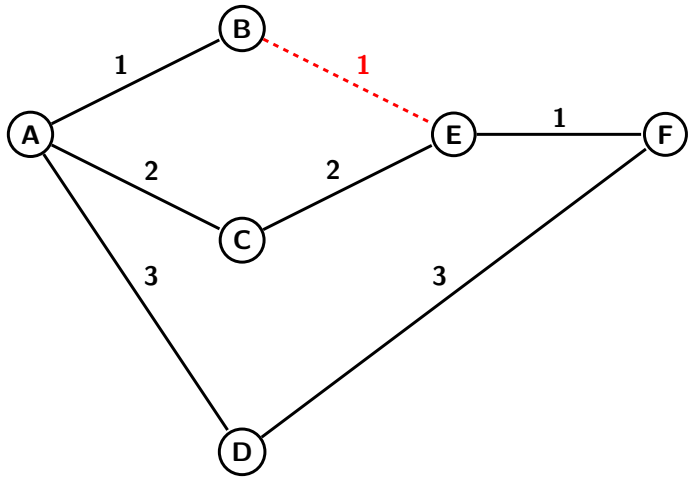


Figure 2.1: A topology used for illustration of FIFR and progressive updates.

Under FIFR, however, A infers potential failures along the shortest path to F that would cause the packet to arrive at A from its usual next hop B (i.e., through the *unusual* incoming interface B→A). It is apparent that the failure of link B–E would cause the packet for F to arrive through interface B→A. Similarly, the failure of E–F would also cause the packet destined for F to arrive through interface B→A, as the shortest path from E to F without link E–F is E→B→A→D→F. Since A does

not know which of these links actually failed, it excludes all such candidate links to find an alternate next hop (which we refer to as *back hop*), which is D, for packets destined for F arriving through interface B→A.

We observe that the resulting back hop would be the same, if we were to exclude just one of those candidates links, referred to as the *key link*, that is closest to the destination. In this example, the key link for destination F and interface B→A is E–F. Router A can pre-compute a key link per destination for each of its interfaces (which may be none if there are no candidate links), and compute the corresponding back hops. Table 2.1 lists the key links and back hops for all combinations of unusual incoming interfaces and destinations in Fig. 2.1.

Table 2.1: Key Links and back hops for unusual incoming interfaces for the topology in Fig. 2.1

Interface	Destination	Key Link	Back Hop
B→A	E	B–E	C
B→A	F	E–F	D
A→B	C	A–C	E
A→B	D	A–D	E
E→B	C	C–E	A
E→B	F	E–F	A
A→C	B	A–B	E
E→C	B	B–E	A
F→D	E	E–F	A
B→E	A	A–B	C
F→E	D	D–F	B

Effectively, FIFR computes interface-specific forwarding entries, i.e., a packet's next hop depends not only on its destination but also on the incoming interface, i.e.,

the previous hop. Most of these entries would be identical to usual forwarding entries based on destination only, independent of the incoming interface (as if the packet originated at this router). Only the unusual interfaces that lie along the reverse shortest path from the point of failure to the destination have back hops.

To put this formally, under FIFR, each router i has a forwarding entry $\mathcal{F}_i^d$ per each destination d. In addition, it keeps a *back hop* entry $\mathcal{B}_{j\rightarrow i}^d$ per each destination d and neighbor $j \in \mathcal{F}_i^d$ (both $\mathcal{F}_i^d$ and $\mathcal{B}_{j\rightarrow i}^d$ are sets, as there could be multiple shortest paths of equal cost). Let $\mathcal{K}_{j\rightarrow i}^d$ be the farthest link along a shortest path from i to d, whose failure would cause the packet to d arrive through interface $j\rightarrow i$. While $\mathcal{F}_i^d$ is the set of usual next hops from i to d, $\mathcal{B}_{j\rightarrow i}^d$ is the set of next hops from i to d without the link $\mathcal{K}_{j\rightarrow i}^d$. When $\mathcal{K}_{j\rightarrow i}^d$ is $\varnothing$, $\mathcal{B}_{j\rightarrow i}^d$ is the same as $\mathcal{F}_i^d$. Once the set of next hops and back hops for each destination are computed, packets can be forwarded as follows. A packet originating at i to destination d is forwarded to $\mathcal{F}_i^d$. A packet destined for d arriving at i through neighbor j is forwarded to $\mathcal{B}_{j\rightarrow i}^d$ if $j \in \mathcal{F}_i^d$, otherwise to $\mathcal{F}_i^d$. Moreover, a packet to d that were to be forwarded to the next hop j is rerouted by i to $\mathcal{B}_{j\rightarrow i}^d$ when the link i–j is down and all other routers are not yet notified and converged to the new topology.

The forward hop and back hop entries can effectively be combined into one interface-specific forwarding table per interface, as shown in Table 2.4 for router A (the first row shows the next hops for packets originating at A). Note that most entries are usual next hops along the shortest paths to the destinations. But for the interface B→A, the next hops for destinations B, E and F are different from the usual forwarding entries. So, a packet to F is forwarded to the usual next hop B if it originates at A (using A→A entry) or if it arrives at A from C (using C→A entry) or D (using D→A entry). On the other hand, if the packet to F arrives at A from B, it is forwarded to D (using the B→A entry for F). Also, when an A–B link is down, packets to B, E, and F that were to be forwarded to B are rerouted to

back hops C, C, D respectively (using B→A entries), *without involving the control plane*. It is also important to emphasize that these entries *are computed apriori and not when necessary*. Moreover, since routers nowadays maintain a copy of the FIB at each interface to perform forwarding at line speed, interface-specific forwarding can be implemented *without any changes to the forwarding plane*.

Table 2.2: Interface-specific forwarding entries at router A from the topology in Fig. 2.1 (A→A is meant for packets originating at A).

Interface	Destination				
	B	C	D	E	F
A→A	B	C	D	B	B
B→A	C	C	D	C	D
C→A	B	B	D	B	B
D→A	B	C	B	B	B

FIFR for Node Failures

Thus far, we have described the FIFR approach for handling link failures. It has been shown that FIFR can route around node failures by inferring key nodes, instead of key links [63]. Here, we illustrate FIFR for dealing with node failures using a simple example. Consider the topology shown in Fig. 2.2, where each link is labeled with its cost. Now suppose node E in Fig. 2 failed. Imagine forwarding a packet from source A to destination F. Based on the link costs, the shortest path from A to F is A→B→E→F. So, router A will forward the packet to its next hop B. Router B, being adjacent to the failed node E, initiates local rerouting. Since the shortest path to destination F, without the router E, is B→A→C→E→F, it forwards the packet back to A. Normally, this would cause the packet destined for F to go back and forth between routers A and B, resulting in a forwarding loop.

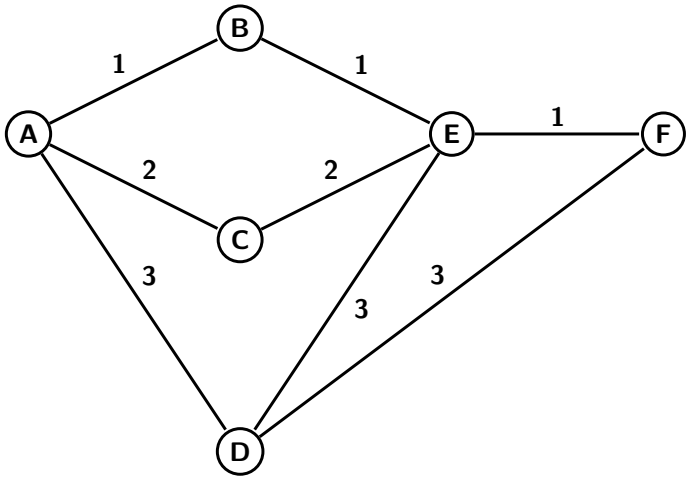


Figure 2.2: A topology used for illustration of FIFR for Node Failures.

Using FIFR for dealing with node failures, however, A infers potential failures of nodes along the shortest path to F that would cause the packet to arrive at A from its usual next hop B (i.e., through the *unusual* incoming interface B→A). It is apparent that the failure of node E would cause the packet for F to arrive through interface B→A.

The computation of forwarding table entries of an interface involves identifying a set of key nodes $K^d_{j\to i}$, whose failure causes a packet to arrive at the node through that interface. In this example, the key node for destination F and interface B→A is node E. Router A can pre-compute a key node per destination for each of its interfaces (which may be none), and compute the corresponding back hops. Table 2.3 lists the key nodes and back hops for all combinations of unusual incoming interfaces and destinations in Fig. 2.2.

The forward hop and back hop entries can effectively be combined into one interface-specific forwarding table per interface, as shown in Table 2.4 for router A (the first row shows the next hops for packets originating at A). Note that most entries are usual next hops along the shortest paths to the destinations. But for the interface B→A, the next hops for destinations B, E and F are different from the usual forwarding entries. So, a packet to F is forwarded to the usual next hop B if it

Table 2.3: Key Nodes and back hops for unusual incoming interfaces for the topology in Fig. 2.2

Interface	Destination	Key Node	Back Hop
B→A	F	E	D
A→B	C	C	E
A→B	D	D	E
E→B	C	C	A
E→B	F	−−	A
A→C	B	B	E
E→C	B	B	A
F→D	E	E	A
B→E	A	A	C
F→E	D	D	B

originates at A (using A→A entry) or if it arrives at A from C (using C→A entry) or D (using D→A entry). On the other hand, if the packet to F arrives at A from B, it is forwarded to D (using the B→A entry for F). Also, when an A–B link is down, packets to B, E, and F that were to be forwarded to B are rerouted to back hops C, C, D respectively (using B→A entries), *without involving the control plane.*

Table 2.4: Interface-specific forwarding entries at router A from the topology in Fig. 2.2 (A→A is meant for packets originating at A).

Interface	Destination				
	B	C	D	E	F
A→A	B	C	D	B	B
B→A	C	C	D	C	D
C→A	B	B	D	B	B
D→A	B	C	B	B	B

It has been shown that FIFR, by employing interface-specific forwarding, can guarantee loop-free forwarding to all reachable destinations in case of a single link or router failure [63, 45]. However, because the packets are rerouted along alternate paths only from the point of failure, the resulting paths are sub-optimal. For instance, in the above example, when link B−E is down, a packet from A to F traverses the path A→B→A→D→F, compared to the optimal path A→C→E→F. Therefore, it is of interest to initiate a link state update in case of a long-lasting failure, even while performing fast reroute, so that all routers can forward along optimal routes.

2.2 Convergence with Progressive Link State Updates

An obvious approach to deal with any link state changes is to propagate them network-wide so that all routers can recompute their next hops and forward packets along the optimal routes. The problem is that, during the convergence period, i.e., between the time a link state update is initiated and the time all routers install new forwarding entries, packets may be forwarded by some routers based on new state and others based on old state, potentially causing forwarding loops.

Suppose the link B−E in Fig. 2.1 is being shut down for maintenance by changing its cost from 1 to ∞, and all routers are notified of this change. Due to the delays in propagation of the link state advertisements and re-computation/installation of new forwarding entries, it is possible that B is in the *ac* (after change) era, i.e., starts forwarding according to the new cost, whereas A is still in the *bc* (before change) era, i.e., continues forwarding based on the old cost. Then, as per the new shortest path B→A→C→E, a packet destined for E from B is forwarded to A. Being in the *bc* era, router A, as per the old cost, forwards it back to B, resulting in a loop.

To prevent such forwarding loops during convergence, [19] proposed a scheme that imposes a certain order among the FIB updates of different routers. In this example, B installs new FIB entries only after A has done the same (and the link B−E is shut

down only after its adjacent nodes B and E update their FIBs). Then, a packet from B is forwarded to E, either along the old path B→E before the update or the new path B→A→C→E after the update, avoiding loops at any time instant. Enforcing such an order requires some form of coordination between routers. As we are interested in an approach that does not require any such coordination, we consider the progressive link metric increments approach proposed in [20].

The main idea behind the progressive link metric increments is as follows. Instead of sending an update with B−E cost of ∞, suppose we send a sequence of updates with a cost of 2, 3, and so on, each with a progressively higher cost, until B−E is not along any shortest path. Note that a subsequent update is sent only after the network has converged to the previous update. Consider the first update in this sequence, where the cost of B−E is increased from 1 to 2. With the old cost of 1 or the new cost of 2, the shortest paths from A to E and B to E remain the same, and hence this update would not cause a forwarding loop, even if A and B are in different eras.

When B−E cost is updated to 3, A→C→E becomes a shortest path with the same cost as that of A→B→E, while the shortest path from B to E remains B→E. Again, regardless of the eras A and B are in, this update does not cause a forwarding loop. Now, by increasing B−E cost to 4, A→B→E ceases to be a shortest path. The following update of cost to 5 makes A one of the next hops from B to E. Since B is no longer a next hop from A to E after the previous update, this update does not cause any back and forth between A and B. Next, updating the B−E cost to 6 or higher, effectively eliminates the link from the topology, since it is not along the shortest path between its adjacent nodes. We can then send a final update with B−E cost of ∞, to inform all routers that the link is actually down. Thus, each update yields loop-free convergence and progressively brings the network to the desired target topology without B−E, allowing the B−E link to be shut down gracefully.

It is not necessary that the link cost has to be incremented in steps of 1 to avoid loops. It has been shown that loop-freedom can be assured with a shorter metric sequence [20], which is specific to each link and depends on the topology. Table 2.5 shows the sequence of metric updates needed to bring down a link in the topology in Fig. 2.1 gracefully.

Table 2.5: Metric sequence required to bring each link down, for the topology in Fig. 2.1

Link	Without FIFR	With FIFR
A–B	2, 4, ∞	∞
A–C	3, ∞	∞
A–D	5, ∞	∞
B–E	4, ∞	∞
C–E	3, ∞	∞
D–F	5, ∞	∞
F–E	3, 5, 7, ∞	∞

The downside of the above incremental update method is that it takes longer to converge to the target state. In this dissertation, we explore how we can achieve fast loop-free convergence while performing fast reroute with FIFR, without necessitating coordination between routers.

Chapter 3

Handling Contingencies in Wired Networks

This chapter presents how we can build upon the interface-specific forwarding employed by the FIFR approach described in the previous section to ensure disruption-free forwarding before, during, and after network convergence in response to link state changes. We show that the proposed approach, referred to as FIFR++, four key features that make it attractive for deployment. 1) It guarantees loop-free fast rerouting around a failure to any reachable destination. 2) It provides loop-free fast convergence to optimal forwarding after a failure. 3) It does not require any changes to the IP datagram. 4) It does not require any coordination between routers. In the following, we describe the FIFR++ approach in stages and show that it can handle both link down and up events. Then, we present how it can deal with node failures and also mitigate packet loss due to network congestion.

3.1 Network Convergence during Link State Down Event[1]

We now explain how we incorporate the above ideas of FIFR and incremental updates towards developing the FIFR++ approach for providing loop-free fast reroute/convergence. First, we observe that even with traditional non-incremental link state updates FIFR mitigates the problem of forwarding loops during convergence. Next, we find that most of the loops can be avoided by installing new interface-specific backwarding

[1]Phani Krishna Penumarthi, Aaron Pecora, Jason O'Kane, Srihari Nelakuditi, Failure-Inference-Based Fast Reroute with Progressive Link Metric Increments, in Proceedings of IEEE ICCCN 2018.

entries only after interface-independent forwarding entries are updated at all routers. Finally, we show that only in some rare cases we need to employ progressive link state updates to ensure loop-free convergence and disruption-free forwarding.

3.1.1 Traditional Link State Updates

We have already seen in Section 2.2 that, without FIFR, a straightforward updating of B−E's cost from 1 to ∞ causes transient forwarding loops during convergence. FIFR, however, helps mitigate this problem. Again, consider the scenario of A forwarding a packet destined for F, when A is in the *bc* era and B is in the *ac* era. A, being unaware that B−E link is down, forwards that packet to its usual next hop B. Since B is in the *ac* era, it will forward to its new next hop A. Without FIFR, this makes the packet to loop between A and B. With FIFR, on the other hand, when B forwards a packet destined for F to A, instead of forwarding it back to B, node A will forward that to D, using interface-specific forwarding. D, no matter what era it is in, will in turn forward the packet to F. Thus the packet from A to F takes the path A→B→A→D→F.

Now suppose A is in the *ac* era. Then its forwarding and backwarding table entries will be updated as shown in Table 3.1. The new next hop from A to F is C. So, A will forward the packet destined for F to C. Since B-E link is not along the shortest path from C to F, regardless of the era C is in, it will forward the packet to E, which in turn will forward to F. Thus, when A is in the *ac* era, packets destined for F take the path A→C→E→F. Hence, packets from A to F are delivered along the path A→B→A→D→F or A→C→E→F without causing any forwarding loop irrespective of the order in which A and B are updated. For the topology in Fig. 2.1, we find that updating about any link failure, not just B−E, does not cause loops during convergence with FIFR, even with traditional link state updates, regardless of the order of routers updating their FIBs.

Table 3.1: Interface-specific forwarding entries at router A in Fig. 2.1 when it is updated (in *ac* era) with the failure of B−E. Note that the set of back hops for rerouting to B is empty in case the link A−B also goes down later making B unreachable.

Interface	Destination				
	B	C	D	E	F
A→A	B	C	D	C	C
B→A	–	C	D	C	C
C→A	B	D	D	D	D
D→A	B	C	C	C	C

3.1.2 Deferred Updating of Back Hops

The above observation does not hold across all topologies. Consider the topology in Fig. 3.1, where the link U−W is down and a link state update is triggered with cost ∞. Assume that router T enters the *ac* era, as it has updated its forwarding and backwarding entries, whereas the rest of the routers are still in the *bc* era. Suppose P has a packet for destination W. Table 3.2 shows the next hop and back hop for destination W at each router in the *bc* era. As per the *bc* topology, the shortest path from P to W is P→R→T→U→W. So P forwards it to R. Router R, which is also in the *bc* era, forwards it to T.

According to T, which is in the *ac* era, the new shortest path, without U−W link, is T→R→S→V→X→W. Therefore, its new next hop to W is R. Accordingly, the new back hop for packets destined for W arriving at T from R is V. This is because, T determines, based on the *ac* topology without U−W link, that S−V is the keylink, whose failure would cause a packet for W to arrive, along the (new) reverse shortest path, at T from R. Consequently, T forwards the packet to its new back hop V (Table 3.3 shows the next hops and back hops for destination W at each router according to the *ac* topology). But, router V, which is still in the *bc* era, forwards it

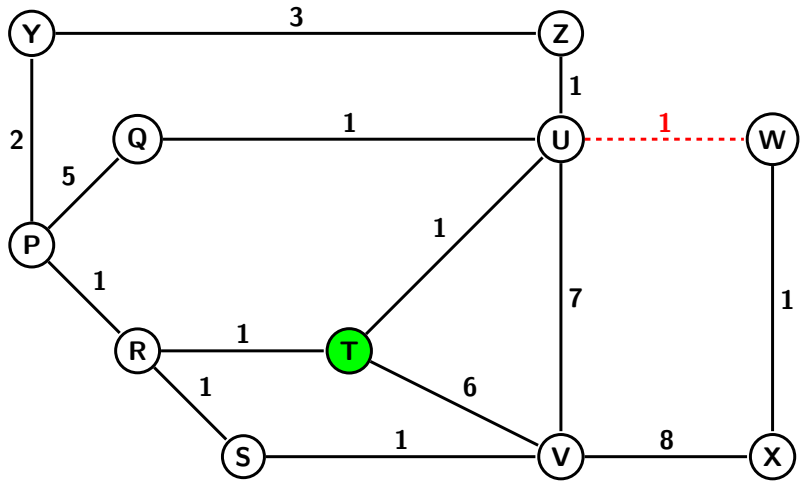


Figure 3.1: An illustrative scenario for pointing out the need for deferring updating of back hops during convergence with FIFR. Link U−W is down and not all routers have updated their tables. Only router ***T*** is in the *ac* era. Its new next hop and back hop to W are R and V respectively. With the new back hop, packets from P to W traverse a loop P→R→T→V→S→R→T. With deferred updating of back hops, they take loop-free path P→R→T→R→S→V→X→W.

to its usual next hop S. Being in the *bc* era, S forwards it to R. Again, R forwards it to T. Thus, the packet traverses the path P→R→T→V→S→R→T, forming a loop.

The reason for the above looping scenario is that T, which is in *ac* era, misinterprets the usual forwarding from R, which is in *bc* era, as unusual back hop due to another failure along the new shortest path T→R→S→V→X→W, which obviously is not the case. This does not happen if T does not treat R→T as a back hop interface. Therefore, *we propose to eliminate the possibility of such loops during convergence by deferring the updating of back hops.* In other words, *a router in the* ac *era uses* ac *next hop but* bc *back hop during convergence.*

Table 3.4 shows this combination of *ac* next hops and *bc* back hops for destination W. In this table, T has updated its next hop from U to R based on the *ac* topology, but keeps the back hop as R for packets arriving at T from U (instead of changing it to V as per Table 3.3). The new *ac* back hops as in Table 3.3 are installed once the network converges to the *ac* era. Note that deferred updating of back hops does not

Table 3.2: Next hops and back hops to destination W in *bc* era (with link U−W). For instance, a packet destined for W is forwarded by node R to the usual next hop T if it arrives from any node other than T. But if a packet to W arrives at R from T, it is forwarded to the back hop S (avoiding the key link U−W).

Node	Next Hop	Back Interface	Back Hop
P	R	R→P	Q, Y
Q	U	U→Q	P
R	T	T→R	S
S	R	R→S	V
T	U	U→T	R
U	W	W→U	T
V	S	S→V	X
X	W	W→X	V
Y	Z	Z→Y	P
Z	U	U→Z	Y

delay the network's convergence to optimal routing, since back hops are needed only for protection against a subsequent failure.

Now, let us revisit the above scenario of a packet being forwarded from P to W. As before, the packet will go from P to T along the path P→R→T. Then, with the deferred updating of back hops, T will forward it based on the *bc* back hop to R. Thereafter, the packet will reach V from R, along the reverse shortest path R→S→V. Therefore, V will forward it to its back hop X. Thus, the packet reaches its destination, along the loop-free path P→R→T→R→S→V→X→W.

3.1.3 Progressive Link Metric Increments

While the above approach works fine in most cases, it is possible to handcraft a topology where a particular scenario of link state updates can cause looping. Consider

Table 3.3: Next hops and back hops to destination W in *ac* era (without link U–W). The next hop from T to W is R and the back hop for packets to W arriving at T from R is V.

Node	Next Hop	Back Interface	Back Hop
P	R	R→P	Q, Y
Q	U	U→Q	P
R	S	S→R	T
S	V	V→S	R
T	R	R→T	V
U	T	T→U	V
V	X	X→V	–
X	W	W→X	–
Y	P	P→Y	Z
Z	U	U→Z	Y

the topology shown in Fig. 3.2, where the link I–M is down and a link state update is triggered with cost ∞. Suppose J has a packet for M. Assume that I, J, K, and L have recomputed their forwarding tables and are in the *ac* era, whereas the rest are in the *bc* era.

Table 3.5 shows the next hop and back hop at each router for destination M. Only the next hop entries at J, K, and L are based on the *ac* topology. As per the *ac* topology, the shortest path is J→L→N→M, and so J forwards it to L. L, which is also in the *ac* era, forwards it to N (here the deferred *bc* back hop and *ac* next hop happens to be the same). According to N, which is in the *bc* era, the usual shortest path is N→L→J→H→I→M. Based on its inference on *bc* view, only the failure of J–L can cause a packet for M to arrive through L→N interface along the reverse shortest path. Therefore, since the backward path from N to M without the key link J→L is N→O→K→I→M, the packet is forwarded to O, which in turn forwards to K.

Table 3.4: Next hops and back hops to destination W with deferred updating of back hops. T, which is in *ac* era, updates its next hop but uses *bc* back hop. Note that back interface for T is U→T instead of R→T and its back hop is R, which happens to be the same as its next hop. All other nodes are in the *bc* era and hence use *bc* next hops and back hops.

Node	Next Hop	Back Interface	Back Hop
P	R	R→P	Q, Y
Q	U	U→Q	P
R	T	T→R	S
S	R	R→S	V
T	R	U→T	R
U	W	W→U	T
V	S	S→V	X
X	W	W→X	V
Y	Z	Z→Y	P
Z	U	U→Z	Y

Since K is in the *ac* era, its shortest path to M, without I–M, is K→J→L→N→M. Hence, K forwards to J, which again forwards to L, causing a loop.

We propose to employ progressive updates for such links. The looping scenario discussed above does not occur with this approach, since the cost of I–M is increased gradually with a sequence of updates. One such metric sequence is 10, 14, 19, ∞. Imagine the first update, where the cost of I–M is set to 10. With this cost, regardless of the eras each router is in, packet from J to M is forwarded to I, which then reroutes it to H, along the path I→H→G→M. H infers the failure of I–M, as the packet to M arrives through unusual interface I→H, and forwards it to G. G does the same and forwards it to M. Thus, with each update step, one or more source-destination pairs' shortest paths are made equal to that passing through I–M or turned away from I–M,

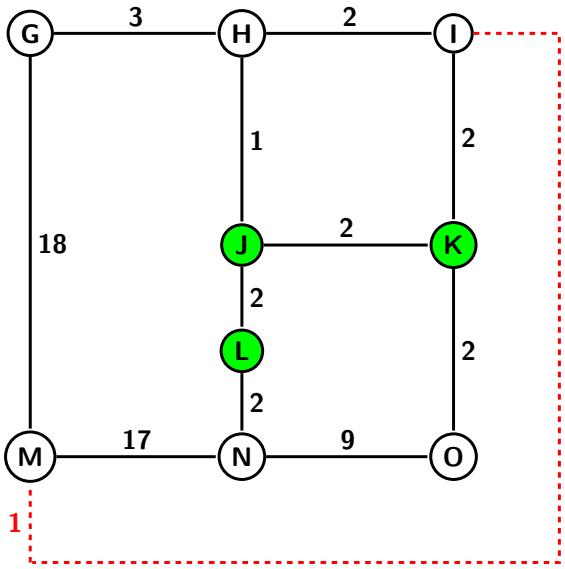


Figure 3.2: An illustration of transient forward loops during convergence with FIFR. Suppose the link I–M is down and only J, K, and L are in the *ac* era and the rest are in the *bc* era. Even with deferred updating of back hops, packets from J to M loop along the path J→L→N→O→K→J→L.

while packets arriving at I are rerouted to the destination along a loop-free path with the help of FIFR. At the end of progressive updates, each router can independently, without any need for synchronization, install its new back hops corresponding to the topology without the failed link, in preparation for a potential subsequent failure.

We refer to *this combination of failure inference based fast reroute before and during convergence, deferred updating of back hops, and progressive link metric increments only when necessary as FIFR++*. Operations under FIFR++ can be summarized as in Algorithm 1 with notation in Table 3.6. A key aspect of FIFR++ is that while the forwarding entries at a router i, $\mathcal{F}_i$, are recomputed upon receiving each progressive link state update, its backwarding entries corresponding to each neighbor j, $\mathcal{B}_{j \to i}$ are recomputed only when the link is finally shut down. The advantage of FIFR++ is that it guarantees loop-free forwarding both before and during convergence in case of single link failures. We prove it in the next section.

Table 3.5: Next hops and back hops to destination M. Next hops at J, K, and L are based on the *ac* topology, whereas the rest of the entries including all back hops are based on the *bc* topology.

Node	Next Hop	Back Interface	Back Hop
G	H	H→G	M
H	I	I→H	G
I	M	M→I	H
J	L	H→J	K
K	J	I→K	J
L	N	J→L	N
N	L	L→N	O
O	K	K→O	N

Table 3.6: Notation

$\mathcal{F}_i^d$	set of next hops from i to d
$\mathcal{B}_{j \rightarrow i}^d$	set of back hops from $j \rightarrow i$ to d
$\mathcal{K}_{j \rightarrow i}^d$	key link corresponding to $j \rightarrow i$ to d
$\mathcal{C}_{u-v}$	cost of the link $u-v$
$\mathcal{T}_i$	shortest path tree (SPT) rooted at i
$\mathcal{P}(\mathcal{T}, d)$	shortest path from root of $\mathcal{T}$ to d
$\mathcal{D}(\mathcal{T}, d)$	shortest distance from root of $\mathcal{T}$ to d

3.1.4 Performance Evaluation

We have validated the proposed FIFR++ [49] approach using 6 Rocketfuel topologies [47] (details of which are listed in Table 3.7) and 262 topologies in the Internet Topology Zoo collection [32]. We have also generated 12 random topologies using BRITE [40], varying the number of nodes from 25 to 150, each with average degrees of 4 and 6. All together our evaluation set includes 280 topologies with a total of

Algorithm 1 : Operations at router i under FIFR++

if originates a packet p to destination d
  Forward p to $\mathcal{F}_i^d$
end if

if receives a packet p to d from neighbor j
  if $j \rightarrow i$ is a back interface
    Forward p to $\mathcal{B}_{j \rightarrow i}^d$
  else
    Forward p to $\mathcal{F}_i^d$
  end if
end if

if detects a failure of an adjacent link to v
  Reroute packets to d with next hop v to $\mathcal{B}_{v \rightarrow i}^d$
  if Link i–v needs metric increments
    Send LSAs to progressively update $C_{i \rightarrow v}$ to ∞
  else
    Send LSA to update $C_{i \rightarrow v}$ to ∞
  end if
  Recompute $\mathcal{F}_i$ without i–v
  Recompute $\mathcal{B}_{j \rightarrow i}$ without i–v for each neighbor j
end if

if receives an LSA with new cost c for link u–v
  $C_{u \rightarrow v} \leftarrow c$
  Recompute $\mathcal{F}_i$ with new cost of u–v
  if $c = \infty$
    Recompute $\mathcal{B}_{j \rightarrow i}$ without u–v for each neighbor j
  end if
end if

17339 bidirectional links. We have developed a customized simulator that brings down one link at a time in the given topology and verifies if a packet from each *affected* (whose shortest path passes through that link) pair of source and destination nodes gets caught in a loop, considering all the possible combinations where each node can be in either bc or ac state.

Figure 3.3 compares how different mechanisms fare in avoiding transient forwarding loops during convergence. They are labeled NoFIFR (traditional link state up-

Table 3.7: Summary of Rocketfuel topologies

AS Number	Name	Nodes	Edges
1221	Telstra	108	306
1239	Sprint	315	1944
1755	Ebone	87	322
3257	Tiscali	161	656
3967	Exodus	79	294
6461	Abovenet	141	748

dates without FIFR), FIFR (traditional link state updates with FIFR), FIFR+ (FIFR with deferred updating of back hops), and FIFR++ (FIFR+ with progressive metric increments when necessary). The results show that without FIFR around 35% of links in Rocketfuel topologies can cause loops during convergence. By employing FIFR alone even with traditional link state updates, this fraction can be brought down drastically to around 2%. FIFR when coupled with deferred updating of back hops completely eliminates the looping problem, obviating the need for progressive metric increments, in Rocketfuel topologies.

Figure 3.3 also presents our evaluation results for Zoo and Random topologies. It shows that in these topologies up to nearly 50% of the links when failed can cause transient loops during convergence. FIFR by itself helps avoid most of these loops, reducing the fraction of loopy links to below 3% in Random and below 0.5% in Zoo topologies. Deferred updating of backhops along with FIFR nearly eliminates the looping problem — only 2 links in Zoo topologies, labeled Missouri and Oteglobe, and 1 link in Random topologies cause loops.

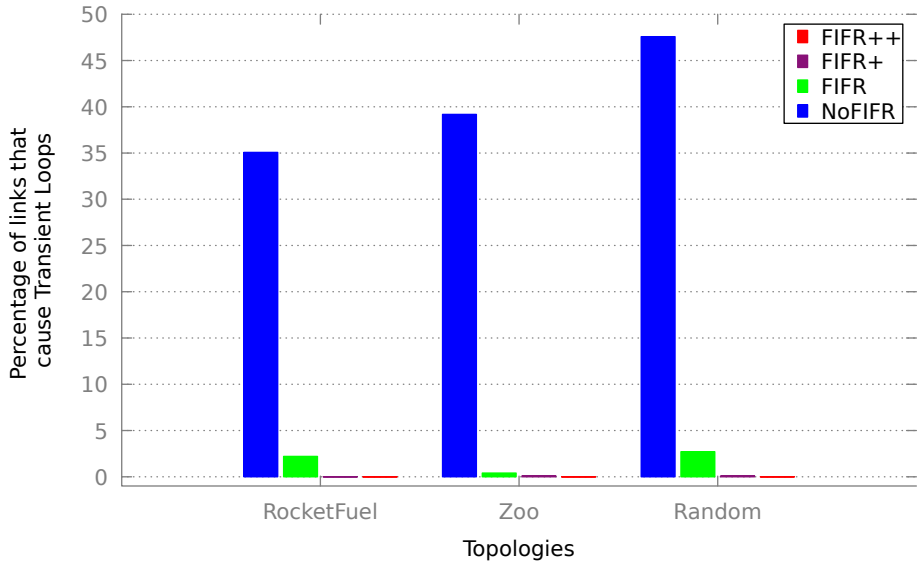


Figure 3.3: Percentage of links that cause transient loops during convergence.

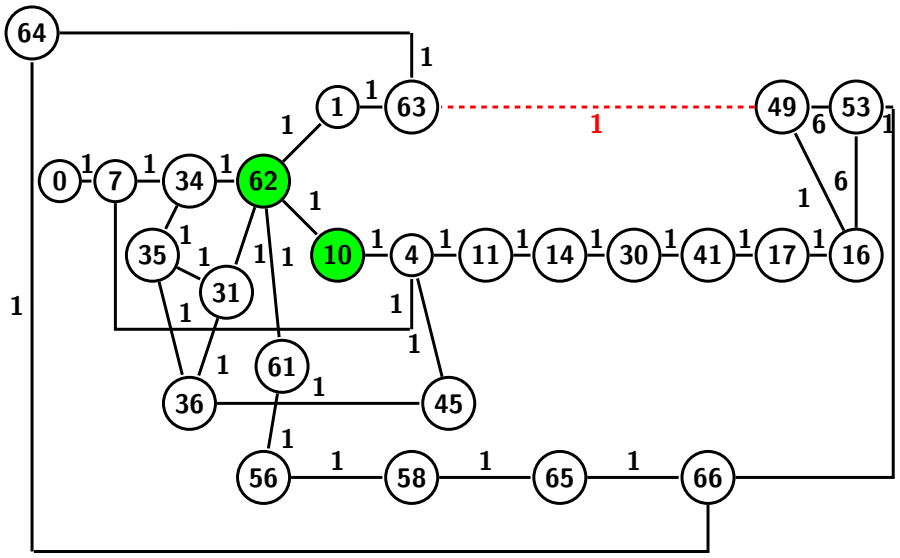


Figure 3.4: A part of the Missouri topology. Failure of link $49 - -63$ can cause transient forwarding loop between 7 to 49 during convergence when only 10 and 62 are in the *ac* era.

Missouri Topology

Figure 3.4 shows a part of the Missouri topology with a link which when goes down can cause a forwarding loop during convergence, even with deferred updating of back hops. Suppose the link $63 - -49$ is down and only the routers 10 and 62 are in the *ac* era with updated FIBs. A quick observation at Tables 3.8, 3.9 reveal that packets

Table 3.8: Nexthops for a few routers in the Missouri topology, in *bc* and *ac* eras.

Router	0	7	34	62	1	63	10	4	11	14	30	41	17	16
NH in *bc*	7	34	62	1	63	49	62	7,11	14	30	41	17	16	49
NH in *ac*	7	4	62	34,10	62	1	4	11	14	30	41	17	16	49

Table 3.9: Keylinks for a few edges in the Missouri topology.

Interface	$7 \rightarrow 4$	$1 \rightarrow 62$	$62 \rightarrow 10$	$62 \rightarrow 34$
Observed Keylinks	NONE	$63 \rightarrow 49$	$63 \rightarrow 49$	NONE

from 0 to 49 should follow the path $0 \rightarrow 7 \rightarrow 34 \rightarrow 62 \rightarrow 1 \rightarrow 63 \rightarrow 49$ in *bc* era, and $0 \rightarrow 7 \rightarrow 34 \rightarrow 62 \rightarrow 10 \rightarrow 4 \rightarrow 11 \rightarrow 14 \rightarrow 30 \rightarrow 41 \rightarrow 17 \rightarrow 16 \rightarrow 49$ in the *ac* era (where cost of edge $63 \rightarrow 49$ is ∞). When Routers $\{62, 10\}$ are in updated *ac* era, packets from 0 destined to 49 will observe loops along the path $0 \rightarrow 7 \rightarrow 34 \rightarrow 62 \rightarrow 10 \rightarrow 7 \rightarrow 34 \rightarrow 62 \rightarrow 10 \rightarrow 7 \rightarrow 34$.

Oteglobe Topology

Table 3.10: Nexthops for a few routers in the Oteglobe topology, in *bc* and *ac* eras.

Router	2	73	74	80	81	82	75	76	90	87	83	3	5	79	4	17
NH in *bc*	73	74	80	81, 82	75	76	90	90	87	83	-	17	16	80	79	4
NH in *ac*	73	74	82	79	80	80	81	82	75,76	90	-	17	16	4	17, 49	15

Table 3.11: Keylinks for a few edges in the Oteglobe topology.

Interface	$80 \rightarrow 79$	$79 \rightarrow 4$	$90 \rightarrow 76$	$90 \rightarrow 75$	$4 \rightarrow 5$
Observed Keylinks	NONE	$80 \rightarrow 79$	NONE	NONE	$79 \rightarrow 80$

Similarly, in Oteglobe topology (Figure 3.5 shows the relevant part) the shortest path between 2 and 83 is 2→73→74→80→81→75→90→87→83. Next hops for rel-

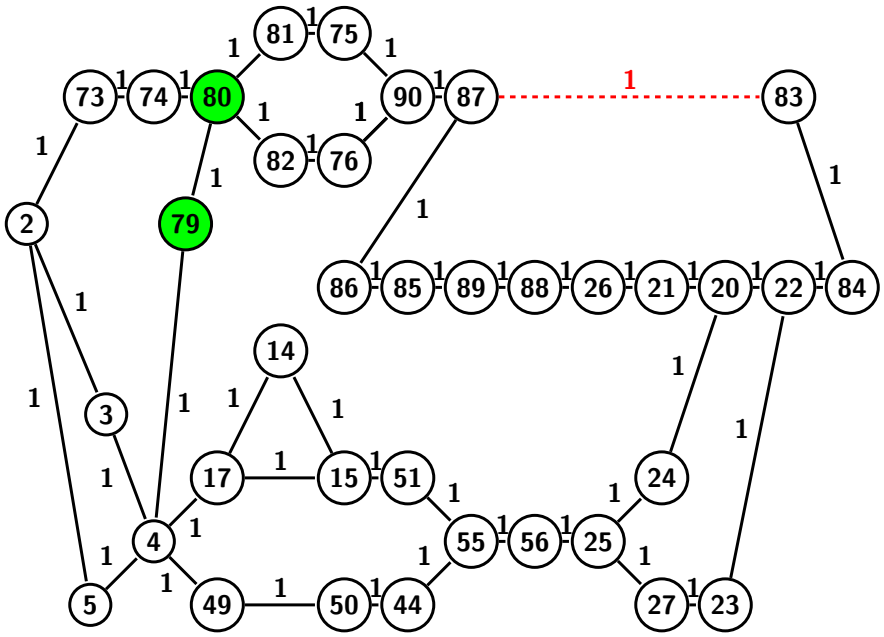


Figure 3.5: A part of the Oteglobe topology. Failure of link $87 - -83$ can cause transient forwarding loop between 2 to 83 during convergence when only 79 and 80 are in the *ac* era.

evant nodes is represented in Table 3.10, while the keylinks for some of the links is represented in Table 3.11. A quick observation at Tables 3.10, 3.11 reveal that packets from 2 to 83 should follow the path $2 \rightarrow 73 \rightarrow 74 \rightarrow 80 \rightarrow 81 \rightarrow 75 \rightarrow 90 \rightarrow 87 \rightarrow 83$ in *bc* era, and $2 \rightarrow 5 \rightarrow 4 \rightarrow 17 \rightarrow 15 \rightarrow 51 \rightarrow 55 \rightarrow 56 \rightarrow 25 \rightarrow 24 \rightarrow 20 \rightarrow 22 \rightarrow 84 \rightarrow 83$ in the *ac* era (where cost of edge $83 \rightarrow 87$ is ∞). When Routers $\{80, 79\}$ are in updated *ac* era, packets from 2 destined to 83 will observe loops along the path $2 \rightarrow 73 \rightarrow 74 \rightarrow 80 \rightarrow 79 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 73....$

Random Topology form BRITE Topology Simulator

Table 3.12: Nexthops for a few routers in a random topology, in *bc* and *ac* eras.

Router	1	4	5	6	7	2	3	6
NH in *bc* era	4	5	6	7	9	3	④	9
NH in *ac* era	2	③	4	5	8	6	2	9

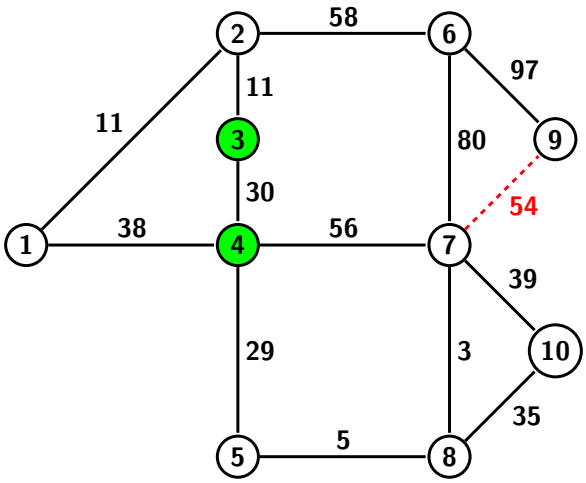


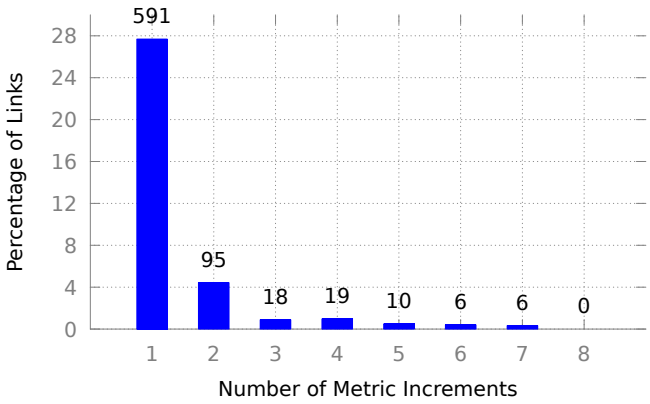
Figure 3.6: A part of the Random topology. Failure of link 7–9 can cause loop transient loop between 1 and 9 when only 3 and 4 are in the *ac* era.

Table 3.13: Keylinks for a few edges in a Random topology.

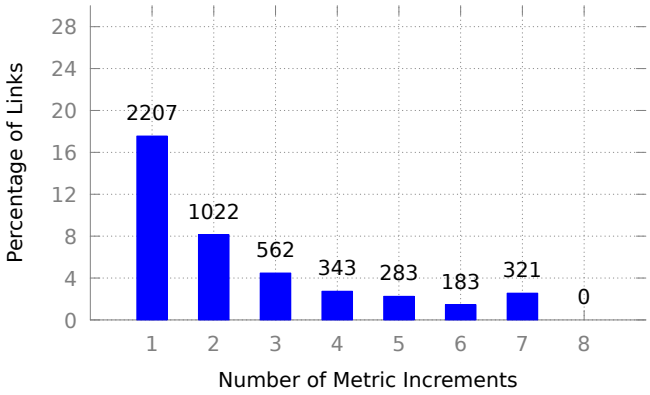
Interface	$3 \rightarrow 2$	$4 \rightarrow 3$	$5 \rightarrow 4$	$8 \rightarrow 5$	$7 \rightarrow 8$	$2 \rightarrow 1$
Observed Keylinks	$3 \rightarrow 4$	NONE	$5 \rightarrow 8$	NONE	NONE	$3 \rightarrow 4$

Finally, in a Random topology we generated (a subgraph of which is shown in Figure 3.6) the link 7–9 when down causes packets from 1 to 9, that were traversing 1→4→5→8→7→9, to loop along 1→4→3→2→1→4 when only nodes 3 and 4 are in *ac* era. Next hops for relevant nodes is represented in Table 3.12, while the keylinks for some of the links is represented in Table 3.13. A quick observation at Tables 3.12, 3.13 reveal that packets from 1 to 9 should follow the path $1 \rightarrow 4 \rightarrow 5 \rightarrow 7 \rightarrow 8 \rightarrow 9$ in *bc* era, and $1 \rightarrow 2 \rightarrow 6 \rightarrow 9$ in the *ac* era (where cost of edge $7 \rightarrow 9$ is ∞). When Routers $\{3, 4\}$ only are in the updated *ac* era, packets from 1 destined to 9 will observe loops along the path $1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 4 \rightarrow 3 \rightarrow 2$.

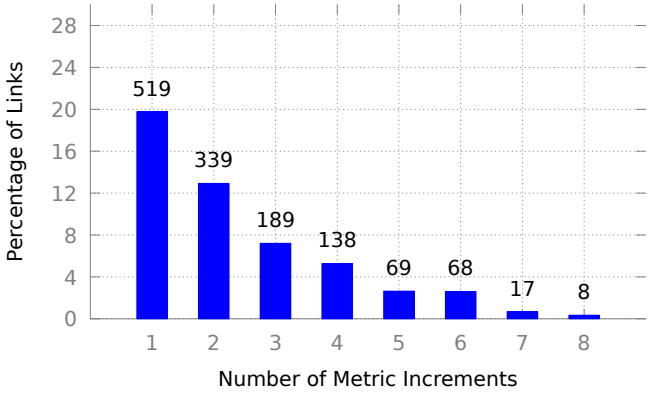
Except in these 3 specific instances, where a particular combination of nodes are in the *ac* era while all others are in the *bc* era, deferred updating of back hops do not cause forwarding loops in any other scenarios. Even in those 3 rare cases out of the total 17339 links in 280 topologies we have considered, by employing



(a) Rocketfuel Topologies



(b) Internet Zoo Topologies



(c) Random Topologies

Figure 3.7: Percentage of links with different lengths of progressive metric increment sequences (each bar is labeled with the absolute number of links).

progressive metric increments, FIFR++ ensures loop-free convergence and disruption-free forwarding.

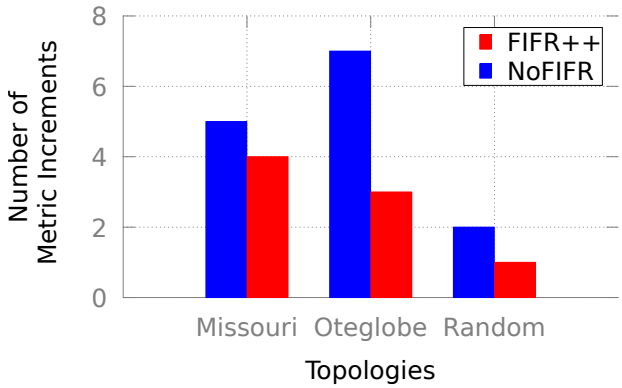


Figure 3.8: Comparison of metric sequences without FIFR and with FIFR++.

Apart from avoiding loops during convergence, FIFR++ approach also helps accelerate convergence after a failure for the following reasons: i) Except in very rare in cases, FIFR++ can directly announce the link cost as ∞ instead of progressively incrementing the cost to ∞. ii) Each router can independently update its FIB without having to wait for signaling from other routers to enforce a particular order. To illustrate this, in Figure 3.7 we plot the percentage of links in each category of topologies that need a specific progressive metric sequence length to ensure loop-free convergence. Note that the sequence length indicates the number of incremental changes needed before the link cost is set to ∞. So the sequence length is considered to be 0 if the link cost can be updated directly to ∞. It is evident from Figure 3.7 that without FIFR, many links require multiple rounds of progressive updates before the network is converged to a state without the failed link. The progressive metric sequence length for some of the links could up to 8, causing a significant delay in the network convergence.

With FIFR++, on the other hand, only 3 links out of all the topologies need progressive updates. Furthermore, even for those links, the length of incremental metric sequence with FIFR++ is less than that without FIFR, as shown in Figure 3.8. Overall, these results demonstrate that order of FIB updates rarely matters with FIFR and thus FIFR++ can not only achieve fast route but also fast convergence.

3.1.5 Related Work

An approach most related to this work uses the combination of NotVia [8] and interface specific forwarding [55] for loop-free convergence and fast reroute. The relative merit of FIFR++, unlike NotVia which relies on encapsulation, is that it provides the same service without any modification to the IP datagram.

Safeguard [35] aims to provide both fast route and transient loop prevention, by carrying remaining path length in all packets. Compared with such schemes, FIFR++ provides failure resilience without additional information in the packet header.

Along the lines of progressive updates, [10] and [11] provide efficient algorithms for determining metric sequences for shutting down a link or a router. Similar to [20], they are, however, meant for planned maintenance. FIFR++ can leverage them and handle both unplanned and planned failures gracefully.

An approach for mitigating transient forwarding loops during convergence using interface-specific forwarding was proposed in [46]. But this method prevents loops by discarding packets that arrive through unusual interfaces. In contrast, FIFR++ ensures loop-free convergence without any packet loss.

Failure carrying packets [34] does away with the need for convergence by carrying the list of failed links in the packet. Packet Recycling [38] can deal with more than one failure by rerouting packets leveraging cellular graph embeddings, but needs multiple bits in the header. Both these require a significant modification to the packet header.

Software defined networking (SDN), with a centralized control plane, has been gaining in popularity [16]. Yet, there have been attempts to combine the advantages of SDN and traditional routing. In [62], Vissicchio et al propose a hybrid approach called Fibbing, which introduces fake nodes in the network to induce the desired forwarding tables. Since link failures will necessitate changes in the forwarding plane and fake nodes, we believe approaches like FIFR++ for providing failure protection and loop-free convergence are still relevant.

A recent work [39] shows resiliency to 4 simultaneous failures using interface-specific forwarding. We plan to study this approach and extend FIFR++ to provide loop-free forwarding and convergence even in the presence of multiple failures.

3.1.6 Proof of Loop-Free Convergence with FIFR++

Now, we sketch a proof that FIFR++ is loop-free, based on the following assumptions: i) There is only one failed link in the network that is protected with FIFR; ii) Only one link state update is being propagating throughout the network; iii) Each progressive update increments the failed link cost by 1;[2] iv) Links are bidirectional with symmetric costs.

Let u–v, with original cost of $C_{u\leftrightarrow v}$, be the failed link. Its cost is progressively updated and currently it is being changed from $\overleftarrow{C}_{u\leftrightarrow v}$ (cost in the *bc* era) to $\overrightarrow{C}_{u\leftrightarrow v}$ (cost in the *ac* era). For other symbols too, we use the overhead left arrow to refer to the *bc* era state and right arrow to refer to that in the *ac* era.

Suppose a packet is being forwarded from source s to destination d. Without loss of generality, let us assume that a packet from s to d, if it were to cross the link u–v, will pass in the direction $u\rightarrow v$. In the following, we show that, under FIFR++, this packet does not get caught in a loop, i.e., does not traverse the same link in the same direction more than once.

Property 1: A packet does not loop if $u\rightarrow v \notin \mathcal{P}(\overleftarrow{\mathcal{T}}_s, d)$

If $u\rightarrow v \notin \mathcal{P}(\overleftarrow{\mathcal{T}}_s, d)$, then $u\rightarrow v \notin \mathcal{P}(\overrightarrow{\mathcal{T}}_s, d)$, i.e., if the shortest path from s to d in the *bc* topology does not include $u\rightarrow v$, then same is the case in the *ac* topology too. This is true because the path through $u\rightarrow v$ gets costlier as the cost $C_{u\leftrightarrow v}$ is increased from *bc* to *ac*. Since this is the only change between the *bc* and *ac* topologies, $\mathcal{P}(\overleftarrow{\mathcal{T}}_s, d) = \mathcal{P}(\overrightarrow{\mathcal{T}}_s, d)$. Therefore, all routers along that path forward the packet consistently to d.

[2]Using the arguments analogous to that in [20], which shows that updating with optimized metric sequence is loop-free, it is possible to prove that similar sequence with larger metric increments also does not cause loops with FIFR.

Property 2: A packet does not loop if it is not rerouted by u.

As per FIFR++, a packet may be forwarded using back hops in $\mathcal{B}$ tables, only after it reaches u and gets rerouted. In all other cases, it is forwarded using usual next hops in $\mathcal{F}$ tables. Suppose that is not the case and a router i forwards the packet using the back hop entry $\mathcal{B}^d_{j\rightarrow i}$. For this to happen, i should be a next hop to d according to j and vice versa. Obviously this can not happen if i and j are in the same era.

Let us assume that router i is in the *ac* era and j in the *bc* era. Then, in j's view of the topology, $\mathcal{D}(\overleftarrow{\mathcal{T}}_j, d) > \mathcal{D}(\overleftarrow{\mathcal{T}}_i, d)$ and in i's view, $\mathcal{D}(\overrightarrow{\mathcal{T}}_i, d) > \mathcal{D}(\overrightarrow{\mathcal{T}}_j, d)$. This is not possible considering that $\mathcal{D}(\overrightarrow{\mathcal{T}}_j, d) = \mathcal{D}(\overleftarrow{\mathcal{T}}_j, d)$ or $\mathcal{D}(\overrightarrow{\mathcal{T}}_j, d) = \mathcal{D}(\overleftarrow{\mathcal{T}}_j, d) + 1$, and $\mathcal{D}(\overrightarrow{\mathcal{T}}_i, d) = \mathcal{D}(\overleftarrow{\mathcal{T}}_i, d)$ or $\mathcal{D}(\overrightarrow{\mathcal{T}}_i, d) = \mathcal{D}(\overleftarrow{\mathcal{T}}_i, d) + 1$. Similarly, when i is in the *bc* era and j in the *ac* era, no back hops are used to forward the packet.

In other words, when a packet is not rerouted by u, all routers along the path forward it using $\mathcal{F}$ tables only and FIFR plays no role in forwarding it. This scenario is already shown be loop-free due to progressive link metric increments [20].

Property 3: A packet does not loop if it is rerouted by u.

The path taken by a packet rerouted by u can be split into 3 segments: a forward segment to the point of failure $s \rightsquigarrow u$, a backward segment $u \rightsquigarrow x \rightarrow y$ to a *turning point* y (where it switches from backwarding to forwarding), and an additional forward segment $y \rightarrow z \rightsquigarrow d$ to destination. Note that s and x can be u itself and similarly z can be same as d. It follows from Property 2 that no looping occurs in the path $s \rightsquigarrow u$. Next, we prove that the other two segments are also loop-free.

From the properties of FIFR, it is known that both the segments are loop-free in the original topology (with original cost of $C_{u\text{–}v}$ for u–v). Since back hops $\mathcal{B}$ are based on the original topology, backwarding is done consistently by the routers following u. The only deviation is that some unusual incoming interfaces with associated back hops in the original topology may no longer be unusual incoming interfaces in the *bc* or

ac topology. Without loss of generality, let y be the turning point from backwarding segment to forwarding segment.

We show that y can *safely* switch from using backhops, $\mathcal{B}^d_{x \to y}$, to either $\overleftarrow{\mathcal{F}}^d_y$ or $\overrightarrow{\mathcal{F}}^d_y$, when x is no longer a next hop from y to d. First, consider the scenario when router y is in the *bc* era. Let z be the next hop from y to d in the *bc* topology (with cost $\overleftarrow{C}_{u \leftarrow v}$ greater than original cost of $C_{u \leftarrow v}$ for link $u-v$). Then, we can show that the packet from z to d would not arrive back at u, avoiding any potential for looping. For z to be a next hop from y in the *bc* topology, we must have $\mathcal{D}(\overleftarrow{\mathcal{T}}_x, d) + C_{x-y} > \mathcal{D}(\overleftarrow{\mathcal{T}}_z, d) + C_{y-z}$, whereas $\mathcal{D}(\mathcal{T}_x, d) + C_{x-y} < \mathcal{D}(\mathcal{T}_z, d) + C_{y-z}$ in the original topology. Considering that there is no change in the cost of any other link except $u-v$, this is possible only if $u \to v \notin \mathcal{P}(\overleftarrow{\mathcal{T}}_z, d)$, i.e., the shortest path from z to d does not pass through $u \to v$. Therefore, a packet gets forwarded from z to d without getting caught in a loop according to Property 1.

Now, suppose router y is in the *ac* era with z as its next hop to d, whereas x was the next hop in the *bc* topology. Since z or any other downstream routers may be in the *bc* era, the question is whether the packet from z to d reaches u and then gets rerouted to y, resulting in a forwarding loop. For that to happen, the backward path $u \rightsquigarrow x \to y$ must be shorter than $u \rightsquigarrow z \to y$, while the forward path $y \to z \rightsquigarrow d$ must be shorter than $y \to x \rightsquigarrow d$. This is an impossibility if $z \rightsquigarrow d$ were to pass through u, considering that link costs are symmetric.

Finally, if the failure of $u-v$ partitions the network and there is no path from s to d, then the packet is dropped, instead of getting rerouted, by u or v. Thus, FIFR++ can guarantee loop-free forwarding to reachable destinations during convergence.

3.1.7 Calculating FIFR++ Metric Sequence Efficiently

Our evaluation has shown that the number of links that need progressive metric increments with FIFR++ approach is quite low. In this section, we focus on how to determine such links efficiently.

Authors of [20] propose an algorithm to generate Route Metric sequences for every link in the network, which ensures loop-freedom for packets between any source-destination pairs. In this section, we update the original optimal route metric sequence generation algorithm, such that it is optimized for usage with FIFR++. The updated algorithm is presented in Algorithm 2. The algorithm to calculate metric sequences for interface independent forwarding is presented in Algorithm 3 and arguments about loop freedom for that algorithm can be referred in detail at Fig.3 of [20]. We update the function OptimizeRMS such that Route metric sequences are calculated only when FIFR does not handle packets incurring loops in the network.

The condition can be explained as follows: When a node in new era forwards the packet to a node in old era, and the node in old era forwards the packet using next hops, we claim that loops may occur in the network. One way to determine if FIFR is not applicable between two neighboring nodes in the network is if: one node forwards packets to neighboring node in old era, and the other node forwards packets in the updated era, and the keylinks observed from next-node to initial-node are empty. Mathematically speaking, the proposed condition can be represented as: $v \in \mathcal{F}_u^d$ & $u \in (\overrightarrow{\mathcal{F}_v^d})$ & $\mathcal{K}_{v \rightarrow u}^d = \phi \ \forall u \in \mathcal{N}, \forall v \in \mathcal{N}$. We list some of the properties and study the efficiency of proposed condition.

Property 4: If the proposed condition appears in the network, some packets can incur loops in the network.

Proof (by Contradiction): Assume none of the packets incur loops when this condition is satisfied. Consider a packet originated at node u destined for d. Assume u is in the before change (bc) era, v is in the updated (after change, ac era). u

Algorithm 2 Route Metric Sequences for link $A \rightarrow B$ using FIFR++

$\mathcal{N}, \mathcal{E}$: Number of Nodes, Edges in the Topology
$\mathcal{F}_u^d$: Forward hops from node u to destination d with link $A \rightarrow B$
$\overrightarrow{\mathcal{F}_u^d}$: Forward hops from node u to destination d without link $A \rightarrow B$
for destination $d \in \mathcal{N}$
  for $(u, v) \in \mathcal{E}$
    if $v \in \mathcal{F}_u^d$ **and** $u \in (\overrightarrow{\mathcal{F}_v^d})$ **and** $\mathcal{K}_{v \rightarrow u}^d = \phi$
      $M_{A \rightarrow B}^d = M_{A \rightarrow B}^d \bigcup$ getORMS(u, v, A, B, d)
      //getORMS defined in Algorithm 3.
    else
      $M_{A \rightarrow B}^d = \phi$
    end if
  end for
end for

$M_{A \rightarrow B} = \bigcup_{d=1}^{\mathcal{N}} M_{A \rightarrow B}^d$
$OptimizedMetric_{A \rightarrow B} = \phi$
$compareIndex = 0$
for $i \in |M_{A \rightarrow B}| - 1$
  When the metric changes from $M_{A \rightarrow B}^{compare}$ to $M_{A \rightarrow B}^{i+1}$
  for $(src, dest) \in (N, N)$
    if Packets from src Reaches Destination $dest$
      no loops // do nothing
    else
      compareIndex=$i + 1$
      $OptimizedMetric_{A \rightarrow B} = OptimizedMetric_{A \rightarrow B} \bigcup M_{A \rightarrow B}^{i+1}$
    end if
  end for
end for
return $OptimizedMetric_{A \rightarrow B}$

forwards the packet to v, v in its updated state forwards to u, $v \rightarrow u$ does not infer any link failure as $KL_{u \rightarrow v}^d = \phi$ and forwards the packet back to v. If $KL_{v \rightarrow u}^d \neq \phi$, v forwards the packet through backhops a again, and packets do not incur loops. Otherwise, i.e., $KL_{v \rightarrow u}^d = \phi$, v forwards the packet to u again, and the Loop occurs, leading to contradiction. Clearly, *some* of the packets will incur loops in the network.

Property 5: If some of the packets incur loops, then this condition occurs in the network.

Algorithm 3 Route Metric Sequences for link $A \rightarrow B$ using traditional routing

$\mathcal{N}$: Number of Nodes in the Topology;
$\overleftarrow{\mathcal{SPT}}_x^d$: Shortest path tree to destination d from node x with cost of link $A \rightarrow B$ as $currCost$
$\overrightarrow{\mathcal{SPT}}_x^d$: Shortest path tree to destination d from node u with cost of link $A \rightarrow B$ as $newCost$

procedure getORMS(node u, node v, node A, node B, dest d)
/*for the link $A \rightarrow B$, destination d */
$currCost = cost\ of\ link\ A \rightarrow B$
$lastCost = \infty$
$newCost = currCost + 1$
$M_{A\rightarrow B}^d = \phi$
while newCost < lastCost
  for each node $x \in (u, v)$
    $\overleftarrow{SPT_x^d}$ = getSPT(x, d, A, B, currCost)
    $\overrightarrow{SPT_x^d}$ = getSPT(x, d, A, B, newCost)
    SPT_x^d = merge($\overleftarrow{SPT_x^d}$,$\overrightarrow{SPT_x^d}$)
    if cycles exist in SPT_x^d
      $M_{A\rightarrow B}^d = M_{A\rightarrow B}^d \bigcup$ newCost
    end if
    $currCost = newCost$
    $newCost++$
  end for
end while
return $M_{A\rightarrow B}^d$

Before we provide the argument, some basic axioms that can be formed from the definition of FIFR techniques are listed here. S1: If a packet just follows F^d at every node in its path, packet will NOT incur loops. S2: If a packet just follows $\overrightarrow{F^d}$ at every node in its path, packet will NOT incur loops. S3: If a packet just follows B^d at every node in its path, packet will NOT incur loops. S4: If a packet just follows B^d and F^d at all nodes, packet will NOT incur loops. S5: If a packet just follows B^d and $\overrightarrow{F^d}$ at all nodes, packet will NOT incur loops. S6: If a packet follows $\mathcal{F}^d$ initially and follows ($\overrightarrow{\mathcal{F}^d}$) later, proposed condition can occur, packet may incur loops. S7:

If a packet follows ($\overrightarrow{\mathcal{F}^d}$) initially and follows $\mathcal{F}^d$ later, proposed condition can occur, packet may incur loops.

Proof: If a packet changes its next hops from F^d to $\overrightarrow{F^d}$ due to the router's change of state, and some of the nodes in its path are still following old state, then packet will NOT incur loops if every node in its path follows B^d. This is due to the loop freedom provided by the FIFR. A packet will follow B^d, only when $KL^d \neq \phi$ from previous node to current node.

At one of the nodes in a packet's path, if $KL^d \neq \phi$ from previous node to current node, packet can follow B^d or F^d depending on the previous node. If previous node appears in SPT^d of current node, the node infers link failure and forwards the packet using B^d, otherwise the node forwards the packet using F^d. In either case, packet will NOT incur loops, due to S3, S4.

At one of the nodes in a packet's path, if $KL^d = \phi$ from previous node to current node, packet can follow F^d or $\overrightarrow{F^d}$ depending on the state of the node. If the packet follows F^d, we can conclude that packet will never incur loops, due to S1. If the packet follows $\overrightarrow{F^d}$, we can not conclude that packet will not incur loops. And the proposed condition occurs in the network.

In essence: If a packet moves from a node in new state (ac era) to a node in old state (bc era), and the observed keylinks are null at the node in old state; some of the packets can incur loops in the network. The existence of proposed condition in the 3 rare topologies, is mentioned in Tables 3.9, 3.11, 3.13. All the links that match the condition are circled in their appropriate cells.

Efficacy of the Proposed Condition in Real Production Networks

We simulate 268 different real world topologies and verify if any of the link failures will cause the proposed condition to appear in each of those topologies. The results of our simulations are plotted in Figure 3.9.

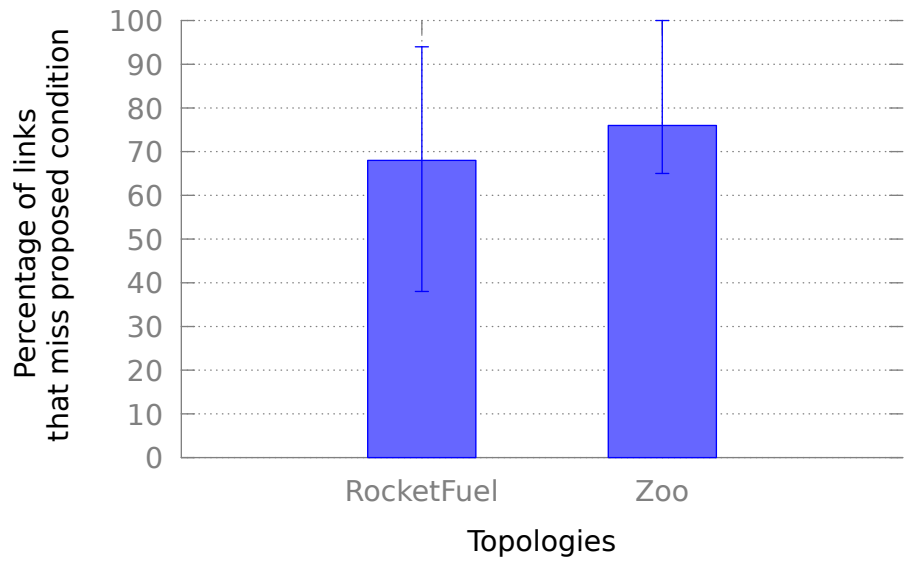


Figure 3.9: Accuracy of the proposed condition among topologies available in the internet of Zoo (262) and RocketFuel topologies (6): Percentage of link failures that will miss the proposed condition in network, in turn eliminating the need for simulation.

The plot in Figure 3.9 shows that the condition mentioned above appears for a minimum of 2% and a maximum of 35% of link failures among different topologies of Zoo, suggesting that metric sequences may be necessary for only these 35% of links in each topology. For the links where we need sequences, we suggest to use the algorithm 3 and verify the resulting metric sequence via simulations. When we compare the results: we can observe that only 2 of these total number of links really need route metric sequences, even though we need to verify around 35% of the total links among all the topologies available in the Internet of Zoo.

The plot in Figure 3.9 shows the number of times the proposed condition occurs among Rocketfuel topologies, suggesting a minimum of 5% and maximum of 62% of links among different RocketFuel topologies need to be simulated for finding the FIFR++ metric sequence.

While the condition proposed above does not provide any false negatives, it can be clearly seen that the number of false positives is high. While the number of links for which we need progressive increments is 2 (in the extremely rare scenarios), we

have around 35% of the links needing simulations for topologies in the Internet of Zoo. A condition that determines those links accurately, is considered as part of the future work.

This section investigated the possibility of using failure inference based fast reroute (FIFR) along with a link state update mechanism for loop-free fast convergence while performing fast reroute. We have observed that FIFR alone drastically reduces the potential forwarding loops during convergence, even with traditional link state updates. Furthermore, when coupled with deferred updating of back hops, it nearly eliminates the looping problem. In our evaluation, only 3 out of 17339 links corresponding to 280 real and random topologies could cause transient loops for a particular combination of updated and not-yet-updated nodes. We proved that loop-free convergence can be guaranteed by employing FIFR with deferred updating of back hops and progressive metric increments (called FIFR++). We showed that FIFR++ rarely requires incremental updates and in those rare instances needs fewer metric increments. Next, we present how to provide loop-free convergence with link up events and then discuss how node failures can be handled with FIFR++.

3.2 Network Convergence during Link State Up Event

A link that is down because of an unplanned failure or routine maintenance is typically brought up again, necessitating a network-wide convergence. Similar to link down scenario, a link up event can also cause forwarding loops during convergence. These loops can be avoided by maintaining a strict order for updating the routers. However, routers need to communicate among themselves to ensure that this ordering is met. In this section, we show that we can make a creative reuse of interface-specific forwarding entries to extend the FIFR++ approach to provide loop-free fast convergence for link up events also without any coordination between routers.

First, let us see the need for a new mechanism for loop-free convergence in case of a link up event. Consider the topology shown in Figure 3.10. Assume that all the solid lines correspond to the current (bc) topology, while the dashed link between H–M is a new link (corresponding to the ac topology) and all the routers received the link up event. While each node in the network received the link state updates, assume that only nodes K and O already processed the LSA update and are in the updated (ac) state. Consider the packets destined towards the node M. The forward and back hops to the destination M at each node of the topology is mentioned in Table 3.14. Now suppose a packet originates at K that is destined to M. If each node just follows the FIFR approach as above, a packet destined to M from K will traverse the path of $K \rightarrow J \rightarrow L \rightarrow N \rightarrow O \rightarrow K \rightarrow J$, resulting in a loop.

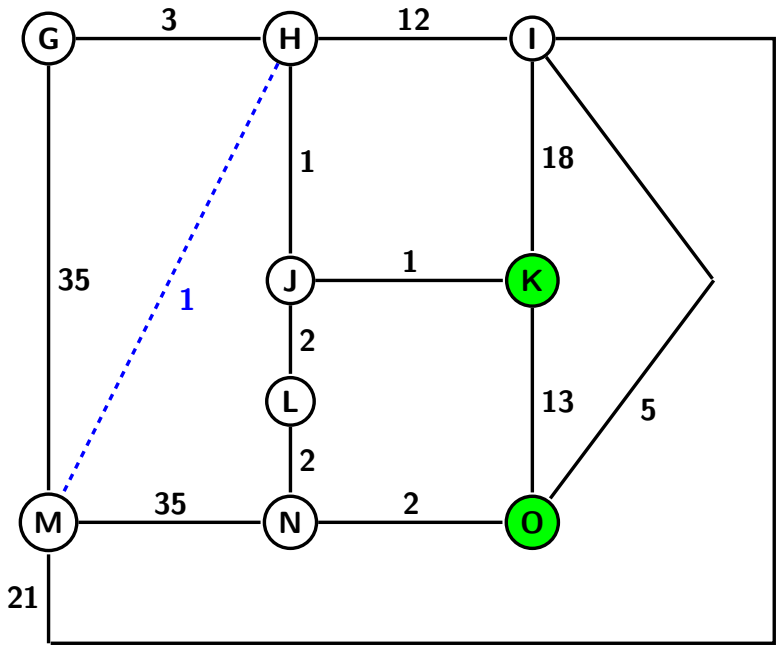


Figure 3.10: A topology used for illustration of FIFR for Link State Up Event.

3.2.1 Proposed Method

We now present how FIFR++ achieves fast loop-free convergence by making a creative use of interface-specific forwarding. Interface-specific forwarding implies that

Table 3.14: Next and back hops for destination M at each node of the topology in Fig. 3.10

Node	NH (bc)	BH (bc)	NH (ac)	BH (ac)
G	M	-	H	M
H	I	G	M	-
I	M	O	H	M
J	L	H	H	L
K	J	O,I	J	O,I
L	N	J	J	N
N	O	M	L	O
O	I	N	N	K

a packet's next-hop is determined based on both its destination and incoming interface. Let $\mathcal{ISF}_R$[prv] be the forwarding table corresponding to the incoming interface prv$\rightarrow R$. Then, a packet p arriving at R from the previous hop prv gets forwarded to the next-hop $\mathcal{ISF}_R$[prv](p.dst). Apart from interface-specific forwarding, FIFR++ assumes that all the links in the network are bidirectional with equal weight in both directions, which is generally true for backbone networks. With symmetric link weights, a forwarding loop happens only when a packet traverses two neighboring routers that are in different eras and each router is the other router's next-hop for the packet's destination according to their view of the network topology. Therefore, a router R in the ac era can infer that a packet's previous hop Q is in the bc era, if in the ac era the next hop from R to destination is Q. In that case, R can forward the packet according to its bc table and avoid a potential forwarding loop. When R has only the bc table, a packet is forwarded under FIFR++ based on its destination alone regardless of the incoming interface.

We describe the proposed approach using an example. Consider again the topology shown in Figure 3.10 where a packet is being forwarded from source K to destination M. As mentioned earlier, suppose nodes K and O are in *ac* state while all other nodes in *bc* state. In this scenario, the initial path remains the same as before, $K \rightarrow J \rightarrow L \rightarrow N \rightarrow O$. But at O, the *ac* next hop is N and the back hop for packets arriving from N is its *bc* next hop which is I. So, when O receives the packet, its forwards it to I. Therefore, with the proposed approach, the packet traverses the path of $K \rightarrow J \rightarrow L \rightarrow N \rightarrow O \rightarrow I \rightarrow M$ and reaches the destination. Thus, the proposed method can be used to eliminate loops during network convergence of a link up event.

FIFR++ operations are formally specified in Algorithm 4. When a packet p arrives at R from the previous hop prv, it is simply forwarded to $\mathcal{ISF}_R[\texttt{prv}](p.\texttt{dst})$. Upon computing a new FIB, the new next hops are pushed to all the interfaces as usual, except for the interfaces corresponding to the next-hops (could be multiple with ECMP) of each destination. Only for those interfaces, where the next hop is neighbor, will be set to the next-hops corresponding to the `bc` table. When the router receives an LSA, these entries are reset to the usual next-hops, and thus effectively purging the `bc` table.

3.2.2 Proof of Loop-Free Convergence with FIFR++

We now prove that FIFR++ is loop-free, under the following constraints: 1) Only one network event propagates throughout the network until the convergence for that event is complete; 2) A router R does not install a new FIB and change its $R.\texttt{era}$ until its direct neighbors are notified of the corresponding LSA. 3) Links are bidirectional with symmetric weights. We show that under FIFR++, a packet does not traverse the same link in the same direction more than once.

We first analyze the loop freedom property of a shortest path routing protocol. The only property assumed in this generic routing protocol is that each router delivers

Algorithm 4 : Operations under FIFR++

1: **if** Router R receives a packet p from previous hop `prv`
2: forward p to $\mathcal{ISF}_R$[`prv`](p.`dst`)
3: **end if**
4:
5: **if** Router R receives a new LSA
6: **for each** destination `dst`
7: $\mathcal{ISF}_R[\mathcal{F}_R[R.\texttt{era}](\texttt{dst})](\texttt{dst}) \leftarrow \mathcal{F}_R[R.\texttt{era}](\texttt{dst})$
8: **end for**
9: purge $\mathcal{F}_R[\overline{R.\texttt{era}}]$
10: **end if**
11:
12: **if** Router R has recomputed a new FIB
13: $R.\texttt{era} \leftarrow \overline{R.\texttt{era}}$
14: $\mathcal{F}_R[R.\texttt{era}] \leftarrow$ new FIB
15: **for each** neighbor `prv` of router R
16: $\mathcal{ISF}_R[prv] \leftarrow \mathcal{F}_R[R.\texttt{era}]$
17: **end for**
18: **for each** destination `dst`
19: $\mathcal{ISF}_R[\mathcal{F}_R[R.\texttt{era}](\texttt{dst})](\texttt{dst}) \leftarrow \mathcal{F}_R[\overline{R.\texttt{era}}](\texttt{dst})$
20: **end for**
21: **end if**
22:
23: **if** Packet p originates at router R
24: forward p to $\mathcal{F}_R[R.\texttt{era}](p.\texttt{dst})$
25: **end if**
26:
27: **if** Router R initializes its state
28: $R.\texttt{era} \leftarrow 0$
29: $\mathcal{F}_R[1] \leftarrow \mathcal{F}_R[0]$
30: **end if**

a packet to the next hop in the shortest path to the destination according to its view of network. We show that shortest path routing can avoid a certain type of routing loop. Once loop freedom is established for the shortest path routing, we extend the argument for FIFR++ to prove its loop freedom in general.

Property 6: The generic shortest path routing protocol is free from circular loops in symmetric link networks.

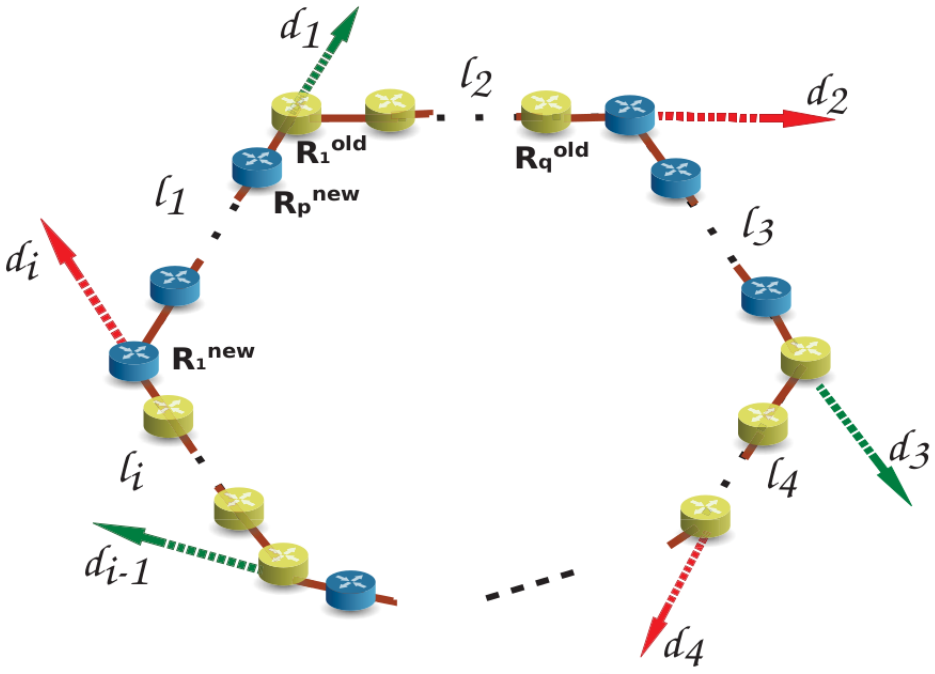


Figure 3.11: A generic circular loop scenario in a symmetric link network.

We use the term *circular routing loop* to refer to a scenario where a packet reaches the initial router (can be any of the participating routers that is considered as the starting point of the looping packet) without traversing any edge twice in the same direction, Consider an example scenario shown in Fig. 3.11. In this loop, the updated and non-updated routers are arbitrarily chosen. The darker nodes depict the updated routers that use the new topology and lighter nodes are for non-updated/uninformed ones that use the old topology to forward a packet. Uninformed nodes can not be neighbors of any updated node as per our assumptions, however they would use the same topology as the non-updated nodes. Since the path used by uninformed nodes does not differ from a non-updated neighbor, we can consider uninformed nodes as non-updated ones in this proof without loss of generality.

The circular loop consists of segments of consecutive updated/non-updated nodes that follow the same path to the destination. These segments can contain one or more possible nodes. For example, the segment consisting of routers R_1^{new} through R_p^{new}

follows the new topology path to the destination. The last router in this segment R_p^{new} forwards the packet to R_1^{old}, the first node of the next segment. This node uses the old topology shown by the dotted lines from router R_1^{old} to router R_q^{old}. We define the collective edge weight of the links from R_1^{new} to R_1^{old} as l_1 and that of the links along the old path from R_1^{old} to the destination as d_1. Similarly, the values for next segment are l_2, d_2 respectively and so on. We consider i such segments in the circular loop. To show that the shortest path routing will not complete any circular loop in a symmetric link network, it will be sufficient to show that the loops are not possible with shortest path routing.

We first assume that a loop exists and then we prove the loop free property by showing a contradiction. In a circular loop as shown in Figure 3.11, the router R_1^{new} can forward the packet from $R_1^{new} \rightsquigarrow R_1^{old}$ and then follow the new path shown with the dotted line from router R_1^{old}. Alternately, R_1^{new} can forward the packet from $R_1^{new} \rightsquigarrow R_k^{old}$ and then follow the path shown with the dotted line from the router R_k^{old}. R_1^{new} chooses the first path as the shortest path to the destination, therefore the total weight of the first path must be less than the second one as captured in the following inequality:

$$l_1 + d_1 < l_i + d_{i-1}$$

Following the similar argument for all of the i segments, we can obtain the following inequalities.

$$l_2 + d_2 < l_1 + d_i$$

$$l_3 + d_3 < l_2 + d_1$$

$$\vdots$$

$$l_{i-1} + d_{i-1} < l_{i-2} + d_{i-3}$$

$$l_i + d_i < l_{i-1} + d_{i-2}$$

Now adding the left and right hand sides yields a contradiction,

$$\sum_{n=1}^{i}(l_n + d_n) < \sum_{n=1}^{i}(l_n + d_n)$$

Therefore, in symmetric link networks, no generic shortest path routing protocol will cause circular loops.

Property 7: FIFR++ is free from circular routing loops.

In FIFR++ an updated router does not follow the old or new topology independently. The previous hop decides the choice of topology in updated routers. However, a non-updated router, which is a neighbor of any updated one, has only the old table and therefore no choice of topology. It always forwards with the old topology view. As the generic routing algorithm proves freedom from circular loops without any restriction on the choice of topology in updated nodes and FIFR++ does not contradict any assumption made therein, it is evident that FIFR++ is also free from a circular loop.

Next, we consider what we refer to as the *linear routing loop*, where a packet reaches the initial router by traversing all the edges of the loop exactly once in each direction, as in Figure 3.12. We prove that FIFR++ does not cause a packet to go around a linear routing loop more than once. Towards that end, we establish some invariants under FIFR++.

Invariant 1: $\forall R_n \in N(R), R.\texttt{era} \neq R_n.\texttt{era}$ if exactly one of $\mathcal{F}_R[\overline{R.\texttt{era}}]$ and $\mathcal{F}_{R_n}[\overline{R_n.\texttt{era}}]$ does not exist.

According to the FIFR++ scheme, a router R changes its `era` value only when it has completed a FIB update. Therefore, the value of the `era` bit for R differs from its neighbor R_n only if one of them, say R_n, has installed a new FIB and the other hasn't. Then, according to our assumptions, the neighbor R of the updated router R_n

should have received the LSA and purged its forwarding table $\mathcal{F}_R[\overline{R.\texttt{era}}]$ immediately after receiving it. Therefore, $\mathcal{F}_R[\overline{R.\texttt{era}}]$ will not exist until R completes a FIB update. Conversely, if $\mathcal{F}_{R_n}[\overline{R_n.\texttt{era}}]$ does not exist, that implies R_n is computing an update and would have the same `era` value as R until its FIB update is complete.

Invariant 2: In FIFR++ an updated node can not return a packet to the router that last forwarded the packet to it.

Suppose an updated node R gets a packet from a neighboring node Q. For R to return the packet to Q, in the new topology Q must be its next hop, which implies that Q must be using the old topology. In this case, R will forward this packet using the old topology. When both Q and R use the same old topology for forwarding, R can not return the packet to Q.

Invariant 3: Though an uninformed node has two forwarding tables, it always uses the old topology to forward packets.

FIFR++ assumes that there can not be any uninformed node as a neighbor of an updated node. Therefore an uninformed node can only get a packet from another uninformed node or a non-updated node. Moreover, at any time there can be at most a single event which an uninformed node is unaware about. Therefore, when an uninformed node A forwards a packet to another uninformed node B, it always uses its $\mathcal{F}_A[A.\texttt{era}]$ table and it in turn leads B to use its $\mathcal{F}_B[B.\texttt{era}]$, both new tables in their view. But, $\mathcal{F}_A[A.\texttt{era}]$ and $\mathcal{F}_B[B.\texttt{era}]$, in uninformed nodes A and B respectively, actually correspond to the old topology. Now, in case a non-updated node C forwards a packet to B, C uses its only table $\mathcal{F}_C[\overline{C.\texttt{era}}]$ which represents the topology before the failure according to the FIFR++ actions. Therefore, from B's view the packet

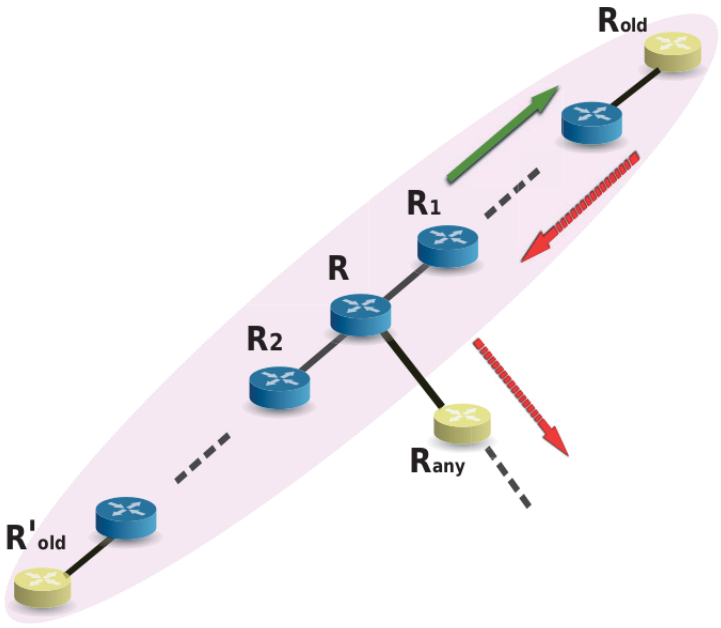


Figure 3.12: A generic linear loop scenario in a symmetric link network.

comes following a new topology and it uses its $\mathcal{F}_B[B.\texttt{era}]$ to forward this packet. This shows that under all circumstances an uninformed node uses the old topology to forward a packet.

Invariant 4: In FIFR++, an uninformed node does not get a packet returned from the node it forwarded the packet to.

We use the similar argument as in Invariant 1 to claim that an uninformed node can only pass a packet to another uninformed node or a non-updated node. Now, according to the Invariant 1, in both of the cases the recipient node uses the old topology, which is also used by the uninformed sender node. As both the sender and receiver nodes use the same topology to forward the packet, the receiver will not return it to the sender.

Property 8: FIFR++ does not cause a packet to go through a linear routing loop more than once.

We consider a generic linear loop repetition scenario in a symmetric link network as shown in Fig. 3.12 and argue that the scenario is not possible with FIFR++. According to the Invariant 2, such a loop must have more than two nodes. Moreover, the nodes at the extreme ends of the loop must return the packet to the router that last forwarded it. Therefore, these two nodes must be non-updated as indicated by Invariant 2. The intermediate nodes have to forward a packet to two different neighbors and it is possible if the nodes contain two tables for old and new topologies. In FIFR++, a router can have two tables if it is an uninformed or updated node. Note that an uninformed node can not be a neighbor of an updated node. So, the intermediate nodes of the repeatable linear loop must be either all uninformed or all updated. According to Invariant 4, the node adjacent to the end nodes must be updated, therefore all intermediate nodes are updated. The two non-updated nodes at the extreme ends of the loop forward a packet with old topology paths in opposite directions in the loop. To make this possible, there must be an old topology path from the router R_{old} to the destination that deviates from the loop at an intermediate node, R, as shown in the figure.

The linear loop as shown in Fig. 3.12 must involve forwarding a packet from R'_{old} to R_{old} and back twice to be an incident of loop repetition. Lets consider a packet being forwarded from R_1 to R. If $R \rightarrow R_1$ exists in the new shortest path tree, then R uses the old path and forwards it to R_{any} and thus breaks the loop. Otherwise, the packet will follow the new shortest path from R to R_2, and then through the consecutive updated nodes to R'_{old}. Now, suppose packet comes back to the router R from R_2. Since we have considered $R \rightarrow R_2$ as being part of the new shortest path, when the packet goes from R to R'_{old}, $R_2 \rightarrow R$ can not be part of the new path. So, R

forwards the packet to R_{any} and thus breaks the loop again. Thus, in networks with symmetric links, forwarding with FIFR++ is loop-free [55].

In this section, we extended the FIFR++ approach for preventing forwarding loops during the convergence period of a link up event, and restore the network to an optimal routing state as soon as possible. We have proved that it is possible to achieve the fast convergence with fast rerouting using interface-specific forwarding in networks with symmetric link weights, without any changes to the packet format.

3.3 Integrated Approach for Handling All Link State Events

In this section, we integrate the proposed methods for ensuring loop-free convergence for both link up and link down events. A fine state machine diagram for the integrated approach is depicted in Fig 3.13.

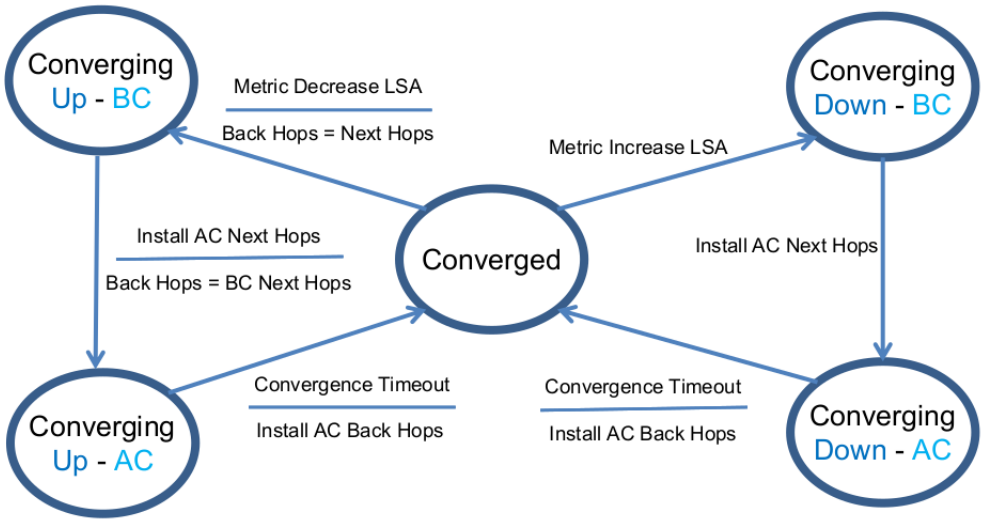


Figure 3.13: A Router's state diagram, while handling link state advertisement messages.

Each router remains in its *Converged* conventional state. In its Converged state, each router follows interface specific forwarding, in the sense that the next hop for forwarding a packet is based not only on the destination, but also on the previous hop of that packet.

When an LSA is received and it corresponds to a new link that has come up, the router enters *Converging Up - BC* state. In *Converging Up - BC* state, the router stops using back hops and forwards packets from all interfaces using interface-independent next hops. Once the new *ac* next hops are calculated, the forwarding entries are updated to the new next hops while the back hops are set to the *bc* next hops, and the router switches to the *Converging Up - AC* state. Once all the routers have updated their next hops (after the maximum convergence time), every router in the network returns to the *Converged* state by installing the updated *ac* backhops.

When a link cost increase LSA is received, the router infers one of the existing links is going down and reaches *Converging Down - BC* state. In its *Converging Down - BC* state, the router follows Interface specific forwarding. Once the new next hops are calculated, the forwarding entries are updated to the new *ac* next hops while the backhops remain as the previous *bc* backhops. Once all the routers have updated their next hops (after the maximum convergence time), every router in the network returns back to the *Converged* state by installing the updated backhops.

Note that we assume that there is only ONE network wide event at any point of time. In other words, If we observe a link cost decrease event once, we will keep observing link cost decrease event before we observe any other event. Therefore, all the routers in the network would either be in *Converging Up* state or all in *Converging down* state. We will not observe some nodes in *Converging Up* state and some others in *Converging Down* state at the same time. Under these assumptions, the proposed integrated approach ensures loop-free convergence for both link up and link down events in a network with symmetric link weights.

3.4 Network Convergence during Node Down Event

We have shown thus far that FIFR++ approach can ensure loop-free forwarding before, during, and after network convergence in case of any single link state change.

In this section, we discuss how this approach can provide the same in case of a single router failure. Note that when a router fails, that amounts to the failure of all its adjacent links. So, one way to address a router failure is as a multiple link failure scenario. Unfortunately, FIFR and consequently FIFR++ can not deal with multiple independent link failures. However, by treating the router failure as a single event, it is still possible to address it and forward packets to their destinations without incurring loops. We briefly discuss the approach below and then evaluate its effectiveness.

First, when a node fails, we need to perform fast reroute around the failure without causing loops. We have already shown in Chapter 2 that by inferring key nodes, instead of key links, FIFR provides loop-free forwarding in case of a single node failure. Second, we need to ensure that during network convergence, due to LSAs generated by the node down event, packets do not loop. Note that a node failure causes multiple link failures and they in turn trigger several LSAs. A straightforward application FIFR++ approach discussed earlier does not work, as it assumes that a single link state event is being propagated in the network. Instead, we propose that a router should wait to collate multiple LSAs, compute the next hops corresponding to the state change, and update its FIB once, treating it as a single event. In other words, in the finite state machine shown in Figure 3.13, the router should stay in *Converging Down - BC* state till it gathers all the LSAs associated with the router failure event and then switch to *Converging Down - AC* state once it computes the new forwarding table. With this change, FIFR++ can deal with node failures too.

The remaining question then is how fast FIFR++ will converge in case of a node failure. As is the case with link state changes, we would like to avoid progressive metric increments if possible, otherwise convergence would be slow. To study the effectiveness of FIFR++ in case of node failures, we considered a wide variety of network topologies and brought down each router in the topology to identify if any of the packets will incur transient loops. We used Rocket Fuel (6) topologies, topolo-

gies from the Internet of Zoo (262), and Random topologies (12). Results from the simulations are presented in Figure 3.14. The percentage of router failures that cause packets to incur loops in the network for each of the topologies is shown without FIFR and with FIFR++. Note that FIFR++ here refers to the case that does not include progressive metric increments.

The results show that without FIFR, nearly 30%, 60%, and 40% of routers in Rocketfuel, Random, and Zoo topologies respectively when down can cause loops during convergence. On the other hand, none of the router failures in Rocketfuel topologies cause loops with FIFR++, resulting in fast convergence. In case of Zoo topologies, only around 0.57% of all routers (62 out of 10877 routers) cause loops with FIFR++ without progressive increments. Similarly, only 2 routers (0.07%) in Random topologies require progressive increments with FIFR++. In all of the studied topologies, we observed that a packet does not loop if it reaches a node adjacent to the failed node. Otherwise, it is possible for a packet loop between two intermediate nodes. However, our evaluation has shown that only a small fraction of routing failures could cause looping with FIFR++ during convergence and they can be avoided by employing progressive increments. Overall, in almost all situations FIFR++ provides fast convergence, while always ensuring loop-free forwarding.

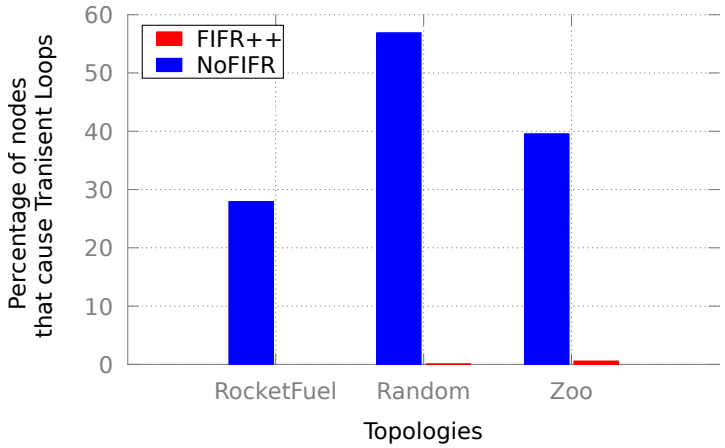


Figure 3.14: Percentage of node failures that cause packets to incur transient loops during network convergence

3.5 Alleviating Micro-Congestion

The above sections discussed how FIFR++ can avoid packet drops due to forwarding loops during convergence. In this section, we study another cause for packet drops: Network Congestion. Congestion occurs fairly frequently in modern networks [66, 42]. A recent study on network data sets (measured every second) of one large cloud network operator observed that packet drops occur, but are uncorrelated or weakly correlated with observed link utilization [66]. Surprisingly, authors found that packet drops occur even when a link is less than 1% utilized, when the utilization is computed over a 4 second interval. On the other hand, when link load statistics are gathered every 25μ seconds, authors observe a very important phenomena, the link queue builds up and becomes empty very fast. Clearly, most of the congestion events are too short-lived to be characterized by the average link utilization over 4 second interval. It is this phenomena, named as *Micro-Congestion*, where the congestion occurs due to bursty traffic for a very short period of time, that we study in this section.

Traditionally, congestion at a link is handled by end-hosts using variants of TCP (for example: DCTCP, TCP Reno). The router that observes congestion, sets the ECN bit of the packet and forwards it to the receiver. When the destination receives a packet with ECN bit set, it notifies the source about congestion in the network and then the source reduces rate of transmitting packets into the network. However, due to the time lag in setting ECN bit and destination notifying the source, some packets in the network get dropped during high congestion (traffic burst). Since the destination needs to receive and process the ECN bit and update the source about congestion, it might take multiple (>1) RTTs for the sender to reduce the rate of transmission (at least RTT/2 for the sender to receive congested signal and some more time to adapt to the congested signal). The authors of [66] show not only that the bursts exist in network, but also that almost all the high utilization in the network is part of a burst that starts and ends in a few μ seconds. Clearly, the RTT is in the

order a few *ms*, which is far longer than any burst duration. It is this observation that motivates the study of this section.

As mentioned in the previous chapter, FIFR can be used to reroute packets in the network in case of transient link failures. In this section, we explore the idea of using FIFR when micro-congestion is detected. When a micro burst occurs in the network, we propose that routers adjacent to that link use backup paths provided by the FIFR to reroute the traffic. One main challenge in this scheme is to show that the packets do not observe loops, when they are rerouted. In the remainder of this section, we handle the case of controlling congestion using a variant of FIFR method. Subsection 3.5.1 provides a comprehensive study of the work related to this problem setting. In Subsection 3.5.2, we propose a design that mitigates packet loss due to congestion in the network. In Subsection 3.5.3, we explain the evaluation details for the proposed idea and present the observed results.

3.5.1 Related Work

The authors of [2] show that congestion in the network can be detected by (un)setting the ECN bits of packets. Any intermediary switch that observes congestion sets the ECN bit in packet header . Based on the received packet, destination of a flow notifies the sender to reduce the rate of transmission. On other hand, our proposal does not necessitate to modify any of the packets.

Authors of [59] suggest the intermediary switches append their switch Identifiers and epoch Ids into the packet headers, before forwarding a packet. The edge destination device that receives the packet, uses the Identifiers listed in packet header to debug the congestion (or) link failure event in the network.

Authors of EverFlow [67] suggest that controller configures a set of rules on each switch and any packet that matches those rules to be mirrored in the network. These mirrored packets are sent to the analyzers, which processes a set of packets for loops,

drops, link load counters etc. All the information about the network is gathered from these mirrored packets. Since these packets act as a representative sample of the flows in the network, configuring these rules is very important for working on EverFlow.

Authors of SFlow [12] suggest to monitor flows on all ports of a switch continuously, by sampling packets of each flow, using which congested links are instantly highlighted. Additionally, SFlow suggests to use statistical properties of packet sampling, to provide quantifiable measurements. Authors of [43] suggest that counters maintained in software provide more control in the network than maintaining them on ASICs.

Authors of [4] propose that end host votes for every link in the network path of a flow to be the faulty link whenever it observes TCP Re-transmissions for that flow. They further show that as long as a flow goes through one faulty (failed/congested) link at any time, it can be proved that the failed link will be determined accurately by using their proposed scheme.

Authors of [56, 65] propose to correlate transport layer metrics and network system call delay with the path taken by each flow, and apply statistical techniques to localize the faulty link. Some of the switches in the network are responsible for signaling the network path for each flow. Hence, this proposal needs modifications to the switch functionality, which is not a common option to many other proposed techniques.

Authors of [13] propose a congestion control algorithm in the hypervisor host OS, that is decoupled from the congestion control algorithm used in guest OS. The authors claim that though the guest OS believes the Congestion Control algorithm is private. However, their proposed algorithm is common across all the guest OSes (dependent on the CC policy on hypervisor). The authors propose to add a translational layer, that translates the legacy TCP into a newer variant of TCP. The main advantage of this approach is that the classical guest OSes need not be modified with updated TCP versions, but can get the functionality of recent TCP variations.

3.5.2 Proposed Design

The problem studied in this section can be concisely stated as: *'Can we handle bursts occurring and ending at the granularity of* ***microsecond*** *in a network, by using Fast Reroute techniques?*' Since micro-congestion occurs for a very short period of time, we can model micro-bursts occurring at a link as transient link failure for a micro-second interval. We can then apply FIFR techniques for routing around congestion similar to the way it routes around failures. We propose to extend the FIFR technique to infer not only link failure, but also link congestion (micro-bursts). Inferring which link is congested will be same as inferring which link is failed in the network. Therefore, packets do not incur loops by inferring congestion, just as they do not incur loops by inferring failures (routers avoid such links in both cases).

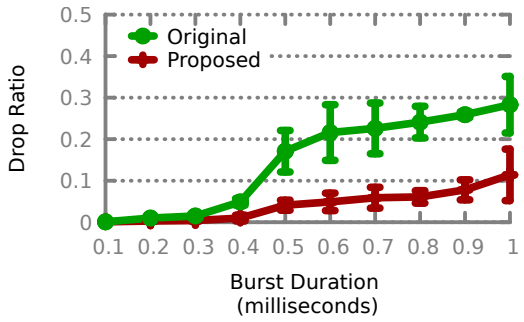
When a link is congested and an arriving packet finds that the link queue is full, normally, this would result in a packet drop. Under the proposed approach, however, any packet that were to be dropped due to an overflowing queue would be rerouted using a back hop for that link, that was computed for performing fast reroute in case that link fails. In other words, a router adjacent to a congested link with a full queue as a failed link and initiates fast reroute. All other routers perform interface-specific forwarding as they typically do under FIFR to route around a failure. So, only change needed to employ the proposed approach on a network that already uses FIFR is to treat a congested link as a failed link and initiate fast reroute. While the proposed approach is appealing, the downside is that FIFR can only handle single link or router failures. Therefore, it can guarantee loop-free forwarding when only one link is congested in the network at any time instant. But when multiple links are simultaneously congested and they appear on the primary and backup paths to a destination, it is possible under the proposed approach to have packet drops due to looping. To study how often this occurs in practice, we evaluate the proposed approach on multiple real world IP backbone networks.

3.5.3 Performance Evaluation

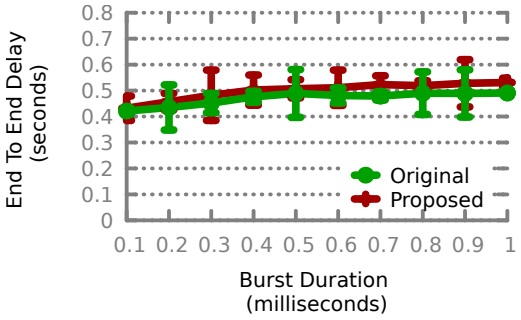
We used the standard network simulator (ns-3) [24, 58, 30, 14] tool to evaluate the proposed approach to alleviate micro-congestion. Some of the parameters considered while evaluating the scheme are: the topology, amount of traffic entering into the network, the location through which bursty traffic enters into the network, link that observes micro-congestion due to the bursty traffic, the total number of packets that get affected due to micro-congestion (size of each burst), the presence of backup paths at each node, the number of links that cause micro-congestion at the same time, the location of multiple links that cause micro-congestion at the same time, and the average degree of the network.

We evaluated on three different topologies: A standard Abilene and Exodus topologies and a Random topology generated from BRITE topology simulator [40]. Abilene topology has 11 nodes and 14 links associated; Exodus topology has 79 nodes and 294 links; while the random topology has 25 nodes and 60 links. The fact that each node Exodus topology has a higher degree compared to other topologies is an important parameter to be observed. Across all the topologies, we create enough number of flows between every source destination pair such that the average link utilization reaches a threshold (70%) [65]. Once the network contains enough traffic, we start the bursty traffic from one of the nodes to another node (both randomly picked). We vary several parameters of the bursty traffic and measure the metrics such as average end to end delay and observed packet drop ratio in the network. For each of the varying parameter, the results are provided below.

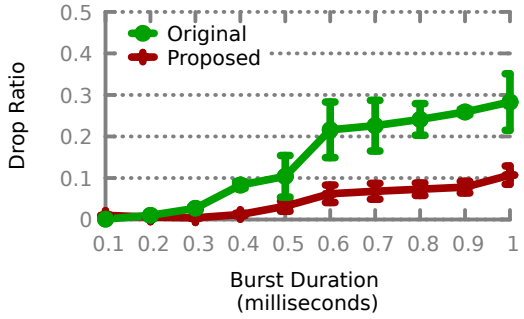
Burst Duration: We varied the amount of time each burst lasts, once it is generated by a flow. The results for different topologies are provided in Figure 3.15. The peak burst and queue size are maintained constant at $0.4ms$ and 400 packets. From the results, it can be clearly observed that the proposed technique reduces the drop ratio with a minimal increase in the end to end delay. As the burst duration



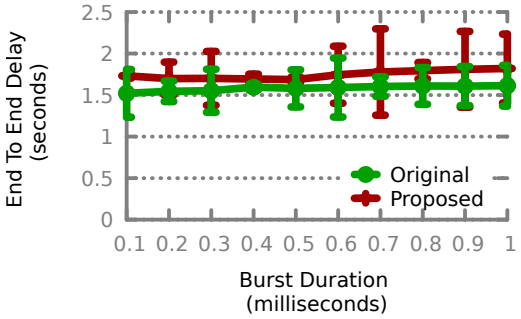
(a) Average Packet Drop Ratio for the packets generated from all Source Destination pairs



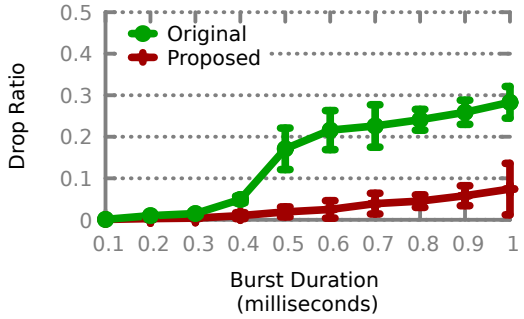
(b) Average End to End delay between all source destination pairs



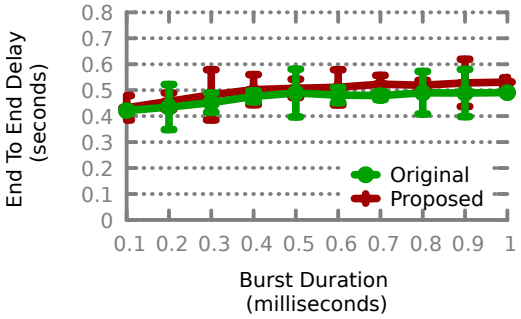
(c) Average Packet Drop Ratio for the packets generated from all Source Destination pairs



(d) Average End to End delay between all source destination pairs



(e) Average Packet Drop Ratio for the packets generated from all Source Destination pairs



(f) Average End to End delay between all source destination pairs

Figure 3.15: Varying the Burst Duration for Abilene (a,b), Random (c,d) and Exodus (e,f) Topologies

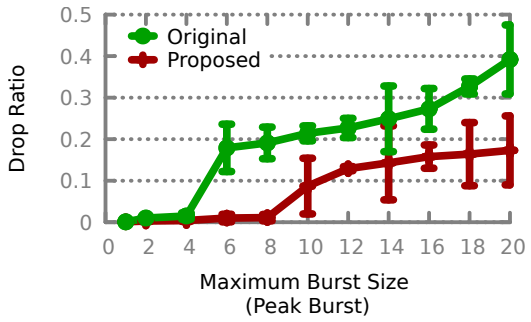
increases, the effect of using the proposed technique is more visible. The difference in the drop ratio becomes significant with increase in the degree of the network.

Peak Burst: We varied the ratio between the maximum number of packets generated at the time of burst and non-burst time intervals. The results for different topologies are provided in Figure 3.16. The queue size and burst duration are kept constant at 200 packets and $0.4ms$. From the results, it can be clearly seen that the drop ratio is significantly lower by using the proposed technique, with a very minute increase in the end to end delay. Also, as the peak burst increases, we observe that both drop ratio and end to end delay are significantly different with Abilene topology, but not that different for Random topology. This is due to the presence of very few links in Abilene topology, Abilene has 2 bottleneck links that affect the results more rapidly than others. Due to the presence of multiple links at each node, nodes in Exodus topology can maintain the drop rate consistently.

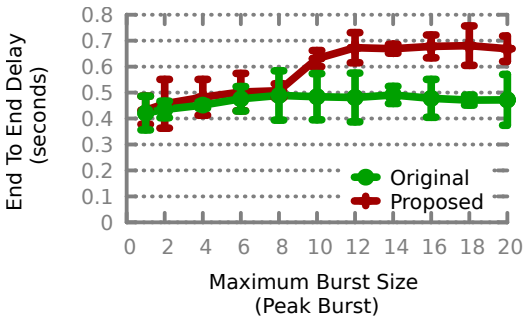
Queue Size: We also varied the number of packets each node can store on one interface, or the size of buffer at each interface. The results for different topologies are provided in Figure 3.17. The peak burst and burst duration are kept constant at 8 and $0.4ms$. From the results, it is clear that increasing the queue size reduces the drop ratio and end to end delay, to some extent. Once a threshold is reached, increasing the queue size do not have any affect on the drop ratio. This is because all the peak burst is already handled at that queue size.

3.6 Conclusion

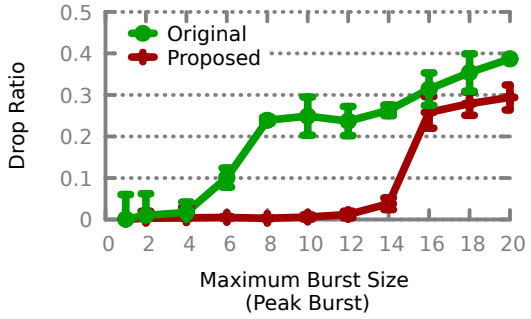
In this chapter, we presented an approach called FIFR++, that extends a previously proposed scheme called FIFR, for dealing with network contingencies such as link and router failures and micro bursts. We have shown that FIFR++ provides loop-free convergence during link/router down/up events an IP back bone network. We have evaluated FIFR++ using 280 real and random topologies and our results confirm that the order of updates does not matter with FIFR++ for 17,336 out of 17,339 links and 11,707 of the 11,772 routers in those topologies, leading to fast convergence. More-



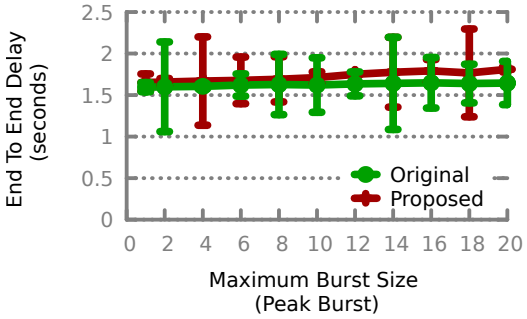
(a) Average Packet Drop Ratio for the packets generated from all Source Destination pairs



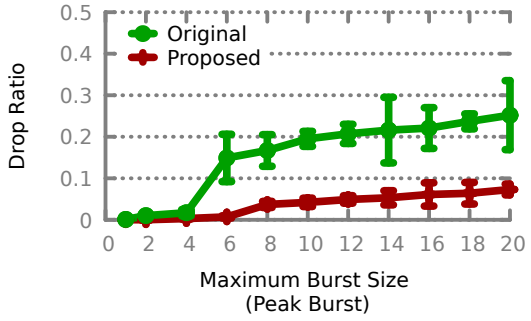
(b) Average End to End delay between all source destination pairs



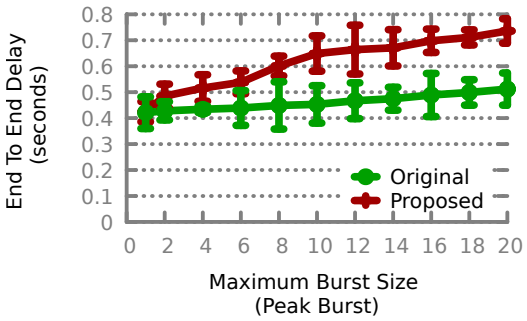
(c) Average Packet Drop Ratio for the packets generated from all Source Destination pairs



(d) Average End to End delay between all source destination pairs



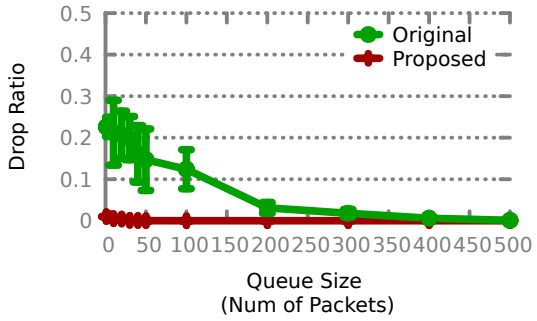
(e) Average Packet Drop Ratio for the packets generated from all Source Destination pairs



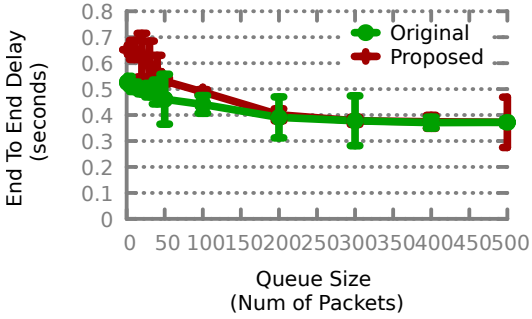
(f) Average End to End delay between all source destination pairs

Figure 3.16: Varying the Maximum Burst (Peak) for Abilene (a,b), Random (c,d) and Exodus (e,f) Topologies

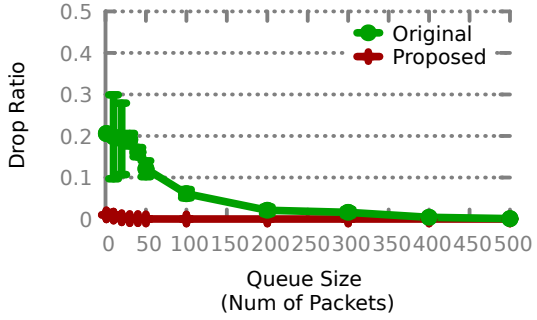
over, for those outlier links and routers, FIFR++ uses progressive metric increments to ensure loop-free convergence. We have also demonstrated that FIFR++ can be employed to mitigate packet loss due to micro traffic bursts in IP backbone networks.



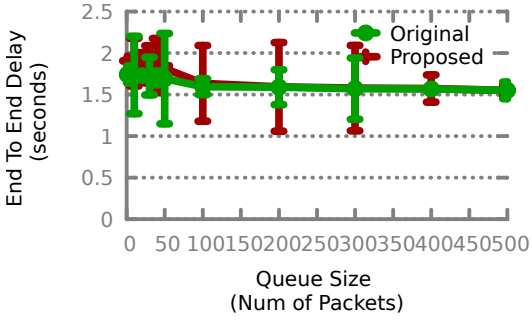
(a) Average Packet Drop Ratio for the packets generated from all Source Destination pairs



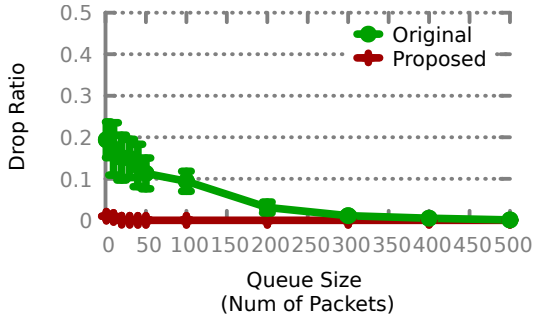
(b) Average End to End delay between all source destination pairs



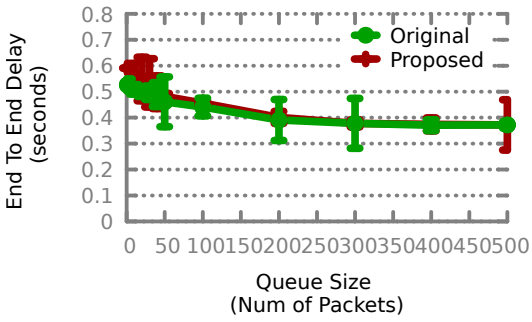
(c) Average Packet Drop Ratio for the packets generated from all Source Destination pairs



(d) Average End to End delay between all source destination pairs



(e) Average Packet Drop Ratio for the packets generated from all Source Destination pairs



(f) Average End to End delay between all source destination pairs

Figure 3.17: Varying the Queue Size at each router for Abilene (a,b), Random (c,d) and Exodus (e,f) Topologies

Part II

Robotic Wireless Networks

Chapter 4

Building Wireless Communication Maps for Avoiding Network Disruptions [1]

[1] Alberto Quattrini Li, Phani Krishna Penumarthi, Jacopo Banfi, Nicola Basilico, Jason M O'Kane, Ioannis Rekleitis, Srihari Nelakuditi, Francesco Amigoni, Multi-robot online sensing strategies for the construction of communication maps, Springer Autonomous Robots, 1-21, 2019.
Reprinted here with permission of publisher.

4.1 Introduction

This chapter aims at increasing the efficiency of a multirobot system that builds an ad-hoc network *communication map* by using prior information from the environment map.

A communication map encodes the information about whether robots in two arbitrary locations can communicate or not. Building communication map can not only be a standalone task—for example to decide where to optimally place routers in an indoor environment—but also is important to efficiently accomplish other robotic tasks, such as exploration [64, 50, 6], environmental monitoring [15], and search and rescue [44]. Indeed, it is experimentally shown that communication constraints degrade the system performance [60]. Many recent works are explicitly considering communication in the multi-robot systems design [27, 6, 22]. All these works share in common the assumption that: robots have a communication map readily available, which is not available in practice [37], or that a conservative communication model is used, such as limited range line-of-sight, limiting the capabilities of the robots. Having a reliable communication map allows the robots not to be hindered by the communication constraints, when choosing where to go, and to have a more efficient multirobot coordination.

Figure 4.1: The robots, TurtleBots 2 equipped with a Wi-Fi dongle, deployed in an environment to build its communication map.

In this chapter, we build on a previous work [29], in which a system for the efficient construction of communication maps, where the communication source is not stationary, is generated. We use a team of robots capable of measuring signal strength between them and a Gaussian Process (GP) to model the communication map. Considering every location in the free space where robots can take measurements would make the robots travel extensively and so the construction of a communication map would be too time consuming to be feasible. Thus, limiting the numbers of candidate locations to informative places is important to accomplish such a task in an efficient manner. Specifically, we use *a priori* communication models that can be built out of the physical map of an environment to reduce the number of candidate locations to those that provide some distinctiveness, decreasing the exploration time and the total traveled distance. In addition, we filter out input measurements to the GP to reduce GP's computational complexity; which is $O(n^3)$ [53] for n observations, resulting in a system that scales better over time and space. We describe four different communication models for Wi-Fi communication along with experiments to test how close to real data they are. We then present how such models can be used to filter observations to update the communication map and to generate candidate locations used by the sampling strategies. A series of experiments with a team of TurtleBots 2 (see 4.1) demonstrate the effectiveness compared to methods which do not exploit such information.

In the literature, typically, the problem of building a communication map with respect to a fixed router in the environment is considered [41, 17]. Such a communication map can be used to improve indoor localization [33]. Some previous work involved the use of hand-held devices to create a radio map [57]. Other approaches exploited a single robot exploring an environment and localizing a radio source [61] and multiple robots to map a stationary source without any coordination [18]. Kemppainen et al. [31] proposed a method for a single robot to explore a magnetic field that

can be measured in an environment so that it can localize. Hsieh et al. [28] propose some offline methods to compute efficient joint paths for small teams of robots, with the aim of collecting signal strength measurements from a set of predefined locations.

The chapter is structured as follows: the next section describes the problem in detail. Section 4.2.1 presents an overview of the system that our contributions are based on, highlighting the proposed approach in this chapter. Section 4.3 describes communication models for Wi-Fi communication and how such information is used by the robots to make the communication map building process more efficient. Section 4.4 shows experimental results from numerous experiments with real robots. The chapter concludes with lessons learned and a discussion with interesting research directions. This chapter is concluded in section 4.5.

4.2 Problem Statement

Similar to [29], m mobile robots are deployed in a known bounded environment with obstacles, where free space is denoted as $\mathcal{A} \subset \mathcal{R}^2$ and $\mathbf{p} \in \mathcal{A}$ denotes a location that can be occupied. Robots can localize themselves within a global coordinate system with a laser range finder. Further, they are equipped with an omni-directional Wi-Fi transceiver, to communicate with peers over the radio channel within a maximum communication range allowed by the device.

The robots select a sequence of candidate locations in an online fashion, to measure the signal strength between two locations. Measurements are included in the communication map. Note that robots can collect data while traveling to a selected location.

A *communication map* represents information about communication links availability between ordered pairs of locations in $\mathcal{A}$. It is defined as a function $\hat{f} : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}_{\leq 0}$ estimating the received radio signal strength, RSSI in dBm, f between any two locations $\mathbf{p}_i$ and $\mathbf{p}_j$. The closer it gets to zero, the higher the reliability and

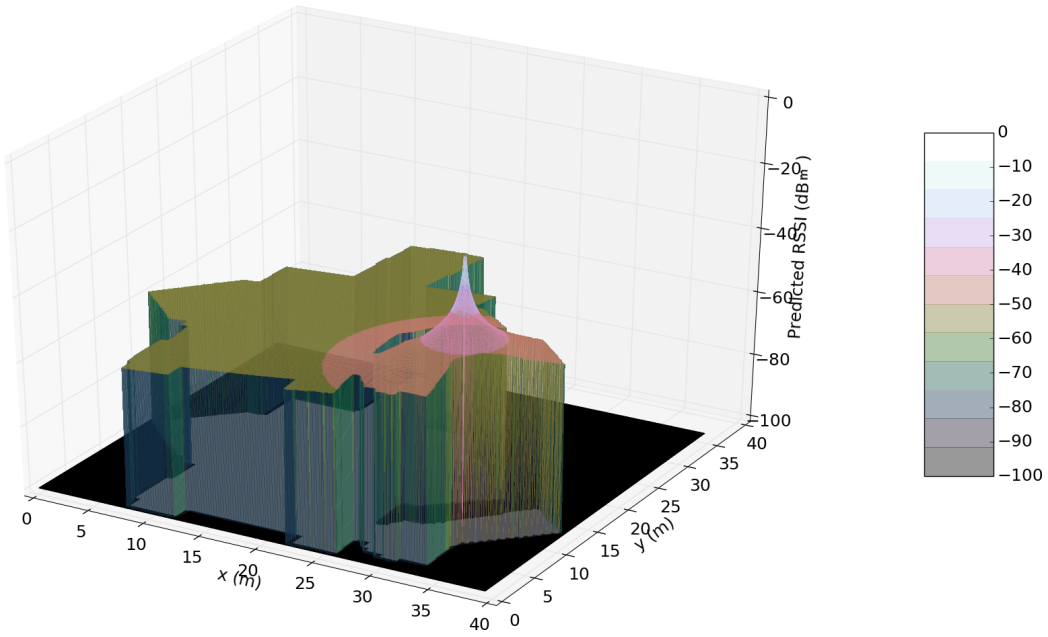


Figure 4.2: Example of communication map. For clearly differentiating the locations where robot can not visit in the physical map, we are not allowing the GP to predict (and plot) RSSI values at locations inaccessible to the robots.

reachable bandwidth between $\mathbf{p}_i$ and $\mathbf{p}_j$. Let us call $\mathbf{x}_{ij} = (\mathbf{p}_i, \mathbf{p}_j)$ pair of locations in the freespace and $\hat{f}(\mathbf{x}_{ij})$ the estimate of the signal strength from $\mathbf{p}_i$ to $\mathbf{p}_j$. As communication links not necessarily are symmetric, in general, $f(\mathbf{x}_{ij}) \neq f(\mathbf{x}_{ji})$ [26].

An example of a 2-Dimensional communication map is presented in Figure 4.2, by fixing the location of the transmitting robot. Obviously, locations with highest RSSI value will be the locations closest to the transmitting robot.

This chapter, differently from [29] that considers all possible locations in an environment (up to a discretization), focuses on the following two questions: First, which measurements $\mathbf{x}_{ij}, f(\mathbf{x}_{ij})$ should be used for updating the communication map; Second, which candidate locations $\mathbf{p}_j$ should be considered by the robot, to visit and collect data, for a fixed $\mathbf{p}_i$. Given that the free space of an environment can be arbitrarily large, reducing the number of observations both for building on-line the communication map and deciding where to go is important for reducing the traveled distance and increase the scalability.

4.2.1 Background

A communication map maps the physical location to its communication strength and is helpful in determining the next destination for the robot [3]. We are interested in a multi-robot system that learns and updates the communication map *during* the space exploration. The problem of choosing optimal locations is studied in [7]. The multi-robot system that the proposed approach is going to be integrated in is composed of two main components, that are presented in the following two subsections: a modeling of the communication map, and a decision on the next destination location. A detailed description is available at [29].

4.2.2 Gaussian Process based Communication Map

The communication map is generated from a GP [53], given the spatial correlation that radio signal strength displays. Such a model can also be used to predict signal strengths in areas where measurements have not been collected yet, with an associated uncertainty. Specifically, $\hat{f}$ can be estimated as a posterior distribution fitted over a set of noisy observations made by robots which explore and coordinate in the environment to collect signal strength measurements. Assume that the robot team as a whole collected q measurements over the environment. Let $\mathbf{Y} = [y^1, y^2, \ldots, y^q]^T$ and $\mathbf{X} = [\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^q]^T$ be the set of those measurements and the set of the corresponding pairs of locations from where they have been collected, respectively; recall that $\mathbf{x}^i \in \mathcal{A}^2$. The signal strength observation $y^i = f(\mathbf{x}^i) + \epsilon$ is affected by additive sensing error, which is assumed to be i.i.d. and $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$. The covariance function expresses the spatial correlation between any two values of f. It takes as input the two location pairs corresponding to each value. Let us denote the covariance function as $k(\mathbf{x}, \mathbf{x}')$. A radial basis kernel (RBF) is used, as done in the mainstream approach:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\Big(- \frac{|\mathbf{x} - \mathbf{x}'|^2}{2l^2} \Big). \tag{4.1}$$

where the signal variance σ_f^2 and length scale l^2 are parameters that indicate the amplitude and the smoothness.

To make notation easy to follow, given $X_1 = [x_1^1, \ldots, x_1^a]^T$ and $X_2 = [x_2^1, \ldots, x_2^b]^T$, we identify with $K(X_1, X_2)$ the $a \times b$ matrix, where $K_{ij} = k(x_1^i, x_2^j)$ and with I_q the $q \times q$ identity matrix. The correlation between the observed function values is represented by the following equation:

$$\text{cov}(\mathbf{Y}) = K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 I_q \tag{4.2}$$

Here, the GP is assumed to have a zero mean function, as typically done in literature [53]; therefore it is fully specified by the parameter vector $\theta = [\sigma_n^2, \sigma_f^2, l^2]^T$. Such a parameter vector is computed as the one maximizing the observations log-likelihood, that is, $\theta^* = \arg\max_\theta \log p(\mathbf{Y} \mid \mathbf{X}, \theta)$ where:

$$\log p(\mathbf{Y} \mid \mathbf{X}, \theta) = -\frac{1}{2}\Big(\mathbf{Y}^\mathbf{T}\text{cov}(\mathbf{Y})^{-1}\mathbf{Y} - \log|\text{cov}(\mathbf{Y})| - n\log 2\pi\Big). \tag{4.3}$$

To calculate an estimate of the signal strength, θ^* optimal parameter vector is used in unobserved regions by evaluating the posterior. Specifically, called $\mathbf{W} = [\mathbf{w}^1, \mathbf{w}^2, \ldots, \mathbf{w}^l]^T$ a set of arbitrary location pairs for which a signal strength estimate is requested, $p(f(\mathbf{W}) \mid \mathbf{X}, \mathbf{Y}) \sim \mathcal{N}(\mu_\mathbf{W}, \mathbf{\Sigma_W})$, where the mean vector is obtained as $\mu_\mathbf{W} = K(\mathbf{W}, \mathbf{X})\text{cov}(\mathbf{Y})^{-1}\mathbf{Y}$ and represents the estimate $\hat{f}(\mathbf{W})$, while the covariance matrix is given by $\mathbf{\Sigma_W} = K(\mathbf{W}, \mathbf{W}) - K(\mathbf{W}, \mathbf{X})\text{cov}(\mathbf{Y})^{-1}K(\mathbf{W}, \mathbf{X})^T$. Note that the main diagonal of $\mathbf{\Sigma_W}$ is called *predictive variance* and quantifies the uncertainty of estimates in $\mathbf{W}$.

Because of the inverse operation of the covariance matrix, the complexity of updating the GP with n number of observations is $O(n^3)$. As such, in this chapter we address the following question: is it possible to wisely select observations to include in the update of the GP so that the model is still accurate, but at the same time the multirobot online system is not overloaded?

4.2.3 Sensing strategies

Computing and planning the optimal set of locations robots should reach during the exploration of an environment is a hard problem [7]. In this work, two main sensing strategies are adopted to decide pairs of location for collecting measurements, and both are based on a *leader-follower* scheme. Specifically, the first sensing strategy, called *Pairwise Mapping* (PM), divides a team of robots into pairs, where one acts as a leader, and the other one as a follower. Pairs of locations are selected by the leader preferring pairs of locations that display high predictive variance inferred from the GP. In addition, other robots' plan is considered in this decision: if two selected locations are close to locations selected by other robots, they are discarded. To ensure robustness, a backup pair of locations where robots could communicate is also selected, in case robots cannot communicate from the new locations. The two locations are then assigned to the robots to minimize the maximum traveled distance.

The second sensing strategy, called *Region Mapping* (RM), allows one leader to have more than one follower. The idea is that instead of minimizing the uncertainty of the communication map by iteratively selecting pairs of locations, with RM, the objective is to select and minimize the uncertainty of a given region centered in a selected location from the leader. First, the leader randomly selects a location $\mathbf{p}_c$, as a center for the region, taking into account the associated uncertainty and possible overlaps with other teams of robots. In that region, locations to be assigned to the followers are iteratively chosen, considering the highest sum of predictive variance when paired with $\mathbf{p}_c$ and sufficiently far apart from the already chosen waypoints. Also with the RM strategy, backup locations are decided to avoid any significant disconnection between robots.

In [29], the whole free space was discretized according to a minimum distance set between locations and sampled, up to a maximum communication range for an arbitrary location. This results in a large number of possible observations. In this

chapter, we pose the question: can we utilize some prior models to reduce the set of possible candidate locations, to improve the performance of the system?

4.3 Filtering based on Empirical Channel Models

In this section, first, we present the *a priori* communication models used, with an evaluation of their fidelity; second, we show how such models can be used to filter observations for updating the communication map and locations where robots should go; see Figure 4.3 to see how the proposed approach modifies the one in [29].

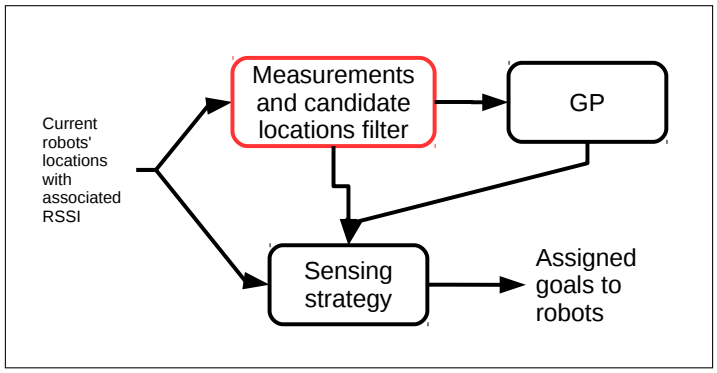


Figure 4.3: Block diagram representing the proposed system integrated with that of [29]; in red, the proposed modification to the communication exploration system.

4.3.1 Prior from Communication Models

In general, it is hard to directly estimate the RSSI value knowing the map of an environment, because the compositions of the obstacles are not known. However, in the communication literature, some models—such as Free space, Two-ray, Ten-ray, Wall Attenuation Factor model, and Multi Wall Attenuation Model [21]—have been proposed, by estimating the signal's power loss during propagation (also known as *path-loss*)[2]. Each of these models vary in terms of computation complexity and accuracy. Further, each model usually has several parameters and quantifying them

[2]It should be noted that all these path-loss models are independent of the used communication frequency, i,e., not restricting the analysis to Wi-Fi/LTE etc.

accurately is hard, as they depend on the considered physical environment. However, such models have been shown to perform relatively well, when the parameters are predicted from the training data obtained from the actual measurements [5].

We selected and evaluated four communication path-loss models with varying complexity that have been tested in indoor environments. In the following the equations for the different models[3], referring to transmitter $\mathbf{p}_i$ and receiver $\mathbf{p}_j$:

Distance Model (DIST) A free space distance based path-loss model assumes the signal is passing through vacuum; the path-loss observed by the signal depends only on the Euclidean distance between locations of transmitter and receiver [52, 21]:

$$L_{\text{dist}}(\mathbf{p}_i, \mathbf{p}_j) = -10 \log_{10} \left[\frac{\sqrt{G_L}\lambda}{4\pi d(\mathbf{p}_i, \mathbf{p}_j)} \right]^2 \tag{4.4}$$

where, G_L is the product of transmitted and receiver antenna gains[4]; λ is the wavelength of the transmitted signal; and $d()$ is the Euclidean distance between transmitter and receiver locations.

Wall Attenuation Factor Model (WAF) An empirical path loss model [5], which assumes physical map of the environment to be available beforehand, as path-loss is influenced by the number of walls between transmitter and receiver, in addition to the euclidean distance:

$$L_{\text{waf}}(\mathbf{p}_i, \mathbf{p}_j) = L_{\text{dist}}(\mathbf{p}_i, \mathbf{p}_0) - 10n \log_{10} \frac{(d(\mathbf{p}_i, \mathbf{p}_j))}{d_0} - \{w(\mathbf{p}_i, \mathbf{p}_j) \times WAF if w(\mathbf{p}_i, \mathbf{p}_j) < C, C \times WAF\ Otherwise\} \tag{4.5}$$

where, $L_{\text{dist}}(\mathbf{p}_i, \mathbf{p}_0)$ is the path loss at reference distance between transmitter $\mathbf{p}_i$ and arbitrary $\mathbf{p}_0$; n indicates the rate of change in path-loss; $w(\mathbf{p}_i, \mathbf{p}_j)$ is the number of

[3]Note a slight change of notation compared to the previous works to make notation uniform and highlight variables and parameters.

[4]Gain is defined in terms of the antenna's capability to send/receive signals in a direction.

walls on a straight line between transmitter and receiver, C is an empirical constant—i.e., the maximum number of walls that can make a difference in path-loss; WAF is a constant factor specific to the type of each wall.

Multi-Wall Model (MWM) Another empirical path loss model [68], whcih assumes the number of walls on ceiling and floow to be known follows Equation 4.6:

$$L_{\text{mwm}}(\mathbf{p}_i, \mathbf{p}_j) = L_{\text{FSL}}(\mathbf{p}_i, \mathbf{p}_j) + \sum_{l=1}^{N} k_l w_l(\mathbf{p}_i, \mathbf{p}_j) + k_f \mathfrak{f}(\mathbf{p}_i, \mathbf{p}_j) \tag{4.6}$$

where, $L_{\text{FSL}}(\mathbf{p}_i, \mathbf{p}_j) = L_{\text{dist}}(\mathbf{p}_i, \mathbf{p}_0) + 10n \log(d(\mathbf{p}_i, \mathbf{p}_j))$ models a free space path-loss model; $w_l()$ is the number of walls of l^{th} type between transmitter and receiver, k_l is a parameter for the attenuation affecting the signal for wall of type l; $\mathfrak{f}(\mathbf{p}_i, \mathbf{p}_j)$ is the number of floors between transmitter and receiver, and k_f is the attenuation parameter observed by signal due to the type of that floor.

ITU Radio communication Model (ITU) An empirical model, used by *IEEE 802.15 Working Group for Wireless Personal Area Networks* [25], for testing the proposed channel model of a signal propagating in an arbitrary environment:

$$L_{\text{itu}}(\mathbf{p}_i, \mathbf{p}_j) = 20 \log_{10} f + n \log_{10} d((\mathbf{p}_i, \mathbf{p}_j)) + k_f \mathfrak{f}((\mathbf{p}_i, \mathbf{p}_j)) - 28 \tag{4.7}$$

where, n is the distance power loss coefficient; f is the communication frequency (MHz); $d()$ is the distance between the transmitter and receiver (in meters); k_f is the floor penetration loss factor (dBm); $\mathfrak{f}()$ is the number of floors between transmitter and receiver.

Each parameter should be fine-tuned according to the specific environment. Values for parameters are heuristically suggested in the related papers, usually for a communication signal at $2.4GHz$ (WiFi) for different scenarios, including indoor office environment.

Considering the transmitting power T_{power} (dBm), the RSSI between transmitter and receiver can be then calculated as:

$$\text{RSSI}(\mathbf{p}_i, \mathbf{p}_j) = T_{\text{power}} - L_{()}(\mathbf{p}_i, \mathbf{p}_j) \tag{4.8}$$

As a physical map of the environment is available, a prior communication map can be computed for every location reachable by a receiver robot, given a fixed location for a transmitting robot. Note that as robots cannot access some locations—e.g., because of doors—and maps are pre-built by the robots, the number of walls is an estimate of the actual number of walls. Specifically, every change from freespace cell and occupied cell in the grid is counted as one wall. Maps are preprocessed in such a way that small objects are removed from the map and thus not counted as wall. Generating a prior communication map using the different models shows that locations closer to the fixed robot observe higher RSSI values, while distant locations have lower values. While the trend seems to be similar, the RSSI values from these priors are different; among them, WAF model seems to weigh more the different terms and as such the returned values are smaller.

Table 4.1: Errors observed in the calculated RSSI values, for SIX experiments performed varying locations of fixed robot; while moving robot follows a stable fixed path and collects data

Path-loss	Experiment 1		Experiment 2		Experiment 3		Experiment 4		Experiment 5		Experiment 6	
Model	Mean	stdev	Mean	stdev	Mean	stdev	Mean	stdev	Mean	stdev	Mean	stdev
Distance	8.09	4.98	9.40	7.27	9.40	7.28	17.56	9.27	10.89	8.54	6.14	5.62
WAF	12.02	10.24	15.33	12.08	15.34	12.09	27.29	10.87	20.06	11.85	17.47	7.04
MWM	7.98	5.01	9.37	7.26	9.38	7.27	17.66	9.29	11.15	8.58	4.03	3.66
ITU	11.33	7.84	15.49	10.28	17.79	7.09	29.02	11.74	20.51	9.66	15.40	6.43

We evaluate the accuracy of such models, calculating the difference between the measurements collected by two robots described in Section 4.4 and the RSSI values from the communication models (error). In particular, we conducted *six* different experiments in the engineering building of the University of South Carolina[5]. Each experiment involved one robot fixed at a different location, and the other robot following a precomputed coverage path. Each robot measures the Wi-Fi signal 10 times a second along with its position in the map. The physical environment used for these experiments is depicted in Figure 4.4a.

Table 4.1 shows mean (and standard deviation) error for the six different experiments. It is worth mentioning that: we observed a change of 8 dBm to 10 dBm in the RSSI value while changing the height of the antenna (by a few centimetres) at the moving robot multiple times. An accuracy error of < 20 dBm is comparable to what is shown, for example in [5]. As such models display a relatively low error, especially MWM, it is justified to use such models as priors for the robots constructing a communication map, as shown in the next section.

4.3.2 Use of Communication Models Prior

For a given location of a transmitting robot—leader in the strategies described in Section 4.2.1—we propose an algorithm to automatically generate a set of locations that can be provided as goal to the followers and can be used also for integrating measurements in the GP. The main idea is to choose locations that are informative, namely those which present some change in the field, as constant values can be easily approximated by a model.

Mathematically speaking, the slope of RSSI is determined at each location, based on the prior communication map built from the communication models. This is

[5]All experiments were conducted at night time, so the interference due to moving objects/humans is minimal, except for the people performing the experiment.

basically the first order derivative of the RSSI value at every location. We also determine the change of slope between neighbor locations in the map. If the change of slope between two locations crosses a threshold (τ), we select that location as one of the possible goals of the moving robot (follower). This idea is explained in detail in the Algorithm 5.

Algorithm 5 Goals for Moving Robot to Pick Observations

INPUT:$\mathcal{A}$ (physical map of the environment), $\mathbf{p}_i$ (possible locations that robots can occupy)
$N = |\mathcal{A}|$, $comm\_model$,
$(x_1, y_1) \in \mathcal{A}$ (considered location of transmitter), τ (threshold)
Notation: $comm\_model = \{itu, waf, dist, mwm\}$
{Communication model of the environment}
Location of the fixed robot, **Notation**: $(x_1, y_1) \in P_i$
Ensure: List of candidate goals for the moving robot $\{(x_i^d, y_i^d)\},\ \forall i \in goals$
for $(x_2, y_2) \in p_i - (x_1, y_1)$
    $rssi\ (x_1, y_1, x_2, y_2)\ =\ calculate\_rssi\ (x_1, y_1, x_2, y_2, comm\_model)$;
    {Using appropriate Equation from 4.4, 4.5, 4.6, 4.7}
end for
$goals = get\_goals(rssi, x_1, y_1, N, p_i)$;
return *goals*

procedure: $get\_goals(rssi, x_1, y_1, N, p_i)$
$goals$={}
{Return variable}
for $(x_2, y_2) \in p_i$
    $rssi\_slope(x_1, y_1, x_2, y_2) = calculate\_slope(x_1, y_1, x_2, y_2, rssi)$
    {Calculate the slope of RSSI}
end for
for $(x_2, y_2) \in \mathcal{A}$
    $rssi\_change\_slope(x_1, y_1, x_2, y_2) = calculate\_slope(x_1, y_1, x_2, y_2, rssi\_slope)$
    {Calculate the slope of rate of change of RSSI }
end for
for $(x_2, y_2) \in p_i$
    if $rssi\_change\_slope(x_1, y_1, x_2, y_2) > \tau$
        $goals.add((x_2, y_2))$
    end if
end for
return *goals*

Additionally, locations generated from the algorithm can be used to filter the measurements. If measurements are taken close to those locations within a given range, they are included in the communication model. It is important to note that: τ needs to be generated heuristically, for each communication model separately. When τ is high, robot may not receive sufficient number of goals to predict accurately, while lower τ could result in too many goals for the robot. From our preliminary experiments, τ value of 11 seemed reasonable ($\pm 10\%$) to all the four models, and provided between $30 - 120$ goals for various communication models–among 212 possible locations–in the tested environments. Note that to account for inaccuracies of the prior model, locations within a given radius are added to the list of candidate locations.

The proposed method to select locations is integrated in the PM and RM strategies by filtering the locations considered for the follower. PM and RM algorithm chooses the set of goals, i.e., leader choosing one (or) many followers according to the strategy briefly described in Section 4.2.1. Further, such selected locations are also used to determine which measurement to include in the communication map.

Using the data collected during our experiments described in the previous section—i.e., one robot fixed at a location, while the other moving along a fixed, known path—we evaluate the GP with all measurements, and the GP with filtered measurements from the proposed method.

In general, the GP with filtered data maintains low variance in GP predictions and low Root Mean Square Error (RMSE), and at the same time the GP training time reduces by 50%, compared to the GP with all data. For example, the Root Mean Square error between the observed data and the predicted values from the GPs, in one of the experiments, is 11.014 for the GP with all data and 11.03 for the GP with filtered data.

This experimental results validate the use of such a filtering approach. Note that such a priori information could also be used within a GP, by changing the mean function, however, the standard definition for the GP worked well enough, without adding complexities for estimating the hyperparameters.

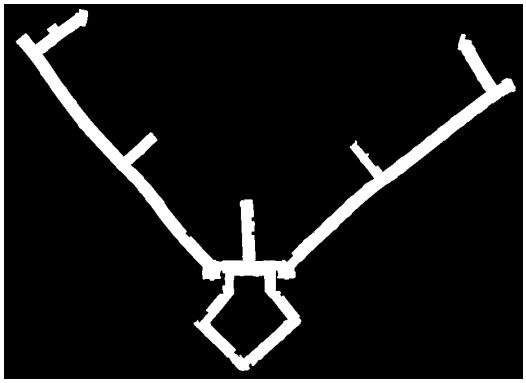
The next section shows the proposed method in an online scenario where robots decide "where to go" to build the communication map.

4.4 Experimental Evaluation using TurtleBot2 Robots

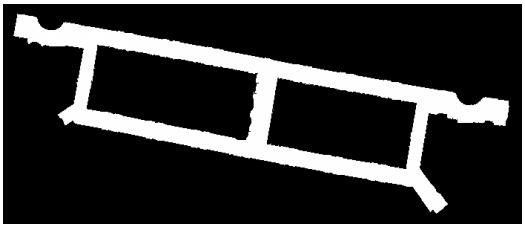
As we already showed a validation for filtering the training data on the GP and as we are assessing whether such prior communication models could work in a real scenario, simulations are not run as they would not reflect the real world communication channel performance. We use a fleet of TurtleBot 2 platforms[6], equipped with an RGB-d sensor (Microsoft Kinect) that allows the robots to localize together with odometry information [48, 36, 51, 7, 3]. The maps used for localization are built in a setup phase and are represented as an occupancy grid. Specifically, a single robot is manually driven around the environment to collect RGB-d readings which are then processed using the ROS gmapping package [23]. Figure 4.4 shows two different environments with different characteristics in Swearingen Engineering Center at University of South Carolina: one map depicting a corridor passing through the labs and the other map depicting a corridor passing through faculty workspace.

We considered two relatively big indoor environments with different characteristics in the Swearingen Engineering Center at the University of South Carolina, depicted in Figure 4.4. Figure 4.4a is characterized by long corridors with some intersecting short corridors and one small loop. Note that between the two long corridors, there is an outdoor space, which the robots cannot access. Figure 4.4b is instead an environment with corridors surrounding small office rooms.

[6] `http://www.turtlebot.com`



(a) Corridor surrounding Lab Workspace (Lab-Corridors 66m x 92m)

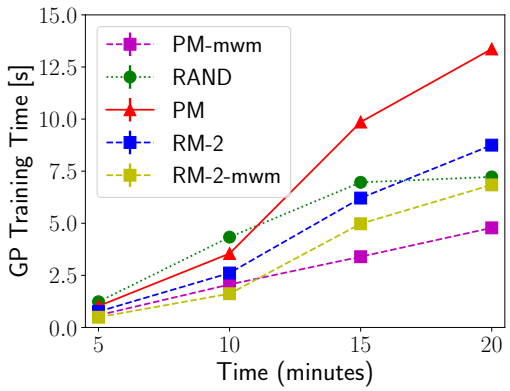


(b) Corridor surrounding the Faculty Workspace (Office-Corridors 30m x 70m).

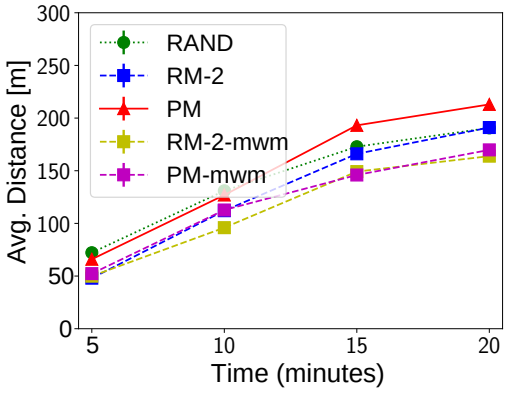
Figure 4.4: Two environments, portions of third floor at Swearingen Engineering Center, University of South Carolina.

After running some preliminary experiments, we decided to use *MWM* as the communication model to generate candidate locations using Algorithm 5, because it provides a good number of candidate locations and has good accuracy (see Section 4.3.1). We execute the updated algorithm on a real system with two Turtlebot 2 robots. The modified versions of PM and RM are compared against the basic version defined in [29], and on a baseline strategy, RAND, where robots independently move to random destinations over the environment while taking Wi-Fi measurements with respect to other teammates.

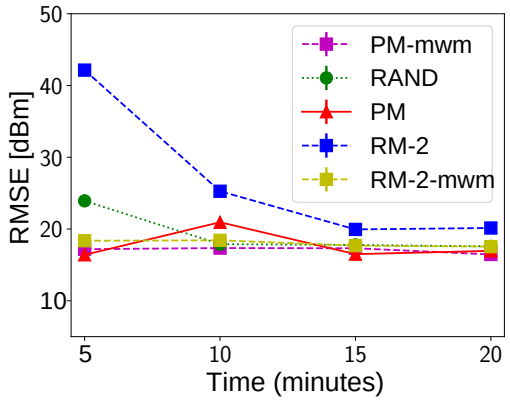
For each of the two physical environments mentioned in Figure 4.4, we verified the performance of both proposed and base line methods; by using 2 Turtlebot 2 robots, for a 20 minute duration. The strategies are evaluated by considering the quality of the GPs that is obtained; by merging all the collected data in a global rendez-vous after every 5 minute, i.e., traveled distance and the GP processing time. Quality is measured both in terms of RMSE, calculated as the difference between measurements collected with locations along fixed trajectories (described in the previous section) and the predicted values at those locations by the GP trained online, as well in terms of the average predictive standard deviation of the predictions. Note that, obviously, it



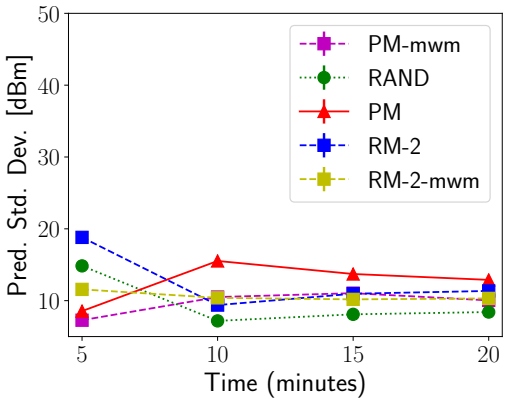
(a) Time consumed by Gaussian Process to Converge, varied with time.



(b) Distance Covered by Turtlebot 2, varied with time.



(c) RMSE calculated between Predictions from GP and Measured Values.



(d) Predicted standard deviation of the Gaussian Process.

Figure 4.5: Evaluating the Proposed (PM-MWM and RM-MWM) Approaches with Experiments Performed by TurtleBot 2 in Lab-Corridors.

is impossible to have a full ground truth as it would take too long for the robots to cover continuously the 4D space.

Figures 4.5 shows results for experiments performed on the map shown in Figure 4.4a. Our proposed PM-MWM approach spends less time updating the GP-based communication map—e.g., at 20 minute, about 1.2 seconds for PM-MWM, compared to 3 sec for PM. Moreover, robots consistently travel less distance at a given time slice (about 30%), preserving good quality in terms of RMSE and predictive standard deviation compared to the other strategies without communication prior—e.g., at 5 minute with PM-MWM the traveled distance is about 15 m with a predictive

standard deviation of 10 dBm, compared to 20 m with PM and 12 dBm as standard deviation. Also, note that with Random strategy robots cover longer distance but has higher predictive standard deviation. Figures 4.5c and 4.5d show that the predictive standard deviation of the proposed models is lower than remaining models while maintaining approximately the same RMSE value. Similar results are obtained in the other environment.

4.5 Conclusion

In this chapter, we presented a method for making the process of building communication maps more efficient. In particular, measurements to be integrated into a communication map and candidate locations to be chosen as goals by robots are filtered, by using communication models as prior. Communication models built starting from physical environment map, can provide information about distinctiveness of locations, by calculating first and second order derivatives. Experiments with real robots validate the accuracy of such models for the purpose of building communication maps. Moreover, the proposed approach showed an improvement over other methods in terms of traveled distance and computation time, while maintaining comparable RMSE and predictive variance.

Chapter 5

Conclusion

In this dissertation, we handled the problem of packet losses due to various network events, across both traditional wired networks and robotic wireless networks.

In Part 1, we addressed the problem of forwarding loops during convergence in response to various changes in the network such as link/router down/up. We presented an approach called FIFR++, which extends a previously proposed scheme called FIFR, for dealing with network contingencies such as link and router failures and micro bursts. We have shown that FIFR++ provides loop-free convergence during link/router down/up events in an IP backbone network. We have evaluated FIFR++ using 280 real and random topologies and our results confirm that the order of updates does not matter with FIFR++ for 17,336 out of 17,339 links and 11,707 of the 11,772 routers in those topologies, leading to fast convergence. Moreover, for the remaining outlier links and routers, FIFR++ uses progressive metric increments to ensure loop-free convergence. We have also demonstrated that FIFR++ can be employed to mitigate packet loss due to micro traffic bursts in IP backbone networks.

One major limitation of FIFR++ is that it assumes that at most one link or router down or up event is being propagated in the network. Otherwise, routers' notion of *bc* and *ac* eras can get mixed up leading to unexpected behavior and forwarding loops. Therefore, it is desirable to revise the approach to ensure graceful degradation when network needs to absorb multiple independent link state changes simultaneously, which is part of our future work. Another limitation is that the proposed approach works only for networks consisting of bidirectional links with symmetric link weights.

So, another avenue for future work is to extend the FIFR++ approach for networks with asymmetric link weights.

In Part 2, utilizing a Gaussian Process representation of the WiFi signal strength distribution, we designed and tested multi-robot online sensing strategies for mapping the quality of WiFi communication links between pairs of locations in an environment. We used communication models as prior to improve the overall performance and computational efficiency. Experiments using simulations and 4 Turtlebots show how distributed coordination schemes can effectively perform such a mapping task. The proposed schemes require robots to take fewer measurements and travel less distance, and yet get similar accuracy as methods that consider all the locations in the environment. Future work along these lines would be to extend the proposed approach to explicitly consider temporal variations and employ more complex communication models that account for multiple paths. Also, extrapolating this work to outdoors, under-water, and aerial environments opens several interesting research directions.

LIST OF PUBLICATIONS

1. Phani Krishna Penumarthi, Aaron Pecora, Sanjib Sur, Jason M O'Kane, and Srihari Nelakuditi, *Order of FIB Updates Rarely Matters with Failure Inference based Fast Reroute*, Submitted to IEEE Transaction on Network and Service Management, May 2020.

2. Alberto Quattrini Li, Phani Krishna Penumarthi, Jacopo Banfi, Nicola Basilico, Francesco Amigoni, Jason O'Kane, Ioannis Rekleitis, and Srihari Nelakuditi, *Multi-robot Online Sensing Strategies for the Construction of Communication Maps*, Springer Autonomous Robots, 44, pages, 299–319, 2020.

3. Phani Krishna Penumarthi, Aaron Pecora, Jason M O'Kane, and Srihari Nelakuditi, *Failure-Inference-Based Fast Reroute with Progressive Link Metric Increments*, in Proceedings of IEEE ICCCN 2018.

4. Phani Krishna Penumarthi, Alberto Quattrini Li, Jacopo Banfi, Nicola Basilico, Francesco Amigoni, Jason M O'Kane, Ioannis Rekleitis, and Srihari Nelakuditi, *Multi-robot Exploration for Building Communication Maps with Prior from Communication Models*, in Proceedings of IEEE MRS 2017.

5. Alberto Quattrini Li, Phani Krishna Penumarthi, Jacopo Banfi, Nicola Basilico, Francesco Amigoni, Ioannis Rekleitis, Jason M O'Kane, and Srihari Nelakuditi, *On Building Communication Maps for Reliable Multirobot Deployments*, in Proceedings of RSS Workshop on *Robot Communication in the Wild*, 2017.

6. Glenn Robertson, Nirupam Roy, Phani Krishna Penumarthi, Srihari Nelakuditi, and Jason M. O'Kane , *Loop-Free Convergence with Unordered Updates*, IEEE Transaction on Network and Service Management, Volume: 14, Issue:2, June 2017.

7. Zafar Ayyub Qazi, Phani Krishna Penumarthi, Vyas Sekar, Vijay Gopalakrishnan, Kaustubh Joshi, and Samir R Das, *KLEIN: A Minimally Disruptive Design for an Elastic Cellular Core*, in Proceedings of ACM SOSR 2016.

Bibliography

[1] A. Kvalbein, et al. Fast IP network recovery using multiple routing configurations. In *IEEE Infocom*, Apr. 2006.

[2] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). In *in Proceedings of the ACM SIGCOMM*, 2010.

[3] F. Amigoni, J. Banfi, N. Basilico, I. Rekleitis, and A. Quattrini Li. Online update of communication maps for exploring multirobot systems under connectivity constraints. In *Distributed Autonomous Robotic Systems*, pages 513–526. Springer, Oct. 2018. doi: https://doi.org/10.1007/978-3-030-05816-6_36.

[4] B. Arzani, S. Ciraci, L. Chamon, Y. Zhu, H. H. Liu, J. Padhye, B. T. Loo, and G. Outhred. 007: Democratically finding the cause of packet drops. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 419–435, 2018.

[5] P. Bahl and V. N. Padmanabhan. Radar: an in-building rf-based user location and tracking system. In *Proc. IEEE Infocom*, volume 2, pages 775–784 vol.2, 2000. doi: 10.1109/INFCOM.2000.832252.

[6] J. Banfi, A. Quattrini Li, N. Basilico, I. Rekleitis, and F. Amigoni. Asynchronous multirobot exploration under recurrent connectivity constraints. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 5491–5498, May 2016. doi: 10.1109/ICRA.2016.7487763.

[7] J. Banfi, A. Q. Li, I. Rekleitis, F. Amigoni, and N. Basilico. Strategies for coordinated multirobot exploration with recurrent connectivity constraints. *Autonomous Robots*, pages 1–20, July 2017.

[8] S. Bryant, S. Previdi, and M. Shand. A framework for IP and MPLS fast reroute using not-via addresses. RFC 6981, Aug. 2013.

[9] S. Bryant, C. Filsfils, S. Previdi, M. Shand, and N. So. Remote Loop-Free Alternate (LFA) Fast Reroute (FRR). RFC 7490, Apr. 2015.

[10] F. Clad, P. Merindol, J.-J. Pansiot, P. Francois, and O. Bonaventure. Graceful convergence in link-state ip networks: A lightweight algorithm ensuring minimal operational impact. *IEEE/ACM Trans. Netw.*, 22(1):300–312, Feb. 2014. ISSN 1063-6692.

[11] F. Clad, S. Vissicchio, P. Mérindol, P. Francois, and J.-J. Pansiot. Computing minimal update sequences for graceful router-wide reconfigurations. *IEEE/ACM Trans. Netw.*, 23(5):1373–1386, Oct. 2015. ISSN 1063-6692.

[12] S. Consortium. Sflow. `https://sflow.org/`, 2018.

[13] B. Cronkite-Ratcliff, A. Bergman, S. Vargaftik, M. Ravi, N. McKeown, I. Abraham, and I. Keslassy. Virtualized congestion control. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 230–243, 2016.

[14] E. Dong, X. Fu, M. Xu, and Y. Yang. Low-cost datacenter load balancing with multipath transport and top-of-rack switches. *IEEE Transactions on Parallel and Distributed Systems*, 31(10):2232–2247, 2020.

[15] M. Dunbabin and L. Marques. Robots for environmental monitoring: Significant advancements and applications. *IEEE Robotics Automation Magazine*, 19(1):24–39, 2012.

[16] N. Feamster, J. Rexford, and E. Zegura. The road to SDN. *Queue*, 11(12):20, 2013.

[17] B. Ferris, D. Fox, and N. D. Lawrence. WiFi-SLAM using Gaussian Process Latent Variable Models. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2480–2485, 2007. URL `http://ijcai.org/Proceedings/07/Papers/399.pdf`.

[18] J. Fink and V. Kumar. Online methods for radio signal mapping with mobile robots. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 1940–1945, 2010.

[19] P. Francois and O. Bonaventure. Avoiding Transient Loops during IGP Convergence in IP Networks. *ACM Transactions on Networking*, 15(6):1280–1292, Dec. 2007.

[20] P. Francois, M. Shand, and O. Bonaventure. Disruption free topology reconfiguration in OSPF networks. In *IEEE INFOCOM*, May 2007.

[21] A. Goldsmith. *Wireless Communications*. Cambridge University Press, 2005.

[22] J. Gregory, J. R. Fink, E. Stump, J. N. Twigg, J. G. Rogers, D. Baran, N. Fung, and S. Young. Application of multi-robot systems to disaster-relief scenarios with limited communication. In *Proc. International Conference on Field and Service Robotics (FSR)*, pages 639–653, 2015.

[23] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.

[24] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 14(14): 527, 2008.

[25] M. Hernandez, H.-B. Li, I. Dotlić, and R. Miura. Reference channel models for proposals evaluation to tg8. In *IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs)*, pages 1–20, 2012. URL `https://mentor.ieee.org/802.15/dcn/12/15-12-0459-00-0008-tg8-channel-models.doc`.

[26] K. Heurtefeux and F. Valois. Is rssi a good choice for localization in wireless sensor network? In *Proc. IEEE International Conference on Advanced Information Networking and Applications*, pages 732–739, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4651-3. doi: 10.1109/AINA.2012.19. URL `http://dx.doi.org/10.1109/AINA.2012.19`.

[27] G. A. Hollinger and S. Singh. Multirobot coordination with periodic connectivity: Theory and experiments. *IEEE Transactions on Robotics*, 28(4):967–973, 2012.

[28] M. A. Hsieh, A. Cowley, V. Kumar, and C. J. Taylor. Maintaining network connectivity and performance in robot teams: Research articles. *Journal of Field Robotics*, 25(1-2):111–131, Jan. 2008. ISSN 1556-4959. doi: 10.1002/rob.v25:1/2. URL `http://dx.doi.org/10.1002/rob.v25:1/2`.

[29] Jacopo Banfi and Alberto Quattrini Li and Nicola Basilico and Ioannis Rekleitis and Francesco Amigoni. Multirobot online construction of communication maps. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[30] S. Jin, S. Roy, W. Jiang, and T. R. Henderson. Efficient abstractions for implementing tgn channel and ofdm-mimo links in ns-3. In *Proceedings of the 2020 Workshop on ns-3*, pages 33–40, 2020.

[31] A. Kemppainen, J. Haverinen, I. Vallivaara, and J. RÃűning. Near-optimal slam exploration in gaussian processes. In *IEEE Conference on Multisensor Fusion and Integration*, pages 7–13, Sept 2010. doi: 10.1109/MFI.2010.5604467.

[32] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, October 2011.

[33] A. M. Ladd, K. E. Bekris, A. Rudys, L. E. Kavraki, and D. S. Wallach. Robotics-based location sensing using wireless ethernet. *Wireless Networks*, 11(1-2):189–204, Jan 2005.

[34] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica. Achieving convergence-free routing using failure-carrying packets. In *ACM SIGCOMM*, Aug. 2007.

[35] A. Li, X. Yang, and D. Wetherall. SafeGuard: Safe Forwarding during Routing Changes. In *ACM CoNEXT*, 2009.

[36] A. Q. Li, P. K. Penumarthi, J. Banfi, N. Basilico, J. M. O'Kane, I. Rekleitis, S. Nelakuditi, and F. Amigoni. Multi-robot online sensing strategies for the construction of communication maps. *Autonomous Robots*, pages 1–21, May 2019. doi: https://doi.org/10.1007/s10514-019-09862-3.

[37] T. Liu and A. E. Cerpa. Foresee (4C): Wireless link prediction using link features. In *Proc. Conference on Information Processing in Sensor Networks IPSN*, pages 294–305, 2011.

[38] S. S. Lor, R. Landa, and M. Rio. Packet re-cycling: Eliminating packet losses due to network failures. In *ACM HotNets*, Oct. 2010.

[39] M. Chiesa and I. Nikolaevskiy and S. MitroviÄĞ and A. Gurtov and A. Madry and M. Schapira and S. Shenker. On the resiliency of static forwarding tables. *IEEE/ACM Transactions on Networking*, 25(2):1133–1146, April 2017.

[40] A. Medina, A. Lakhina, I. Matta, and J. Byers. Brite: An approach to universal topology generation. In *Proceedings of MASCOTS 2001*, August 2001.

[41] P. Mirowski, T. K. Ho, and P. Whiting. Building optimal radio-frequency signal maps. In *Proc. International Conference on Pattern Recognition*, pages 978–983, Washington, DC, USA, 2014. IEEE Computer Society. ISBN 978-1-4799-5209-0. doi: 10.1109/ICPR.2014.178. URL `http://dx.doi.org/10.1109/ICPR.2014.178`.

[42] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats. Timely: Rtt-based congestion control for the datacenter. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, 2015.

[43] J. C. Mogul and P. Congdon. Hey, You Darned Counters!: Get off My ASIC! In *in Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, 2012.

[44] R. R. Murphy, S. Tadokoro, and A. Kleiner. Disaster robotics. In *Handbook of Robotics*, pages 1577–1604. Springer, 2016.

[45] S. Nelakuditi, S. Lee, Y. Yu, Z.-L. Zhang, and C.-N. Chuah. Fast Local Rerouting for Handling Transient Link Failures. *IEEE/ACM Trans. Networking*, 15(2):359–372, Apr. 2007.

[46] S. Nelakuditi, Z. Zhong, J. Wang, R. Keralapura, and C.-N. Chuah. Mitigating transient loops through interface-specific forwarding. *Computer Networks*, 52(3): 593–609, 2008.

[47] N.Spring, R. Mahajan, and D. Wetherall. Measureing ISP topologies with Rocketfuel. In *ACM SIGCOMM*, Aug. 2002.

[48] P. K. Penumarthi, A. Q. Li, J. Banfi, N. Basilico, F. Amigoni, I. Rekleitis, J. M. O'Kane, and S. Nelakuditi. Multirobot exploration for building communication maps with prior from communication models. In *International Symposium on*

Multi-Robot and Multi-Agent Systems, pages 90–96, Los Angeles, CA, USA, Dec. 2017. doi: https://doi.org/10.1109/mrs.2017.8250936.

[49] P. K. Penumarthi, A. Pecora, J. M. O'Kane, and S. Nelakuditi. Failure-inference-based fast reroute with progressive link metric increments. In *International Conference on Computer Communications and Networks*, 2018.

[50] A. Quattrini Li, R. Cipolleschi, M. Giusto, and F. Amigoni. A semantically-informed multirobot system for exploration of relevant areas in search and rescue settings. *Autonomous Robots*, 40(4):581–597, 2016.

[51] A. Quattrini Li, P. K. Penumarthi, J. Banfi, N. Basilico, F. Amigoni, I. Rekleitis, J. O'Kane, and S. Nelakuditi. On building communication maps for reliable multirobot deployments. In *RSS2017 (Robotics Science and Systems) Workshop on "Robot Communication in the Wild"*, Boston, MA, USA, 2017.

[52] T. S. Rappaport. *Wireless communications: principles and practice*. Prentice Hall PTR New Jersey, 1996.

[53] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[54] G. Retvari, J. Tapolcai, G. Enyedi, and A. Csaszar. Ip fast reroute: Loop free alternates revisited. In *INFOCOM*, pages 2948–2956, April 2011. doi: 10.1109/INFCOM.2011.5935135.

[55] G. Robertson, N. Roy, P. K. Penumarthi, S. Nelakuditi, and J. M. O'Kane. Loop-free convergence with unordered updates. *IEEE Transactions on Network and Service Management*, 14(2):373–385, 2017.

[56] A. Roy, H. Zeng, J. Bagga, and A. C. Snoeren. Passive realtime datacenter fault detection and localization. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 595–612, 2017.

[57] P. M. Scholl, S. Kohlbrecher, V. Sachidananda, and K. V. Laerhoven. Fast indoor radio-map building for rssi-based localization systems. In *Proc. International Conference on Networked Sensing (INSS)*, pages 1–2, 2012.

[58] G. Seguin. Multi-core parallelism for ns-3 simulator. *INRIA Sophia-Antipolis, Tech. Rep*, 106:110, 2009.

[59] P. Tammana, R. Agarwal, and M. Lee. Distributed network monitoring and debugging with switchpointer. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 453–456, 2018.

[60] G. Tuna, K. Gulez, and V. C. Gungor. The effects of exploration strategies and communication models on the performance of cooperative exploration. *Ad Hoc Networks*, 11(7):1931–1941, 2013.

[61] J. N. Twigg, J. R. Fink, L. Y. Paul, and B. M. Sadler. RSS gradient-assisted frontier exploration and radio source localization. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 889–895, 2012.

[62] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford. Central control over distributed routing. In *ACM SIGCOMM*, Aug. 2015.

[63] J. Wang and S. Nelakuditi. IP Fast Reroute with Failure Inferencing. In *INM*, Aug. 2007.

[64] B. Yamauchi. Frontier-based exploration using multiple robots. In *Proc. International Conference on Autonomous Agents*, pages 47–53, 1998.

[65] K. Zarifis, R. Miao, M. Calder, E. Katz-Bassett, M. Yu, and J. Padhye. Dibs: Just-in-time congestion mitigation for data centers. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys 2014, 2014.

[66] Q. Zhang, V. Liu, H. Zeng, and A. Krishnamurthy. High-resolution measurement of data center microbursts. In *Proceedings of the 2017 Internet Measurement Conference*, IMC '17, pages 78–85, 2017.

[67] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Y. Zhao, and H. Zheng. Packet-level telemetry in large datacenter networks. In *in Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015.

[68] S. Zvanovec, P. Pechac, and M. Klepal. Wireless LAN networks design: Site survey or propagation modeling? *Radio Engineering*, 12(4):42–49, 2003.