Permutation Based Routing for Increased Robustness in IP Networks

by

Hung Quoc Vo

Doctoral Dissertation submitted to
the Faculty of Mathematics and Natural Sciences
at the University of Oslo
in partial fulfilment of the requirements for
the degree of Philosophiae Doctor

October 2013

Abstract

The Internet has emerged as a preferred transport medium for many services of critical importance for business and individuals. In particular, an increasing number of time-critical services such as trading systems, remote monitoring and control systems, online gaming, telephony and video conferencing place strong demands on network on timely recovery from failures. For these applications, even short outages in the order of a few seconds will cause severe problems or impede the user experience. This has fostered the development of a number of proposals for more robust routing protocols, which are able to continue packet forwarding immediately after a component failure, without the need for a protocol re-convergence. Such solutions add robustness either by changing the routing protocol so that it installs more than one next-hop towards a destination in the forwarding table or by adding back-up next-hops a posteriori to the forwarding entries found by a standard shortest path routing protocol. Unfortunately, few of these solutions have seen widespread deployment, due to added complexity or incompatibility with existing routing protocols.

In this work, we argue that a new routing method is feasibly adopted if it satisfies serval important properties. First, it is designed in hop-by-hop manner, which is fundamental in the IP networks. Second, the method does not require additional information to be placed in the IP headers which normally have no unused bits. Third, it must be loop-free in all fault situations. In fact, routing loops significantly degrade the network performance, and are considered to be more severe than an interruption caused by the failures. Finally, it does not introduce a new protocol for information exchange between routers. The method following those properties can be incremental deployed like ECMP and LFA in the IP networks.

We propose a new multi-path routing method that strictly follows the listed four properties for the adoption and investigate it in different versions of the IP networks. First, we start by considering networks with the most basic functionality. This means that there is one routing table in each router, and that the table can support more than one output ports defined per prefix or address. Second, we study the case where routers have functionality that allow them to prohibit U-turns on back-up paths. That is if there is a

forwarding option for an incoming packet that makes the packet go back out on the same link that it came in on, then this option is prohibited. And the third one is the IP networks with interface-specific forwarding. In this type of IP networks, the routers forwards packets based on their incoming interfaces and destination addresses. We evaluate our routing methods over inferred ISP networks with realistic link weight setting. The results show that our method can significantly improve the routing robustness over LFA while still retaining a low computational complexity that is needed for a distributed routing system. This work also seeks an answer for the non-trivial question to decide how traffic should be routed over the available paths to achieve the optimal network performance. Based on our multi-path routing solution and existing load-balancing mechanisms, we draw a strategic mechanism of using multiple paths to balance the load for least network congestion in either normal operation or a failure situation.

Acknowledgements

It takes me almost four years to complete this work, the dissertation for the Doctor of Philosophy at the Department of Informatics, University of Oslo. First of all, I would like to thank the Simula Research Laboratory for providing me with financial support and excellent working conditions.

This dissertation is the result of both my patience and the supervisions of my two sophisticated advisers, Dr. Amund Kvalbein and Prof. Olav Lysne. Foremost, I would like to thank Amund and Olav for their contributions in my final outcome. I also would like to thank Prof. Paal Engelstad and Dr. Audun Hansen for selecting me out of more than three hundred Ph.D. applicants, opening the door for my Ph.D. career. Particularly, I would like to express my thank to Dr. Ahmed Ehmokashfi who very often gave me inspiring advices to overcome difficulties in the Ph.D. life.

Working towards Ph.D. has never been easy, being often in uncertainty, and sometimes frustrating, but all my colleagues are fun and understanding. I would like to thank Saif Shams, Kristian Evensen, Dominik Kaspar, Tomas Kupka, Jie Xiang, and Hai Ngoc Pham for their helps and instructions during the course of working at Simula.

Finally, a special thank goes to my wife Ngan Mai who frequently pulls me out of all worries and cooks for me good dinners.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The last decade has witnessed the adoption of the Internet as the preferred transport medium for services of critical importance for business and individuals. However, the Internet faces challenges. On one hand, the traditional Internet routing protocols such as OSPF and IS-IS only offer a single path for each source-destination (S-D) pair, which is sensitive to network failures since there is no back-up path to take over the traffic on the shortest path when it fails. Although OSPF and IS-IS own the self-recovery ability, meaning that the network can heal the fault and come back to normal operation by itself, the recovery process taking seconds can not be tolerated by the modern Internet applications. On the other hand, measurements indicate that network failures are frequent, occurring daily with various reasons and time scales and most of them are unfortunately unplanned [50]. Those issues have fostered the development of a more robust routing algorithm for the Internet routing protocols that can provide multiple paths for each S-D pair. The main focus of our work is to propose such robust routing algorithms.

1.1 Context of this work

The Internet is constituted by a large number of interconnected networks, which are generally named Autonomous Systems (ASes). Each AS is also a network of a few to hundreds of routers that share a common administrative control. In such a structural administration, routing protocols in the Internet are classified in two main classes: Interior Gateway Protocols (IGPs) and Exterior Gateway Protocols (EGPs), which correspond to finding paths within an AS and among ASes, respectively. The routing protocols for a single AS are also known as the intra-domain routing protocols.

IGPs exchange topology information among routers within an AS, from which paths are computed for all S-D pairs. IGPs are divided into two categories according to their dissemination methods of routing information: distance-vector routing protocols and link-

state routing protocols. In the former, a router uses Bellman-Ford algorithm to compute paths and the needed information for such a computation is the distances from its neighboring routers to the destinations. It means that the dissemination takes place only among adjacent routers. In contrast, the link-state routing protocols require states of all links be flooded all over the networks so that every router has a consistent view of the entire network for the path computation purpose. Despite suffering longer time to process the routing information, the link-state routing protocols result in much faster network convergence compared to that of the distance-vector protocols, and consequently have been widely deployed in the intra-domain networks. Two standard link-state routing protocols are OSPF and IS-IS that are currently used in the Internet.

The central component of the link-state routing protocols is Shortest Path First (SPF) algorithm, also known as Dijkstra's algorithm, which computes a *single* shortest path for each S-D pair. Being widely deployed in the Internet routing protocols for its simplicity, the SPF soon exposes two severe limitations. First, a single shortest path can not balance the traffic in case the currently used path is over-loaded. Second, any failure will break the shortest path and disrupt the traffic forwarding, triggering global flooding of routing information in the network, during the course of which traffic forwarding is interrupted. Much effort has been made to lower the convergence time at different steps of the routing protocols, i.e. minimizing the failure detection time, tweaking the protocol timer parameters [1] [26], restricting the flooding and the path re-calculation to only the affected components [14] [89], but the convergence time is still relatively large due to the nature of distributed routing systems. The multi-path routing protocols are therefore apparently needed to increase network robustness.

1.1.1 Network robustness

A routing protocol results in a routing which is the assignment of a set of next-hops for each destination at each source router. Furthermore, a multi-path routing offers multiple paths (next-hops) for each S-D pair to a destination, and has obvious advantages with respect to increased network robustness. First, it provides several paths for balancing the load, which can be used to absorb short-lived spikes in traffic demands and thus avoid congestion. Second, it provides fast re-routing with alternative routes that are readily available after a link failure. These two properties are related: if a path is opted as a back-up path, it will not be used to balance the load in normal operation.

Internet Engineering Task Force (IETF) defines an IP Fast ReRoute (IPFRR) framework [75] for the fast re-routing in context of the link-state routing protocols that seeks to resume packet forwarding within 50ms under network failures. IPFRR consists of two main principles: back-up path establishing and local re-routing. Specifically, every router

first computes alternate paths for a given primary one right after the link-state IGP's convergence. As a failure occurs on the primary path, the router adjacent to the failure will locally re-route the affected traffic over the back-up paths without the need of a network re-convergence. If the failure lasts sufficiently long, the global flooding of IGP protocols then is triggered to calculate new primary paths while re-routed traffic is still safe on the back-up paths. As a new primary path is available, the re-routed traffic is switched back to it.

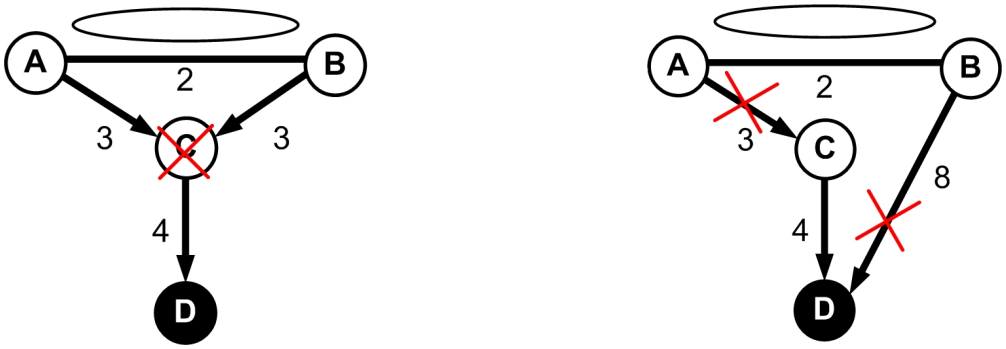
(a) B is a link-protecting alternate for link AC (b) D is a node-protecting alternate for link AC

Figure 1.1: Two types of alternates: link-protecting and node-protecting alternates

In a routing towards a destination, a router has the IPFRR capability when it installs *at least* two next-hops. We consider the case in which a router has only two next-hops. In this case, one next-hop is the primary next-hop and another one is the back-up next-hop for the primary. For example, router A in Figure 1.1(a) has two next-hops, B and C. If B is the primary, then C is the back-up or vice versa. As a result, router A is safe when link AB *or* link AC fails. Both alternates, however, do not work when the failure occurs at router C. Therefore, B and C are further called link-protecting alternates. Router A, on the other hand, can have a valid route to the destination under the failure of router C if one of its next-hop is D as shown in Figure 1.1(b). We call D the node-protecting alternate for primary link AC. Since an alternate like D does not always exist in practical topologies, a router then should admit a link-protecting alternate instead. For that reason, in this work we generally refer to a router with at least two next-hops as a protected router and to protection coverage as fraction of protected routers in the network.

It is crucial that the determination of alternates for a given primary path must guarantee that the traffic re-routed through those alternates will reach the destination. Otherwise, the traffic is caught in a loop, seriously degrading network performance. The

alternates that do not cause loops in a fault situation are called loop-free alternates. However, the terminology in many solutions is only accurate with specific failure scenarios. For example, a proposal being loop-free under any single link failure may not be loop-free if the number of link faults is greater than one. For example, we use the loop-free criterion [3] against a single link fault to find the alternates for routers A and B on the existing shortest path routing in Figure 1.2(a). As the two following inequalities are satisfied: $dist(A) < dist(B) + dist(A, B)$ and $dist(B) < dist(A) + dist(B, A)$ where $dist(X)$ is the shortest path distance from router X to the destination and $dist(A, B)$ is the link weight of link AB, the A is allowed to take B as its alternate when link AC fails and A is also an alternate of B when link BC fails. However, if router C fails, a loop will occur between router A and router B. Clearly, the failure of router C is more extensive than a single link failure in the design. Likewise, the failures of two independent links, AC and BD in Figure 1.2(b) also cause a loop between A and B when the loop-free criterion is used to compute the alternates for A and B. Since the failure scenarios are diverse and often more extensive than the desired ones, it is important to design a routing method that can provide loop-freeness under arbitrary failures.



(a) B is a link-protecting alternate for link AC (b) D is a node-protecting alternate for link AC

Figure 1.2: Forwarding loops under extensive failures

1.1.2 Problems and scope we address

Currently deployed multi-path routing algorithms in the link-state IGPs are Equal-Cost Multi-Path (ECMP) and Loop Free Alternates (LFA) [3] [25], both of which are extensions of the SPF algorithm. ECMP computes multiple equal-cost shortest paths for each S-D pair while LFA does not limit its path computation on the shortest paths. As a result, LFA can produce higher path diversity than that of ECMP, and therefore is more robust

against network failures. LFA comes in two main versions. The first version uses only loop-free criterion to provide alternates for a primary next-hop. This version is only loop-free for single link fault situations and suffers from routing loops when a node failure or a multiple link failure occurs as we illustrated in the above example. The second version uses both loop-free criterion and downstream path condition to prevent loops against any fault situation. The downstream path condition will let some of links unused since utilizing those links in the routing possibly causes routing loops under failures. Consequently, the latter LFA limits significantly the protection coverage.

The main reason behind the limitation of the LFA lies in its path computation which depends on a link weight setting. Obviously, we can tune the link weights in order to eliminate the adverse effect of the downstream path condition, but the approach, typically accompanying with a local search heuristic, will result in high complexity even in a small network. Therefore, such a solution is not feasible to be deployed in a distributed routing system. In addition, the design of LFA only aims to find as many alternates as possible for an S-D pair, often leading to a situation that one primary next-hop can have many alternates while others are left with only one. Certainly, the tuning link weight approach would be more complicated if it has to take the next-hop distribution into account.

Recent solutions are proposed to overcome the limitation of LFA, but unfortunately few of solutions have seen widespread deployment. We observe that those solutions do not follow at least one of four important properties that make LFA adoptable. First, the forwarding mechanism of the solution should be hop-by-hop, which is optimized for fault-free case in the IP networks. Second, the solution does not require information associated with network faults be included in the packet headers. Since unused bits in the IP headers are an extremely scarce commodity, such a solution is hard to deploy. Third, it does not suffer from routing loops, even when there are multiple faulty components in the network. And finally, it does not introduce a new routing protocol for exchanging routing information among routers. In other words, it can work with existing link-state routing protocols such as OSPF or IS-IS. A solution with those properties can incrementally deployed in the IP networks since it only uses the features which are present in the current routers.

In this work, we propose a novel multi-path routing method that can significantly increase the routing robustness for the IP networks. Our method strictly follows the four listed properties since we argue that those properties are crucial for adopting a new routing method over the existing IP network infrastructure. In our method, we do not use the loop-free criteria given in the LFA specification due to their tight relation with finding an optimized link weight setting which is known as a complicated problem. We suggest, instead, a new single loop-free criterion that can ensure loop-freeness for a

routing under any type of failures. Our proposed criterion is independent to a specific link weight setting, but can work compatibly with the routings calculated from any link weight setting. That is our routing method can be used to improve the coverage of alternates for an existing shortest path routing. Additionally, our method can solve the uneven next-hop distribution that occurs with the LFA, meaning that a router is prevented from having many alternates while its neighboring routers starve of alternates.

1.2 Contributions

This work presents permutation routing as a novel and flexible approach for calculating multiple loop-free next-hops for each source router in the networks with the traditional hop-by-hop forwarding. Permutation routing is based on the observation that routing in any network consists of using a set of resources (routers or links) in *sequence* (or called *permutation*). A routing strategy can therefore be expressed as a permutation of the routers (or links) that are involved in traffic forwarding to a destination. In addition, a routing resulted from a permutation routing will be loop-free as long as traffic can only be forwarded in one direction with respect to the router ordering in that permutation. The permutation routing offers a simple but powerful abstraction that can be used to implement many different routing objectives, by adjusting constraints that govern the creation of the sequence. In this work, we first propose a generic algorithm for computing a permutation routing for a certain routing objective. The algorithm framework then is used to create the permutation routings for the specific routing objectives in three different variants of the traditional IP networks.

1.2.1 Permutation routing framework

We start by introducing the permutation routing concept and proposing its mathematical model. The main idea of the permutation routing is its relationship with a loop-free routing graph, typically called a directed acyclic graph. Specifically, any loop-free routing can be represented by a permutation routing. As a result, we can construct a desired routing by finding its permutation routing whose construction algorithm is expected to be simple and suitable for distributed routing systems.

The problem of finding a permutation routing for a desired loop-free routing is to construct a sequence of routers whose ordering leads to produce that routing with a forwarding rule associated with the permutation routing. Intuitively, the brute force search can be used to find such a sequence. However, the computational complexity for that exhaustive search is high when the number of routers in the network is greater than

15. For that reason, we give a couple of principles to limit the search space so that the permutation routing method can be suitable for the distributed routing systems.

The permutation routing concept is flexible since it does not tie with a special forwarding strategy, e.g. a forwarding strategy that depends non-standard added bits in the IP headers. Instead, it can be used for any loop-free hop-by-hop destination-based IP forwarding. Consequently, the construction framework is helpful for creating different loop-free routings for various versions of the IP networks.

1.2.2 Next-hop permutation routing

Our first permutation routing method is designed for the traditional IP networks with the most basic functionality. Accordingly, there is one routing table in each router, and the table can support multi-path forwarding. In other words, there can be one or more output interfaces defined per prefix or IP address. We require that a routing must be loop-free, and in the given setting this means no packet can visit the same router twice on its path from a source router to a destination router. Hence a given routing corresponds to a directed acyclic graph rooted at each destination node consisting of the links and nodes involved in packet forwarding.

Based on such a premise, we propose two routing objectives, Next-Hop Optimal Routing and Shortest Path compatible Next-Hop Optimal Routing, both of which aim to maximize the number of routers with at least two next-hops towards a destination. The difference between the two is that the latter guarantees that the shortest path is always included in the resulting loop-free routing. The routing objectives then are interpreted into the routing constraint functions at source routers so that they are comprehensible to the algorithm framework. The outputs of the construction algorithm are two permutation routings whose corresponding routings approximate the objectives.

1.2.3 Next-hop permutation routing with Disallowed U-turns

The second permutation routing method studies the case where routers have functionality that allows them to prohibit U-turns on back-up paths. That is if there is a forwarding option for an incoming packet that makes the packet go back out on the same link that it came in on, then this option is prohibited. At the time of writing, it is unclear to what extent this feature is widespread in current networked equipments. This feature is described in Internet standards [74], and we can demonstrate that this feature significantly increases the potential for protection coverage compared to the case we described in the previous contribution.

We re-use the above two routing objectives. Yet their mathematical interpretations

are different from those of the previous contribution and the permutation routing definition is also extended to incorporate the added functionality. The construction algorithm framework then is applied to generate the corresponding permutation routings whose corresponding routings are asymptotic to the routing objectives.

1.2.4 Interface based permutation routing

The case we consider in this contribution is inspired by the fact that many present day routers keep separate copies of the routing tables in their line-cards [11]. The rationale behind this is that table lookup is parallelized so that a central routing table does not become a bottleneck for router performance. This can, however, be exploited by having different routing tables in each line-card [58]. In our scenario as described in the algorithm framework, this will mean that we consider *interfaces* to be the network resources that are to be used in an acyclic manner. In the interface based routing, we will let a router treat a packet differently depending on which interface it entered the router. A router may therefore see the same packet several times, but each time it will enter the router on a different interface.

We first use the line graph transformation technique [34] to turn the topology into a line graph whose vertices are nodes of the topology and whose edges are induced from the connectivity the topology. If the algorithm framework takes the line graph as input, presuming that a certain formalized routing objective in terms of routing constraint functions is given, the output is a permutation routing whose elements are links (interfaces) of the original topology. This contribution can be used to generate routings that maximize link-protecting coverage.

The multi-path routing algorithm offers multiple next-hops for each source router towards a destination. Those next-hops can be used for load-balancing. Our next contribution is to find an efficient load-balancing strategy for the traditional IP networks in either normal operation or in a fault situation.

1.2.5 On load-balancing in multi-path routing protocols

The efficacy of a load-balancing mechanism lies in its ability to engineer given traffic demands across the network with minimum risk of link congestion. In this contribution, we first propose a routing, called Augmented Base Routing, which can be constructed with the permutation routing method. The ABR includes a base routing, and augmented the base routing with additional loop-free forwarding options. The base routing is a routing that is optimized for a given traffic matrix or manually configured by the network operator for a specific purpose. We show that the ABR can offer maximum network

performance in either normal operation or in a fault situation when it is used in such a manner that only the based routing is allowed if there is no network fault. But in a fault situation the affected node should balance the re-routed traffic over *all* available next-hops. Fortunately, the ABR can provide high fraction of source routers with multiple next-hops for that re-routing purpose.

Chapter 2

State of the Art

This chapter will give the state of the art of the multi-path routing protocols, which lay the fundamental means for fast re-routing, multi-path routing, and load-balancing in the intra-domain networks. First of all, the multi-path routing protocols can result in multiple paths between a certain pair of source destination (S-D). That means the source router in the pair can install multiple next-hops towards the destination. Then the source is offered various strategies in utilizing those available next-hops to improve network performance. The most popular strategy for a router is to save some of its next-hops for fast re-routing when its currently used next-hops are suddenly not in service. Since the router will activate the back-up next-hops, to which affected traffic is diverted, as soon as it detects the failure, the recovery time could be small enough not to interrupt the Internet applications. Another important strategy is the multi-path routing where the router simultaneously employs at least two next-hops for traffic forwarding. The routing over multiple paths can help prevent over-loaded links when the traffic increases significantly due to network failures or temporary demands caused during noteworthy events. In addition, the multi-path routing is typically associated with a load-balancing method which determines an amount of traffic on each of the routing path. The load-balancing method is sometimes referred to as the traffic split method. The most common split is equal-split that is currently implemented in OSPF and IS-IS.

There are numerous intra-domain multi-path routing protocols proposed in the past decades, all of which aim to expand the fast re-routing coverage for failures, by that way increasing number of S-D pairs with multi-path capability. Many solutions provide up to 100% protection coverage for single link faults, and either single link faults or single node faults, and more extensive failure scenarios. However, in practice very few solutions are adopted since most of them are required to change the traditional IP forwarding paradigm that is widely used in the IP networks. Particularly, ECMP and LFA are only two multi-path routing algorithms that are currently deployed in routers of a couple of

vendors. Unfortunately, the two give very poor protection coverage, and therefore multi-path routing capability; in most case lower than 50% of S-D pairs is installed with at least two next-hops [27].

One alternative approach, on which many multi-path solutions are built for the intra-domain networks, is Multiple Protocol Label Switching (MPLS) [68]. Different from the traditional IP networks where the forwarding is based on the hop-by-hop technique, MPLS establishes end-to-end tunnels for S-D pairs, and can flexibly govern the traffic over those tunnels. Therefore, MPLS is referred to as the connection-oriented switching approach. Despite having more important properties for traffic engineering, MPLS requires much investment for migrating to from the existing network infrastructure. Since this work is devoted to find a better multi-path routing protocols for the traditional IP networks, we will emphasize on related solutions in our work.

2.1 MPLS networks

An MPLS network is an interconnection of a number of Label Switching Routers (LSRs). The LSRs are located at the ingress point and egress point of an MPLS network and are called Ingress LSRs and Egress LSRs, respectively. In the MPLS networks, Label Switching Paths (LSPs) are pre-established between each pair of Ingress and Egress LSRs. During operation, Ingress LSR adds a fixed length label to an incoming packet, and forwards it along the LSP to the Egress LSR where the label in the packet will be removed before being forwarded out of the MPLS domain. The forwarding of the packet in the LSP is based on such labels. The MPLS network is standardized by IEFT and has shown a greater flexibility in terms of load-balancing and fault tolerance compared to the traditional IP networks.

2.1.1 Load-balancing

Traffic Engineering in MPLS is referred to as MPLS-TE which deals with optimization of traffic distribution on LSPs. The MPLS-TE can be achieved by establishing multiple LSPs with guaranteed bandwidth and balancing the traffic demands among those LSPs to minimize congestion risk. The MPLS-TE has been studied extensively in both offline and online settings.

Offline TE

Offline TE requires a demand traffic matrix (TM) as input. The important property of load-balancing in MPLS-TE is the ability to split incoming traffic arbitrarily among

multiple LSPs to achieve optimal traffic distribution. Such split ratios can be yielded by solving instances of the multi-commodity flow problem [53]. However, the optimal result typically requires numerous LSPs, and therefore is only suitable for a small-sized network. A number of heuristics are proposed to provide more scalable solutions by limiting the number of LSPs in scope [29] [71] [92] [91].

Online TE

Online TE deals with the dynamic nature of network load and therefore does not need full knowledge of the TM. Online TE comes in two approaches: dynamically assign traffic over multiple LSPs according to network-wide load status or dynamically set up LSPs to balance potential load. The former requires that all LSPs be static and pre-established. In addition, a probing mechanism is needed to capture the current distribution of the network load. Two typical examples in this approach are MATE [20] and TeXCP [36]. In the second approach, LSPs only are calculated upon request, and their constructions take into account critical links, whose definitions vary in the different proposals, to minimize interference among existing LSPs and future ones, e.g. DORA [8] and MIRA [37]. Since the MPLS networks can accommodate various QoS requirements, there is then possibility for higher priority LSPs to preempt other LSPs. The LSPs with lower priority can be completely impeded as proposed in [62] or reduced in their transmission rates as in [7] [17].

2.1.2 Fast restoration

The IETF defined a framework [77], called MPLS-based Recovery, for MPLS-TE so as to reduce the forwarding interruption under failures to tens of milliseconds. In this framework, an LSP carrying traffic in normal operation is the primary LSP and an LSP taking over the traffic of the primary path due to its failure is the back-up LSP. Moreover, the framework defines Path Switch LSR or Point of Local Repair (PLR) and Path Merge LSR or Merge Point (MP) as points of repair. Specifically, under failures the PLR will perform switching from a primary LSP to its back-up LSP and the MP will perform merging the re-routed traffic on a back-up LSP back to its primary LSP. The proposals for fast restoration based on the framework can be classified into three categories according to the desired locations of PLRs and MPs: global recovery, local recovery and unrestricted recovery.

Global recovery

In global recovery, each primary LSP has a single back-up LSP. Beside the common end nodes, the back-up LSP does not share any link or node with the primary LSP so

that the back-up path can protect against all node or link failures along the primary path. In Figure 2.1, LSP $A-I-J-K-L-F$ is the back-up LSP for primary LSP $A-B-C-D-E-F$. Both LSPs share two LSRs, Ingress LSR A and Egress LSR F, which work as the PLR and the MP, respectively.

With the global recovery scheme, upon detection of a failure the immediate LSR must signal back to the PLR which is located at the Ingress LSR. In the example, if link CD fails, node C will send the failure notification to downstream node A. The papers in this category, e.g. [35] [49] [78] [85], have modeled the problem of finding back-up paths as instances of Integer Linear Program (ILP) and have proposed heuristics to achieve near-optimal solutions for bandwidth-guaranteed back-up paths. However, the signaling of failure notification could take much time if the meditate LSR is far from the PLR, many papers have proposed local recovery to yield a shorter recovery time.

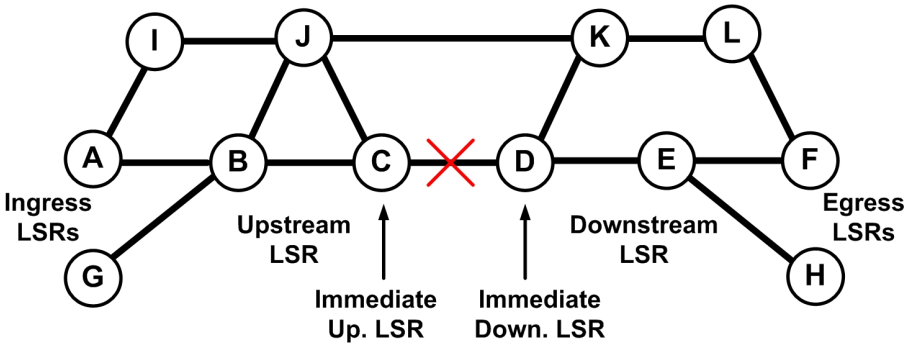


Figure 2.1: An MPLS network with its different components

Local recovery

The local recovery approach suggests that each back-up LSP is needed to protect a component (link or node) of the primary LSP. Correspondingly, the PLR and MP of the back-up LSP are located at the immediate upstream node and a downstream node on the primary LSP of the protected component. In Figure 2.1, $C-J-K-D$ or $C-J-K-L-F$ can be the back-up LSP that protects link CD belonging to primary LSP $A-B-C-D-E-F$. In this case, C works as a PLR and D or F is a MP for the above back-up scenarios, respectively. The recovery time in this in category is typically shorter than that of the global recovery since the node that will process the failure notification and conduct path switching lies more closely to the failure than the Ingress LSR. However, there needs many back-up LSPs for a given primary LSP. For example, we need $(n-1)$ back-up paths to

protect a primary path with n hops. Hence, facility local recovery is proposed to protect each single component in shared path context. In other words, each back-up LSP is associated with *all* primary LSPs that use the protected link or node. Consequently, the PLR and MP of the back-up should be located at the immediate upstream node and the immediate downstream node of the protected component. For example, two primary LSPs $A-B-C-D-E-F$ and $G-B-C-D-E-H$ in Figure 2.1 both can employ LSP $C-J-K-D$ as their back-up LSP to protect their common link CD. Now C is the immediate upstream node of component CD and is assigned to PLR role. Meanwhile, D is the immediate downstream node of link CD and will work as a MP. The finding back-up path problem with the facility local recovery can also be modeled as instances of ILP and can be solved with heuristics such as [4], [12] and p-cycles [82]. Since only one back-up is employed for multiple primary LSPs, the necessary number of back-up LSPs will be significantly less than that of the general local recovery. Furthermore, it is possible to use more than one back-up for multiple primaries to balance the re-routed traffic load [2].

Unrestricted recovery

Proposals in this category [6] [30] [48] [95] do not explicitly limit the location of the PLRs and the MPs of the back-up LSPs and are typically devoted to shared path protection like the facility local recovery schemes. In Figure 2.1, we could have more than ten possible choices for a back-up LSP with regard to link CD shared by two primary LSPs, $A-B-C-D-E-F$ and $G-B-C-D-E-H$. Those back-ups include the cases where the PLR and MP are assigned to Ingress LSR A and Egress LSR F, respectively, or to LSRs C and D, respectively. The unrestricted recovery is flexible since it significantly increases the chance to find sufficient spare-bandwidth paths, but it might introduce much higher overhead compared to the local recovery schemes.

2.2 IP-based networks

Unlike MPLS-TE, the traditional IP networks are connectionless, and consequently do not support explicit routing paths and arbitrary split over multiple paths for each traffic demand. However, the hop-by-hop forwarding is essential to MPLS Label Distribution Protocol (MPLS LDP) which is a process of assigning and releasing labels in the IP networks that support MPLS applications, e.g. MPLS VPNs.

The traditional IP networks use link-state routing protocols to establish shortest paths for all S-D pairs. The shortest path computation requires that each link of the network be assigned a link weight and the length of a path is calculated with respect to these

weights. However, since the shortest path routings, or its extension to allow equal-cost multi-path (ECMP), only offer a single path for most of S-D pairs, it significantly restricts the network robustness. Correspondingly, when occurring in the network, the failure will partition the shortest path tree and interrupt the traffic forwarding on the affected shortest path branches. Right after the failure is detected, the link-state routing protocol is triggered at the routers to disseminate their link states to all over the network. When the dissemination of routing information is finished, the routers will re-compute a new shortest path tree, which excludes the failure, and resume the traffic forwarding. Since the whole re-convergence process is known to be time consuming owing to the distributed manner of the routing systems, it can not be tolerated by many modern Internet applications.

A plethora of solutions are proposed to improve the robustness for IP networks, primarily on computing more than one path to a destination for each S-D pair. With given multiple paths, multi-path routing and subsequently load-balancing or fast re-routing are employed to provide a good traffic distribution or continuity of traffic forwarding under failures, respectively. We first review existing load-balancing methods in the IP networks.

2.2.1 Load-balancing

Multi-path routing protocols compute multiple loop-free next-hops for each source router towards a destination. If the source installs more than one primary next-hops in its routing table, it then conducts multi-path routing to the destination. Following the installation, the source also has to set the split ratio for each of the next-hop in order to balance the load to protect the outgoing links from being over-loaded. Load-balancing methods in the IP networks can be static or adaptive. Static methods determine traffic split ratios merely on topology information. The static ratios will not respond to current network load and may only change when routing tables are updated. On the other hand, an adaptive method dynamically selects next-hops with the knowledge of network states. The adaptive method typically needs to be implemented on the top of a routing protocol.

Static load-balancing

In the static setting, the TM is known in advance. The paths and their traffic split ratios are calculated to optimize a network objective with respect to the given the TM. The first TE solution in terms of the load-balancing for the IP networks is Optimized OSPF [24]. The basic idea of this work is to use the local search heuristic for computing an optimized set of link weights, given the network topology and the TM, such that the ECMP can produce a near-optimal average network cost. However, the authors also prove that finding an optimal link weight setting for ECMP is NP-hard and furthermore construct a topology

pattern in which Optimized OSPF is substantially worse than an optimal routing. For that reason, several proposals, e.g EIGRP [5], DEFT [96], and PEFT [97], suggest that the routings should admit additional paths beside the shortest ones and the traffic should be more flexibly split among the multiple paths. In EIGRP, which is a Cisco proprietary protocol, the traffic is inversely proportional to the path's cost, meaning that the shorter path is likely offered more traffic. Similarly, DEFT and PEFT prefer uneven split among multiple unequal-cost paths, but both employ an exponential split with respect to the path cost difference. DEFT is provably always better than both Optimized OSPF and EIGPR [38]. The authors of DEFT later propose PEFT that gives a stronger guarantee for small performance gap with the optimal routing. PEFT assigns the traffic based on paths, and is different from DEFT which is designed for link-based splitting. This modification allows PEFT to offer arbitrary split on paths like the optimal flow allocation. The performance analysis of static load-balancing methods of packet-based splitting and flow-based splitting can be found in [60].

Adaptive load-balancing

Since an accurate TM is hardly available in practice, the static load-balancing methods could not timely react to short-term dynamics of the Internet traffic such as spikes that last only a few seconds or to shifting of the traffic load due to failures. In the dynamic method, routers must probe the load information of the network and adaptively assign traffic over given paths to minimize traffic congestion. Homeostasis [38] is an example of this load-balancing category and is designed to use only the local load information. Therefore, the method does not introduce signaling overhead. In its operation, Homeostasis picks only the *single* available shortest path whose traffic load is below a threshold. Alternately, the work [52] employs multiple paths simultaneously but computing split ratios on the fly regarding the incoming traffic.

Only a few papers focus on the load-balancing issue in the post-failure scenario, mainly applied for multiple routing configurations [19] [33] [40], all of which use the modified IP forwarding paradigm.

2.2.2 IP fast re-routing

IP fast re-routing (IPFRR) comes in many variants that aim to increase network protection. Figure 2.2 shows a schematic classification of different IPFRR approaches. The following discussion is organized according to the structure of the diagram.

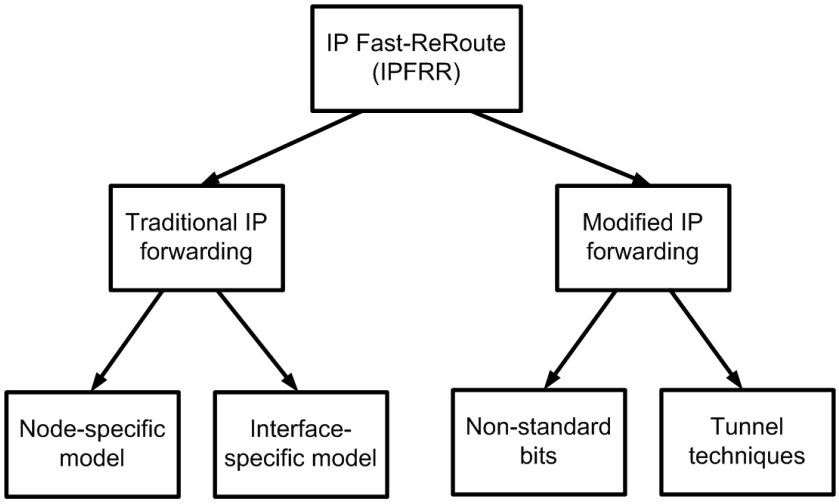


Figure 2.2: Classified different IPFRR approaches

2.2.3 Traditional IP forwarding

The traditional IP forwarding is based on only the destination. We typically call them destination-based forwarding, which are central in two routing protocol classes in the Internet: the distance vector routing protocols and the link-state routing protocols. In the scope of this work, we discuss related work that associates with the link-state routing protocols.

In a network with the traditional IP forwarding, each router maintains a *single* routing table that maps each destination to next-hop routers or each interface of the router owns its distinct routing table that also maps each destination to next-hop routers. We refer to the two models as the node-specific model and the interface-specific model, respectively. The node-specific model is typically associated with the routers with a centralized design, in which forwarding engines, being in charge of processing packets' headers, are independent to routers' interfaces. On the contrary, routers with a distributed design have forwarding engines integrated into their interfaces in order to increase the lookup speed. Such a design is preferred by most high performance routers. As a result, the interface-specific model is dedicated to routers with the distributed design.

ECMP and Loop-Free Alternate (LFA) [3], [25] are two extensions of the traditional OSPF and IS-IS that support multi-path routing for the node-specific model. The ECMP installs the equal shortest paths for each S-D pair while LFA offers path diversity beyond the shortest ones. LFA provides several constraints on path lengths, called loop-free criteria, to compute loop-free link-protecting, node-protecting, and link-and-node-protecting alternates. Such criteria are also the central ideas in other IPFRR methods in this category such as Homeostasis [38], TBFH [54], MPA [55], and MPDA [89]. The benefit of

the proposals built with the loop-free criteria is that the resulting routing tables contain the shortest path next-hops given by the shortest path algorithm. In other words, those proposals are compatible with the current deployed routing protocols in the intra-domain networks such as OSPF or IS-IS. However, a guarantee for being loop-free under *arbitrary* failures by using down-stream path criterion of the LFA can significantly limit the protection coverage.

Typical solutions that do not use the loop-free criteria presented in [3] are O2 [72], PR [41], and MARA [61], all of which share a common goal of building the routing protocol offering more than one next-hop for as many S-D pairs as possible towards a destination. O2 proposes that some links in the network should operate as back-up mutual links and calls them joker-links. Then two algorithms [64] are given to identify those joker-links so that many routers can have two next-hops towards a destination. However, these algorithms are only applicable for centralized routing systems since they may return various sets of joker-links at different routers, meaning that all routers could not have a consistent view of the network. PR shares with O2 the idea of using joker-links but improves over O2 by not limiting the maximum number of next-hops to two. Furthermore, PR selectively assigns the back-up next-hops for each given primary one to maximize network performance under potential failures of nodes. Since the joint optimization problem of protectability and performance requires high computation, PR is only suitable for centralized routing systems, e.g. [23] [28]. MARA proposes a different loop-free criterion that can increase network robustness compared to those of LFA. However, MARA is immature in two aspects: the algorithm is inefficient in many cases due to its large search space and the imprecision of the optimization objective. Our contributions of this work share with MARA the new loop-free criterion, but we propose more realistic routing objectives, low computational algorithms and diverse applications.

The interface-specific forwarding is first proposed in series of Failure Insensitive Routing (FIR) [44] [57] [58] [59] [98], and is central in the interface-specific model. FIRs are designed to avoid all possible single link faults [57] or node faults [98]. The idea behind FIRs is that a router will infer a failure if packets enter the router from *unusual* interfaces which are not used when the network is fault-free, and then forward the packets to next-hops that can avoid the failure. The FIRs' algorithms are based on the shortest path algorithm and each of them is specific for a certain failure scenario, e.g. single link faults, dual link faults, or node faults. With the guarantee of full coverage for the desired failure scenario, FIRs, however, suffer from routing loops when failures are more extensive than those designed [21]. Furthermore, maintaining two routing tables for a destination, the forwarding table and the backwarding table, at each interface in FIRs could consume much memory with a large network. Likewise, SS_LINK/NODE [93] and ESCAP-DL [94]

share with FIR the concept of interface-specific forwarding except that their routing options are not necessarily bound to the shortest paths. SS_LINK/NODE and ESCAP-DL derive ILP problems of assigning back-up paths to cover all single faults and dual link faults, respectively, and then propose algorithms to solve the problems. NISR [45] and [46] further extend the back-up assignment problem to incorporate the load-balancing requirement. However, the routing loop issue is not considered and resolved in those papers. In this work, we also propose a routing method that uses the concept of interface-specific forwarding. Different from previous proposals, our method does not aim to protect all single failures, but it can avoid any routing loops under multiple uncorrelated failures. In practice, routing loops can cause significant network degradation and are considered more serious than failure events.

Pure node-specific model makes the traditional IP networks hardly achieve full protection coverage in any topology. One well-known example for this limitation is the last-hop problem in which at least one router adjacent to the destination has left with a single path to the destination. Theoretically, we can construct a routing such that its protection coverage will increase if the connectivity of a topology get denser but never reach 1, and in the worst case with respect to the ring topologies the protection coverage is not higher than $\frac{1}{n-1}$ where n is the number of routers in the topology [66]. In the node-specific model supporting joker-links, the protection coverage is significantly improved. In many cases, the joker-links help overcome the last-hop problem and therefore full protection coverage can be reached. However, that ideal protection coverage can not be yielded in general. In the worst-case topology, the protection coverage is bounded by $\frac{2}{n-1}$. In addition, the later chapters show that ring topologies also restrict routing methods of the interface-specific model from obtaining the protection coverage of 1. In summary, the protection coverage given by routing method with the traditional IP forwarding depends on the topology structure and in many 2-connected network topologies the full protection coverage is hardly achieved. Consequently, many papers modify the IP forwarding paradigm so that at least one back-up path always exists for a given primary in a 2-connected network.

2.2.4 Modified IP forwarding

This type of forwarding is not merely based on the destination as the previous one, and typically is more complicated. The modified IP forwarding paradigm comes in several variants. The typical examples are MRC [39], DLMRC [32], IDAG [13], FCP [42], SafeGuard [47], and LOLS [67], all of which modify IP packet headers to include additional non-standard information. MRC and DLMRC use multiple routing tables that cover all possible single failures and dual link faults, respectively. Upon detecting a failure on the connected link, the affected router selects another routing table to avoid the network in-

terruption and marks the routing table index in its packets. Other routers will examine such an index in an incoming packet and will select the same configuration. Unlike MRC, IDAG uses two routing tables that are corresponding to two routing configurations, and thus only needs a single marking bit. In order to avoid forwarding loops under arbitrary failures, packets must be transferred from one configuration to another *at most* one.

LOLS and FCP do not let added bits infer the routing configurations as in the MRC. Instead, they store failed components that the packets encounter in its normal forwarding paths in the IP header from which the back-up path is computed on the fly at routers. For example, LOLS introduces a a minimum set of failed links, called blacklist, which a packet must carry. Under LOLS a router is called deadend if its cost to the destination is smaller than those of all other possible next-hops. When a packet arrives at a deadend router whose the only next-hop to the destination has been failed, the router adds failed link in the blacklist and forwards the packet with the recovery mode. The blacklist in the packet will be reset if the packet reaches a router whose cost to the destination is lower than that of the router from which the packet entered the recovery mode. Also computing back-up path on the fly, SafeGuard suggests that IP packets carry the remaining shortest path cost from the leaving router to the destination.

Different from the LSP tunnels employed in MPLS networks that we have already discussed, IPFRR tunnel-based technique [10] is designed for the traditional IP networks. The IPFRR tunnels can be established from the routers that detect the failures to some stating routers from which the traffic can be forwarded towards the destinations without forming loops. Several IP-in-IP mechanisms, e.g. IP-in-IP [79], GRE [31] and L2TPv3 [43], can be used to construct the IPFRR tunnels. Not-via addresses [76] is a pure tunnel-based approach in which each router will construct a tunnel to bypass the possible failure of each of its next-hop. The tunnel, therefore, departs at the router adjacent to the failure and terminates at the second closest router on the shortest path towards the destination. Not-via addresses can protect all link and node failures, but it introduces considerable computational overhead and complexity [22] since a high number of tunnels will be needed in a large network. For that reason, the combination of the LFA and the tunnel-based technique, called remote LFA (rLFA) [9], is proposed to reduce the number of tunnels needed. The rLFA suggests that the tunnels only need to be built for the routers which can not find loop-free alternates for their primaries. Furthermore, the rLFA gives a simple method to identify staging routers to which the repair tunnel should be destined. Those routers are the result of an intersection between P-space and Q-space with respect to a specific failure. Given the failure of link AB that leads an interruption of the forwarding path from B to A towards destination D, the P-space is a set of routers reachable from B without traversing link BA. On the same condition, the Q-space is a set

of routers reaching D without traversing the link BA. Despite significant improvement of protection coverage compared to the typical LFA [16], rLFA does not always offer 100% coverage for single failures.

Chapter 3

Permutation routing framework

This chapter introduces the permutation routing method that we will use as a fundamental background for constructing our desired routings in Chapter 4, 5, 6, and 7. The permutation routing can be either used as a loop-free criterion or as a representation of the corresponding loop-free routing. Importantly, the permutation routing method can significantly increase the robustness for IP networks with the traditional IP forwarding.

The permutation routing, which is formally defined later, is a sequence of routers whose ordering together with a suitable forwarding rule will represent a loop-free routing. More specifically, if we have a permutation routing for a given network topology, we definitely can re-construct the corresponding routing by letting the traffic be forwarded in one direction with respect to the router ordering in the permutation routing. Luckily, for almost loop-free routings we can formulate and construct them in association with creating their corresponding permutation routings. The permutation routing can be constructed without introducing new signaling overhead and therefore can be integrated into the existing link-state routing protocols such as OSPF or IS-IS.

In this chapter, Section 3.1 first presents our motivation on using a permutation routing to represent a loop-free routing and provides the definition showing the relationship between a given loop-free routing and its corresponding permutation routing. Since we hardly know if our expected loop-free routing exists, we propose an algorithm in Section 3.2 to identify those routings by seeking the existence of their corresponding permutation routings. Section 3.3 analyzes complexity of the proposed algorithm by investigating its search space, and recognizes the parameters that help to minimize our search. In addition, we also discuss in this section key requirements that the algorithm should meet in order to be implemented on distributed routing systems. Finally, we conclude the chapter in Section 3.4.

3.1 Permutation routing concept

3.1.1 Motivation

Any loop-free forwarding strategy for one given destination will arrange critical network resources into a directed acyclic graph (DAG). If we define resources to be routers, a DAG with routers as nodes will specify legal paths from any source router to a specific destination. If we define resources to be links (or interfaces), then a DAG with links as vertices will define all legal paths from any first link going out of the source to any link entering into the destination. The observation that forms the basis for our approach is that *any* DAG can be topologically sorted into a sequence, so that all directed edges in the DAG point to resources that appear earlier in that sequence.

One implication of the above observation is that *any* loop-free routing can be represented as a sequence of network resources, in which a packet can use resources only in decreasing order with respect to the position in the sequence. We next propose a mathematical model for permutation routings.

3.1.2 Permutation routing

We first are given the network topology modeled as a connected graph $G = (V, E)$ where V is the set of nodes, which are routers in the network, and $E \subseteq V \times V$ is the set of bidirectional links among those nodes. In this model a bidirectional link from node i to node j is denoted by (i, j), and includes two directed links: $(i \to j)$ from node i to node j and $(j \to i)$ from node j to node i. We begin with a basic definition of a permutation in the graph context.

Definition 3.1. *For a given network topology $G = (V, E)$, permutation $\mathcal{P}$ of nodes is an arrangement of all nodes in V into a particular order.*

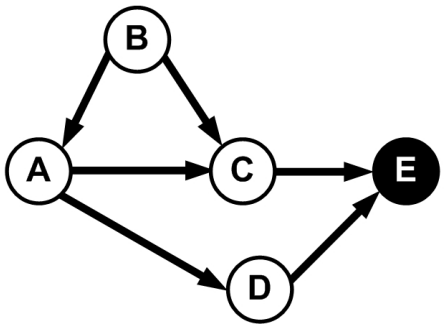
If each node in the graph is identified by a number, then a permutation is simply a sequence of non-repeating numbers. Let N be the cardinality of set V, we can have up to $N!$ such permutations with regard to G. Since we desire to use a permutation to represent a loop-free routing, but not all of the permutations of G can do that, we further select the reasonable permutations for our purpose.

A loop-free routing towards a destination constructed on G consists of next-hop assignments at all nodes for that destination. Without loss of generality, we look at the assignment of next-hops for each destination individually. For destination $d \in V$, we denote by $R_d = (V, E_d)$ a routing graph extracted from G for packets destined to destination d, where E_d is a set of directed links. If each link in G is understood as two directed links, then $E_d \subset E$. In R_d, node j is called the *next-hop* of node i if there exists a directed

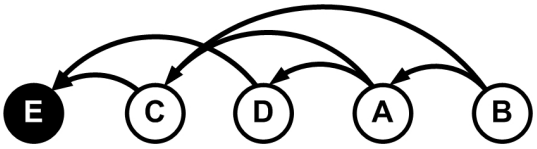
link from node i to node j, denoted by $(i \rightarrow j)$. As we have explained, R_d must be a DAG rooted at destination d. The permutation that can be used to represent R_d on given network topology G is called the permutation routing towards d and is defined as follow:

Definition 3.2. *Permutation $\mathcal{P}$ is a* permutation routing *for R_d if all next-hops of each node occur before it in $\mathcal{P}$. If we write $j < i$ to denote that j occurs before i in $\mathcal{P}$, then $\forall(i \rightarrow j) \in E_d : j < i$.*

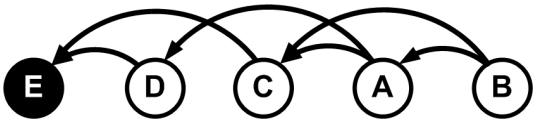
According to this definition, node d will always be the first node in the permutation routing for R_d since the destination does not have any next-hop in that routing, and nodes further out in the routing permutation will be topologically farther from the destination. Furthermore, the permutation routing implies a forwarding rule in which a node can forward packets to its next-hops that always appear before it in the permutation routing towards the destination without causing forwarding loops. The fact is that such permutation routings for R_d can always be computed using the topological sort algorithm [15].



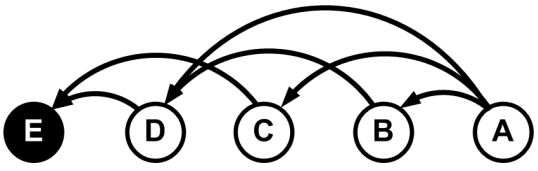
(a) Loop-free routing R_E towards destination E.



(b) A permutation routing for R_E



(c) Another permutation routing for R_E.



(d) A permutation routing not for R_E.

Figure 3.1: Different permutation routings represent a loop-free routing

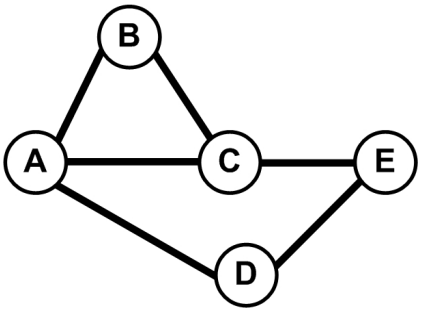
In general, there can be more than one permutation routing for one routing graph R_d. Starting with a routing permutation $\mathcal{P}$, another permutation routing $\mathcal{P}'$ can be generated by swapping *two consecutive* nodes that are not connected by a directed link to each other. For instance, both sequences $\{E\ C\ D\ A\ B\}$ in Figure 3.1(b) and $\{E\ D\ C\ A\ B\}$ in Figure 3.4(b) are valid routing permutations for the DAG in Figure 3.1(a) according to Definition 3.2. Specifically, C and D are not connected in R_E, and thus the swapping of C and D in the two permutation routings does not affect the assigned next-hops of each source node. But if we swap A and B as in sequence $\{E\ D\ C\ B\ A\}$ in Figure 3.4(c), the resulting routing graph would be altered. Correspondingly, A would have three next-hops that correspond to its three neighboring nodes placed before it in the permutation routing while A only has two next-hops in given routing graph R_E.

In the reverse process, a loop-free routing for a destination, typically represented by routing tables at source nodes for that destination, can be generated from its corresponding routing permutation, given a *forwarding rule* that defines the relationship between neighboring nodes. For now, we consider a *greedy* forwarding rule for constructing the routing tables, in which *all* topological neighbors of a node that occur before it in the permutation routing are installed as its next-hops. For example, with such a forwarding rule we can reconstruct the routing in Figure 3.2(c) or the routing tables in Figure 3.2(d) from the permutation routing in Figure 3.2(b) (top) and the topology in Figure 3.2(a). Likewise, the routing and the routing tables in Figure 3.2(e) and in Figure 3.2(f), respectively, are induced from the permutation routing in Figure 3.2(b) (bottom) and the network topology. The greedy forwarding rule will result in a *all-path* routing, where all bidirectional links in the topology are potentially used for traffic forwarding to the destination. This will also maximize the potential for load-balancing and failure recovery. Note that with a provided permutation routing and the network topology, only *one* routing is identified with the greedy forwarding rule.

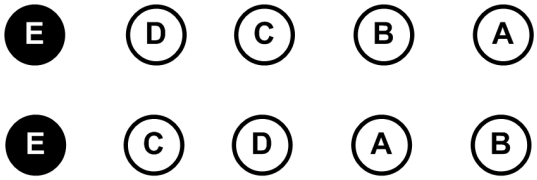
More restrictive forwarding rules could also be considered, which would result in a sparser DAG. This can sometimes be beneficial in order to avoid excessively long paths, or to limit the number of next-hops for a particular destination.

3.2 Algorithm framework

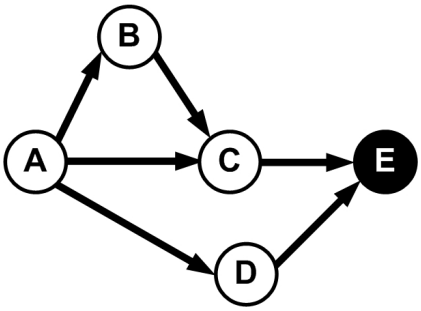
The defined permutation routing concept reveals we can yield our expected loop-free routing on the given topology through its permutation routing. Therefore, in this section we develop an algorithm to construct such a permutation routing. However, the algorithm first requires that our desired routing be formulated as routing constraint functions.



(a) Given network topology



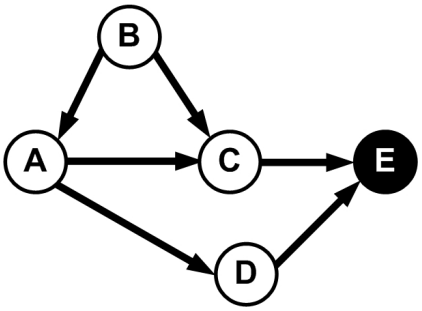
(b) Two permutation routings towards E



(c) Routing corresponds to permutation routing in Figure 3.2(b) (top)

Node	Destination	Next-hops
A	E	B, C, D
B	E	C
C	E	E
D	E	E

(d) Routing tables of the permutation routing in Figure 3.2(b) (top)



(e) Routing corresponds to permutation routing in Figure 3.2(b) (bottom)

Node	Destination	Next-hops
A	E	C, D
B	E	A, C
C	E	E
D	E	E

(f) Routing tables of permutation routing in Figure 3.2(b) (bottom)

Figure 3.2: Illustrations of generating the routings from their permutation routings

3.2.1 Routing constraint functions

A routing towards a destination can be understood as the assignment of a set of next-hops for that destination at each source node. Hence, for a desired routing we can formulate its next-hop assignment at each source node in terms of a routing constraint function as follows:

$$C(u) = \begin{cases} \textbf{True} & \text{if condition } C \text{ for node } u \text{ is satisfied.} \\ \textbf{False} & \text{otherwise} \end{cases}$$

where $C(u)$ is the routing constraint function of node u, and condition C interprets the specific next-hop assignment that node u desires.

Condition C can simply be a specific single next-hop assignment of a node. For instance, if we want that node B would be the next-hop of node A, assuming that there is a bidirectional links between the two nodes, then routing constraint function for node A has become:

$$C(A) = \begin{cases} \textbf{True} & \text{if } B < A \\ \textbf{False} & \text{otherwise} \end{cases} \tag{3.1}$$

where $B < A$, as explained earlier, means that B occurs before A in the resulting permutation routing.

In another case, condition C can be an arithmetic equation. That is when we require that node A have k next-hops, regardless of the identities of the next-hops. The routing constraint function for node A is formulated below:

$$C(A) = \begin{cases} \textbf{True} & \text{if } n[A] = k \\ \textbf{False} & \text{otherwise} \end{cases} \tag{3.2}$$

where $n[A]$ is the number of neighboring nodes of node A that are placed before it in the resulting permutation routing.

Additionally, condition C can be a combination of multiple conditions. For example, node A in the routing is designed to have two next-hops, one of which is node B. Then the condition C for node A would be:

$$C(A) = \begin{cases} \textbf{True} & \text{if } n[A] = 2 \wedge B < A \\ \textbf{False} & \text{otherwise} \end{cases} \tag{3.3}$$

3.2.2 Back-tracking algorithm

We denote by $\mathcal{C}$ the constraint set that includes all routing constraint functions of nodes in the network, $\mathcal{C} = \{C(u_i)\}_{1 \leq i \leq N}$, where N is the number of nodes. The problem of

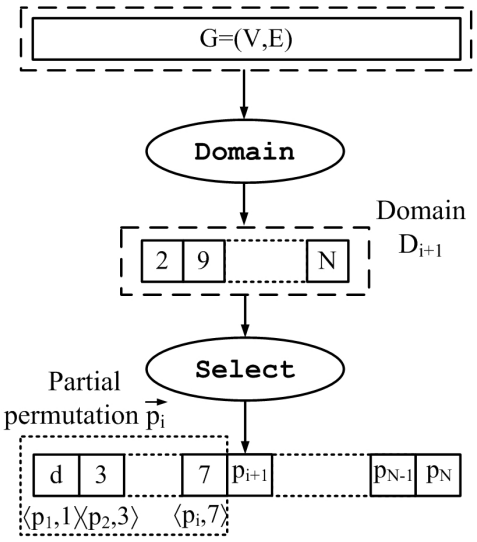


Figure 3.3: Basic assignment procedure for a variable

finding a permutation routing that satisfies the provided constraint set $\mathcal{C}$ would involve solving a constraint satisfaction problem (CSP) [18] [84]. The basic algorithm for the CSP problems is defined below.

Let $\mathcal{P} = \{p_1, p_2, \ldots, p_N\}$ be a set of N variables in a *fixed order* from p_1 to p_N, with respective domains $\mathcal{D} = \{D_1, D_2, \ldots, D_N\}$. We refer to D_i as the *candidate set* for each variable p_i, which consists of the nodes that can be assigned to variable p_i. We further assume that each node u_i is given a constraint function $C(u_i)$ listed in set $\mathcal{C}$.

Initially, routing permutation $\mathcal{P}$ for our desired routing is empty. One at a time we assign a node $u_i \in D_i$ to variable p_i, denoted by $\langle p_i, u_i \rangle$. Such an assignment is valid if constraint function $C(u_i)$ returns **True**. Figure 3.3 illustrates the basic assignment for variable p_{i+1} in which two key functions `Domain` and `Select` work as filters to control the assignment. Specifically, function `Domain` generates the candidate set for variable p_{i+1}, and function `Select` will pick a node from those potential nodes that satisfies its defined constraint function. The successful assignments of nodes to a subset of variables $\{p_1, p_2, \ldots, p_i\}$ given by $\{\langle p_1, u_1 \rangle, \ldots, \langle p_i, u_i \rangle\}$ is called *partial permutation routing* with i nodes. For simplicity, we abbreviate it to $\vec{p}_i$.

The described assignment procedure is an instance of the CSP, and therefore full permutation routing $\mathcal{P}$ can be found with the backtracking algorithm whose pseudo-code has been shown in **Algorithm** 1. The main component of the algorithm is a loop. The loop calls function `Select` which goes through D_i to find a valid node for the current variable p_i. If `Select` succeeds in finding such a node, the algorithm calls function `Domain` to generate domain D_{i+1} and proceeds to next variable, p_{i+1}. Otherwise, a backtrack step

will be executed to revisit variable p_{i-1}. The algorithm terminates if permutation routing $\mathcal{P}$ of N nodes, also denoted by $\vec{p}_N$, is found or a failure notification is returned if all backtracks are examined but no solution is found under a certain $C(u_i)$.

Algorithm 1: `Backtracking`

Input: Network topology G and constraint set $\mathcal{C}$
Output: Either a permutation routing or failure notification

```
1  i ← 1
2  D_i ← {d}
3  V_i ← V \ {d}
4  while 1 ≤ i ≤ N do
5  |  p_i ← Select()
6  |  if p_i = null then
7  |  |  i ← i − 1
8  |  else
9  |  |  i ← i + 1
10 |  |  Domain()
11 if i = 0 then
12 |  return failure
13 else
14 |  return p⃗_N
```

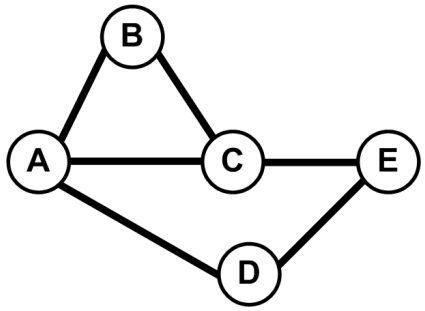
To illustrate the algorithm framework for constructing permutation routings, we consider a simple network topology in Figure 3.4(a) with 5 nodes and 5 bidirectional links, and three sets of constraint functions listed in Table 3.1. In each constraint set, the routing constraint function for each node, $C(u_i)$, has the form of arithmetic equation as shown in formula (3.1).

Table 3.1: Different routing constraint sets

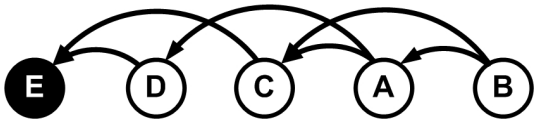
Node u	A	B	C	D	E
$\mathcal{C}_1$	$n[A] = 2$	$n[B] = 2$	$n[C] = 1$	$n[D] = 1$	-
$\mathcal{C}_2$	$n[A] = 3$	$n[B] = 1$	$n[C] = 1$	$n[D] = 1$	-
$\mathcal{C}_3$	-	-	$n[C] = 2$	$n[D] = 2$	-

With the topology and each set of constraint functions in the table, the algorithm returns the corresponding permutation routings. Permutation routings $\{E\ D\ C\ A\ B\}$ in Figure 3.4(b) and $\{E\ D\ C\ B\ A\}$ in Figure 3.4(c) satisfy $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively, but

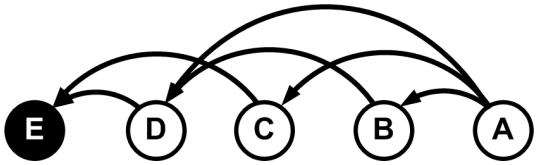
there does not exist a permutation routing for $\mathcal{C}_3$. It is easy to see that if we install two next-hops for both C and D as illustrated in Figure 3.4(d), then a routing loop would occur.



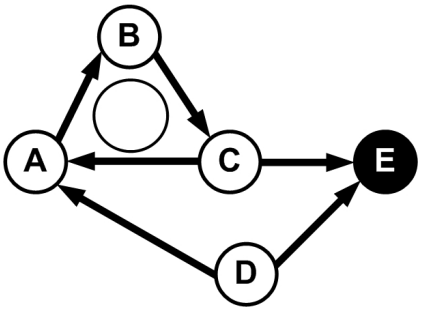
(a) Network topology



(b) Permutation routing satisfies $\mathcal{C}_1$.



(c) Permutation routing satisfies $\mathcal{C}_2$.



(d) Last-hop problem prevents both C and D from having two next-hops.

Figure 3.4: Constructing permutation routings from the constraint sets

3.3 Discussion

3.3.1 Search space and computational complexity

As already illustrated in the previous example, a constraint set could decide the existence of the corresponding permutation routing. However, if constraint set $\mathcal{C}$ allows it, the `Backtracking` algorithm will find *one* routing permutation among all possible solutions by searching through a *search space* shaped by the variables of $\mathcal{P}$ and their domains of values.

In a naïve implementation, the domain for variable p_{i+1} consists of $(N - i)$ nodes that have not been placed in $\vec{p}_i$. Based on that observation, search space S of the permutation assignment problem has the form of a tree of depth N rooted at the initial state $\langle p_1, d \rangle$

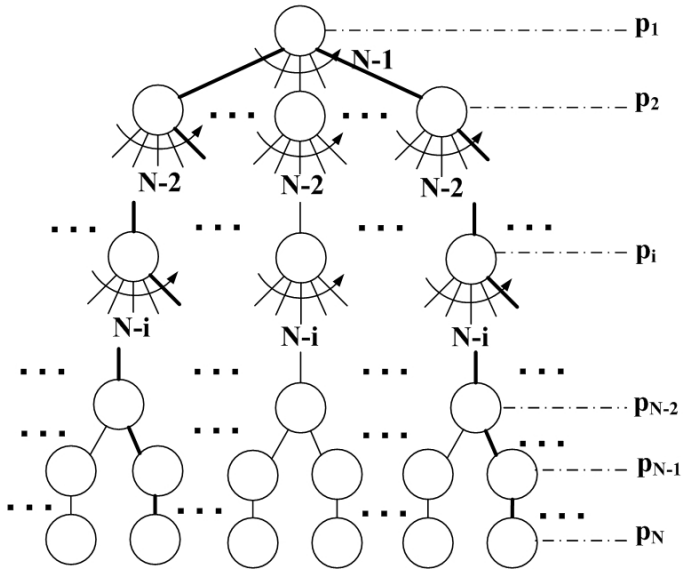


Figure 3.5: Search space for routing permutation problem

as illustrated in Figure 3.5. Solutions $\vec{p}_N$ are be traced from the initial state to state p_N located at the leaves of the tree. Two connected states in the search space refer to two instances of p_i and p_{i+1}. Assume that the computational operations needed on average to move from state p_i to state p_{i+1} do not depend on other states, the complexity in the best case when we do not need any backtrack step (backtrack-free) is $O(N)$. In other extreme, if there is only one solution and we always make the "wrong" choice at each stage, the complexity would be $O(N!)$.

The described naïve implementation above may result in high computational complexity, and is only feasible for small topologies. Hence, it is important to guide the search to avoid exploring the entire search space. We have noticed that the search space can be reduced by using following two approaches:

1. Limit the size of the domain for each variable - Clearly, the smaller domain size for each variable will result in a much smaller search space compared to the original of Figure 3.5. We can narrow the domains by an intuitive observation or using defined routing constraint functions to eliminate in advance the nodes that are impossible to be selected in a specific assignment stage.

2. Constraint function $C(u)$ should be simple to avoid possible backtrack steps.

We will return to this issue when we deal with a specific routing objective.

3.3.2 Distributed implementation

The permutation routing algorithm takes only a network topology as its input, and does not introduce additional signaling overhead. Hence, the algorithm can be implemented as part of a link-state routing protocol. Particularly, the permutation routing that is compatible with the shortest path routing such as ANHOR-SP (discussed in Chapter 4) can be integrated into standard link-state routing protocols, i.e. OSPF and IS-IS, with few changes.

However, the efficacy of the algorithm in a link-state routing protocol should be also considered in context of the distributed manner of routing systems. That is in such a system each node computes its routing table which is consistent with routing tables of others. The condition implies that the resulting permutation routing should be identical at all nodes of the network. In other words, the algorithm should be designed to select the same single node at each assignment. The issue is attributed to stringency of a routing constraint set, $\mathcal{C}$, which we will elaborate in association with a specific routing objective. Another issue contributes to the efficiency of a routing protocol in the distributed routing systems is the convergence time of the routing algorithm, which in turn is subject to the algorithm's computational complexity. We believe that the algorithm will be feasible for the adoption if its complexity is as good as that of the shortest path algorithm, e.g. linearly proportional to the number of nodes. Fortunately, this can be achieved by carefully opting the routing constraint functions so that the search is backtrack-free.

Given a network of N nodes, each node has to compute N permutation routings, one for each destination. However, the node does not necessarily produce N full permutations. Instead, the node can terminate the algorithm right after the node itself has been placed in the permutation. The reason is the node has sufficient information to generate its routing table with such a permutation routing. Let i be the source node and j be the destination, the length of the permutation for node i is the position of node i in the permutation routing which is denoted by $\mathcal{P}_j(i)$. For some destinations that are neighboring to the source node, $\mathcal{P}_j(i)$ could be 1 while for other destinations $\mathcal{P}_j(i)$ could be N. The average length of the permutation routings calculated by node i for all destinations would be calculated by $\overline{N_i} = \frac{1}{N}\sum_{j=1}^{N}\mathcal{P}_j(i)$. Table 3.2 shows the permutation routings towards all destinations constructed at node A and node B of the network topology in Figure 3.4(a) either in the reduced or in the full forms. In the reduced forms, the average length of the permutation routings node A and B would compute are 2.5 and 3.75, respectively. In the full forms, the average length is equal to the number of nodes (5).

Table 3.2: Average lengths of permutation routings in reduced and full forms

Destination u	Permutations at A	Permutations at B	Full permutations
A	-	$\{A\ B\}$	$\{A\ B\ C\ D\ E\}$
B	$\{B\ A\}$	-	$\{B\ A\ C\ D\ E\}$
C	$\{C\ A\}$	$\{C\ A\ B\}$	$\{C\ A\ B\ D\ E\}$
D	$\{D\ A\}$	$\{D\ A\ E\ C\ B\}$	$\{C\ A\ B\ D\ E\}$
E	$\{E\ C\ D\ A\}$	$\{E\ C\ D\ A\ B\}$	$\{E\ C\ D\ A\ B\}$
Average lengths	2.5	3.75	5

3.4 Summary

In this chapter, we present a method that can be used as a loop-free criterion in constructing routings with increased robustness for IP networks. We call the method permutation routing.

The permutation routing is a sequence of all nodes in a network with a particular arrangement together with a forwarding rule that allows a source node to send its packets to its neighboring nodes towards a destination without causing loops. In that sequence, there always exists at least one topological neighbor of a node that is placed before that node in the sequence, except the destination node which is at the left-most position. The permutation routing corresponds to a loop-free routing towards the destination and each of other nodes is allowed to install *all* its neighboring nodes that occur before it in the sequence as its next-hops. The resulting routing from a permutation routing and a greedy forwarding rule is unique, and is called all-path routing since it can use all bidirectional links (in each bidirectional link, only one direction is used for one destination) to forward the packet.

The construction of the permutation routing associated with a given routing objective is a constraint satisfaction problem and can be solved through three steps. First, we model the permutation routing as a set of variables, each of which stands for a position in the sequence and define the domain for each variable. Second, we formulate the routing objective in terms of a routing constraint function at each source node. Finally, the backtracking algorithm will generate the permutation routing, if it exists, by repeatedly assigning a node in the domain to the corresponding variable so that the assignment satisfies the routing constraint function of that node.

The backtracking algorithm can produce a permutation routing after an excessive huge number of assignments due to necessary backtrack steps. Nevertheless, we can reduce the complexity by narrowing the search space. In order to do that, we should minimize the domain for each variable and simplify the routing constraint functions as much as good.

In the best case, we can yield the permutation routing just after N assignments where N is the number of nodes.

The permutation routing algorithm can easily be designed for a distributed routing system where each of node in the network will compute its own permutation routing. We require that all resulting permutation routings be identical so that the routing tables of all nodes are also consistent. That can be done if we define a strict routing constraint function so that each selection is consistent at all source nodes.

Chapter 4

Next-hop permutation routing

In the traditional IP networks, each node installs a single routing table that maps each destination to a next-hop. When a packet arrives at the node, from any incoming interface, it is forwarded to the next-hop according to the destination in its IP header. In addition, the routing tables may be enhanced to support multi-path routing protocols that can offer multiple next-hops for a destination.

An ideal robust IP routing has each node install multiple next-hops for a destination. The additional next-hops are the basis for either load-balancing or fault tolerance under network failures. However, such a routing is hard to achieve for ISP networks with the traditional IP forwarding. One example is the last-hop problem which prevents all neighboring nodes of a destination from having two next-hops, leaving at least one of them with only one next-hop. Even if we do not count the last-hop problem, [41] and [83] prove that the minimum node degree of the network topology should have to obtain the optimal robustness is $\lceil \frac{n}{2} \rceil$. Such a condition makes the network excessively dense when the number of nodes is large, and therefore requires a big investment for the network infrastructure. For that reason, in this chapter we aim to construct a routing that can maximize the robustness for a given network topology. In other words, our algorithm will find a loop-free routing with as many S-D pairs with at least two next-hops as possible. The construction is based on the permutation routing concept and the algorithm framework developed in the previous chapter.

The chapter begins with a motivating example in Section 4.1. The example shows the importance to distribute next-hops for all nodes so that a routing is loop-free and maximizes the protection coverage. From the example, we derive two routing objectives for increased robustness for IP networks. We then use the permutation routing framework described in Chapter 3 to build routings with such objectives by producing the corresponding permutation routings. As required in the construction process, we first formulate the routing objectives in terms of routing constraint functions at source nodes

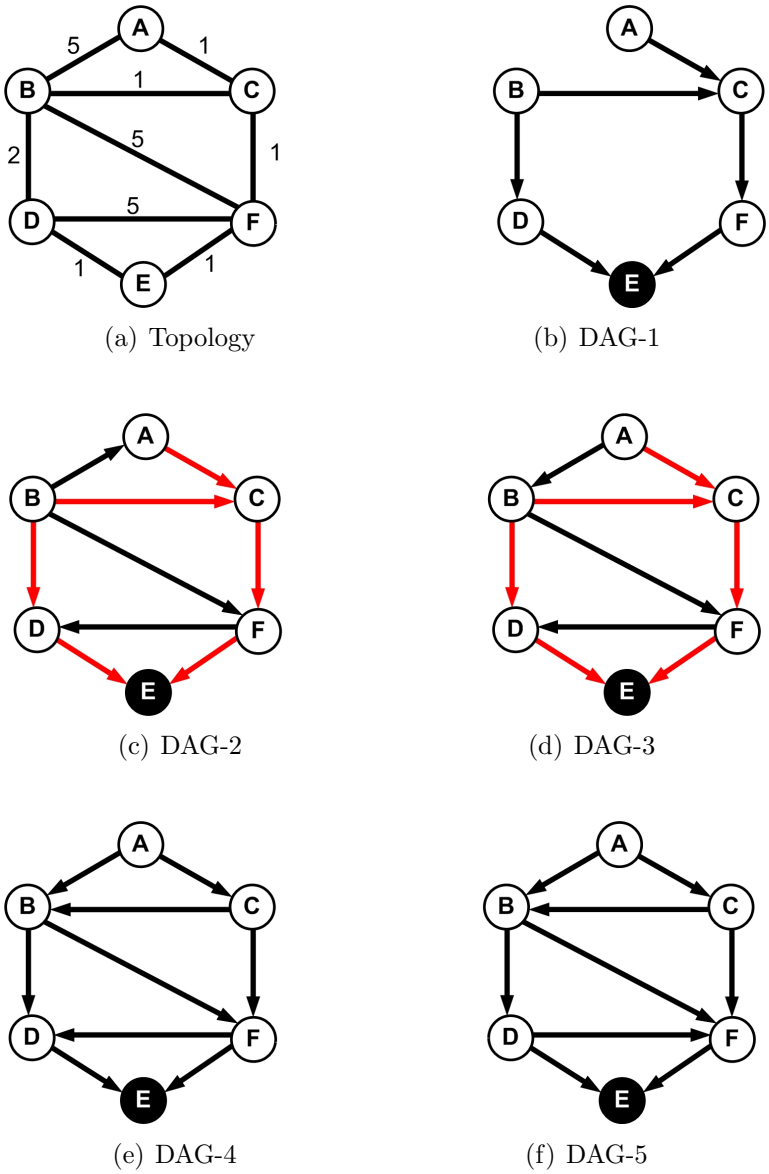
for each destination in Section 4.2. Then in Section 4.3 and Section 4.4 we define the domains for each variable of the permutation routings regarding the two routing objectives. The definitions of the routing constraints and the domains guarantee that we are able to achieve backtrack-free algorithms. The performance analysis of our algorithms is given in Section 4.5. In addition, Section 4.6 provides an enhancement of the algorithm for identifying node-protecting alternates. Finally, we conclude the entire chapter in Section 4.7.

4.1 Motivation

A loop-free routing where all links of the topology are included in the corresponding DAG is called *all-path routing*. With a given network topology, many different all-path routings can normally be constructed. However, they will have different properties with respect to failure recovery as shown in the following illustration.

Figure 4.1 shows a simple network topology (Figure 4.1(a)), with 5 different DAGs for destination node E. In Figure 4.1(b), DAG-1 is given by shortest path routing with ECMP using the link weights indicated in the topology. Node B can split its traffic over two next-hops, while the other nodes have only a single next-hop towards destination E. Links (A, B), (B, F) and (D, F) are left idle, and are neither used for backup or load balancing. The DAGs in Figure 4.1(c), Figure 4.1(d), Figure 4.1(e), and Figure 4.1(f) are all all-path routings, where all links can be used for packet forwarding. They differ, however, in their distributions of next-hops. In DAG-2, there are three nodes that have only a single next-hop (A, C, and D). DAG-3, on the other hand, has only two such nodes (E and D). Note that DAG-2 and DAG-3 are both compatible with the shortest path routing, because they contain all directed links of DAG-1. DAG-4 is not compatible with shortest path routing, by changing the direction of the link ($B \rightarrow C$), but the number of nodes with a single next-hop has been reduced to one (the minimum value). DAG-5 is also not compatible with the shortest path routing, and slightly different from DAG-4, but still has maximum number of nodes with two next-hops.

ECMP and LFA describe ways of increasing the possibility of using multiple paths in the network. They do this by using link weights to maximize the number of links that can be used towards any given destination without creating loops. They are, however, oblivious to whether a link provides the second forwarding alternative for a node, or whether it provides the fourth alternative. From a fault tolerance viewpoint there is a significant difference between these two cases. The source of improvement over ECMP and LFA in our approach is that we build all-path routings for all destinations, and that we can arrange the use of these paths so that the solution optimizes on the number of nodes



(a) Topology (b) DAG-1

(c) DAG-2 (d) DAG-3

(e) DAG-4 (f) DAG-5

Figure 4.1: A network topology with different DAGs

that have at least two forwarding alternatives. This leads us to the following optimization criterion for *Next-Hop Optimal Routing* (NHOR):

Definition 4.1. *An NHOR is an all-path routing that maximizes the number of S-D pairs that have* at least *two next-hops towards a destination.*

Since the definition of the NHOR only pays attention to the number of next-hops at each source node, there may exist more than one NHOR on the network topology. For example, DAG-4 in Figure 4.1(e) and DAG-5 in Figure 4.1(f) are two NHORs towards node E on the same network topology. In addition, an NHOR in general will not be compatible with the shortest path routing, meaning that some of next-hops installed at source nodes conflict with the shortest path tree (SPT). For that reason, we also define the *Shortest-Path compatible NHOR* (NHOR-SP):

Definition 4.2. *An NHOR-SP is an all-path routing that maximizes the number of S-D pairs that have* at least *two next-hops while containing the DAG calculated by a shortest path algorithm.*

The routing that includes the SPT has several advantages from a practical point of view, since networks are often engineered to give good performance with the shortest path routing [24] [81] [65].

Both NHOR and NHOR-SP are loop-free all-path routings, and they can be built by using our described permutation routing framework. Ideally, if we do an exhaustive search of all possible legal permutation routings, we could find an NHOR solution. This is, however, only practical for very small networks. We therefore only look for permutation routings that can be found iteratively, and without backtracking. The routing constraint functions for nodes presented in the next section are designed to balance between two goals: backtrack-freeness of the algorithm and the optimal number S-D pairs with at least two next-hops.

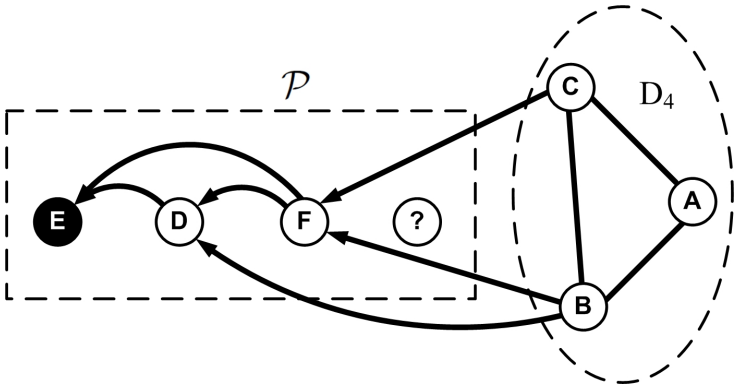
4.2 Routing constraint functions

The number of next-hops of a source node in a permutation routing is the number of its neighboring nodes that we place before the source node in the permutation routing. It implies that the optimal assignment of a node to a variable should take into account the current partial permutation routing and the domain of the variable. Let $\vec{p_i}$ be the partial permutation routing of i nodes for destination d, and by applying the greedy forwarding rule on $\vec{p_i}$, we achieve a loop-free routing sub-graph towards node d. We denote by $R_d^i = (V(\vec{p_i}), E_d(\vec{p_i}))$ such a sub-graph where $V(\vec{p_i})$ is the set of i nodes in $\vec{p_i}$ and $E_d(\vec{p_i})$ is

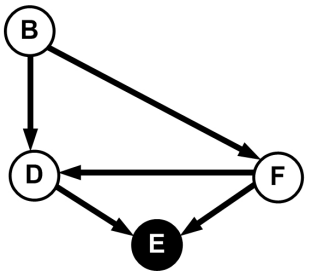
the set of directed links between connecting pairs of nodes in $V(\vec{p}_i)$. Assume that R_d^i is the most robust routing sub-graph possible, and D_{i+1} is the candidate set for variable p_{i+1}, our strategy is to select a node in D_{i+1} to produce $\vec{p}_{i+1}$ whose corresponding sub-graph R_d^{i+1} is also the most robust. Intuitively, the selected node should be the one that has the highest number of neighboring nodes already placed in $\vec{p}_i$. Consequently, the number of directed links of $E_d(\vec{p}_{i+1})$, resulted from the assignment $\langle p_{i+1}, u\rangle$, must be maximized:

$$|E_d(\vec{p}_{i+1})| = \max_{\forall u \in D_{i+1}} |E_d(\vec{p}_i, \langle p_{i+1}, u\rangle)| \tag{4.1}$$

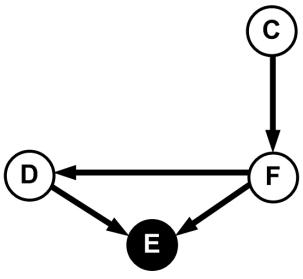
We clarify the above formulation with an example. Assume that we are given topology G in Figure 4.1(a) and partial permutation $\vec{p}_3 = \{E\ D\ F\}$ towards destination E. Candidate set D_4 for variable p_4, in a naïve implementation, can consist of three nodes C, B, and A. The scenario is illustrated in Figure 4.2(a). It is obvious that node A can not be selected since it does not has any neighbor in $\vec{p}_3$. We then have two possibilities of $\vec{p}_4$, $\{E\ D\ F\ C\}$ with respect to assignment $\langle p_4, C\rangle$ and {E D F B} with assignment $\langle p_4, B\rangle$. Since $|E_E(\{E\ D\ F\ B\})| = 5$ in Figure 4.2(b) and $|E_E(\{E\ D\ F\ C\})| = 4$ in Figure 4.2(c), we pick B for p_4.



(a) Assigning a node from the domain to the variable



(b) Routing graph when B is selected



(c) Routing graph when C is selected.

Figure 4.2: Illustration of the routing constraint function

We derive the routing constraint function for node u, which is assigned to variable p_{i+1}, to realize expression (4.1) as follow:

$$C(\langle p_{i+1}, u\rangle) \text{ or } C(u) = \begin{cases} \textbf{True} & \text{if } c[u] = \max_{\forall v \in D_{i+1}} c[v] \\ \textbf{False} & \text{otherwise} \end{cases} \tag{4.2}$$

where $c[u]$ is the counter that keeps track the number of outgoing links from u to $\vec{p_i}$. In other words, $c[u]$ corresponds to the number of next-hops node u will have if it is selected for the variable.

The condition in the constraint function has a simple form of an arithmetic equation. Different from what have discussed in Section 3.2 of Chapter 3, the constraint function for a node in formula (4.2) can not be known in advance due to the dependency of the current assignment to the previous one. As the result, we provide the algorithm framework with an initial constraint set, $\mathcal{C}_0 = \{C_1, \mathsf{null}, \ldots, \mathsf{null}\}$, where C_1 associates with the destination, and C_i will be computed at assignment i-th. Specifically, function `Select` given below first computes the maximum connectivity value from all nodes in the candidate set to the latest partial permutation routing, c_{max}, which is the central of the arithmetic equation of C_i. A loop then goes through domain D_i, one at a time, picks a node from that and compares the next-hop counter of that node with c_{max}. A node will be returned if the comparison is equal. It is clear that function `Select` will only return `null` if the given domain is empty.

Furthermore, in order to guarantee a consistent selection among all nodes in the network, since domain D_i typically has more than one node u satisfying its constraint $C(u)$, we should pick the node with higher ID. As an ID, we can for instance use the loopback IP address of a node.

Function Select

```
1 c_max ← max_{∀v∈D_i} c[v]
2 while D_i ≠ ∅ do
3 |   an arbitrary node u from D_i
4 |   D_i ← D_i \ {u}
5 |   if c[u] = c_max then
6 |   |   return u
7 return null
```

In the following sections, we use the algorithm framework and the defined routing constraint functions to generate permutation routings that approximate NHOR.

4.3 Approximate NHOR (ANHOR)

The efficiency of the backtracking algorithm is determined by the way we define domains for variables. Intuitively, a feasible routing requires that the potential nodes for variable p_{i+1} be those that have at least one connection to nodes already placed in partial permutation routing $\vec{p_i}$. The domain is, therefore, generated following the recursive relation:

$$D_{i+1} = D_i \cup \{\ v \in V_i \mid (u,v) \in E\} \setminus \{u\} \tag{4.3}$$

where u is the node that has been assigned to variable p_i in i-th assignment and V_i is a set of nodes from V, excluding all nodes in $\vec{p_i}$ and D_i.

Formula (4.3) means that the domain for the next assignment is the domain of the previous assignment after removing out the previous assigned node, say u, and adding into nodes having connections with u. This recursive form also reduces the computational complexity in the implementation. The pseudo code for computing the domain is shown in function `Domain`.

The main input for function `Domain` is network topology G while the output is the candidate set for the next assignment. First of all, previous assigned node u is removed out of the domain (line 1). Then the first loop goes through the remaining nodes in node set V that we never consider and adds into the domain those nodes that have connections to u (lines 2-3). Next a second loop applies to the new domain set to update next-hop counters for nodes in that domain (lines 4-5). The counters for the newly added nodes in the domain are always 1 while the counters of existing nodes with connections to u are increased by 1.

Function Domain
Input: Topology G
Output: Domain D_i

1 $D_i \leftarrow D_i \setminus \{u\}$
2 **for** $v \in V_i$ *such that* $(u,v) \in E$ **do**
3 $\quad D_i \leftarrow D_i \cup \{v\}$
4 **for** $v \in D_i$ *such that* $(u,v) \in E$ **do**
5 $\quad c[v] \leftarrow c[v] + 1$
6 $V_i \leftarrow V_i \setminus D_i$

The computational complexity of ANHOR is the product of the average number of operations to make a move between two successive states and the total number of states

visited in the search space. Before we go into a further discussion of complexity, we show that $C(u)$ always allows us to find a valid assignment iteratively:

Proposition 4.1. *Routing constraint function* (4.2) *and the domain defined in* (4.3) *give a backtrack-free algorithm for all connected input topologies.*

Proof. The proof is by contradiction. Assume that the algorithm with the constraint function $C(u)$ is not backtrack-free. This means that constraint function returns **False** for all $u \in D_{i+1}$ at some iteration. That can not happen because D_{i+1} can never be empty in a connected topology before all nodes have been placed in the permutation and all nodes in domain D_{i+1} always have at least one next-hop in $\vec{p}_i$. □

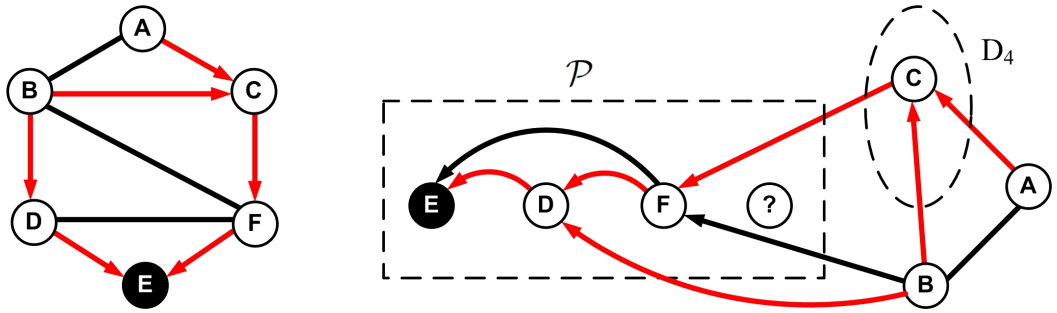
Given the backtrack-free property of our algorithm, the complexity of calculating a permutation for *each* destination is $O(|E| + N \times |D|)$, where $|D|$ denotes the average size of the domain. Since $|D|$ typically depends solely on the average node degree of the network topology, the complexity can be written as $O(|E| + N)$. In dense topologies, the complexity would be $O(N^2)$.

4.4 Approximate NHOR-SP (ANHOR-SP)

As previously mentioned, the objective of NHOR-SP has two properties: first to include the existing shortest path routing, and second to maximize protection coverage. In other words, given the shortest path routing, $R_d^* = (V, E_d^*)$ to a certain destination d, we desire to compute routing $R_d = (V, E_d)$ such that E_d is a superset of E_d^*. It is easy to see that the second property is the heart of NHOR and its construction is described above, the first property is added to the NHOR so that it can be compatible with the shortest path routing. The first property is realized by re-defining the candidate sets for variables. We observe that *not all* nodes in the domain of formula (4.3) are compatible with the existing shortest path routing, meaning that those nodes, if being picked, may result in a routing graphs whose directed links break the current ordering of the shortest path routing. For example, we already have had the SPT in Figure 4.3(a) and partial permutation routing $\vec{p}_3$ in Figure 4.3(b), $\{\langle p_1, E\rangle, \langle p_2, D\rangle, \langle p_3, F\rangle\}$, and are making decision on the assignment for variable p_4. Intuitively, we have two choices for p_4, node B and node C. Furthermore, $\langle p_4, C\rangle$ is a relevant assignment while $\langle p_4, B\rangle$ may result in the routing that contains $(C \rightarrow B)$, and conflicts with $(B \rightarrow C)$ of the shortest path routing. As the example suggests, candidate set D_{i+1} will include all nodes that have *all* its next-hops of the SPT already placed in the permutation routing, formalized as follow:

$$D_{i+1} = D_i \cup \{v \in V \mid c^*[v] = n^*[v]\} \setminus \{u\} \tag{4.4}$$

where u is the node that has been assigned to variable p_i, $c^*[v]$ is the counter of node v to keep track the number of its outgoing links in the SPT from itself to its neighboring nodes already placed in the sequence, and $n^*[v]$, precomputed by node v from the SPT, is the total number of its next-hops in the SPT. In Figure 4.3(b) we have $c^*[B] = 1$ and $c^*[C] = 1$. Meanwhile, the calculation from the SPT in Figure. 4.3(a) gives $n^*[C] = 1$ and $n^*[B] = 2$. According to (4.4), D_4 should contain only node 4, and thus assignment $\langle p_4, C \rangle$ is valid.



(a) The SPT towards E (b) Only node C is suitable for domain D_4

Figure 4.3: Illustration of forming of the candidate set

Function Domain

Input: Topology G and shortest path routing R_d^*
Output: Domain D_i

1 $D_i \leftarrow D_i \setminus \{u\}$
2 **for** $v \in (V_i + D_i)$ *such that* $(v \rightarrow u) \in E_d^*$ **do**
3 $\quad c^*[v] \leftarrow c^*[v] + 1$
4 **for** $v \in (V_i + D_i)$ *such that* $(u, v) \in E$ **do**
5 $\quad c[v] \leftarrow c[v] + 1$
6 **for** $v \in V_i$ **do**
7 $\quad$ **if** $c^*[v] = n^*[v]$ **then**
8 $\quad\quad D_i \leftarrow D_i \cup \{v\}$
9 $V_i \leftarrow V_i \setminus D_i$

Function `Domain` of ANHOR-SP takes both the network topology and the shortest path tree as its parameters. The function first removes previously assigned node u out of the domain (line 1). Next the first loop checks all remaining nodes in V and the current

domain to find the nodes which take u as their shortest path next-hops, and then updates their shortest path counters (lines 2-3). Note that we should not update the domain for the next assignment right now since we also need to compute the next-hop counters for nodes, which is necessary for `Select` function in its selection strategy, upon the moving of u into the permutation routing (lines 6-8). Finally, we generate the new domain, following formula (4.4), by looping through only the remaining nodes in V to pick those nodes whose all shortest path next-hops are already placed in the partial permutation routing (lines 4-5).

Proposition 4.2. *Routing constraint function* (4.2) *and the domain defined in* (4.4) *give a backtrack-free algorithm for all connected input topologies.*

Proof. According to formula (4.2), routing constraint function $C(u)$ always returns **True** unless D_i is empty. We show here that D_i can never be empty before all nodes have been placed in the permutation routing. If D_i is empty, there is no node that has all its shortest path descendants in $\vec{p_i}$. In other words, we can follow the shortest path DAG R_d^* from any node that has not been placed and always find a next-hop node that is not placed in $\vec{p_i}$. But this is impossible: since R_d^* is connected and loop-free, any path along the shortest path DAG will eventually reach the destination, which is the first node that was placed in the permutation routing. □

The computational complexity of ANHOR-SP towards one destination includes two parts: the shortest path calculation using Dijkstra's algorithm and the permutation routing construction. Due to the property of backtrack-freeness, with sparse topologies the complexity of second part towards *one* destination would be $O(|E| + |E_d^*| + N \times |D|)$ where $|D|$ denotes the average size of the domain. Since $|E_d^*|$ is typically much smaller than $|E|$ and $|D|$ only depends on the average node degree of the network, the complexity can be written as $O(|E| + N)$. In dense topologies, the complexity would be $O(N^2)$.

4.5 Performance evaluation

We evaluate performance of the proposed algorithms by measuring how well they realize NHOR and NHOR-SP defined in Definition 4.1 and Definition 4.2, respectively. Since a multi-path routing may lead to path inflation, we also measure path length distribution. ANHOR and ANHOR-SP are compared with ECMP and LFA. Comparisons with other robust routing methods, e.g. MRC [39] and FIR [58], are less relevant, because of their different objectives and implementation costs.

4.5.1 Environment settings

The algorithms for producing ANHOR and ANHOR-SP require that the topology information, including link weights, be available.

Refined network topologies

We select six representative network topologies from Rocketfuel project [80] for our evaluations. For each topology, we remove all nodes that will not contribute on routing (e.g. single degree node). The refined topologies are bi-connected graphs, listed in Table 4.1 in increasing order of their average node degrees.

Table 4.1: Network topologies

AS	Name	Nodes	Links	Avg. Degree
1221	Telstra(au)	50	97	3.88
3967	Exodus(us)	72	140	3.89
1755	Ebone(eu)	75	149	4.00
3257	Tiscali(eu)	115	282	4.90
6461	Abovenet(us)	129	363	5.60
1239	Sprint(us)	284	941	6.62

Link weight assignment

The results for ECMP and LFA depend heavily on the link weight settings used in the topologies. To obtain realistic link weight settings, we run local search heuristic with link load objective function proposed in [24], using a traffic matrix generated by the gravity model [56]. For AS1239, we use unit link weights, because the local search heuristic described in [24] does not scale to a topology of this size.

4.5.2 Evaluation metrics

We select three metrics that reflect routing robustness and examine each of them on the six topologies with different routing algorithms.

- *The fraction of nodes with at least two next-hops* represents the level of fault tolerance and load-balancing of a network. We also refer to this as the link-protecting coverage. A routing with higher link-protecting coverage tends to be more robust under single link failures.

- *Routing Efficiency (RE) coefficient* is defined as the fraction of directed links that are used for traffic forwarding and topology links when the number of next-hops installed in the forwarding table is limited to *at most* K, and is given by

$$RE = \frac{|E_d(K)|}{|E|} \tag{4.5}$$

  where $|E_d(K)|$ is the number of directed links in the DAG when each node can have at most K next-hops and $|E|$ is the number of links in the network topology. According to this definition, $0 \leq RE \leq 1$.

  In practice, there are good reasons to limit the number of next-hops that are installed in the forwarding table for a particular destination. Installing more than a few next-hops will not give much benefit with respect to robustness or load-balancing. It will, however, require more fast memory in the forwarding table, and may lead to the unnecessary inclusion of paths that are much longer than the shortest path.

- *Static path inflation* is the fraction of the static longest path and the shortest path given by a source towards a destination. By static path lengths, we mean the worst case path when the available route is selected at the source and at each intermediate hop towards the destination. Since our routings, ANHOR and ANHOR-SP, both produce next-hops that may not be shortest, the static path inflation metric is helpful for predicting the performance of traffic distribution. We do not expect that traffic takes significant long paths due to two reasons: introducing long latency for traffic reception at the destination and increasing load in the network.

- *Relative running time* is the fraction between the average running time of our routings and the average running time of the shortest path algorithm for all destinations for a specific network topology.

4.5.3 Robustness evaluation

Maximizing multi-path capability

Figure 4.4 shows the fraction of nodes with at least two next-hops with the different routing methods. We observe that the multi-path capability varies strongly among topologies; it is generally higher in more well-connected networks. ANHOR achieves significant improvements over ECMP and LFA in all networks. For instance, the differences are up to 78% and 28%, respectively, in AS3257 and 33% and 31%, respectively, in AS1239.

Note that the number of next-hops achieved with ANHOR is independent of link weight settings, while ANHOR-SP is constrained to including the shortest paths in the

routing. ANHOR-SP performance in most case is much closer to ANHOR compared to ECMP and LFA. For example, the differences in performance between ANHOR-SP and ECMP is approximately 67% in AS6461 and between ANHOR-SP and LFA reaches 31% in AS1239. This shows that permutation routings can give a significant gain in terms of link protection compared to existing solutions, while being compatible with the shortest path routing with realistic link weight settings.

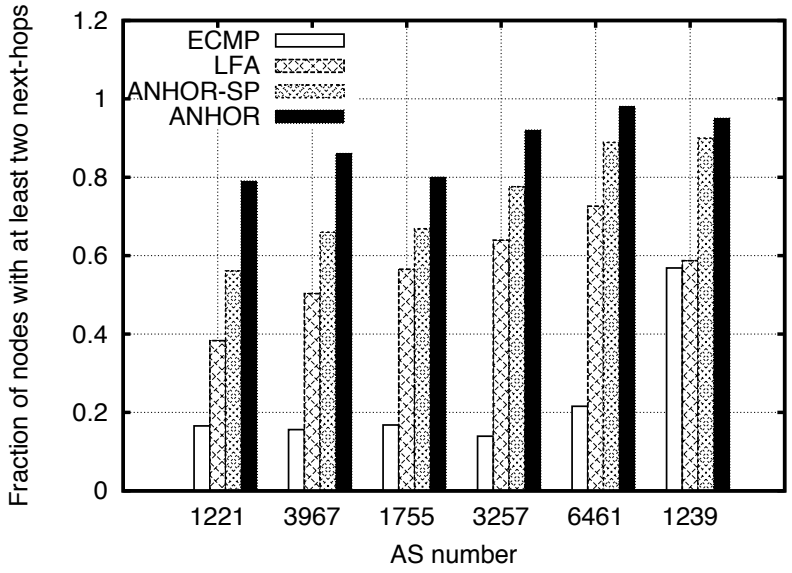


Figure 4.4: Fraction of S-D pairs with at least two next-hops.

Routing efficiency coefficient

Table 4.2 shows the RE values for three values of K in the selected topologies. The presented values are the average over all S-D pairs. We see that for all routing methods, a higher K gives a higher RE value. ANHOR and ANHOR-SP give the highest RE values, sometimes with significant improvements over ECMP and LFA even for $K = 2$. The RE values in more well-connected topologies (AS1239) are lower than those in sparse topologies. Such topologies contain high numbers of nodes with very high degrees (39% nodes has their degrees greater than 15 in AS1239), and a low K will hence exclude many valid (but often unnecessary) paths.

The high RE values of ANHOR and ANHOR-SP also imply that the numbers of next-hops of nodes distribute more evenly across the network. In other words, our proposals can avoid the case that a few nodes have high numbers of next-hops, while others might be left with only one.

Table 4.2: Routing Efficiency coefficients

		AS1221	AS1755	AS3967	AS3257	AS6461	AS1239
$K = 2$	ECMP	0.74	0.61	0.61	0.54	0.43	0.49
	LFA	0.80	0.79	0.77	0.77	0.62	0.49
	ANHOR-SP	0.86	0.84	0.85	0.76	0.67	0.58
	ANHOR	0.94	0.90	0.95	0.81	0.81	0.60
$K = 3$	ECMP	0.76	0.62	0.62	0.54	0.46	0.56
	LFA	0.86	0.90	0.87	0.82	0.77	0.57
	ANHOR-SP	0.92	0.95	0.94	0.88	0.85	0.75
	ANHOR	0.98	0.99	1.00	0.95	0.95	0.80
$K = 4$	ECMP	0.77	0.62	0.63	0.54	0.47	0.60
	LFA	0.90	0.94	0.91	0.88	0.85	0.61
	ANHOR-SP	0.96	0.99	0.97	0.93	0.93	0.85
	ANHOR	1.00	1.00	1.00	0.99	0.99	0.92

Path inflation

Next, we look at the distribution of path lengths. We focus on path lengths in terms of hop counts, since this metric is independent of the link weight settings. Figure 4.5 shows the average *path stretch* regarding K of 3 with different routings, where the length of each valid path has been normalized with the shortest path length for that S-D pair. We observe that the superior path diversity in ANHOR and ANHOR-SP comes at the cost of some path inflation, but that the average path lengths are still comparable to those of the shortest path routing. The path inflation introduced with multi-path routing can be ameliorated with more intelligent load balancing methods [38] [96].

4.5.4 Running time of the algorithms

The complexity of our proposed algorithms depends on the number of nodes and on how efficiently the size of the candidate set can be reduced. The average size of candidate set turns out to be about 18% (AS1755) to 25% (AS6461) of all nodes in our tested topologies. In addition, the average length of permutations that source node i should calculate for all destinations, $\overline{N_i}$, is $0.5 \times N$. Figure 4.6 shows the relative running time of each routing method across six topologies. The AS topologies are listed in an increasing order of number of nodes. The results are achieved with an Intel Core 2 CPU 6300 @ 1.86 GHz machine. ANHOR has low running time that is comparable to a normal ECMP routing computation. For all destinations, the total time difference is less than 10% for all topologies. As for ANHOR-SP, calculating routing permutations for all destinations take less than twice the time of ECMP.

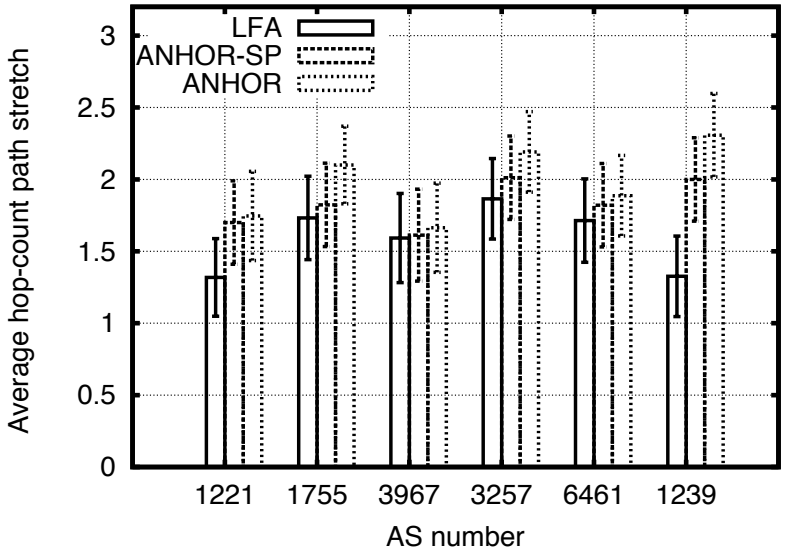


Figure 4.5: Average hop-count path stretch with $K = 3$

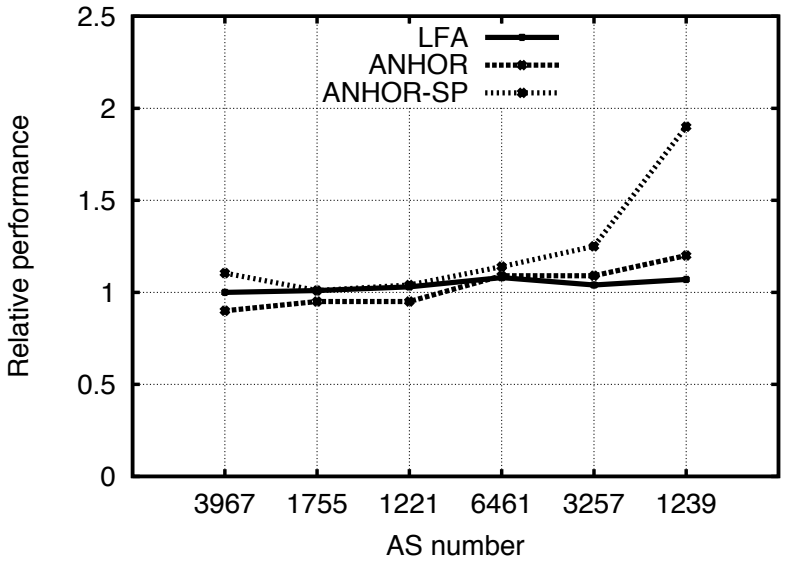
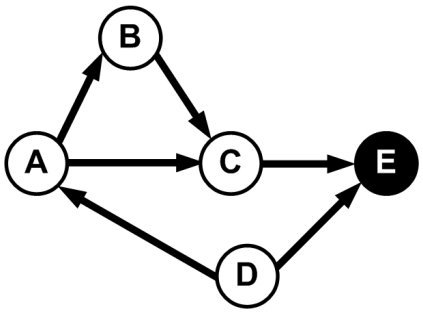


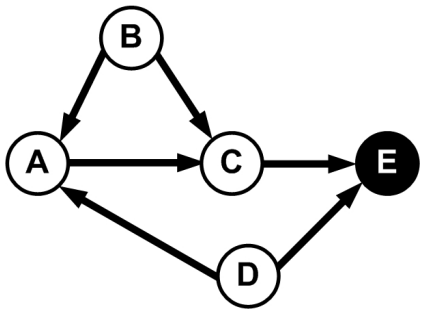
Figure 4.6: Relative running time

4.6 Node-protecting alternates

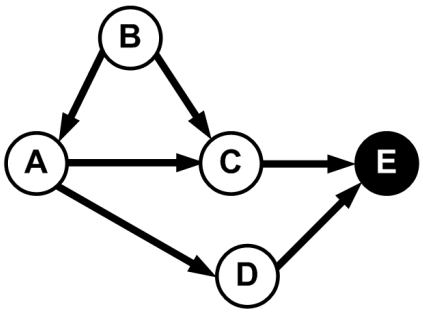
An S-D pair with at least two next-hops is robust under a single fink fault, but may not work when the failure is more extensive, e.g. a node fault. However, if one of the next-hop is a node-protecting alternate, then the S-D pair can be robust under a node fault. In this section, we improve the algorithms of ANHOR and ANHOR-SP to maximize the chance that one of the resulting next-hops given by the corresponding permutation routing is a node-protecting alternate. We describe the idea in the following example.



(a) Routing R_E: node A and node D both have two next-hops, but only D has a valid route to E when C fails.



(b) Routing R'_E: node B and node D both have two next-hops, but only D has a valid route to E when C fails.



(c) Routing R''_E: node A and node B both have two next-hops, and all nodes have valid routes to E when C fails.

Figure 4.7: Maximum protection coverage with node-protecting alternates

Routings R_E, R'_E, and R''_E in Figure 4.7(a), Figure 4.7(b), and Figure 4.7(c), respectively, all have two nodes with two next-hops towards destination E. If node C fails, only node D in both R_E and R'_E can reach the destination through $(D \rightarrow E)$. Two next-hops of node A in R_E and two next-hops of B in R'_E are interrupted after the failure of C. However, the interruption will not occur in R''_E when C is not in service. The reason is node A in R''_E has node D as its node-protecting alternate with respect to the failure.

We will show how to improve the ANHOR and ANHOR-SP algorithms to produce more node-protecting alternates among resulting next-hops.

Assume that we already have partial permutation routing $\vec{p}_i$ and domain D_{i+1} containing multiple nodes for variable p_{i+1}, we will select a node in the domain that satisfies routing constraint function (4.2). We further assume that the maximum connectivity value ($\max_{\forall v \in D_{i+1}} c[v]$) is 1. We consider such a scenario since the topology is sufficiently sparse for the algorithm not possible to distinguish the consequences of different selections if not further scrutinizing them. We observe that the node for variable p_{i+1}, say u_{i+1}, can have a valid route to the destination under the failure of the node assigned to p_i if there exists at least one connection from u_{i+1} to partial permutation $\vec{p}_{i-1}$. That means we should select a node with the maximum connectivity value to partial permutation $\vec{p}_{i-1}$. We implement `newSelect` as an enhancement of `Select` function that incorporates the procedure for computing node-protecting alternates. First of all, c_{max} is computed from domain D_i (line 1). We particularly consider the case when c_{max} is equal 1. If so, we compute the number of connections from a node to partial permutation routing $\vec{p}_{i-1}$, say $\Delta_{i-1}[v]$, for all node $v \in D_i$ in lines 1-2, and then find the maximum value of connectivity $\max_{\forall v \in D_i} \Delta_{i-1}[v]$ (c'_{max}) in line 5. The node selected is the one that that has c'_{max} (lines 6-10).

Function newSelect

Input: Candidate set D_i

Output: Node u that satisfies routing constraint functions.

```
1  c_max ← max_{∀v∈D_i} c[v]
2  if c_max = 1 then
3      for v ∈ D_i do
4          c'[v] ← Δ_{i-1}[v]
5      c'_max ← max_{∀v∈D_i} Δ_{i-1}[v]
6      while D_i ≠ ∅ do
7          an arbitrary node u from D_i
8          D_i ← D_i \ {u}
9          if c'[u] = c'_max then
10             return u
11 else
12     while D_i ≠ ∅ do
13         an arbitrary node u from D_i
14         D_i ← D_i \ {u}
15         if c[u] = c_max then
16             return u
17 return null
```

4.7 Summary

The chapter presents two routing objectives, called NHOR and NHOR-SP, for increased robustness in the traditional IP networks. Both of them are designed to work with the link-state routing protocols and aim to maximize the number of S-D pairs with at least two next-hops. However, they differ in that NHOR-SP contains the shortest path tree given by the shortest path routing algorithm.

We believe that both NHORs and NHOR-SPs are not easy to reach with moderate computational complexity even in a medium scale networks. While retaining the simplicity for the algorithm so that it can be implemented in the distributed routing systems, we propose heuristics that are based on the permutation routing framework to produce approximations for our routing objectives. The two heuristics are called ANHOR and ANHOR-SP.

The heuristics begin with formulation of routing objectives in terms of routing constraint functions at source nodes. The routing constraint functions form the basis for our selection strategy which we select the node that has the maximum connectivity value to nodes already placed in the permutation routing, and by doing that we can locally maximize the number of alternates for each of the assigned node. The heuristics also define a domain for each selection so that we can obtain linear computational complexity. The domain for ANHOR relies on the fact that any node in a feasible routing should have at least one next-hop, and therefore its computation is straightforward, and mostly based on the topological connectivity. Meanwhile, the domain for ANHOR-SP is rather complicated since it has to comply with the ordering of the given shortest path tree. It is then implicitly proven that if the domain contains only nodes that have all their shortest path next-hops placed in the permutation routing, then the domain does not break the exisiting shortest path tree. Our defined routing constraint functions and domain sets give backtrack-free algorithms for ANHOR and ANHOR-SP for all connected topologies.

We have tested our algorithms in six inferred ISP topologies from Rocketfuel project. The simulation results show that our ANHOR provides significantly higher link-protecting coverage than those of ECMP and LFA. In one topology, the performance difference between ANHOR and ECMP reaches 78% while in another topology the performance improvement over LFA is up to 31%. ANHOR-SP also outperforms ECMP and LFA, but gives lower protection rate than that of ANHOR due to the shortest path constraint in ANHOR-SP. As the result of increasing path diversity, ANHOR and ANHOR-SP also introduce path inflation through employing non-shortest paths. However, the average static path lengths measurements reveal that they are comparable to the shortest paths. Furthermore, the computational complexities of the algorithms are also close to those of ECMP and LFA.

Chapter 5

Next-hop permutation routing with Disallowed U-Turns

Previously we construct two routings that can increase the network robustness compared to LFA for the traditional IP networks where nodes have very basic forwarding functionality. In this chapter we study the case where nodes have added functionality that allows them to prohibit U-turns on back-up paths. That is, if there is a forwarding option for an incoming packet that makes the packet go back out on the same link that it came in on, then this option is prohibited. At the time of writing, it is unclear to what extent this feature is widespread in current equipments. This feature is described in the Internet standard [74], and we can demonstrate that it significantly increases the potential for fault coverage compared to the case we described in Chapter 4.

In a routing where routers are added with disallowed U-turn functionality, each source node has at least one primary next-hop and possible back-up next-hop(s). Furthermore, some of the back-up links which are called joker-links [72] can be shared between their two end nodes. Such mutual back-up operation can help augment one back-up next-hop for those end nodes which previously have a single primary next-hop. The sharing a back-up path between its two end nodes, however, should associate with the U-turn disallowance functionality in order to avoid routing loops on the back-up path or general routing loops when multiple failures occur at the same time.

Section 5.1 illustrates the working of joker-links in routing, and shows the benefit of those joker-links in expanding protection coverage for the traditional IP networks. The example and our further observations motivate two routing objectives for increased robustness by taking advantage of the joker-links. In Section 5.2, we extend the permutation routing definition in Chapter 4 so that the new definition of permutation routing in this chapter can incorporate the operation of joker-links, and result in loop-free routings with joker-links. Next, we use the permutation routing framework described in Chapter 3 to

construct permutation routings with joker-links. To do that, we have to formulate the routing objectives in form of routing constraint functions for source nodes and define two functions, Domain and Select, to govern the assignments. Luckily, those elements are similar to those defined for ANHOR or ANHOR-SP in the previous chapter. The only added element for the construction is a procedure to identify joker-links that we describe in Section 5.3. With the joker-link identification procedure, we detail necessary steps in Section 5.4 to achieve permutation routings that can realize our routing objectives. Section 5.5 is dedicated to performance evaluation of the new algorithms in terms of the resulting number of S-D pairs with at least two next-hops. Finally, we improve the algorithms to augment node-protecting alternates among the resulted next-hops.

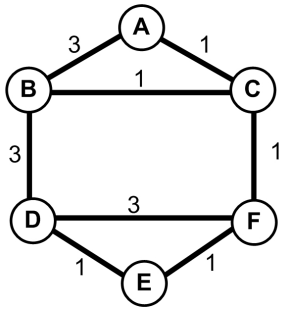
5.1 Motivation

When routers are able to disallow U-turns, then this can be exploited to let two neighboring routers use each other as back-up next-hops. We shall call such links *joker-links*, and their strength is illustrated in Figure 5.1.

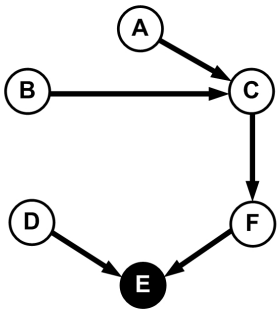
The figures show a simple network topology, with 4 different routings for destination node E. The network topology in Figure 5.1 has 6 nodes and 8 bidirectional links labeled with their link weights. Figure 5.1(b) is a shortest path tree (SPT) rooted at node E, in which no source node is protected. The method we described in the previous chapter can improve the SPT by growing *non-shortest* path branches to form a better-connected routing graph, mandatorily a directed acyclic graph (DAG). Figure 5.1(c) is such an instance. Although Figure 5.1(c) is obviously the most robust loop-free routing graph that the traditional node-based forwarding routing can provide on the given topology, two of five source nodes, node C and node D, are unprotected. We observe that node D in Figure 5.1(d) can be protected if we let node D, at the same time, take node F as its next-hop. In other words, we establish a joker-link between node D and node F, denoted by $(D \leftrightarrow F)$. Additionally, with ability to disallow U-turns the forwarding loop will not occur over joker-link $(D \leftrightarrow F)$. Specifically, when D receives packets from F over joker-link $(D \leftrightarrow F)$, D will never send those packets back over the joker-link. Instead, D will forward the packets to its primary next-hop E. We might think that node C in Figure 5.1(c) could be protected by joker-link $(C \leftrightarrow B)$. However, joker-link $(C \leftrightarrow B)$ would break the SPT-constraint when we require that the resulting routing must include the SPT. Therefore, the only way to protect all nodes in the routing towards node E is to relax the shortest path compatibility constraint and then set up a joker-link between node C and node B as in Figure 5.1(e). Two last routings are called *joker-capable* routings.

Clearly, the SPT compatibility constraint will affect the robustness of joker-capable

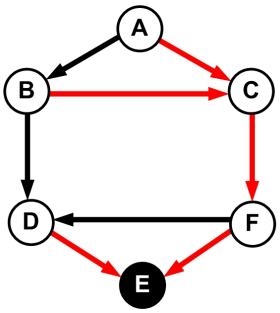
routings. We further show that topology structure also restricts fast re-routing capability of joker-capable routings. For a given topology G, single link fault coverage, denoted by $\Gamma(G)$, is the fraction between the number of protected S-D pairs and all S-D pairs towards all destinations.



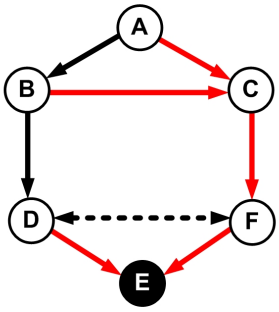
(a) Network topology



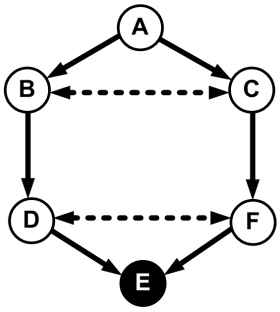
(b) An SPT rooted at node E



(c) An NHOR-SP towards E.



(d) A joker-capable routing towards E that contains the SPT



(e) A joker-capable routing towards E that does not contain the SPT

Figure 5.1: Joker-links can help increase routing robustness.

Theorem 5.1. *The* $\Gamma(G)$ *of a joker-capable routing on a ring topology* G *with* n *nodes is* $\frac{2}{(n-1)}$ $(n \geq 3)$.

Proof. There are n nodes and n links in a ring topology with n nodes ($n \geq 3$). A joker-capable routing on that topology towards a certain destination d has $(n-1)$ source nodes, all of which may have one primary next-hops. Hence, there exists one link connecting two nodes i and j $(i, j \neq d)$ that can be set as a joker-link. In such joker-capable routing, there will have two nodes protected out of $n-1$ nodes. As a result, we obtain $\Gamma(G) = \frac{2}{(n-1)}$ for all destinations. □

It is obvious that $\Gamma(G) < 1$ when $n \geq 4$. Thus, the ring topology with 4 nodes is the smallest topology unit that restricts full single link fault coverage of joker-capable routings. In practice, joker-capable routings also could not provide the ideal fault coverage in more complicated topologies because of the presence of ring structures. Consequently, our goal is to construct joker-permutation routings with minimum numbers of unprotected nodes. This becomes our main routing objectives which will be presented shortly.

As mentioned earlier, a joker-link is simply a bi-directional forwarding link in a routing for a destination. More generally, we define it as follows:

Definition 5.1. *A joker-link is a bi-directional forwarding link in a routing for a destination that connects two nodes, both of which currently have the same number of primary next-hop(s).*

It is obvious that a joker-link helps its two end nodes both have at least two next-hops. Therefore, identifying all possible joker-links will increase the robustness for a routing. That leads us to the following optimization criterion for Joker-capable Next-Hop Optimality Routing (JNHOR):

Definition 5.2. *Given a topology, a JNHOR is a joker-capable routing that maximizes the number of S-D pairs that have* at least *two next-hops towards a destination.*

Note that this definition makes it meaningless for a node to have more than one joker-link for a given destination. For example, in a routing towards destination d, assume that node i has one primary next-hop and two joker-links with node j and node k, each of which also must have one primary next-hop. In other words, node i ends up with three next-hops while j and k both have two next-hops or three nodes, i, j, and k all have at least two next-hops. If we replace joker-link $(i \leftrightarrow j)$ with directed link $(j \rightarrow i)$, the three nodes still have at least two next-hops. In the following we therefore restrict our search to routings where no node has more than one joker-link for a given destination. Additionally, for simplicity we assume that next-hops which are not on joker-links are primary next-hops.

A JNHOR in general will not always be compatible with the shortest path routing. However, in practice such SPT inclusion constraint is required for traffic engineering purpose, e.g. shortest paths likely provide low transmission latency for voice or video traffic. For that reason, we also define the Shortest Path compatible JNHOR (JNHOR-SP):

Definition 5.3. *Given a topology, a JNHOR-SP is a joker-capable routing that maximizes the number of S-D pairs that have* at least *two next-hops while containing the SPT towards a destination.*

Following definitions of the joker-capable routing and its two instances, JNHOR and JNHOR-SP, two questions arise with regard to joker-links:

1. How to identify which links to be joker-links for maximizing the protection coverage?

2. How to avoid typical loops as the disallowed U-turn functionality can avoid single hop loops on joker-links under multiple failures in the network?

We start by giving a firm definition of what we mean by joker-capable routing in terms of permutation routing.

5.2 Permutation routings for joker-capable routings

The joker-capable routing differs from the traditional routing in that the former includes joker-links which require disallowed U-turn forwarding. Hence, in order to represent a joker-capable routing in terms of a permutation routing we depict in Chapter 3, we need to modify its forwarding rule to incorporate the special forwarding for joker-links. The definition of the new forwarding rule is given as follows:

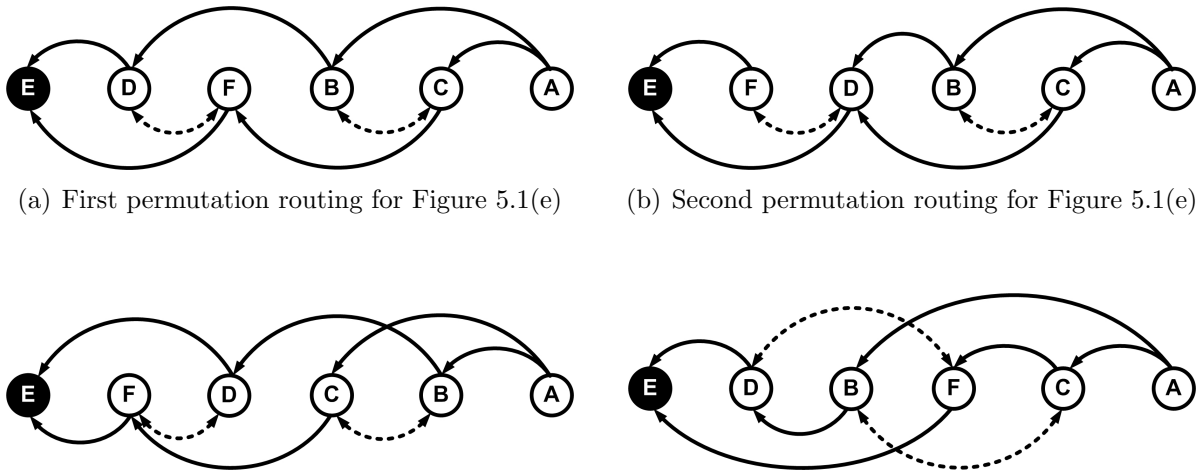
Definition 5.4. *Given permutation routing $\mathcal{P}$ of N nodes that represents for a joker-capable routing for destination d, :*

1. *A node can forward its packets to its all primary next-hops that occur before it in $\mathcal{P}$.*

2. *A node only forwards its packets over its joker-link to the back-up next-hop only when there is no primary next-hop available.*

3. *When a node receives a packet over its joker-link, it would never send the packet back over that joker-link.*

Condition 1 of Definition 5.4 is the greedy forwarding rule of the typical permutation routing, and will result in all-path routing. Conditions 2 and 3 are added to undertake the unusual forwarding operation of the joker-links in the routing. The second condition implies that the next-hop over a joker-link of a node should always have the back-up role and be only used in failure scenarios. And if the back-up next-hop is activated, it should follow the forwarding rule designed for it in Condition 3, and by doing that routing loops over joker-links can be avoided. The third condition is called disallowed U-turn functionality, and can further guarantee loop-free forwarding under multiple independent failures. We will present a proof for the loop-free property later, but now continue our discussion on the definition.



(a) First permutation routing for Figure 5.1(e)

(b) Second permutation routing for Figure 5.1(e)

(c) Third permutation routing for Figure 5.1(e)

(d) Forth permutation routing for Figure 5.1(e)

Figure 5.2: Possible permutation routings for a joker-capable routing.

In general, we can construct more than one permutation routing for a given joker-capable routing. From the first achieved permutation routing, $\mathcal{P}$, we can generate a new one, $\mathcal{P}'$, by simply swapping two successive nodes connecting by a joker-link or two successive nodes that are not connected by a link. We illustrate that fact in the following example. Sequence $\{E\ D\ F\ B\ C\ A\}$ in Figure 5.2(a) is a permutation routing for the joker-capable routing in Figure 5.1(e). By swapping D and F which connect each other through a joker-link, we have another valid permutation routing in Figure 5.2(b). We continue to do that with nodes B and C to generate another one in Figure 5.2(c). The permutation routing in Figure 5.2(d) is the result of swapping node B and node F of the permutation routing in Figure 5.2(a) which do not share a common link. There may exist numerous permutation routings for a single joker-capable routing on a medium

scale topology. Consequently, it would be difficult for nodes to produce a consistent permutation routing in the distributed routing systems, even though the nodes have the same view of the network topology. Thus, we find necessary to limit the number of possible permutation routings by adding a restriction on the sequence. We suggest that two nodes that share a joker-link should be placed next to each other in the sequence. This design is simple and will be easily followed by all nodes. Now we provide a new definition for the permutation routing.

We have network graph $G = (V, E)$ where V is the set of nodes and E is the set of bidirectional links, provided that each bidirectional link is modeled as two directed links. Assume that on G we can build a joker-capable routing, denoted by $R_d = (V, E_d)$ where E_d is the set of routing links (directed links), and $E_d \subset E$. In R_d, node j_1 is called a *primary* next-hop of node i if there exists a directed link between node i and node j_1, $(i \rightarrow j_1)$, and node j_2 is a *back-up* next-hop of i if there exists a joker-link between node i and node j_2, $(i \leftrightarrow j_2)$. The permutation routing that represents R_d is defined as follows:

Definition 5.5. *Permutation $\mathcal{P}$ is a* permutation routing *for R_d if*

1. *all primary next-hops of each node occur before it in $\mathcal{P}$. If we write $j_1 < i$ to denote that j_1 occurs before i in $\mathcal{P}$, then $\forall (i \rightarrow j_1) \in E_d : j_1 < i$.*

2. *two nodes connected by a joker-link are adjacent in $\mathcal{P}$. We write $j_2 = i$ if $\exists (i \leftrightarrow j_2) \in E_d : j_2 = i$.*

According to this definition, node d will always be the first node in the permutation routing for R_d since the destination does not have any next-hop, either primary or back-up, in that routing. Furthermore, such permutation routing for R_d can also be computed using the topological sort algorithm [15] for *partially* ordered set [73]. After we define the permutation routing for a joker-capable routing, we proceed with answering the question of how to identify all possible joker-links in the routing.

5.3 Joker-link identification

Before we construct permutation routings for two particular joker-capable routings, JNHOR and JNHOR-SP, with our algorithm framework described in Chapter 3, we have to formulate each routing objective by defining a routing constraint function for every node. Like what we have done in the previous chapter, we investigate the relationship between partial permutation $\vec{p_i}$ and candidate set D_{i+1} of variable p_{i+1}.

Let $\vec{p_i}$ be the partial permutation routing of i nodes for destination d, and by applying the forwarding rule defined in 5.4 on $\vec{p_i}$, we achieve a joker-capable routing sub-graph

towards node d. We denote by $R_d^i = (V(\vec{p}_i), E_d(\vec{p}_i))$ such a sub-graph where $V(\vec{p}_i)$ is the set of i nodes in $\vec{p}_i$ and $E_d(\vec{p}_i)$ is the set of directed links between connecting pairs of nodes in $V(\vec{p}_i)$. In the next stage, we select a node from D_{i+1}, and assign it to variable p_{i+1} such that R_d^{i+1} induced from $\vec{p}_{i+1}$ is the most robust. Obviously, the right selection is the one with the maximum connectivity value to $\vec{p}_i$, and therefore the routing constraint function for the assignment is exactly the same as (4.2) that is rewritten as follow:

$$C(u) = \begin{cases} \textbf{True} & \text{if } c[u] = \max_{\forall v \in D_{i+1}} c[v] \\ \textbf{False} & \text{otherwise} \end{cases} \tag{5.1}$$

where $c[u]$ is a counter that keeps track the number of outgoing links from u to $\vec{p}_i$. In other words, $c[u]$ corresponds to the number of next-hops node u will have if it is selected.

The description is so far identical to the way we construct the routing constraint functions for ANHOR. That is relevant since JNHOR and ANHOR have a common routing objective. However, JNHOR can improve the robustness by identifying as many joker-links as possible. By looking into set $\mathcal{D}_{i+1}$ that contains all nodes satisfying (5.1), we see two cases:

- $\mathcal{D}_{i+1}$ includes multiple nodes that do not share any common link.
- $\mathcal{D}_{i+1}$ includes multiple nodes, some of which connect to each other.

In the first case, we place the node with highest ID in the permutation and proceed to the next assignment. But in the latter, we assign the link that connects two nodes as the joker-link and place the two nodes in the permutation one right after another. We develop `Select` function to realize this procedure.

Let assume that we are already provided with domain D_i, from which we make our selection. First of all, the function calculates the maximum next-hop counter of the nodes in the domain (line 1). Then, a loop goes through the domain to extract nodes whose next-hop counters are maximum, and groups those nodes into $\mathcal{D}_i$, which we call the sub-domain (lines 2-5). The selection strategy for the JNHOR differs from its counterpart for the ANHOR in the procedure to identify joker-links which is implemented from line 6 to line 16. In this procedure, function `findJokerSet` is responsible for inspecting the connectivity of nodes in the sub-domain to identify a *single* joker-link. We defer to give details of function `findJokerSet` for now since the function may take a specific routing objective into account. After `findJokerSet` is called, it will return two nodes in `jokerSet` whose connection is later set the joker-link and raise flag `isJoker` (line 9) or empty `jokerSet` and subsequently we do not need to set the flag.

If `jokerSet` is not empty, meaning that it contains two nodes, v_1 and v_2 (we further assume that v_1's ID is greater than v_2's ID), function `Select` will pick from it a node

with higher ID and assign it to the permutation routing. At the same time, the function sets the next-hop counter of the other node with a significant large number, say L, which guarantees that the assigned node in the next assignment is v_2. The reason is that with a large next-hop counter, v_2 would be the single node in the sub-domain, and will definitely be selected according to the selection rule. When `jokerSet` is empty, the selection is determined exactly like that of ANHOR (lines 18-20).

Function Select

```
1  c_max ← max_{∀v∈D_i} c[v]
2  D_i ← ∅
3  for v ∈ D_i do
4      if c[v] = c_max then
5          D_i ← D_i ∪ {v}
6  isJoker ← false
7  jokerSet ← findJokerSet()
8  if jokerSet ≠ ∅ then
9      isJoker ← true
10 if isJoker = true then
11     for v_1, v_2 ∈ jokerSet do
12         u ← v_1
13         c[v_2] ← L
14     return u
15     D_i ← ∅
16     jokerSet ← ∅
17 else
18     an arbitrary node u from D_i
19     return u
20     D_i ← ∅
21 return null
```

5.4 Constructions of JNHOR and JNHOR-SP

5.4.1 JNHOR

In JNHOR, computing domains for variables is straightforward. Domain D_{i+1} for variable p_{i+1} will consist all nodes that have at least one neighboring node already placed in $\vec{p}_i$. Otherwise, the resulting permutation routing would be disconnected. Therefore, D_{i+1} can

be written in the recursive form as follow:

$$D_{i+1} = D_i \cup \{\ v \in V_i \mid (v,u) \in E\} \setminus \{u\} \tag{5.2}$$

where u is the node that has been assigned to variable p_i in i-th assignment and V_i is a set of nodes from V, excluding all nodes in $\vec{p_i}$ and D_i.

Function Domain-JNHOR

```
1 D_i ← D_i \ {u}
2 for v ∈ V_i such that (v,u) ∈ E do
3 |  D_i ← D_i ∪ {v}
4 for v ∈ D_i such that (v,u) ∈ E do
5 |  c[v] ← c[v] + 1
6 V_i ← V_i \ D_i
```

In JNHOR, the joker-link is the link connecting two nodes in the sub-domain. To this end, function `findJokerSet` examines whether there exists any connecting pair of nodes in $\mathcal{D}_i$. If there exists such a single pair, we immediately assign them to `jokerSet`, and set flag `isJoker` true. The pseudo code of function `findJokerSet` is shown below. If there are multiple connecting pairs in $\mathcal{D}_i$, we only need to select a pair whose sum of IDs is the highest among pairs. We do that regarding the above agreement that limiting unnecessary joker-links benefits the fast re-routing management.

Function findJokerSet

```
1 for ∀v1, v2 ∈ D_i do
2 |  if ∃(v1, v2) ∈ E then
3 |  |  jokerSet ← {v1, v2}
```

5.4.2 JNHOR-SP

More restrictively, domain D_{i+1} of an JNHOR-SP includes all nodes that *both* have at least one neighboring node already placed in $\vec{p_i}$ and do not violate the ordering of nodes in the existing SPT. Let $c^*[v]$ be the number of shortest path next-hops already appeared in $\vec{p_i}$ and $n^*[v]$ be the total number of shortest path next-hops calculated from the SPT of node v, domain D_{i+1} for variable p_{i+1} follows the recursive relation:

$$D_{i+1} = D_i \cup \{v \in V_i \mid c^*[v] = n^*[v]\} \setminus \{u\} \tag{5.3}$$

where u is the node that has been assigned to variable p_i in i-th assignment and V_i is a set of nodes from V, excluding all nodes in $\vec{p_i}$ and D_i.

Since formulas (5.3) and (4.3) are the same, function `Domain` for JNHOR-SP shown below is identical to that of ANHOR-SP. The function, therefore, composes of three loops. The first loop goes through remaining nodes in the node set and the current domain ($(V_i + D_i)$ or simply $\overline{\mathcal{P}}$) to update the shortest next-hop counter for each of considered node. Note that the shortest next-hop counter of a node is aimed to keep track the number of next-hops on the SPT that are already placed in the permutation of that node. The second loop also examines each node in $\overline{\mathcal{P}}$ to update next-hops counters for the nodes. Different from the shortest next-hop counter, the next-hop counter of a node is the number of next-hops, including those on the SPT. The last loop limits its checking in remaining nodes of the node set, and moves those nodes that satisfy formula (5.3) into the the domain.

Function `findJokerSet` in JNHOR-SP is not so simple as that of JNHOR since either directions of the joker-links should not include any of directed links of the SPT. As explained in Section 5.1, we suppose that all next-hops on the SPT will be set as primary next-hops. Meanwhile next-hops on joker-links are all secondary next-hops, and therefore should be separated from the primaries. To this end, in existence of a connecting pair, say (v_1, v_2) in the sub-domain, we should further check if any of the directed links $(v_1 \to v_2)$ and $(v_2 \to v_1)$ is on the SPT. If none of the directed links is on the SPT, we allow that pair to enter `jokerSet`. Otherwise the `jokerSet` is empty or we keep examining other connecting pairs if there are multiple ones.

Function Domain-JNHOR-SP

```
1 D_i ← D_i \ {u}
2 for v ∈ P̄ such that (v → u) ∈ E*_d do
3 |  c*[v] ← c*[v] + 1
4 for v ∈ V_i such that (u, v) ∈ E do
5 |  c[v] ← c[v] + 1
6 for v ∈ V_i do
7 |  if c*[v] = n*[v] then
8 |  |  D_i ← D_i ∪ {v}
9 V_i ← V_i \ D_i
```

The constructions of JNHOR and JNHOR-SP are similar to those of the typical permutation routings, ANHOR and ANHOR-SP, respectively, and the added joker-link identification procedure works with either empty or non-empty *JokerSet*. Therefore, the

Function findJokerSet

```
for ∀v1, v2 ∈ D_i do
    if ∃(v1, v2) ∈ E ∧ (v1 → v2) ∉ E*_d ∧ (v2 → v1) ∉ E*_d then
        jokerSet ← {v1, v2}
```

algorithms for both JNHOR and JNHOR-SP will also be backtrack-free. In practical networks, the size of $\mathcal{D}_i$ is typically small ($|\mathcal{D}_i| << N$). The computational complexity for N permutations towards N destinations would be $O(M+N)N$ for a sparse topology and $O(N^3)$ for a dense one where M is the cardinality of the link set E.

5.4.3 Loop-freeness

First, we are facing the single-hop loop problem caused by a joker-link whose all primary next-hops of both end nodes are simultaneously unavailable. Fortunately, routers with disallowed U-turn functionality can prevent routing packets from being sent back to on the same interface it was received, and therefore single-hop loops are avoided. Based on that simple rule, we further prove that our JNHOR (or JNHOR-SP) is free of typical loops when the network suffers from multiple independent failures.

Proposition 5.1. *Given a topology, a JNHOR (or a JNHOR-SP) towards a destination where the nodes are equipped with disallowed U-turn functionality will be free of all loops.*

Proof. Assume that there is a permutation routing with joker-links, say $\mathcal{P}$, which suffers from multiple failures on the primary links. We write $j < i$ to denote that j occurs before i in $\mathcal{P}$. First observe that immediately after a hop along a joker-link backward in the permutation routing from node i to node i', there must follow a hop along a primary link to a node k such that $k < i$ (see Figure 5.3). In the same manner, if k faces another link failure, it will forward the packet backward to k' through its joker-link and from k' the packet will be forwarded to m such that $m < k$. Due to the loop-free characteristic of the permutation routing and the observation that the packet moves closer to the destination after it has to travel backward over the joker-link and forward over the primary link, the packet finally reaches the destination if those failures do not partition the network.

□

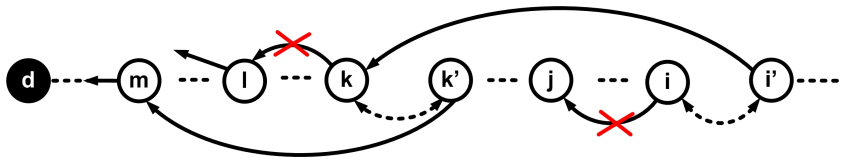


Figure 5.3: A permutation routing with joker-links under multiple failures

5.5 Performance analysis

We assess the path diversity of JNHOR and JNHOR-SP in comparison of their counterparts, ANHOR and ANHOR-SP, respectively, and two currently deployed methods, ECMP and LFA. Clearly, in order to increase path diversity, we must minimize the number of S-D pairs with one next-hop.

5.5.1 Evaluation setups

Network topologies

We select six representative network topologies from Rocketfuel project [80] for our evaluations. For each topology, we remove all nodes that will not contribute on routing (e.g. single degree node). The refined topologies are bi-connected graphs, listed in Table 5.1 in increasing order of their average node degrees.

Table 5.1: Refined network topologies

AS	Name	Nodes	Links	Avg. Degree
1221	Telstra(au)	50	97	3.88
3967	Exodus(us)	72	140	3.89
1755	Ebone(eu)	75	149	4.00
3257	Tiscali(eu)	115	282	4.90
6461	Abovenet(us)	129	362	5.60
1239	Sprint(us)	284	941	6.62

Traffic matrices and link weight settings

For each topology, we first create a traffic matrix (TM) by using the simplified gravity model [56]. We apply the local search heuristic [24] to obtain an optimized link weight setting with respect to the TM. Since the heuristic does not well scale for a large topology, we assign unit link weights for AS1239.

5.5.2 Path diversity

Figures 5.4-5.9 show fractions of protected S-D pairs with six routing methods in the six network topologies. We observe that JNHOR provides up to more than 99% of protected nodes in most tested topologies, except for AS1221 with 95%. In addition, JNHOR performs significantly better than ANHOR. For instance, in AS1221 JNHOR offers 95% protection coverage while that of ANHOR is only 79%. Meanwhile, JNHOR-SP and ANHOR-SP are only slightly different in the topologies. The remarkable difference happens in AS3969 where JNHOR-SP and ANHOR-SP provide 76% and 66%, respectively, S-D pairs with at least two next-hops. For all topologies, ECMP and LFA give much less protection coverages than either JNHOR or JNHOR-SP. Those figures also show that the SPT constraint property in JNHOR-SP substantially reduces the number of joker-links. Consequently, in terms of protected S-D pairs JNHOR-SP, despite performing better than ANHOR-SP, is lower than ANHOR. Those results confirm that different structures of SPTs for different destinations and the constraint which joker-links are not allowed to contain SPT links contributes to the low improvement of JNHOR-SP.

We further investigate the impact of the SPT constraint by illustrating the number of unprotected nodes for each destination of JNHOR-SP and JNHOR in AS1239 (Figure 5.9). JNHOR has a stable number of unprotected nodes towards each destination (from 3 to 6 unprotected nodes). These good results show that JNHOR significantly mitigates the impact of topologies' structure on routings. For JNHOR-SP, the number of unprotected nodes, however, varies strongly for each destination in a topology. For instance, in AS1239 the destination with ID of 104 has 14 unprotected nodes while the destination with ID of 247 has 62 unprotected nodes. Different structures of SPTs for different destinations and the constraint, which joker-links are not allowed to be SPT links, contributes to the uneven protection coverages across the destinations.

5.5.3 Running time of the algorithms

We measure the relative running time of JNHOR and JNHOR-SP to ECMP across six topologies with an Intel Core 2 CPU 6300 @ 1.86 GHz machine. Since JNHORs and ANHORs are similar in their computations, the running times of both methods are not much different. For all destinations, the relative running times between JNHOR and ECMP are slightly more than 1.5 and those of between JNHOR-SP and ECMP are less than 3.

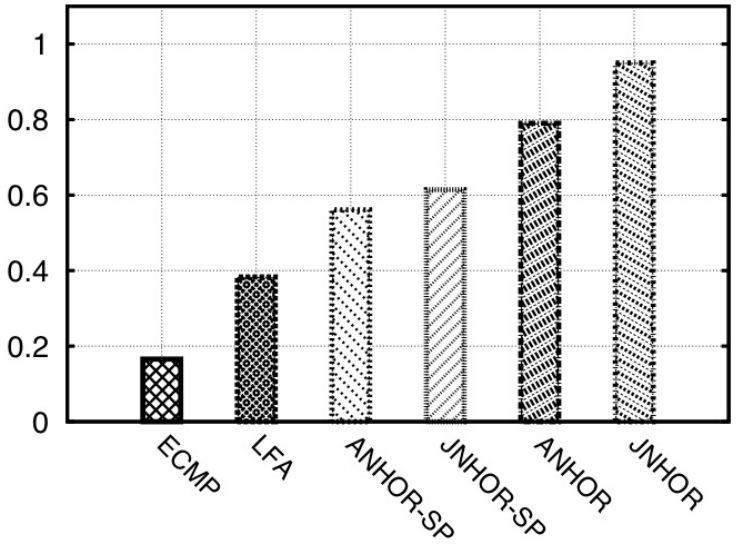


Figure 5.4: Fraction of protected S-D pairs in AS1221

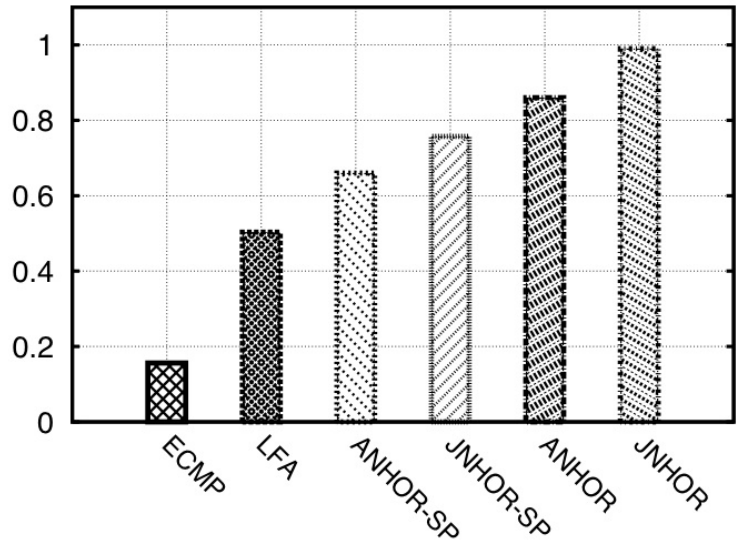


Figure 5.5: Fraction of protected S-D pairs in AS3967

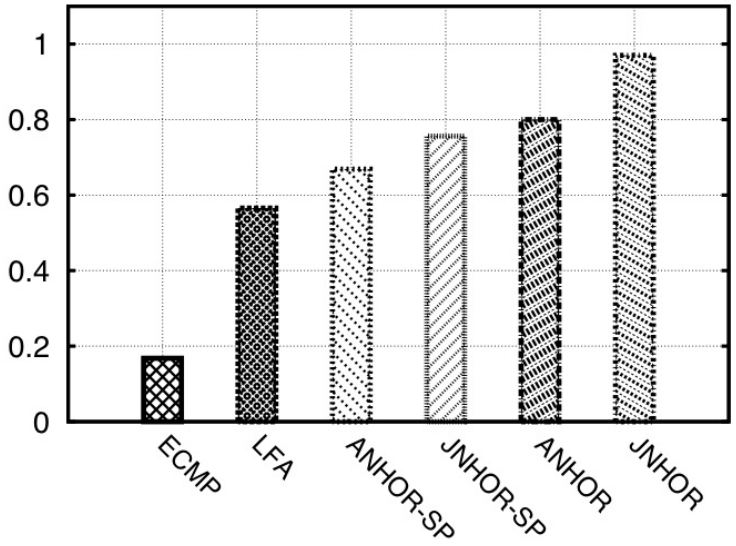


Figure 5.6: Fraction of protected S-D pairs in AS1755

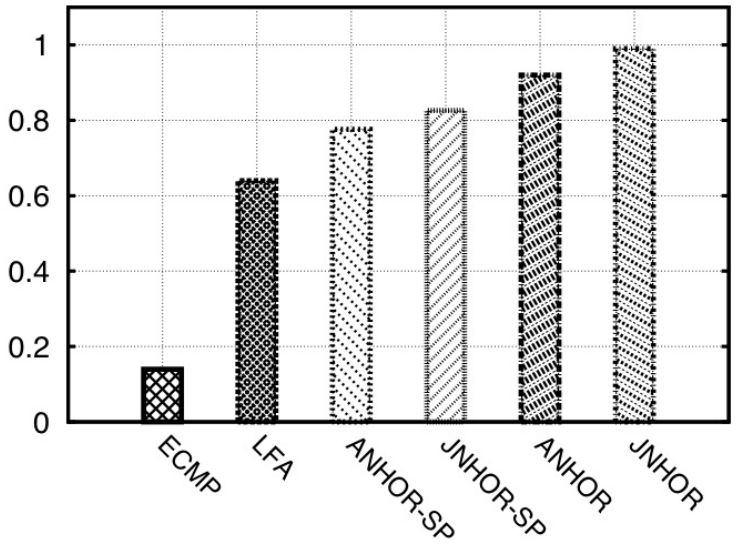


Figure 5.7: Fraction of protected S-D pairs in AS3257

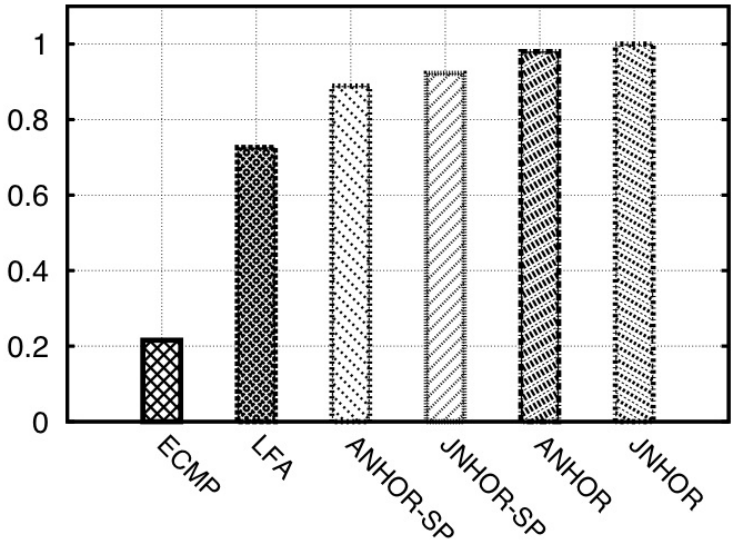


Figure 5.8: Fraction of protected S-D pairs in AS6461

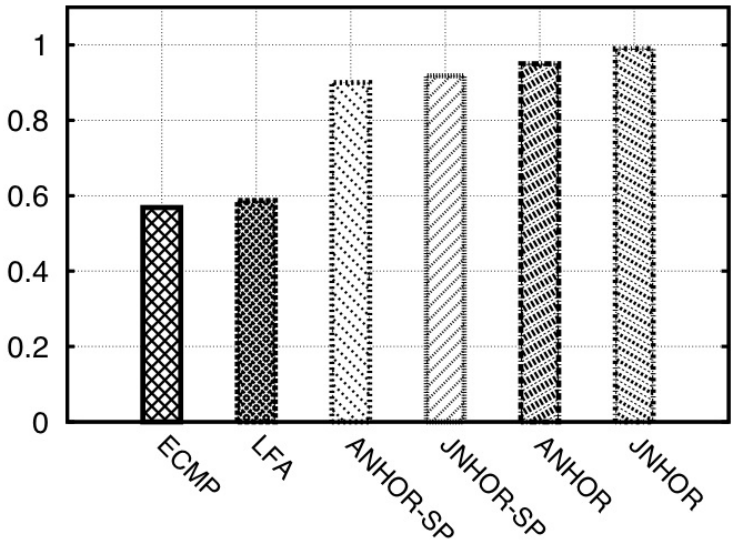


Figure 5.9: Fraction of protected S-D pairs in AS1239

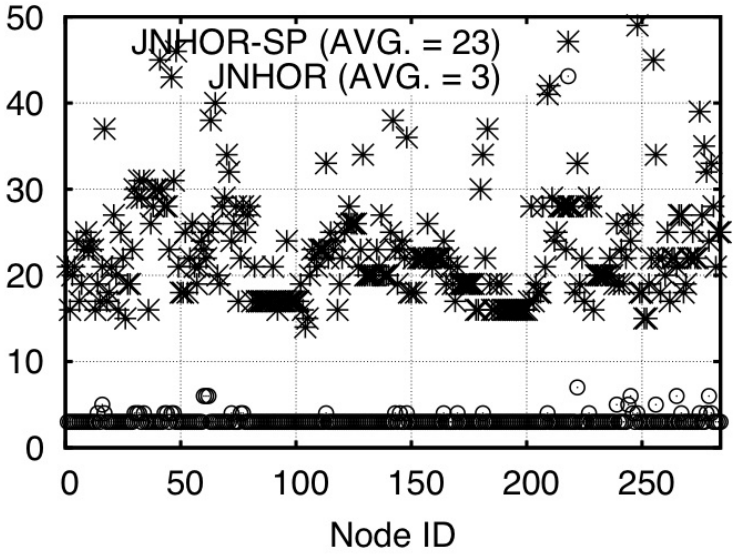


Figure 5.10: Number of protected nodes for different destinations in AS1239

5.6 Discussion

5.6.1 Optimality

We specify earlier that the topology structure may pose limitation on failure protection. However, we show in the following proposition that if the topology allows to construct a routing providing full coverage of single link faults, our algorithm will find the corresponding permutation routing.

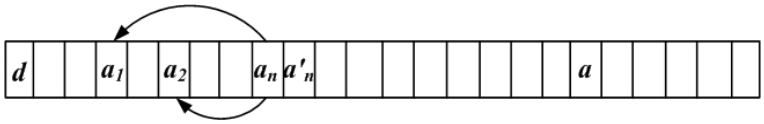
Proposition 5.2. *Assume that our algorithm has resulted in permutation routing $\mathcal{P}$ without backtracking for a given destination d. If there exists a permutation $\mathcal{P}'$ that gives 100% coverage for destination d, then $\mathcal{P}$ will also give 100% coverage.*

Proof. Assume that $\mathcal{P}$ does not give 100% coverage. Then at one point, our algorithm will insert a node a into a position in the permutation, such a does not have two routing alternatives. We shall show that in that case the algorithm would have chosen another node, a_n, instead of a, giving a contradiction.

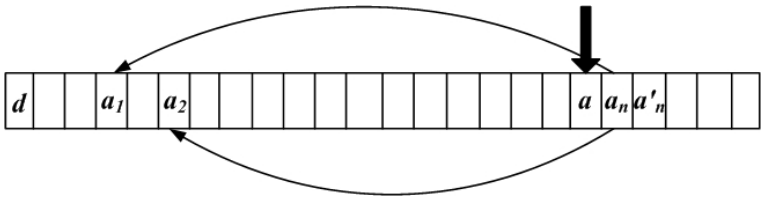
Consider the position that a has in $\mathcal{P}'$. Clearly a has at least two routing alternatives here. Since a has only one routing alternative in $\mathcal{P}$ there must be some nodes that are placed before a in $\mathcal{P}'$ and that had not been placed when a was in placed $\mathcal{P}$.

Now identify a_n to be the earliest node placed before a in $\mathcal{P}'$ and that is placed behind a in $\mathcal{P}$. If a_n has a joker-link in $\mathcal{P}'$, we call it a'_n. Now if a'_n does not exist, all of the next hops of a_n in $\mathcal{P}'$ have been placed before a in $\mathcal{P}$. But then a_n has at least two next hops

already placed in $\mathcal{P}$ thus should have gotten the position that a got because the algorithm will always select a node with 2 next hops before a node with 1 next hop. If a'_n exists, then it is either placed before a in $\mathcal{P}$, in which case a_n has two next hops already, or it should have been placed together with a_n with a joker-link between them.



(a) $\mathcal{P}'$



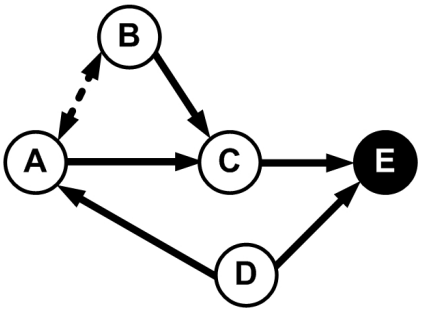
(b) $\mathcal{P}$

Figure 5.11: Illustration of the proof of the Proposition

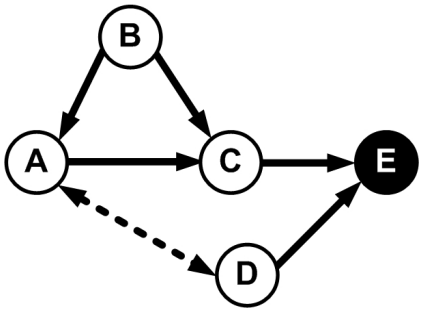
□

5.6.2 Node-protecting alternates

By identifying joker-links, we can improve protection coverage for more S-D pairs. However, in some cases the identified joker-links would not be efficient when facing a node failure. We illustrate the observation in Figure 5.12.



(a) R_E: Only node D has a valid path to E when node C fails.



(b) R'_E: Other nodes have valid paths to E when node C fails.

Figure 5.12: Joker-links and node-protecting alternates

The two joker-capable routings, R_E in Figure 5.12(a) and R'_E in Figure 5.12(b), both have three S-D pairs with two next-hops, two of which have back-up next-hops through the joker-links. In R_E, if node C fails, only node D has a valid path to the destination. Nodes A and B, despite having two next-hops, are unreachable to the destination. The reason is that none of their next-hops is the node-protecting alternate. The interruption, however, would not happen if we arrange the joker-link as in R'_E. Specifically, any failed node in the R'_E does not stop other nodes from reaching the destination. From the idea of the example, we attempt to increase the robustness of JNHOR and JNHOR-SP by more carefully assigning joker-links.

Assume that we already have partial permutation $\vec{p_i}$ and sub-domain $\mathcal{D}_{i+1}$ contains multiple nodes that satisfy routing constraint function (5.1). We observe that the node for variable p_{i+1}, say u_{i+1}, can have a valid route to the destination under the failure of the node assigned to p_i if there exists at least one connection from u_{i+1} to partial permutation $\vec{p}_{i-1}$. That means we should select a node in $\mathcal{D}_{i+1}$ with the maximum connectivity value to partial permutation $\vec{p}_{i-1}$. We implement new `findJokerSet` that incorporates the added procedure for computing node-protecting alternates. First of all, the connectivity value from a node in the sub-domain to partial permutation $\vec{p}_{i-1}$, say Δ_{i-1}, is computed in lines 1-2. Assume that $v_1 \in \mathcal{D}_i$ has c'_{max} ($c'_{max} = \max_{\forall v \in \mathcal{D}_i} c'[v]$), we retrieve v_1 in lines 4-8, and assign the joker-link to link (v_1, v_2) if $v_2 \in \mathcal{D}_i$. If there does not exist such v_2, the function returns a special `jokerSet` which has only v_1 (lines 9-13).

Function findJokerSet

Input: Sub-domain $\mathcal{D}_i$

Output: jokerSet (two nodes connecting through a link)

```
1  for v ∈ D_i do
2      c'[v] ← Δ_{i-1}[v]
3  c'_max ← max_{∀v∈D_i} Δ_{i-1}[v]
4  while D_i ≠ ∅ do
5      an arbitrary node v_1 from D_i
6      D_i ← D_i \ {v_1}
7      if c'[v_1] = c'_max then
8          return v_1
9  for ∀v_2 ∈ D_i do
10     if ∃(v_1, v_2) ∈ E then
11         jokerSet ← {v_1, v_2}
12     else
13         jokerSet ← {v_1, ∅}
14 return null
```

5.7 Summary

Joker-links in a routing towards a specific destination are links that allows packets to travel over it on both directions. In addition, a joker-link connects two nodes that have the same number of primary next-hops towards the destination. A routing that possesses joker-links is called a joker-capable routing. In the joker-capable routing, each router must install disallowed U-turn functionality that prevents packets from being forwarded back to on its incoming interfaces.That is when a packet enters a node over an interface, the node will never send it back over that interface in any circumstances. Disallowed U-turn functionality permits two nodes to share a back-up path without creating routing loops.

The joker-capable routing can increase routing robustness since it can overcome the last-hop problem and reduce the number of S-D pairs with a single next-hop. However, in general 100% protection coverage is not always reached for this type of routing. Hence this chapter presents two routing objectives, JNHOR and JNHOR-SP, which aim to maximize number of S-D pairs with at least two next-hops. JNHOR and JNHOR-SP promise to offer higher protection coverage than those of ANHOR and ANHOR-SP, respectively, presented in Chapter 4.

We construct JNHOR and JNHOR-SP by producing their corresponding permutation routings. To do that, we first modify the original definition of permutation routing in its forwarding rule so that the new permutation routings can be used to represent joker-capable routings. Next we formulate the routing objectives in terms of routing constraint functions at source nodes. The functions are similar to those of ANHORs since they all aim to maximize the number of S-D pairs with at least two next-hops. The main difference in their constructions lies in the joker-link identification procedure of JNHORs. Correspondingly, a link is assigned to be a joker-link if its two end nodes have the same maximum number of primary next-hops towards a destination. There is a subtle difference in identifying joker-links in JNHOR and JNHOR-SP. That is in the latter the joker-links should not coincide with the primary links on the SPT. The candidate sets defined for each assignment in JNHOR and JNHOR-SP are identical to those of ANHOR and ANHOR-SP, respectively, and give backtrack-freeness for the algorithms.

We evaluate JNHOR and JNHOR-SP on six ISP topologies with different node degrees. In most topologies, JNHOR provides more than 99% of S-D pairs with at least two next-hops. JNHOR performs better than JNHOR-SP since the shortest path constraint limits the number of joker-links. However, JNHOR-SP is better than ANHOR-SP which is designed not to accept joker-links. We further prove that JNHOR can achieve 100% protection coverage if the topology allows.

Chapter 6

Interface based permutation routing

The case we consider in this chapter is inspired by the fact that many present day routers keep separate copies of the routing tables in their line-cards [11]. The rationale behind this is that table lookup is parallelized so that a central routing table does not become a bottleneck for router performance. This can, however, be exploited by having different routing tables in each line-card, as was done in Nelakuditi et. al. in [58]. Such distributed table structure forms the basic idea for the interface-specific forwarding where forwarding of a packet at a node to its next-hop relies on not only the destination but also the incoming interface (incoming link) over which the packet enters to the node. We call a routing method that employs the interface-specific forwarding the interface-based routing method. In this chapter, we use the permutation routing concept and its construction framework to create an interface-based routing from a given network topology. Our method can install multiple next-hops for an incoming interface, including next-hops on the shortest path tree and non-shortest-path next-hops, all of which can be used for load-balancing in the fault-free case or fast re-routing under failures. Especially, the method is free of routing loops when the network suffers from multiple independent failures.

Before going into details of the construction algorithm of these permutation routings for desired interface based routings, Section 6.1 begins with a motivating example that shows the operation of an interface based routing works and its dominant benefit compared to the traditional IP routings. After understanding the example, we define a robust interface based routing on context of the multi-path routing protocol. Like two previous chapters, we will construct our desired interface based routing with the permutation routing method. However, it is impossible if we adhere to the original topology from which we only can extract routing tables for nodes, but what we need is routing tables for interfaces (links). For that reason, Section 6.2 introduces a graph transformation technique to compute a line graph of a network topology. In the line graph of a given topology, each vertex corresponds to a directed link of the topology, and from the line graph we

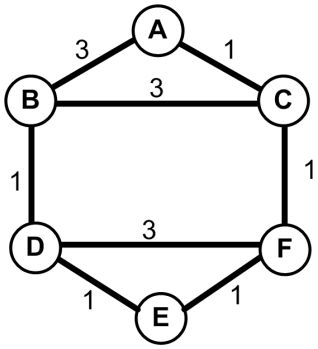
can directly calculate routing tables for interfaces with regular routing algorithms. In Section 6.3, with the resulting line graph of the topology we conduct two steps to achieve the permutation routing of vertices which also means the permutation routing of interfaces. First, we formulate the routing objective in terms of routing constraint functions at vertices. Then we define the candidate set for each assignment so that the algorithm can avoid any back-track step. From the permutation routing of interfaces, we can construct the routing table for each interface with the greedy forwarding rule. We evaluate the performance of our method in Section 6.4, and conclude the chapter in Section 6.5.

6.1 Motivation

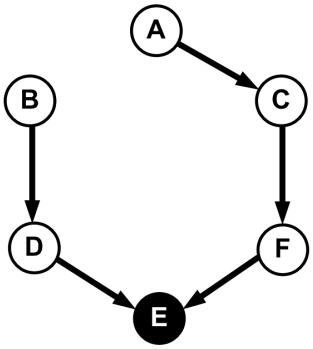
In a interface based routing, we will let a node treat packets differently depending on which incoming interface it enters the node. A node may therefore see the same packet several times, but each time it will enter the node on a different interface. We will later on show empirical results supporting the strength of this concept, but for the moment we support it with an example.

Figure 6.1(a) illustrates a simple topology with six nodes and eight (bidirectional) links with their corresponding link weights. Figure 6.1(b) is the shortest path tree (SPT) rooted at node E. If we say that a node is *protected* if it has *at least* two next-hops, then no node in Figure 6.1(b) is protected. Other routing methods, e.g. ANHOR-SP, can improve the protection coverage by adding *non-shortest* path branches to form a better-connected routing graph, mandatorily a directed acyclic graph (DAG). Figure 6.1(c) is such an instance. Obviously, Figure 6.1(c) is the most robust loop-free routing graph rooted at node E that the traditional IP routing can provide because adding any more directed link will cause loops. Consequently, we fail in protecting node F and node C under single failures.

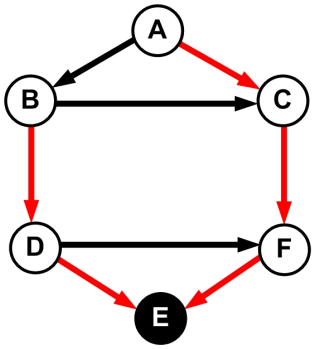
However, node F and node C of the topology can be protected if the interface-specific forwarding is used. Specifically, node F in Figure 6.1(d) will reroute flow 1 (f. 1) to node D under the failure of link (F, E) while node C in Figure 6.1(e) can select node B as its next-hop if link (C, F) fails. Loops will not occur since the forwarding decision bases on both the destination and the incoming interface of packets. That is in Figure 6.1(d) the incoming interface $(F \rightarrow D)$ may have two forwarding options for destination E: outgoing interfaces $(D \rightarrow E)$ and $(D \rightarrow F)$. The former is an easy case since packets are quickly delivered to E. In the latter, node F at the same time must guarantee that incoming interface $(D \rightarrow F)$ does not take outgoing interface $(F \rightarrow D)$ as its next-hop so that the packets would be never forwarded over interface $(D \rightarrow F)$ multiple times, and therefore avoiding loops. Likewise, the outgoing interfaces for re-routed flow 2 (f. 2)



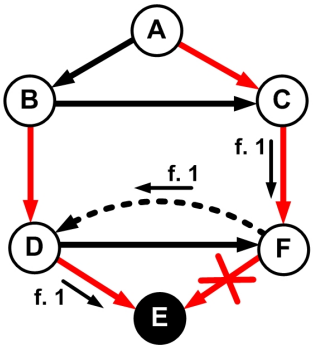
(a) A network topology



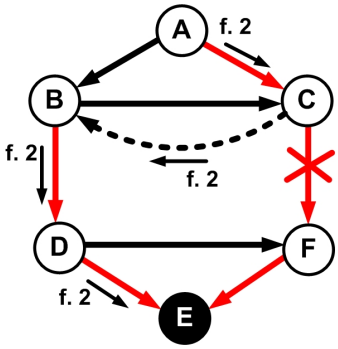
(b) SPT rooted at node E



(c) A DAG, containing the SPT, rooted E



(d) Fast re-routing under interface-specific forwarding if link (F, E) fails



(e) Fast re-routing under interface-specific forwarding if link (C, F) fails

Figure 6.1: Interface-specific forwarding helps increase routing robustness

over incoming interface $(C \rightarrow B)$ are $(B \rightarrow D)$ and $(B \rightarrow C)$. If $(B \rightarrow C)$ is installed at B, then $(C \rightarrow B)$ should not be the outgoing interface for incoming $(B \rightarrow C)$ at C. Figure 6.2 shows the recommended routing tables for the interfaces of node D and node F to illustrate what we have described. In the routing tables, the outgoing interfaces on the SPT are primary next-hops, and others are back-up. We will show how to construct such a routing table for each interface of a node in Section III, but we first introduce our main routing objective for the interface-specific forwarding.

Node	Incoming interface	Next-hop	Primary
D	B → D	E	Yes
		F	No
	F → D	E	Yes
		F	No

(a) Routing tables at interfaces of node D.

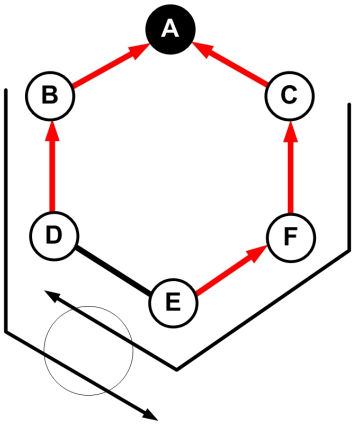
Node	Incoming interface	Next-hop	Primary
F	C → F	E	Yes
		D	No
	D → F	E	Yes

(b) Routing tables at interfaces of node F.

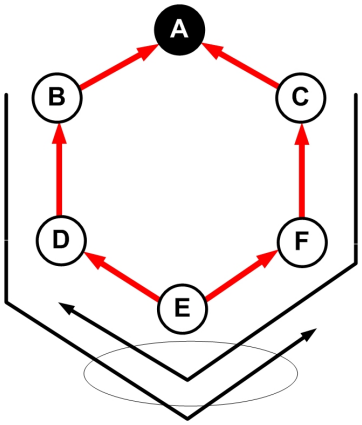
Figure 6.2: Routing tables at interfaces for a loop-free interface based routing

From the example, we observe that the interface based routing can offer 100% protection coverage against single failures in the 2-connected network topology, but we wonder if the observation still holds for all 2-connected topologies. Like previous chapter, we seek the answer by investigating a ring topology. We consider a ring topology of 6 nodes, and two SPTs rooted at node A built on the topology like Figure 6.3(a) and Figure 6.3(b). In Figure 6.3(a), if only link (A, B) fails, the back-up path should be established from B to E where packets can be forwarded on primary path $E \rightarrow F \rightarrow C \rightarrow A$. If only link (A, C) fails, packets should be re-routed to D where they can follow primary path $D \rightarrow B \rightarrow A$. However, if the two failures occur at the same time, the two re-routed scenarios cause a routing loop between node D and node E. The situation similarly occurs to the Figure 6.3(b) with the forwarding loop between node D and node F. Thus, we can conclude that there does not *always* exist a routing with interface-specific forwarding that can handle multiple failures without looping packets while offering 100% coverage for single link failures.

Obviously, the loop-free requirement restricts the interface based routing from having full protection coverage. Therefore, the most robust interface based routing should maximize the number of primary interface with at least two next-hops. We call such a routing



(a) Scenario 1: routing loop occurs between node D and node E if both links (A, B) and (A, C) fail.



(b) Scenario 2: routing loop occurs between node D and node F if both links (A, B) and (A, C) fail.

Figure 6.3: Ring topologies restrict the full protection coverage

Interface Next-hop Optimal Routing (iNHOR), and formally define it as follow:

Definition 6.1. *Given a network topology and an Spt on that topology, an incoming interface on the SPT is called a primary interface (pI) and an incoming interface* not *on the SPT is a secondary interface. An iNHOR is an interface based routing constructed from the network topology that contains the SPT and maximizes the number of primary interface-destination (pI-D) pairs with* at least *two next-hops.*

By maximizing fault tolerance capability for primary interfaces, iNHOR has a great potential to increase robustness for IP networks under failures. Such a routing can be constructed with the permutation routing framework since it shares with NHOR-SP a similar routing objective. However, different from NHOR-SP, we can not apply the framework to the original topology to find the routing table for each node for a destination. Instead, we seek to construct the routing table of each interface of a node for a destination. Fortunately, we can transform the network topology into its line graph version and apply the permutation routing method to the line graph. In the next section, we describe the transformation technique called line graph.

6.2 Line graphs

Let $G = (V, E)$ be the network topology where V is the set of nodes and E is the set of bidirectional links. The line graph of G is $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ is the set of vertices and $\mathbf{E}$ is the set of edges, which are resulted from two transformation rules. The first rule is the content of Definition 6.2

Definition 6.2. *Each vertex in* $\boldsymbol{V}$ *represents a directed link of* E

We denote by $[\mathbf{u},\mathbf{v}]$ the vertex in $\mathbf{V}$, which is directed link $(u \to v)$ in E . Let $H([\mathbf{u},\mathbf{v}])$ and $T([\mathbf{u},\mathbf{v}])$ be two operators which apply to vertex $[\mathbf{u},\mathbf{v}]$ to extract the head node, u, and the tail node, v, of corresponding directed link $(u \to v)$. We induce the connectivity of $\mathbf{G}$ from the connectivity of G in the Definition 6.3.

Definition 6.3. *Two vertices* $\boldsymbol{i}, \boldsymbol{j} \in \boldsymbol{V}$ *form directed edge* $(\boldsymbol{i} \to \boldsymbol{j})$ *in* $\boldsymbol{E}$ *if and only if* $H(\boldsymbol{i}) = T(\boldsymbol{j})$.

For example, $[\mathbf{u},\mathbf{v}]$ and $[\mathbf{v},\mathbf{w}]$ are two vertices that form directed edge $[\mathbf{u},\mathbf{v}] \to [\mathbf{v},\mathbf{w}]$ due to $H([\mathbf{u},\mathbf{v}]) = v = T([\mathbf{v},\mathbf{w}])$. Following Definition 6.2 and Definition 6.3 in that sequence, line graph $\mathbf{G}$ has been identified thoroughly. Let $A_{\mathbf{G}} = (a_{\mathbf{ij}})$ be the adjacency-matrix that represents for $\mathbf{G}$:

$$a_{\mathbf{ij}} = \begin{cases} 1 & H(\mathbf{i}) = T(\mathbf{j}) \\ 0 & \text{otherwise} \end{cases}$$

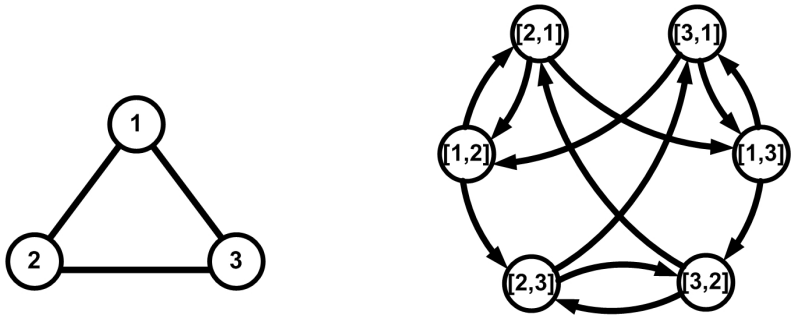
We then provide properties of line graphs in the below theorems. Theorem 6.1 shows the cardinalities of the vertex set and the edge set of a line graph while Theorem 6.2 and Theorem 6.3 prove that line graphs preserve the subset relationship and the acyclic property of the corresponding directed graph.

Theorem 6.1. *Given directed graph from topology* G*, line graph* $\boldsymbol{G}$ *of* G *is satisfied:*

1. $|\boldsymbol{V}| = q$ *where* q *is the cardinality of link set* E.
2. $|\boldsymbol{E}| = \sum_{u=1}^{p} d_u^2$ *where* d_u *is the out-degree of node* $u \in V$ *and* p *is the cardinality of node set* V.

Proof. (1) is trivial due to the Definition 6.2. We prove (2) by examining each node $u \in V$. Let d_u be the out-degree of node u and $N_G(u) = \{v \in V : (u \to v) \in E\}$ be the set of neighboring nodes of u. We observe that there are d_u directed links incident at u and d_u directed links departing from u. Via Definition 6.3, each node $[\mathbf{v}_k, \mathbf{u}] \in \mathbf{V}$, where $u \in N_G(v_k)$, will have d_u neighboring nodes. Therefore, in $\mathbf{G}$, there are d_u nodes and d_u^2 directed links involving node u. That means $|\mathbf{E}| = \sum_{u=1}^{p} d_u^2$. □

According to the theorem, if we have a full-meshed topology of 3 nodes, then the corresponding line graph will have 6 vertices and 12 directed edges. We illustrate the two graphs in Figure 6.5. We verify those numbers with Theorem 6.1. It is easy to see that in the topology there are 3 nodes and 6 directed links and each of the three nodes in the topology has node degree of 2. Therefore, $|\mathbf{V}| = 6$ and $|\mathbf{E}| = 2^2 + 2^2 + 2^2 = 12$.



(a) Topology G (b) Line graph $\mathbf{G}$

Figure 6.4: The topology and its line graph

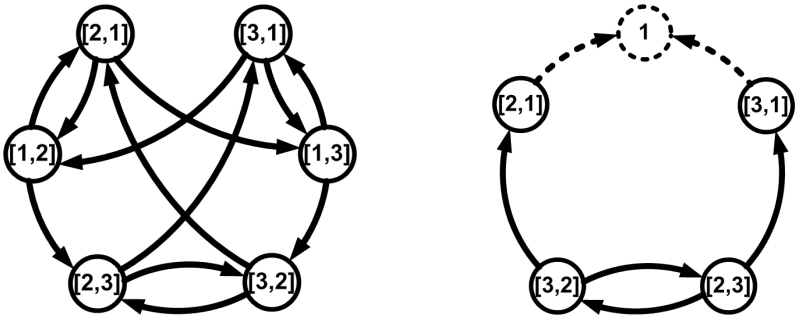
Theorem 6.2. *Let G_1 and G be two directed graphs, if $G_1 \subseteq G$, then line graphs $\boldsymbol{G}_1 \subseteq \boldsymbol{G}$.*

Proof. We prove that $\mathbf{V}_1 \subseteq \mathbf{V}$ and $\mathbf{E}_1 \subseteq \mathbf{E}$ both hold. It is trivial that $\mathbf{V}_1 \subseteq \mathbf{V}$ due to Definition 6.2. Assume that arbitrary directed links $(a \to b) \in E_1$ and $(b \to c) \in E_1$, then we will have $(\mathbf{[a,b]} \to \mathbf{[b,c]}) \in \mathbf{E}_1$. Because $E_1 \subseteq E$, then we have $(a \to b) \in E$ and $(b \to c) \in E$. That means $(\mathbf{[a,b]} \to \mathbf{[b,c]}) \in \mathbf{E}$. So $\mathbf{E}_1 \subseteq \mathbf{E}$ holds. □

Theorem 6.3. *The line graph of a directed acyclic graph is also a directed acyclic graph.*

Proof. The proof is by contradiction. We denote D by the given directed acyclic graph and $\mathbf{D}$ by its corresponding line graph. We assume that $\mathbf{D}$ contains a loop. Namely, the loop is $\mathbf{[a,b]} \to \mathbf{[b,c]} \to \ldots \to \mathbf{[y,z]} \to \mathbf{[z,a]}$ where $a, b, c \ldots y, z$ are nodes of graph D. In addition, the loop implies that there exists path $a \to b \to c \ldots \to y \to z \to a$ which is another loop of D. Since D is directed acyclic graph, our assumption is broken. □

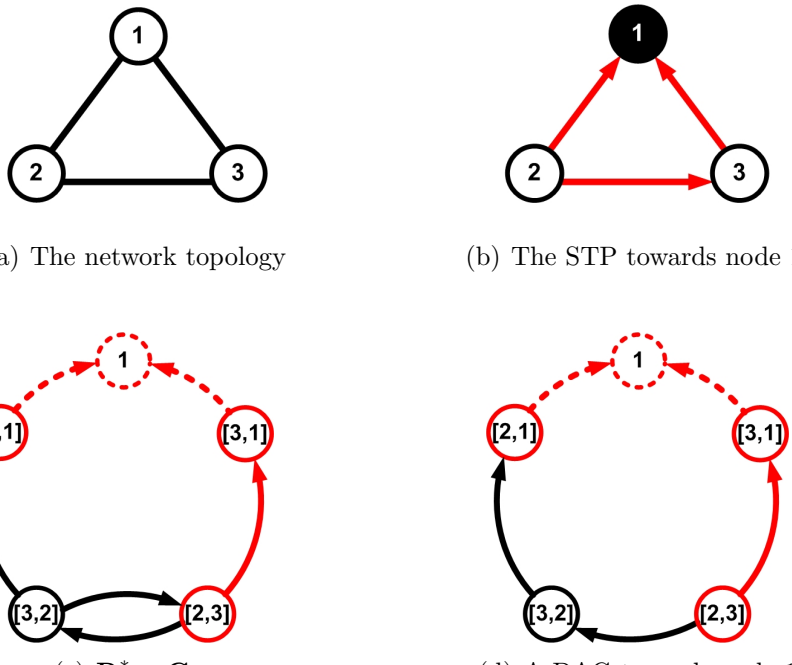
From the definition of line graphs, we might think that $\mathbf{G}$ is a proper input for calculating a permutation routing of interfaces. It is, however, infeasible since $\mathbf{G}$ does not include any vertex that corresponds to the real destination node. For the routing computation purpose, we should modify $\mathbf{G}$ by adding a virtual vertex that represents the destination and taking away vertices and directed edges that will not contribute to routing towards the virtual destination. For simplicity, we also use $\mathbf{G}$ to denote the modified version of $\mathbf{G}$. Figure 6.5 illustrates the line graph $\mathbf{G}$ (Figure 6.5(a)) and its modified version with an added virtual vertex and two added virtual directed edges in dotted lines (Figure 6.5(b)) of the topology in Figure 6.4(a).



(a) Line graph **G** (b) Modified version of **G**

Figure 6.5: Line graph and its modified version with added virtual node 1.

Let R^* be the SPT towards destination d which is constructed on G, we can generate its line graph with virtual destination d which is denoted by $\mathbf{R}^*$. According to Theorem 6.2 and Theorem 6.3, $\mathbf{R}^* \subset \mathbf{G}$ and $\mathbf{R}^*$ is a DAG towards d. Figure 6.6 illustrates those properties. Figure 6.6(b) is the SPT towards node 1, built on the topology in Figure 6.6(a). The line graph of the SPT shown in red lines in Figure 6.6(c) is a subset of the line graph of the topology (Figure 6.5(b)). Now, we can compute a routing on the line graph **G** which includes transformed SPT $\mathbf{R}^*$. Figure 6.6(d) is such a routing.



(a) The network topology (b) The STP towards node 1

(c) $\mathbf{R}^* \subset \mathbf{G}$ (d) A DAG towards node 1

Figure 6.6: Properties of the line graph of the SPT

6.3 Interface based Next-Hop Optimal Routing

Given network topology G, from which we construct SPT R^*, we have their line graph versions, $\mathbf{G}$ and $\mathbf{R}^*$, respectively. The Definition 6.1 means that iNHOR is the interface-based routing which is constructed on $\mathbf{G}$, and includes $\mathbf{R}^*$. It is easy to see that iNHOR is similar to NHOR-SP which has been studied in Chapter 4, and therefore we can use the permutation routing method for the construction. However, like the previous explanation of ANHOR-SP, we only achieve an approximate of iNHOR that we call AiNHOR.

In order to use the permutation framework to produce AiNHOR, we follow two steps: formulating the routing objective in terms of routing constraint functions and defining the domain for each assignment. Although iNHOR is resemble to NHOR-SP in their objectives, the domain for AiNHOR can not follow formula (4.3) defined for ANHOR-SP. The reason is that all nodes of a SPT are all the nodes in the original network topology while in line graph context all vertices of $\mathbf{R}^*$ are only a portion of all vertices of $\mathbf{G}$. For example, in Figure 6.6(c) vertices [**2,1**], [**3,1**], and [**2,3**] belong $\mathbf{R}^*$ but vertex [**3,2**] does not. As a result, we first describe the domains for AiNHOR.

6.3.1 Domain generation

With given line graphs $\mathbf{G}$ and $\mathbf{R}^*$, we then define the domain for each variable. Because a vertex in any valid routing must have at least *one* next-hop towards the destination, domain D_{i+1} for variable p_{i+1} can be the set that contains all vertices with at least one out-neighbor, which has been placed in partial permutation routing $\vec{p}_i$, as follow:

$$D_{i+1} = D_i \cup \{\ \mathbf{v} \in \mathbf{V}_i \mid (\mathbf{v} \to \mathbf{u}) \in \mathbf{E}\} \setminus \{\mathbf{u}\} \tag{6.1}$$

where $\mathbf{u}$ is the vertex that has been assigned to variable p_i and $\mathbf{V}_i$ is a set of vertices of $\mathbf{G}$, excluding all vertices in $\vec{p}_i$ and D_i.

We have noticed that D_{i+1} may contain *three* types of vertices. First, they are vertices in $\mathbf{R}^*$ which have *all* their next-hops in $\mathbf{R}^*$ already placed in $\vec{p}_i$, denoted by D^a_{i+1}. Second, they are vertices which are *not* in $\mathbf{R}^*$, denoted by D^b_{i+1}. Third, they are vertices in $\mathbf{R}^*$ which *do not* have *all* their next-hops in $\mathbf{R}^*$ already placed in $\vec{p}_i$, denoted by D^c_{i+1}.

For vertex $\mathbf{v}$, let $c^*[\mathbf{v}]$ be the number of next-hops in $\mathbf{R}^*$ placed in $\vec{p}_i$ and $n^*[\mathbf{v}]$ be the total number of next-hops in $\mathbf{R}^*$, sub-domain D^a_{i+1} for variable p_{i+1} follows the recursive relation:

$$D^a_{i+1} = D^a_i \cup \{\mathbf{v} \in D_{i+1} \mid c^*[\mathbf{v}] = n^*[\mathbf{v}]\} \tag{6.2}$$

and sub-domain D_{i+1}^b is calculated as follow:

$$D_{i+1}^b = D_i^b \cup \{\mathbf{v} \in D_{i+1} \mid \mathbf{v} \notin \mathbf{R}^*\} \tag{6.3}$$

Function Domain

```
1  D_i ← D_i \ {u}
2  for v ∈ V_i such that (v → u) ∈ E do
3      D_i ← D_i ∪ {v}
4  for v ∈ D_i such that (v → u) ∈ E do
5      c[v] ← c[v] + 1
6  V_i ← V_i \ D_i
7  for v ∈ D_i do
8      if v ∈ V(R*) then
9          if (v → u) ∈ E(R*) then
10             c*[v] ← c*[v] + 1
11             if c*[v] = n*[v] then
12                 D_i^a ← D_i^a ∪ {v}
13     else
14         D_i^b ← D_i^b ∪ {v}
```

Function `Domain` begins with finding domain D_i, following formula (6.1), and next updates the next-hop counters of all vertices in the domain (lines 1-5). Correspondingly, this process comprises of two loops. The first loop (lines 2-3) examines all vertices in the remaining vertex set, $\mathbf{V}_i$, and collects the nodes that have outgoing links to the previous assigned vertex, say vertex $\mathbf{u}$, which we have removed it out of the current domain (line 1). The second loop (lines 4-5) applies to all nodes in the updated domain (D_i after being added with nodes from the first loop) and increases the next-hop counters by 1 for those that have outgoing links to $\mathbf{u}$. In the rest of the function, the domain is divided into two sub-domains as described above: D_i^a and D_i^b. To do that, a loop (lines 7-14) considers each vertex in D_i, and checks if the vertex belongs to the vertex set of $\mathbf{R}^*$, denoted by $V(\mathbf{R}^*)$. If that is not the case, we move that vertex to sub-domain D_i^b (line 14). Otherwise, we further check on its shortest next-hop counter. If the vertex admits $\mathbf{u}$ as its next-hop, and the directed link between them appears in the edge set of $\mathbf{R}^*$, denoted by $E(\mathbf{R}^*)$, the vertex's shortest next-hop counter is increased by 1 (line 10). Furthermore, if the shortest next-hop counter reaches its maximum value (the total number of shortest next-hops of a vertex calculated from $\mathbf{R}^*$, we move that vertex into sub-domain D_i^a (line 12).

We then design a selection strategy to pick a vertex from defined sub-domains for a variable in each assignment to produce a permutation routing representing AiNHOR.

6.3.2 Selection strategy

Our observation is if we *first* chose a vertex in D^a_{i+1} when it is not empty or if we *first* choose a vertex in D^b_{i+1} when it is not empty, the resulting permutation routing in either cases will be compatible with the SPT. The reason is sub-domain D^a_{i+1} in formula (6.2) includes all vertices which do not violate the ordering of $\mathbf{R}^*$ according to Proposition 4.2, and sub-domain D^b_{i+1} encompasses all vertices that do not involve $\mathbf{R}^*$. However, the second strategy will increase the protection coverage because that places in the permutation *all* possible secondary links, which can become next-hops for primary links, before primary links.

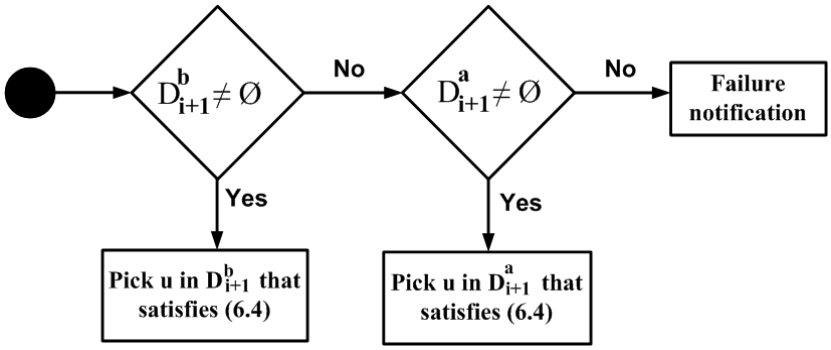


Figure 6.7: The selection procedure for the variable

Given the non-empty sub-domains and followed by the described selection strategy, we also have to pick a vertex from the sub-domains such that the resulting permutation routing corresponds to the routing with near-optimal protection coverage. As described in Chapter 4 and Chapter 5, we should pick a vertex that has the maximum connectivity value to the partial permutation routing. We derive routing constraint function $C(\mathbf{u})$ to realize our selection strategy as follow:

$$C(\mathbf{u}) = \begin{cases} \textbf{True} & \text{if } c[\mathbf{u}] = \max_{\forall \mathbf{v} \in D^x_{i+1}} c[\mathbf{v}] \\ \textbf{False} & \text{otherwise} \end{cases} \tag{6.4}$$

where $c[\mathbf{u}]$ denotes the number of outgoing edges from $\mathbf{u}$ to $\vec{p_i}$ if $\mathbf{u}$ is selected and D^x_{i+1} is D^a_{i+1} or D^b_{i+1}. Figure 6.7 summarizes the selection procedure in which we go through D^b_{i+1} first. In addition, function `Select` below gives more details of the strategy.

We apply our described algorithm to the topology in Figure 6.1(a) and yield the permutation routing of 15 vertices towards destination E: {**E**, [**D,E**], [**F,E**], $[D, F]$, $[F, D]$, [**B,D**], [**C,F**], $[D, B]$, $[C, B]$, $[F, C]$, $[B, C]$, $[A, B]$, [**A,C**], $[B, A]$, $[C, A]$}. In the per-

Function Select

```
1  c^a_max ← max_{∀v∈D^a_i} c[v]
2  c^b_max ← max_{∀v∈D^b_i} c[v]
3  if D^b_i ≠ ∅ then
4      while D^b_i ≠ ∅ do
5          an arbitrary node u from D^b_i
6          D^b_i ← D^b_i \ {u}
7          if c[u] = c^b_max then
8              return u
9  else
10     while D^a_i ≠ ∅ do
11         an arbitrary node u from D^a_i
12         D^a_i ← D^a_i \ {u}
13         if c[u] = c^a_max then
14             return u
15 return null
```

mutation, those boldface vertices are the directed links of the topology that are on the shortest path towards node E (Figure 6.1(b)). Forwarding tables for interfaces then are generated by allowing each vertex in the permutation to forward packets to all its neighboring vertices that occur before it in the permutation. Figure 6.2 shows two examples of forwarding tables for interfaces of node D and node F, extracted from the permutation routing of vertices.

Proposition 6.1. *Our selection procedure in Figure 6.7 gives a backtrack-free algorithm for all connected directed graph.*

Proof. We observe that whenever one of two sub-domains D^b_{i+1} and D^a_{i+1} is not empty, we could find a vertex that satisfies (6.4). Therefore, our algorithm will perform a backtracking step only if both D^b_{i+1} and D^a_{i+1} are empty at the same time in some iteration. However, that can never happen before all vertices have been placed in the permutation. If this is the case, there exists only candidate vertices in D^c_{i+1}. In other words, we can follow $\mathbf{R}^*$ from any vertex that has not been selected and always find a next-hop vertex that is not placed in $\vec{p}_i$. But this is impossible: since $\mathbf{R}^*$ is a DAG (Theorem 6.3), any path along the edges of $\mathbf{R}^*$ will eventually reach the destination, which is the first vertex placed in the permutation. □

Due to the property of backtrack-freeness, with sparse topologies the computational complexity to construct one permutation routing towards *one* destination would be $O(|\mathbf{E}|+|\mathbf{V}|)$. In dense topologies, the complexity becomes $O(|\mathbf{V}|^2)$.

6.4 Performance analysis

We assess routing robustness of AiNHOR in terms of the improved number of next-hops for primary incoming interfaces towards all destinations. Since adopting secondary interfaces for packet transportation can lead to path inflation, we also investigate the hop-count path length distribution with various allowed numbers of next-hops.

6.4.1 Evaluation setup

Network topology

We select six representative network topologies from Rocketfuel project [80] for our evaluations. For each topology, we remove all nodes that will not contribute on routing (e.g. single out-degree node). The refined topologies are bi-connected graphs, listed in Table 6.1 in increasing order of their average node out-degrees. In addition, Table 6.2 shows their corresponding line graphs in increasing order of their average node out-degrees.

Table 6.1: Refined network topologies

AS	Name	Nodes	Links	Avg. Degree
1221	Telstra(au)	50	298	3.88
3967	Exodus(us)	72	280	3.89
1755	Ebone(eu)	75	298	4.00
3257	Tiscali(eu)	115	564	4.90
6461	Abovenet(us)	129	726	5.60
1239	Sprint(us)	284	1882	6.62

Link weight assignments

The ECMP and LFA base their path calculations on link weights. To obtain realistic link weight settings, we implement local search heuristic [24] to optimize link load objective function under the traffic matrix (TM) generated by the gravity model [56]. For AS1239, we, however, use unit link weights, because the local search heuristic does not scale to a topology of this size.

Comparisons

We compare our AiNHOR with the shortest path based routings, ECMP and LFA, and recent solutions ANHOR-SP and JNHOR-SP. Comparisons to other interface based rout-

Table 6.2: Line graphs

AS	Name	Vertices	Edges	Avg. Degree
1755	Ebone(eu)	298	1432	4.80
3967	Exodus(us)	280	1372	4.90
1221	Telstra(au)	298	1030	5.30
6461	Abovenet(us)	726	5434	7.50
3257	Tiscali(eu)	564	4378	7.76
1239	Sprint(us)	1882	25988	13.81

ing methods that include the shortest path routing, e.g. FIR [58] [98], are less relevant because they all are not loop-free under multiple failures.

We evaluate robustness of all mentioned routing methods in terms of links, instead of nodes. Specifically, for given incoming interface $(* \rightarrow i)$, j is called the primary next-hop of $(* \rightarrow i)$ if $(i \rightarrow j)$ is on the SPT towards d. Otherwise, j is called secondary next-hop and $(i \rightarrow j)$ is a secondary link towards d.

6.4.2 Robustness evaluation

Figure 6.8 shows fractions of primary interfaces with at least two next-hops with five methods. We observe that the fractions of AiNHOR vary slightly across six topologies and all are above 97%. Especially, AiNHOR achieves 100% primary interfaces with two next-hops in AS6461 and AS3257. The good results come from the properties of the interface based routing and its operation on 2-connected graphs. That is a node with at least two outgoing interfaces may have the chance to install two loop-free next-hops.

Obviously, AiNHOR gives clear improvements over LFA and ANHOR-SP. Figure 6.8 shows that LFA and ANHOR-SP protect only from 21% to 67% and from 29% to 81% primary interfaces, respectively, in the six tested ISP topologies. In terms of protecting primary interfaces, JNHOR-SP is only subtly better than ANHOR-SP, and still much lower than AiNHOR.

6.4.3 Path length distribution

AiNHOR has significantly improved the multi-path capability by maximizing the number of pI-D with at least two next-hops that allows load-balancing and increased fault-tolerance. However, adopting secondary interfaces for packet forwarding possibly leads to high path inflation which can increase traffic load over non-shortest paths. For that

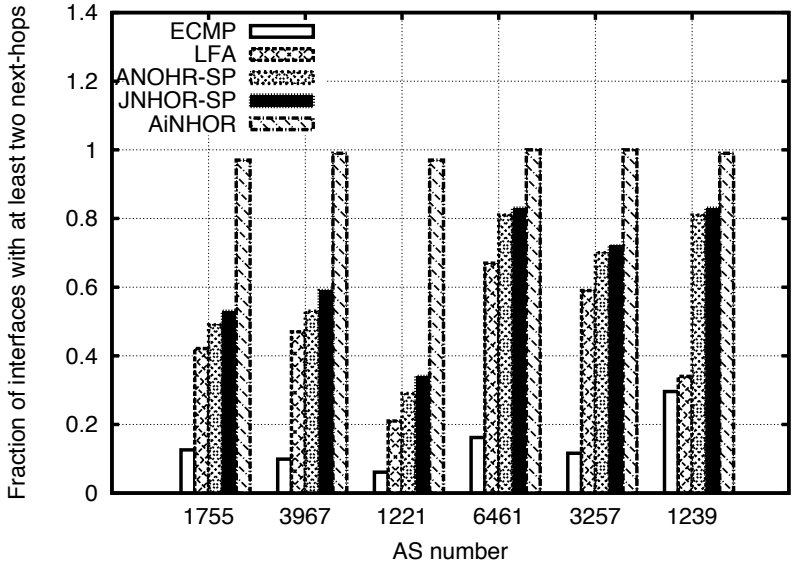


Figure 6.8: Fraction of pI-D pairs with two routing options.

reason, we investigate the distribution of the static path length for different values of the maximum number of next-hops for each interface, denoted by K.

Figures 6.9-6.14 show the distribution of path lengths in hop for K =1, 2 and 3. For $K = 1$, all paths from source nodes to destinations are shortest paths. The average shortest path length is the left-most vertical line in each figure. Increasing K to 2 and 3 will allow more longer paths and therefore shift the average upper bounds of path lengths (vertical lines) to the right. Of all topologies, those upper bounds only increase from 7.5% (AS1239) to 40% (AS6461) for $K = 2$ and up to 100% (AS6461) for $K = 3$. In practice, those path lengths are still comparable to those of shortest path routing.

6.4.4 Running time of the algorithms

The complexity of our proposed algorithm depends on the number of vertices, edges of the line graphs and the average length of the permutations for all destination. For all tested topologies, the average length of the permutations is found equal to half of number of vertices ($0.5 \times |\mathbf{V}|$). That means the average path length of the permutations for all destinations in AS1221 is 149 vertices while that of AS1239 is 941 vertices.

Figure 6.15 shows the relative running time of AiNHOR method to ECMP across the six topologies. The AS topologies are listed in an increasing order of vertex degrees. The results are achieved with an Intel Core 2 CPU 6300 @ 1.86 GHz machine. For the first five topologies, the relative running times are less than four times. The result for AS1239

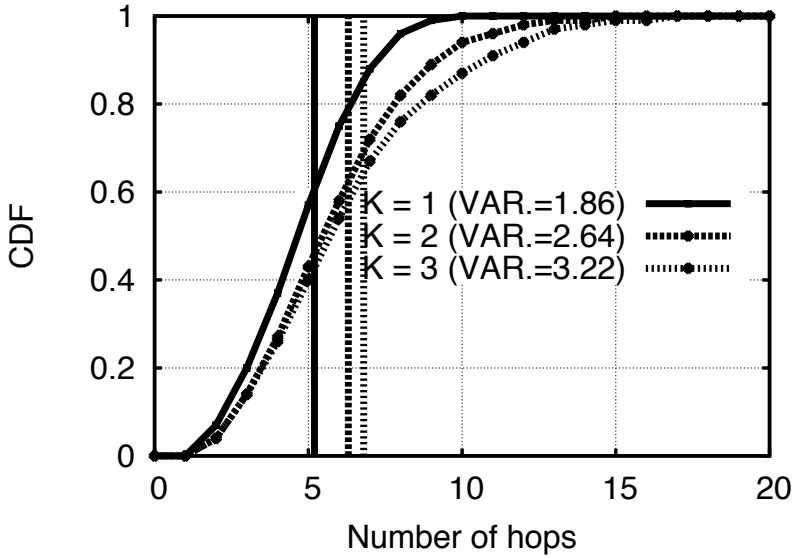


Figure 6.9: Path length distribution in AS1755

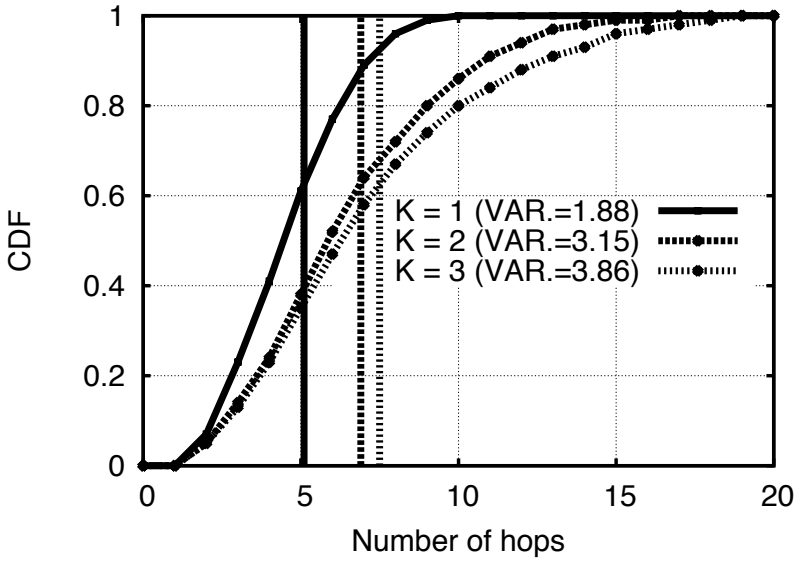


Figure 6.10: Path length distribution in AS3967

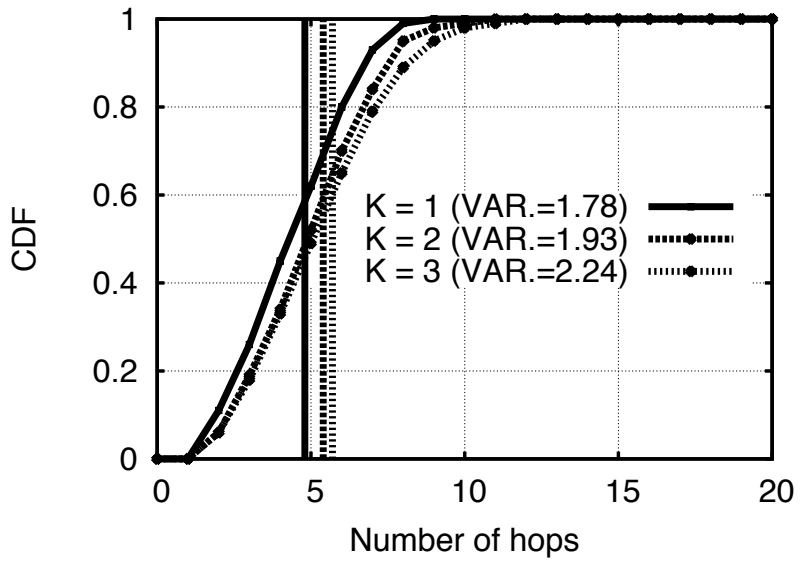


Figure 6.11: Path length distribution in AS1221

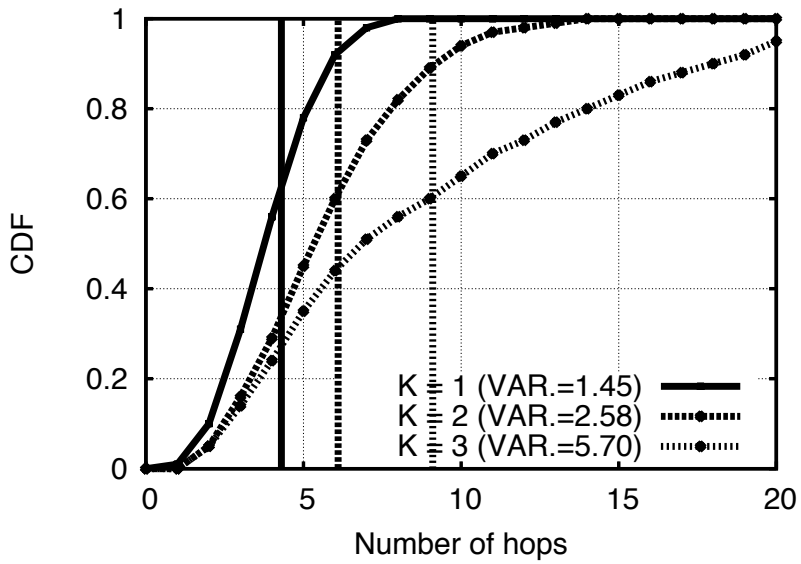


Figure 6.12: Path length distribution in AS6461

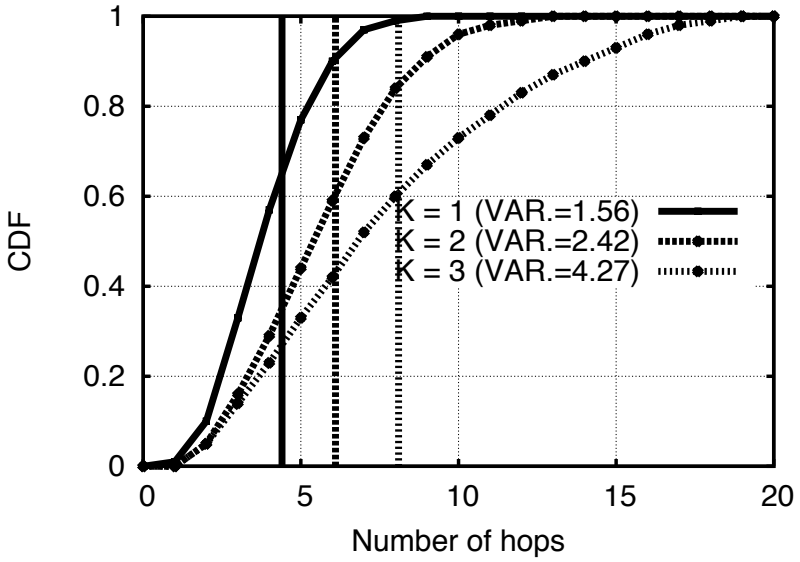


Figure 6.13: Path length distribution in AS3257

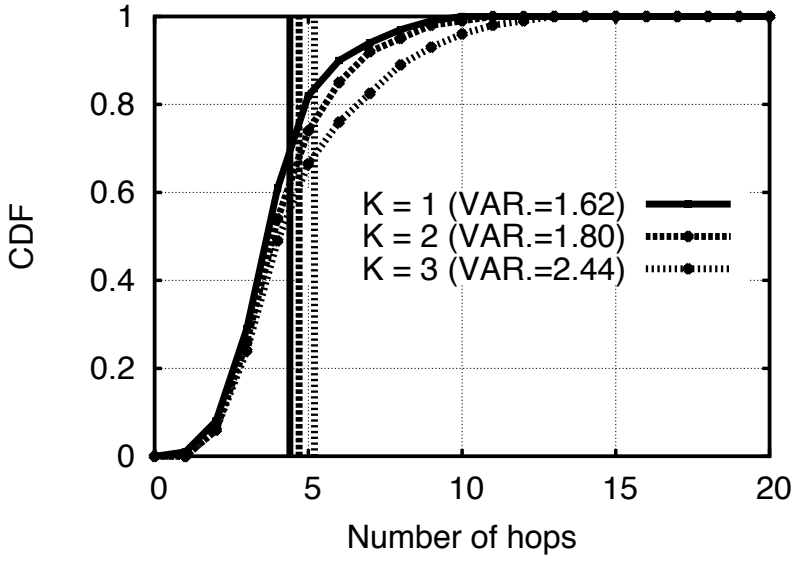


Figure 6.14: Path length distribution in AS1239

is getting a bit worse, but less than 8 times. Explosion in both number of vertices and directed edges in the line graph of AS1239 gives rise in complexity. However, it is still computationally feasible.

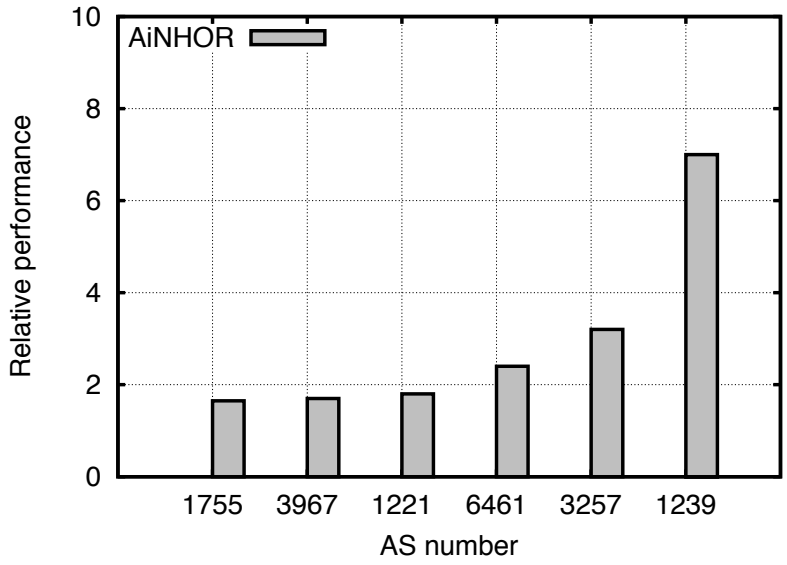


Figure 6.15: Relative running time between AiNHOR and ECMP

6.5 Summary

Interface-specific forwarding bases not only on the destination address but also on the incoming interface of packets. A routing that employs interface-specific forwarding is a called interface based routing. In this type of routing, each interface contains a distinct routing table that maps each destination to outgoing interfaces. A robust interface based routing suggests that each primary interface should install *at least* two outgoing interfaces. Those multiple outgoing interfaces can be used either for fast re-routing or load-balancing. For a given 2-connected topology, the interface based routings can offer higher protection coverages compared to those of the traditional IP routings, and in many cases can protect all single link failures.

In this chapter, we aim to construct the interface based routing, called iNHOR, which can maximize the number of primary interface destination pairs, given that the primary interfaces are those on the shortest path tree built on the same topology. In other words, the iNHOR includes the shortest path tree, and augments the tree with additional forwarding options. In addition, our design targets four important properties for adoption.

First, our routing method employs the hop-by-hop destination based forwarding. Second, it does not require IP headers of packets to install non-standard bits which associates with network faults. Third, it is guaranteed to be free of routing loops, especially when the network suffers from multiple uncorrelated failures. Finally, our method can work with standard link state routing protocols, i.e. OSPF and IS-IS. That means our method can be integrated into existing routing infrastructure without significant changes.

Due to the efficacy and the feasibility of the solution for distributed routing systems, we only achieve AiNHOR which approximates iNHOR. A permutation routing for AiNHOR is a permutation of all interfaces of the network topology. In order to build such a permutation routing, we compute the line graph of the topology, then formulate the routing objective in terms of routing constraint functions of source nodes, and finally define the candidate set for each variable representing a position in the resulting permutation routing. The line graph of the topology is a directed graph whose vertices are directed links given in the topology and connectivity depends on the connectivity of the topology. With the yielded line graph, the construction process is similar to that of ANHOR-SP, but selection strategy is more complicated owing to various types of vertices in the line graph with respect to the given shortest path routing. We have defined a reasonable domain for each variable and a selection strategy that can achieve a back-track free algorithm.

We evaluate AiNHOR over six network topologies like those in Chapter 4 and Chapter 5. The main result from the evaluation is that AiNHOR offers significantly higher link-protecting coverage compared to those of our previous approaches, ANHOR-SP and JNHOR-SP. In addition, the link-protecting coverages given by AiNHOR vary little across topologies with various node-degrees, and are above 97%. At a cost of increased multi-path capability, the path stretch AiNHOR introduces is higher than that of ANHOR-SP, but still comparable to the shortest paths. Finally, AiNHOR has computational feasibility for being adopted in distributed routing systems.

Chapter 7

On load-balancing in multi-path routing protocols

Multi-path routing protocols offer two main benefits. First, the availability of multiple next-hops towards a destination allows fast failover from link or node failures. Second, traffic can be balanced over several next-hops in order to improve network utilization and avoid congestion. Recently, several methods that improves the multi-path capabilities of traditional IP networks have been proposed. It is, however, a non-trivial question to decide how traffic should be routed over the available paths, both during normal operation and in a failure situation. These are the main questions we seek to answer in this chapter.

Previously, we introduce several multi-path routing algorithms for the traditional IP networks, which can be classified into two main types: T_1 includes those compatible with the shortest path algorithm, i.e. ANHOR-SP, JNHOR-SP, and AiNHOR, and T_2 encompasses those independent of the shortest path algorithm, i.e. ANHOR and JNHOR. The performance analysis shows that the methods of T_2 usually offer higher protection coverages for single link faults than those of T_1. However, T_1's methods expose a clear benefit in terms of load-balancing. That is if we optimize the link weights on a given traffic matrix, we can yield a shortest path routing that offers near-optimal performance with respect to that TM. That also implies that if we set those shortest path next-hops as primary next-hops, then the resulting routing is guaranteed to be free of congestion in normal operation. Nevertheless, the problem of assigning back-up roles for additional next-hops beside the shortest ones so that network congestion is minimal after a failure occurs has not been resolved yet.

In this chapter, we seek to deal with back-up assigning issue on context of our permutation routing solutions. In Section 7.1, we introduce a new routing method, called Augmented Base Routing (ABR). ABR is similar to ANHOR-SP, but instead of being only compatible to the shortest path algorithm ABR can include any of existing routings

that the network operators have optimized for their network topology and the TM. With the achieved ABR, Section 7.2 reviews several load-balancing methods and fast re-routing strategies that can be applied to the ABR in normal operation and in a fault situation. The performance results in Section 7.3 show the average network costs of various combinations of load-balancing methods and back-up path selections, which give us an insight on load-balancing in the traditional IP networks. We conclude the chapter in Section 7.4.

7.1 Augmented Base Routing

The basic requirement of any routing scheme is to ensure that traffic demands will be successfully routed, making the network congestion-free. The traffic demands are collected by actual measurements or by traffic models [70], [51] and are represented by a traffic matrix (TM). Given a TM, a routing well engineered for that TM is called the *base routing*. The popular technique to compute base routings is to optimize traffic distribution in association with an objective function, e.g. network-wide cost [24] or minimize maximum link utilization [90]. Such optimized routings, despite near-optimal traffic engineering in normal operation, are usually vulnerable to network failures. For that reason, we propose Augmented Base Routing (ABR).

Definition 7.1. *An ABR is an all-path routing which includes an existing base routing, and augments the base routing with additional forwarding options.*

By all-path routing for a destination, we mean that all bidirectional links in the topology are potentially used for traffic forwarding to the destination. ABR can be constructed with permutation routing framework, which we present below.

7.1.1 ABR algorithm

Let $G = (V, E)$ be the network topology, and $R_d^{br} = (V, E_d^{br})$ be the base routing with respect to the given TM, we will create an ABR, $R_d = (V, E_d)$, such that E_d^{br} is a subset of E_d. In order to generate R_d by the permutation routing method we first determine the candidate set for each variable representing a position in the permutation routing and the routing constraint function for increasing load-balancing capability of each source node. We first address the candidate set.

Let u be a node that has been previously assigned to variable p_i which represents position i-th of the permutation routing. We seek to construct the domain for variable p_{i+1}, denoted by D_{i+1}, given that we already have the domain for variable p_i. Domain D_{i+1} would contain all nodes that have *all* their next-hops in R_d^{br} already placed in the permutation routing, formalized as follow:

$$D_{i+1} = D_i \cup \{v \in V_i \mid c_{br}[v] = n_{br}[v]\} \setminus \{u\} \tag{7.1}$$

where V_i is the remaining nodes of node set V, and $c_{br}[v]$ is the counter of node v to keep track the number of its outgoing links from itself to its neighboring nodes already placed in the permutation, and $n_{br}[v]$, precomputed by node v from R_d^{br}, is the total number of its next-hops in the base routing.

Theorem 7.1. *Given base routing R_d^{br} towards destination d, the candidate set following fomula* (7.1) *containing all nodes that are compatible with R_d^{br}, meaning that the resulting selection will not violate the existing ordering of the base routing.*

Proof. Let V_i be a set of nodes from V, excluding all nodes already in $\mathcal{P}$ and D_i, formula (7.1) means that there does not exist any node v in V_i such that $\exists u_i \in D_i : (u_i \rightarrow v) \in E_d^{br}$ and there does not exist any node $u_j \in D_i$ such that $\exists u_i \in D_i : (u_i \rightarrow u_j) \in E_d^{br}$. Therefore, the permutation routing resulted by selecting a node in candidate set and placing it in $\mathcal{P}$ will not cause any conflict with the ordering of R_d^{br}. □

Let $\vec{p_i}$ be the successive successful assignments of nodes to a subset of i variables, $\{p_1, p_2, \ldots, p_i\} \subseteq \mathcal{P}$, we increase the network robustness for the base routing by picking a node from the domain with the maximum connectivity value to $\vec{p_i}$. The reason behind that is already explained in Chapter 4. The routing constraint function at the source node is therefore identical to that of ANHOR, and is derived as below:

$$C(u) = \begin{cases} \textbf{True} & \text{if } c[u] = \max_{\forall v \in D_{i+1}} c[v] \\ \textbf{False} & \text{otherwise} \end{cases} \tag{7.2}$$

where $c[u]$ denotes the number of outgoing links from u to $\vec{p_i}$. Note that outgoing links from u to $\vec{p_i}$ include those links of the base routing, meaning that $c[u] \geq c_{br}[u]$.

Following the proof of Preposition 4.1, we easily prove that the construction of the ABR is also backtrack-free. Hence, the computational complexity of the ABR towards a destination is $O(M + N)$ for a sparse topology and $O(N^)$ for a dense one where N and M are the cardinalities of node set V and link set E of topology G, respectively.

7.1.2 Routing schemes from an ABR

For a particular destination, ABR can use all bidirectional links (in one direction) of the network topology for traffic forwarding, and therefore results in multiple next-hops at each source node. The source node then assigns to each of its next-hops one of two operational roles: primary or backup. The resulting assignments at all nodes for all destinations form a routing scheme. We have two routing schemes extracted from the ABR.

Prioritized Base Routing Scheme (R_1)

This scheme assigns primary roles for all next-hops that appear in the base routing. Others are backup next-hops. Furthermore, this scheme limits the total number of next-hops, K, at each node, either primaries or backups, to at most 4. In practice, installing more than a few next-hops will not give much benefit with respect to robustness or load-balancing. It will, however, require more fast memory in the forwarding table, and may lead to the unnecessary inclusion of paths that are much longer than the shortest path.

K Next-Hop Scheme (R_2)

In this scheme, all available next-hops are primary next-hops and are used in association with a load-balancing mechanism that we will shortly discuss. Like R_1, scheme R_2 allows at most $4(K = 4)$ active next-hops for the similar reason.

Both R_1 and R_2 are multi-path routing protocols, and typically will associate with a load-balancing method which is a mechanism to determine a split ratio of traffic on each of the path. The following section introduces several load-balancing mechanisms for the traditional IP networks.

7.2 Load-balancing

For a multi-path routing scheme like R_1 or R_2, a source node employs *all* of its next-hops or *some* of its next-hops for traffic forwarding to a destination. The natural question is how to split traffic over those active next-hops to achieve the most load-balanced network in two operational states: fault-free state and post-failure state. We investigate a number of load-balancing methods that use local network information only, first in the network without any failure.

7.2.1 Fault-free case

Load balancing methods in the distributed IP networks can be static or adaptive. Static methods determine traffic split ratios merely on topology information. The static ratios will not respond to current network load and may only change when routing tables are updated. On the other hand, an adaptive method dynamically selects next-hops with knowledge of connected links' loads. The adaptive method typically needs to be implemented on the top of a routing protocol.

Equal-Split Method (M_1)

The Equal-Split method, or simply M_1, evenly distributes traffic over available next-hops. The classical M_1 is only applied to shortest paths, typically called ECMP. However, M_1 can be used for all active paths, regardless of their path lengths. M_1 is a static method. Assume that we use M_1 for n available paths, then each path will get amount f_i of total traffic f:

$$f_i = \frac{1}{n}$$

Proportional-Split Method (M_2)

Proportional-Split method or M_2 is implemented in EIGRP protocol [5]. M_2 decides on each path the amount of traffic that is inversely proportional to the length of that path. Specifically, we assume the existence of n paths with their path lengths $l_1, l_2, \ldots, l_n$ from a source to a destination, M_2 will assign f_i of total traffic f on path i where

$$f_i = \frac{1}{l_i(\sum_{j=1}^{n} \frac{1}{l_j})}$$

The idea of M_2 is to not limit path section on the shortest paths, but the shorter ones likely get more traffic. If all used paths have the same path length, M_2 is identical to M_1.

Exponential-Split Method (M_3)

Exponential-Split method or M_3 is proposed in [96]. Let Δ_i be the path length difference between path i and the shortest path for a certain S-D pair that has n available paths. The fraction of traffic on path i, f_i, is computed as follow:

$$f_i = \frac{1}{\mathrm{e}^{\Delta_i}(\sum_{j=1}^{n} \frac{1}{\mathrm{e}^{\Delta_j}})}$$

M_3 and M_2 share the same idea of using non-shortest paths for traffic forwarding. M_3, however, significantly prioritizes shortest paths with the majority of traffic. M_3 is identical to M_1 when the used paths have the same path length. For example, if $n = 2$, $l_1 = 5$, and $l_2 = 7$, then (f_1, f_2) of M_1, M_2, and M_3 are $(0.5, 0.5)$, $(0.58, 0.42)$, and $(0.88, 0.12)$, respectively. In particular, if we employ M_1 with ECMP, the fraction would be $(1, 0)$ since the longer path does not receive any traffic.

Homeostasis Method (M_4)

Homeostasis is studied in [38]. We simply call it M_4. In M_4, the routing protocol first runs to establish routing tables for each source node. For a destination, the source nodes index their next-hops in the increasing order of path lengths. Upon receipt of an incoming flow, the source will forward it on the shortest path next-hop whose link utilization is under a defined threshold θ (we choose θ of 0.7). When the load of the currently used path reaches the threshold, the residue traffic is switched to the next shortest path.

7.2.2 Post-failure state

Immediately after detecting a failure occurring on a link, the connected nodes will reroute the affected traffic demands over available paths. Such process may face three scenarios. First, some of demands will inevitably get dropped due to the unavailability of alternate next-hops. Second, demands are all re-routed over single available alternate. And finally, affected nodes with more than one alternates have to make decision on how to use them with highest efficiency. In this paper, we address the last scenario with several strategies which are relevant for the distributed routing systems, meaning that a node can only know the path length to the destination of each of its alternates or the current loads on its links to the alternates.

Shortest Path Alternates (S_1)

This decision making is based on path length from the source node to the destination. Among multiple alternates, nodes only select the one on the shortest path from the node to the destination. If there exists more than one, i.e ECMP, all are picked and traffic is split evenly among them.

Least Loaded Alternates (S_2)

This strategy allows an affected node to select the alternate with highest free link capacity between them. Re-routing traffic will be split evenly among multiple alternates if exist.

K-Shortest Alternates (S_3)

S_3 is an extension of S_1, allowing all first K alternates listed in the increasing order of path lengths from the source node to the destination.

K-Least Loaded Alternates (S_4)

Like S_3, S_4 uses up to K alternates, but those are picked according to their increasing order of their link loads.

For a source node with at least two alternates, S_1 and S_2 do not necessarily give different results for *all* destinations. S_1 may be the same as S_2 for a number of destinations, but unlikely for other destinations. This observation also applies to S_3 and S_4. However, with large K, say $K > 3$, S_3 and S_4 are usually similar for most destinations.

7.2.3 Network performance objective

We consider a multi-path routing solution is a combination of three components: routing scheme $r \in \{R_i\}_{1 \le i \le 2}$, load-balancing method $m \in \{M_j\}_{1 \le j \le 4}$, and re-routing strategy $s \in \{S_k\}_{1 \le k \le 4}$. For instance, the routing solution (R_1, M_1, S_2) with the shortest path routing as the base routing means the shortest paths are used when the network is free of faults, and the traffic may be equally split among multiple shortest paths. But when a network fault occurs, the affected node selects only alternates to which the links are least-loaded. We use objective function $\Psi(r, m, s)$ as a metric to measure the performance of traffic distribution for routing solution r. Let $\mathcal{F}$ be the set of possible failures occurring in the network and r' be routing r associating with re-routing strategy s after failure $f \in \mathcal{F}$, we define $\Psi(r, m, s)$ as follows:

$$\Psi(r, m, s) = \begin{cases} \Psi_1 = \Phi(r, m) & \text{if } |\mathcal{F}| = 0 \\ \Psi_2 = \frac{1}{|\mathcal{F}|} \sum_{f \in \mathcal{F}} \Phi(r', m) & \text{otherwise} \end{cases}$$

where $\Phi(r, m) = \sum_{a \in E} \Phi_a(l_a)$ is the network cost function, which is sum of link cost $\Phi_a(l_a)$, defined in [24] and presented below, of each link a with its load l_a. Note that link load l_a is subject to routing r (or r') and load-balancing method m.

$$\Phi_a(l_a) = \begin{cases} l_a & 0 \le l_a/c_a \le \frac{1}{3} \\ 3l_a - \frac{2}{3}c_a & \frac{1}{3} \le l_a/c_a \le \frac{2}{3} \\ 10l_a - \frac{16}{3}c_a & \frac{2}{3} \le l_a/c_a \le \frac{9}{10} \\ 70l_a - \frac{178}{3}c_a & \frac{9}{10} \le l_a/c_a \le 1 \\ 500l_a - \frac{146}{3}c_a & 1 \le l_a/c_a \le \frac{11}{10} \\ 5000l_a - \frac{16318}{3}c_a & \frac{11}{10} \le l_a/c_a < \infty \end{cases}$$

With defined objective function Ψ, we evaluate the routing performance in two stages. First, we minimize Ψ_1 by finding the best combination (r, m) in normal operation, and

with resulted (r, m) we minimize Ψ_2 with the most suitable re-routing strategy s under failure set $\mathcal{F}$.

7.3 Performance evaluation

In this section we assess the robustness of ABR solution in terms of the path diversity and the traffic engineering performance. The latter involves the efficiency of traffic distribution either in normal operation under various discussed load-balancing mechanisms or in post-failure state with different listed re-routing strategies.

7.3.1 Simulation input

We implement the ABR in the flow based network simulator presented in [38]. Input for each simulation is a network topology, a traffic matrix and a link weight setting.

Topologies

We have simulated on three ISP topologies taken from Rocketfuel project [80]. For every topology, single degree nodes are removed since they will not able to have multiple forwarding options. The refined topologies are listed in Table 7.1. Link delays are from an estimate of the propagation delays of the links.

Table 7.1: Network topologies and link delay ranges

Name	POPs	Links	Link delays	Avg. degree
Tiscali	29	73	0.1 - 8.9 ms	2.5
Sprint	32	64	0.1 - 42 ms	2.0
Level3	46	268	0.3 - 38.5 ms	5.8

Since capacity information is not available in Rocketfuel project, we have to generate the capacities for the links in each topology. We take this link capacity information from [38], in which links are assigned with three different capacity values: 100 MB, 400 MB, and 1.6 GB.

Traffic matrix

A TM is generated for each topology by using the simplified gravity model [69]. Accordingly, each node, representing for an urban region, associates with a weight that is

proportional to its population. The estimated demand between two nodes is proportional to the product of the two weights and a scaling factor.

Link weights

We assume that the standard link state routing protocol used is OSPF. Thus, our base routing is the shortest path routing, which is realized by a link weight setting. In this paper, we have two sets of link weights. We use propagation delays of links as link weights (denoted by Delay LWs) and a set of link weights, ranging from 1 to 20, is computed with the local search heuristic [24] on the estimated TM (denoted by Optimized LWs).

Flow generation

The size of a flow, X, is drawn from truncated Pareto distribution with the shape parameter of 1.3, ranging from 8 MB to 8 GB. Flows entering into the network follow global Poisson process in which inter-arrival time of two consecutive flows has negative exponential distribution with $\lambda = \frac{1}{E(X)} \sum_s \sum_t d_s^t$ where d_s^t is the traffic demand of node pair $s - t$ and $E(X)$ is the expected flow size. Traffic rates fall in one of three values: 0.5 Mbps, 1 Mbps, and 10 Mbps with probabilities of 30%, 60%, and 10%, respectively. Those parameters are based on packet traces presented in [63]. Traffic load from all flows on a link is penalized by a link cost $\Phi_a(l_a)$ shown above.

7.3.2 Performance evaluation

Path diversity

Figures 7.1-7.3 show fractions of nodes with *at least* two next-hops and *at least* three next-hops of two base routings, one is shortest path routing with optimized link weights and one is shortest path routing with propagation delay link weights, and their corresponding ABRs. Obviously, ABRs provide significantly better network robustness than the base routings; especially up to 96% nodes with more than one next-hops and up to 87% nodes with more than two next-hops in network Level 3. In addition, the path diversity of ABRs seems to be negligibly dependent on link weight settings.

Load distribution in normal operational state

In this evaluation, we use optimized link weight setting since we believe that it will offer the the most stable network cost compared to other settings. In normal operation, we have five working scenarios: $R_1 + M_1$, $R_2 + M_1$, $R_2 + M_2$, $R_2 + M_3$, and $R_2 + M_4$.

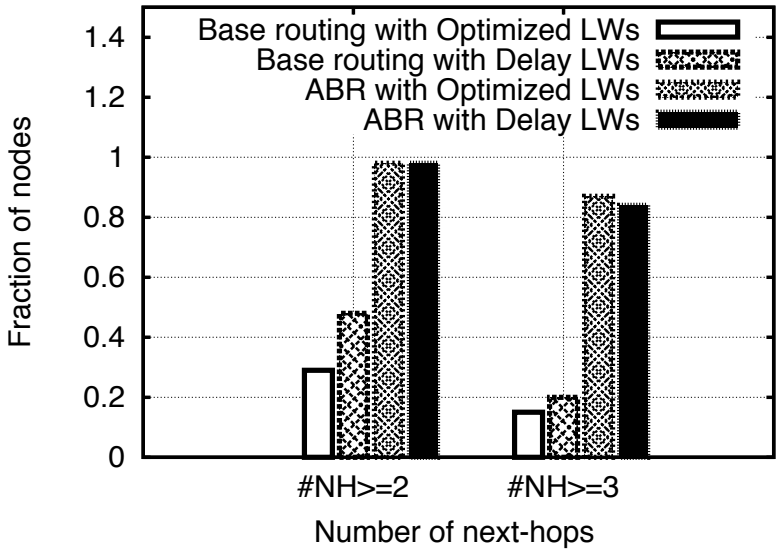


Figure 7.1: Next-hop distribution in network Level3

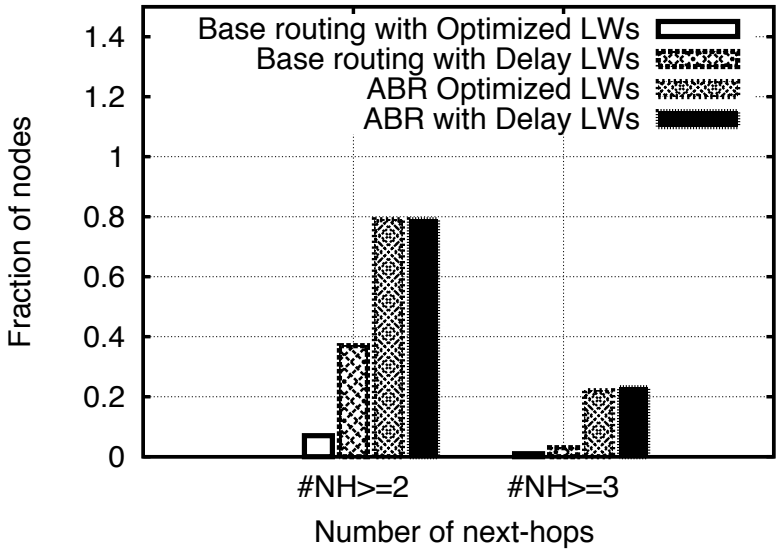


Figure 7.2: Next-hop distribution in network Sprint

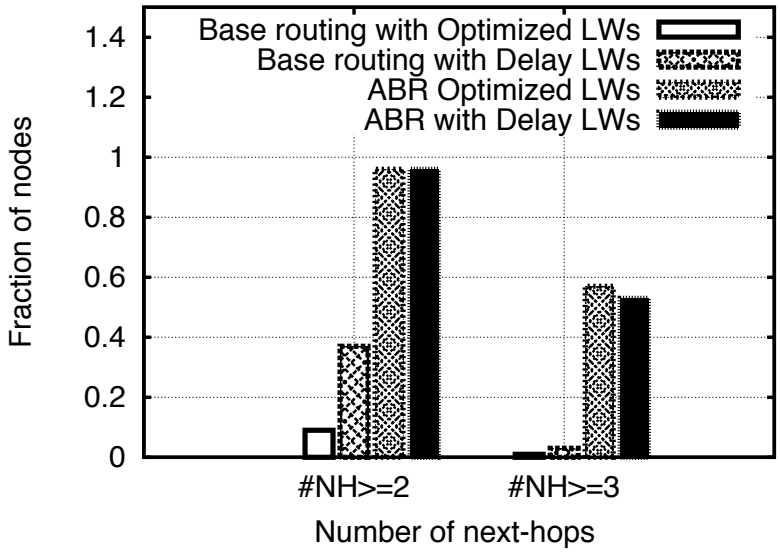


Figure 7.3: Next-hop distribution in network Tiscali Europe

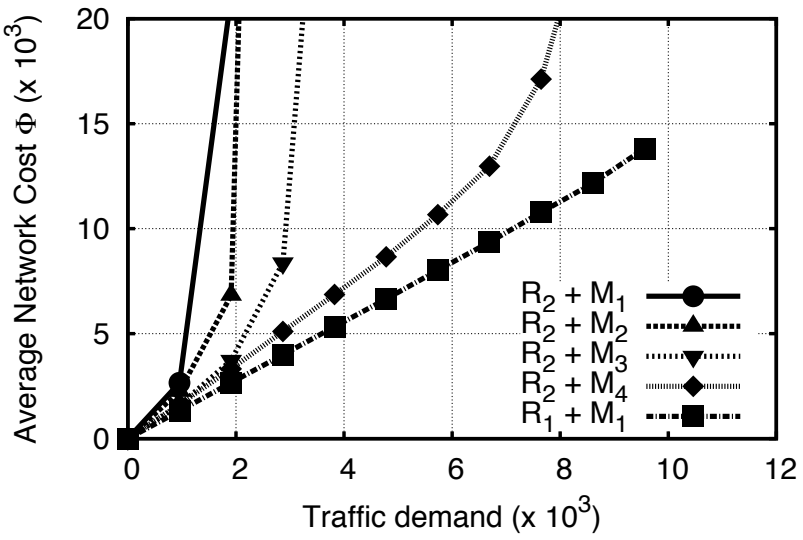


Figure 7.4: Average network costs with increasing TDs in network Level3

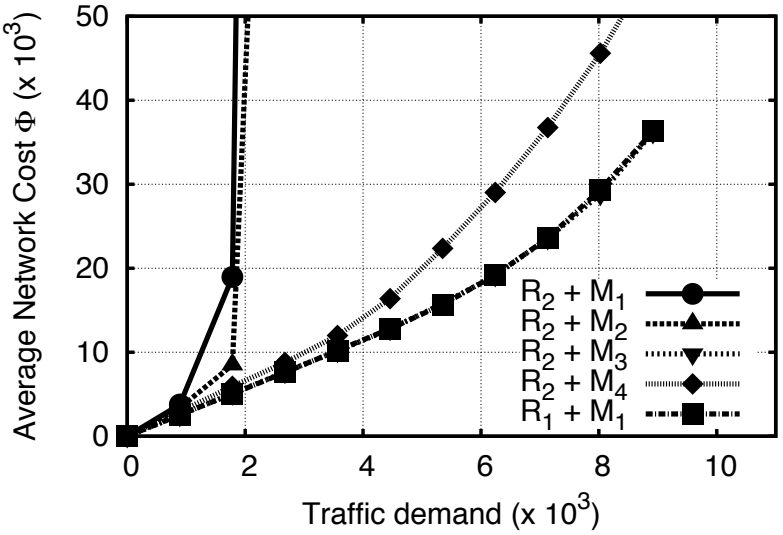


Figure 7.5: Average network costs with increasing TDs in network Sprint

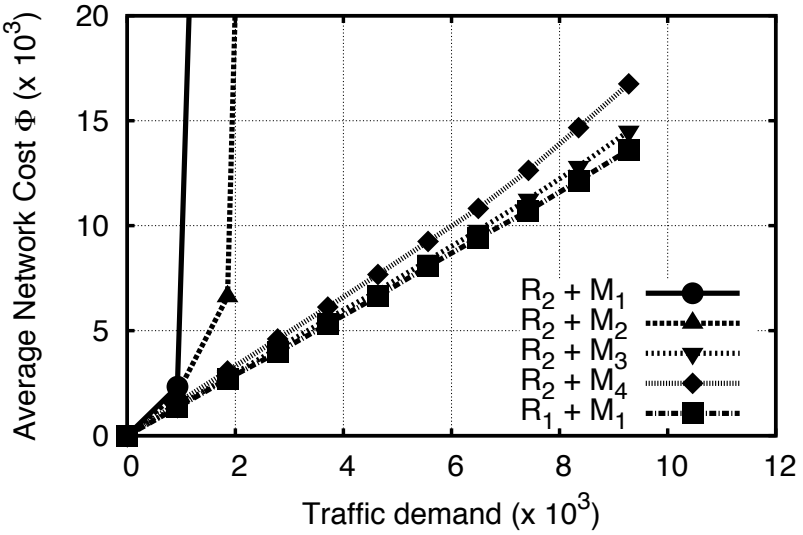


Figure 7.6: Average network costs with increasing TDs in network Tiscali Europe

Figures 7.4-7.6 show network costs against increasing traffic demands (TDs) of the five scenarios. Clearly, $R_1 + M_1$ offers the best traffic distribution in all three topologies. Meanwhile, M_1 shows its inflexibility in scenario $R_2 + M_1$, which results in a high network cost with a small amount of traffic demands. M_2 and M_3 are designed to work with non-equal-cost multiple paths, but both scenarios $R_2 + M_2$ and $R_2 + M_3$ can not compete $R_1 + M_1$. This result, despite being surprising since path diversity of R_1 is poor and M_1 is not flexible, is intuitive. An optimized link weight setting implies a base routing with a corresponding load-balancing method, e.g these in this case are the shortest path routing and the Equal-Split, which can minimize the network cost by carefully selecting high bandwidth paths. Therefore, adding more paths on the base routing will likely degrade the network performance. This also explains why $R_1 + M_1$ performs better than $R_2 + M_4$. Specifically, adaptive method M_4 only seeks to avoid congestion locally, and consequently could be far from an optimal network-wide traffic distribution. PEFT [97] proposed another optimized link weight setting for Exponential-Split, promising a better performance for $R_2 + M_3$. However, such a solution can not work with existing OSPF or IS-IS since PEFT requires not include the shortest path routing.

Load distribution in post-failure state

Given R_1 and M_1 found above, we investigate combination $R_1 + M_1$ with the re-routing strategies. We first select failure set $\mathcal{F}$ of single link faults. However, we will not enumerate all possible single link faults. Instead, a link put in $\mathcal{F}$ should satisfy two conditions:

- Link is critical, meaning that it carries a significant amount of traffic when the base routing is used during normal operation. Clearly, the failure of the critical links will affect significantly the network performance.
- There are at least two alternates backing up for the link towards destination. Otherwise, all rerouting strategies are identical.

Table 7.2 shows the average normalized network costs, each of which is the fraction between Ψ_2 and $\Phi(R_0, M_1)$ where R_0 is the shortest path routing in the network with infinite capacities, with the four re-routing strategies in the three topologies. Failure set $|\mathcal{F}|$ is selected around 25% of total links. From the figures in the table, S_3 has lowest cost, making it the most efficient. S_4 is the runner-up and S_2 is the worst. It means picking alternates with regard to their path lengths is relevant since a shorter path length, given optimized link weights, may be a less loaded path. However, restriction on the shortest one(s) could cause congestion on the selected path since in a well engineered network traffic load is distributed on all links, and few links have enough spare capacity to carry all the back-up traffic after a link failure.

Table 7.2: Load distribution with four re-routing strategies

Topology	Norm. Ψ_2	S_1	S_2	S_3	S_4
Level3	Avg.	4.75	2.89	1.10	1.10
	Max.	15.39	8.26	1.48	1.34
	Min.	1.00	1.00	1.00	1.00
Sprint	Avg.	2.48	3.82	1.57	2.50
	Max.	5.31	9.26	3.02	5.59
	Min.	1.03	1.03	1.04	1.04
Tiscali Europe	Avg.	2.92	8.17	1.05	4.17
	Max.	8.08	28.28	1.12	13.45
	Min.	1.00	1.00	1.00	1.00

7.4 Summary

This chapter seeks to find a good combination between a multi-path routing and a load-balancing method so that the network congestion is minimal either in normal operation or in network fault state.

We propose Augmented Base Routing (ABR) method as a new multi-path routing protocol with the significantly increased robustness for IP networks. There are two major steps in the ABR's construction. First, we assume that the network operators already establish a base routing that results in a congestion-free network for a given TM. Such a base routing can be manually configured or generated by routing protocols such as the shortest path routing with optimized link weights. By including the base routing, the ABR is free of congestion in the fault-free case. Second, the ABR augments the base routing with additional forwarding options, which can be used when the base routing is broken when network failures occur. In general, ABR can work in one of two routing schemes. The first scheme reflects our design goal when constructing the ABR. Specifically, we use the base routing for traffic forwarding in normal operation, and save additional forwarding entries for when there are failures on the base routing. Second, we simply install all next-hops and employ a suitable load-balancing method like Homeostasis. We evaluate both schemes with different load-balancing methods and fast-routing strategies to find a good combination of a routing scheme, a load-balancing method, and a back-up path selection so that we can achieve minimal congestion in a fault situation.

The experiment results reveal that the first routing scheme is better than the second one when we have the TM in advance. That means the base routing optimized on the TM likely minimizes the congestion risk, and therefore adding more forwarding options beside the optimized ones can degrade the network performance. For the base routing which is the shortest path tree with optimized link weights, the idea of only using the shortest

paths for fast-routing at the affected node after the link failure could lead increased congestion. Instead, we should re-route the affected traffic over all possible next-hops to avoid overwhelming a single shortest path. The reason behind it is that a well engineered network traffic load is distributed on all links, and few links have enough spare capacity to carry all the backup traffic after a link failure.

Chapter 8

Conclusion

Since the Internet has evolved to a preferred transportation medium for the modern Internet services, the reliability of the Internet has become crucial in guaranteeing the quality for those services. Unfortunately, currently deployed routing methods offered insufficient protection against diverse network failures which occur frequently and unpredictably. In this work, we propose the permutation routing method as part of a multi-path routing protocol, which can significantly increase the network robustness against network component failures. Particularly, the design of the permutation routing possesses four important properties that make it be easily adopted in the traditional IP networks. First, it does not require the network operators to change the traditional hop-by-hop forwarding strategy that is optimized for the fault free case. Second, it does not require addition of fault-information in the packet headers. Third, it does not suffer from routing loops, even when there are multiple faulty components in the network. Finally, they can work with the existing standard link state routing protocols such as OSPF or IS-IS.

Permutation routing, in general, is a technique that we can use as both the loop-free criteria or the representation for a loop-free routing. Specifically, a permutation routing towards a destination is a sequence of nodes in which except for the destination each node must have at least one neighboring node that is placed before it in that sequence. In an extended version, a permutation routing towards a destination is a sequence of links in which except for the destination each link must have at least one neighboring link that occurs before it in that sequence. The routing will be loop-free as long as traffic can only be forwarded in one direction with respect to the node/link ordering in the sequence. The first type of permutation routing is used to represent a routing in the network with node-specific forwarding while the second one goes with interface-specific forwarding. The above idea implies that we can construct a loop-free routing by means of permutation routing.

In order to construct a routing by the permutation routing method, we first formulate

the routing objective in form of logic functions, also called routing constraint functions, each of which is specific for one source node. The routing constraint function describes a desired next-hop assignment of the source node. The problem of creating a permutation routing with given routing constraints of source nodes is a constraint satisfaction problem and can be solved with the backtracking algorithm.

The main operation of the algorithm is to successively assigning a node to a variable, which stands for a position in the permutation routing. The assignment is valid if the node satisfies the routing constraint designed for it. The central components of the algorithm are two functions, Domain and Select, which work as filters to govern the assignments. Function Domain creates a candidate set which includes all potential nodes for the variable while function Select applies the constraint to the candidate set to extract a valid node for the assignment. The algorithm terminates when there exists a permutation routing of all nodes or no such permutation routing found after all back-track steps.

The permutation routing concept and the construction algorithm lay the basis for the contributions of generating routing with the increased robustness for IP networks. The contributions are three instances of the permutation routing for various versions of IP networks and their applications in terms of fault-tolerance and load-balancing. Subsequently, we summarize each of the contribution.

8.1 Summary of contributions

The networks we first consider have routers with basic functionality. That is each router installs a single routing table which may support ECMP, i.e. there can be more than one output ports defined per prefix. The bidirectional links in this network type are only allowed to carry packets in one direction towards a destination.

We propose two routing objectives for this IP networks. We call them Next-Hop Optimal Routing (NHOR) and Next-Hop Optimal Routing with Shortest Path compatibility (NHOR-SP). Both routings share a common requirement which maximizes the number of S-D pairs with at least two next-hops. Furthermore, the latter is desired to include the shortest paths given by the shortest path routing algorithm.

Theoretically, the backtracking algorithm can help construct permutation routings that result in NHOR and NHOR-SP. However, in the worse cases the computational complexity could be excessively large. For that reason, we simplify the routing constraints at the extent that can balance the two goals: approximating the routing objectives and obtaining a linear complexity. The approximation results are called ANHOR and ANHOR-SP. In comparison with LFA, which is an extension of the shortest path routing, ANHOR-SP can improve up to 50% protection coverage. If shortest path constraint is ignored,

ANHOR can offer 98% protection coverage for a network with the average node degree of greater than 5.

The above forwarding pattern inevitably will leave some nodes with only one routing choice. Then we observe that in carefully chosen cases, we can let forwarding go in the opposite direction without creating forwarding loops. This idea fortunately fits the networks where routers have functionality that allows them to prohibit U-turns on backup paths. That means when the router receives a packet from an interface on a backup path, it will never send the packet back on that backup path. In this type of network, a pair of routers can share a common forwarding link that we called the joker-link. The routing with joker-links are joker-capable routings.

We define two routing objectives for joker-capable routings, JNHOR and JNHOR-SP, as extensions of NHOR and NHOR-SP, respectively. Literally, we assign some directed links in NHOR or NHOR-SP to joker-links such that the resulting JNHOR or JNHOR-SP can maximize the number of S-D pairs with at least two next-hops, either primaries or backups (on the joker-links). Like NHOR-SP, JNHOR-SP includes shortest path next-hops computed by the shortest path routing algorithms. In addition, it is required that the next-hop on the shortest path should not coincide with the joker-link since we for some explained reasons will set those next-hops primary next-hops.

We enhance the above algorithms of ANHOR and ANHOR-SP to produce permutation routings for JNHOR and JNHOR-SP, respectively. The enhancements lie in the new forwarding rule for the permutation routing to handle the special forwarding style need for joker-links and a procedure implemented in function Select to identify joker-links. Those modifications add only small complexity, and therefore the whole algorithm has the complexity that is still linear with the number of nodes in the network. With such a small change, we can achieve significant increase in network robustness. Namely, JNHOR offers an improvement up to 70% compared to ANHOR-SP and up to 20% compared to ANHOR. Especially, JNHOR reaches 100% protection coverage in the network with the average node degree of greater than 5. When the shortest path constraint is included, the improvement of fault-tolerance is limited. Hence, JNHOR-SP only slightly higher than ANHOR-SP.

Our third contribution is to construct robust interface based routing for IP networks. The interface base routing requires that each of its interface install a routing table for a destination. In addition, that routing table can support the multi-path routing which maps each incoming interface to multiple outgoing ones. Typically, the router in this type of network employs interface-specific forwarding which treats packets differently regarding to their incoming interface and destination address. The routing objective we offer for IP networks with the interface-specific forwarding is Interface Next-Hop Optimal Routing

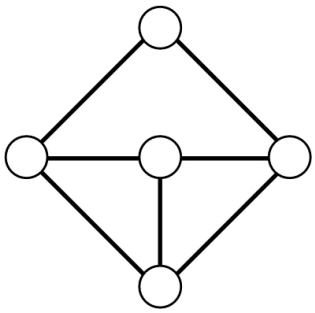
(iNHOR). iNHOR aims to maximize the number of primary interface destination pairs with at least two next-hops towards each destination. Note that the primary interfaces are those on the shortest path tree given by the shortest path algorithm. The permutation routing of interfaces can help us to construct an interface based routing that approximates the iNHOR from a network topology. We call the resulting routing Approximate iNHOR (AiNHOR). The permutation routing of interfaces is actually the permutation routing of vertices of the line graph which is computed from the network topology. The line graph of a topology has vertices which represent the directed links of the topology. In addition, the connectivity of the line graph is calculated from the connectivity of the topology. As a result, the construction of a permutation routing for AiNHOR is similar to that of ANHOR-SP, but the input graph is the line graph of the network topology. The performance analysis shows that the protection coverage given by AiNHOR in six tested ISP topologies are all above 97%. Of the routings that include the shortest path tree that we describe above, i.e. ANHOR-SP and JNHOR-SP, AiNHOR is dominant.

The permutation routings provide significantly high path diversity by maximizing the number of source destination pairs (or primary interface destination pairs) with at least two next-hops. Those next-hops have two main benefits. First, the availability of multiple next-hops towards a destination allows fast failover from link or node failures. Second, traffic can be balanced over several next-hops in order to improve network utilization and avoid congestion. Our last contribution deals with the load-balancing issue in two operational states of the IP networks: fault-free and faulty. In order to guarantee that network is free of congestion in normal operation, we construct a base routing for the given TM. The base routing is a routing that is optimized for the average network-wide cost function and the TM. Despite working well in normal operation, the base routing is typically vulnerable to network failures. We therefore construct a routing called Augmented Based Routing that can augment the base routing with additional forwarding options. We then seek to answer the question of how to assign back-up next-hops for minimal congestion in a fault situation. Our simulation reveals that we should assign *all* additional next-hops beside the next-hops on the base routing to back-up next-hops and the rerouted traffic should use them all to minimize network congestion. Note that the use of all available next-hops for traffic forwarding in the fault-free network will likely degrade the network performance.

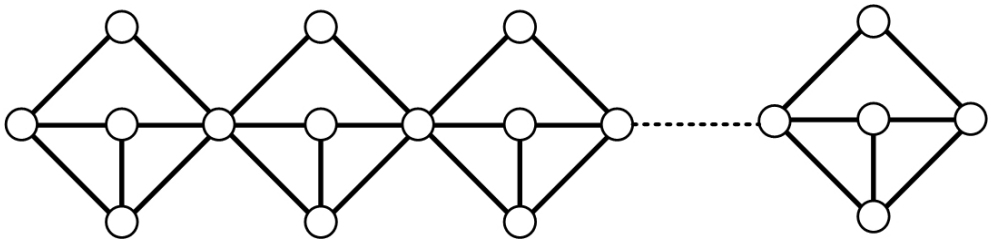
8.2 Future work

We have presented the permutation routing concept and its construction framework in order to realize a general routing objective. We describe three different versions of IP

networks and propose their associated routing objectives to increase routing robustness. However, the permutation routing problem still leaves unanswered questions on which we will keep working on.



(a) Topology unit



(b) Repeating topology constructed from the topology unit

Figure 8.1: Illustration of finding the optimal gap

- The problem of finding a routing towards a destination with maximum protection coverage is proven to be NP-complete [41]. For that reason, we propose the permutation routing method as a heuristic to compute the optimal next-hop routings. Our heuristic has several advantages. For a small topology, less than 15 nodes, the heuristic can compute the optimality by searching through the *reduced* search space. For a topology whose connectivity allows to produce a routing with ideal protection coverage, the heuristic can find such one. For a arbitrary topology, the heuristic can result in near-optimal protection coverages, e.g. ANHOR can reach 98% in AS6461; JNHOR can offer in all the cases above 95%, and above 97% with AiNHOR. However, the gap between the resulting permutation routing and the optimal next-hop routing for each type of the permutation routings is still unknown. One direction to estimate the gap is finding the smallest topology unit that the permutation routing method for a specific kind of permutation routings, without any backtrack step, fails

to compute the maximum protection coverage. Then we compute the gap in the worst case that occurs in a topology constructed from numerous such a topology unit. For example, we can find the gap in the worst case between ANHOR and NHOR of up to 25% in the topology with repeating pattern (Figure 8.1(b)) where the topology unit (Figure 8.1(a)) has five nodes and seven links. The gap in the topology unit is maximum at 10%.

- In Chapter 7, we propose a solution to balance the network load with the Augmented Base Routing (ABR). Specifically, the ABR can be one of these ANHOR-SP, JNHOR-SP, and AiNHOR presented in Chapter 4, Chapter 5, and Chapter 6, respectively, where the routings are designed to include the base routing that is the shortest path routing. So far, we have not proposed an efficient load-balancing mechanism for those permutation routings that are not compatible with the shortest path routing like ANHOR, JNHOR or an instance of the permutation routing of interfaces. However, we observe that ANHOR and JNHOR can work well with Exponential-Split or Homeostasis if the total traffic demands are relatively small compared to the network capacity. Therefore, they are suitable for those networks that prefer fault-tolerance to traffic engineering.

- When multiple failures occur in the network, a part of the routing will be disrupted. The whole routing should be re-computed to establish new paths that avoid the failures to the destination. However, in may cases the routing is not necessarily re-computed entirely because most of the routing graph is not affected by the failures and can remain unchanged. For that reason, a mechanism that allows only affected routers to re-compute their paths to the destination is needed to reduce the convergence time. The new mechanism can be considered as Incremental Permutation Routing and share with the Incremental Shortest Path First the same design goal. However, if only the failure of a single link can partition a large portion of the network, the failure should be more extensive to significantly break down the permutation routing.

- In general, the permutation routing can be understood as a loop-free criterion that many other routing objectives can use to produce the corresponding loop-free routings. For example, one may want to construct multiple routing configurations towards a destination, each of which is required to be loop-free. In that case, each of configuration can be built with the permutation routing method. In addition, the permutation routing can be exploited to support the QoS of traffic. Specifically, each router may construct multiple permutation routing for a destination, each of which is dedicated to a class of service. Other applications of the permutation rout-

ing can be developed to solve dead-lock issues of the flow control problem in the interconnection network.

Appendix A

Publications

The work presented in this thesis has resulted in three conference publications and two in submission papers. This appendix contains summaries of the content of each paper and the individual contributions of authors.

A.1 Conference publications

Title: Permutation Routing for Increased Robustness in IP Networks [86]
Authors: Hung Quoc Vo and Olav Lysne and Amund Kvalbein
Published: IFIP Networking, 2012.
Summary: This paper presents the permutation routing concept and its applications in constructing routings with the increased robustness for the traditional IP networks (described in Chapter 3 and Chapter 4). The arguments in the paper are backed with simulation results.
Contributions: Olav Lysne first gave the idea of the permutation routing. All authors spent plenty of time on discussing the feasibility of the idea and constructing proofs. Hung Vo was responsible for the evaluation with simulations and the writing. Amund Kvalbein gave feedback on the paper's structure and the text.

Title: Increased Robustness with Interface Based Permutation Routing [87]
Authors: Hung Quoc Vo and Olav Lysne and Amund Kvalbein
Published: The 2013 IEEE International Conference on Communications (ICC 2013).
Summary: The work is an extension of the Networking 2012 paper [86]. The paper presents a new permutation routing method to increase the robustness for the IP networks with forwarding-specific forwarding. The simulation is used to evaluate the performance of the method. The main content of this paper is described in Chapter 5.

Contributions: Olav first gave the idea of permutation routing of interfaces. Hung Vo proposed theorems and their proofs involved in the correctness of the idea, and conducted the writing and simulations. Amund Kvalbein supported with the writing and the paper structure. All authors took part in numerous technical discussion and provided the feedback on textual content.

Title: Routing with Joker Links for Maximized Robustness [88]
Authors: Hung Quoc Vo and Olav Lysne and Amund Kvalbein
Published: IFIP Networking, 2013
Summary: In this paper, we propose a routing permutation method to improve the robustness for IP networks which support disallowed U-turns functionality. The paper is adapted in Chapter 5.
Contributions: Hung Vo proposed the idea and conducted the performance evaluation and the writing. Amund and Olav provided feedback on the paper's content and involved in many technical discussion.

Title: On load-balancing in multi-path routing protocols
Authors: Hung Quoc Vo and Olav Lysne and Amund Kvalbein
In submission: The 2014 IEEE International Conference on Communications (ICC 2014)
Summary: This paper explored how to install next-hops in order to maximize the network performance either in normal operation or in a failure state. The content of this paper is described in Chapter 7.
Contributions: Hung Vo proposed the idea and conducted the simulations and the writing. The simulation implementation is based on Amund's previous work [38]. All authors provided feedback on the paper's content.

A.2 Journal articles

Title: Permutation Based Routing
Authors: Hung Quoc Vo and Olav Lysne and Amund Kvalbein
To be submitted
Summary: This journal gives the general idea of permutation routing and its applications that we presented in our previous work. In addition, this work will further improve the algorithms to give a better robustness for IP networks, for example finding node-protecting alternates instead of only focusing on link-protecting alternates as in the previous versions.
Contributions: Olav Lysne constructed the main structure of this journal. Hung Vo developed the writing and added new aspects of the permutation routing that has

not mentioned in the previous work. All authors provided feedback on the textual content.

Bibliography

[1] C. Alaettinoglu, V. Jacobson, and H. Yu. Towards milli-second IGP convergence. *IEFT Internet Draft*, expired in May 2001.

[2] M. Alicherry and R. Bhatia. Simple pre-provisioning scheme to enable fast restoration. *IEEE/ACM Transaction on Networking*, 15:400–412, April 2007.

[3] A. Atlas and A. Zinin. Basic specification for IP fast reroute: Loop-free alternates. *RFC 5286*, September 2008.

[4] R. B., , and M. Raman. A heuristic approach to service restoration in MPLS networks. *2001 IEEE International Conference on Communications (ICC 2001)*, pages 117–121, June 2001.

[5] J. B. B. Albrightson, J. Garcia-Luna-Aceves. EIGRP - a fast routing protocol based on distance vectors. *Networld/Interop*, 1994.

[6] Y. Bejerano, Y. Breitbart, A. Orda, R. Rastogi, and A. Sprintson. Algorithms for computing QoS paths with restoration. *IEEE/ACM Transactions on Networking*, 13:648–661, June 2005.

[7] F. Blanchy, L. Melon, and G. Leduc. A preemption-aware on-line routing algorithm for MPLS networks. *Telecommunication System*, pages 187–206, October 2003.

[8] R. Boutaba, W. Szeto, and Y. Iraqi. DORA: Efficient routing for MPLS traffic engineering. *Journal of Network and Systems Management*, 10:309–325, August 2002.

[9] M. C. S. Bryant and C. Filsfils. Remote LFA FRR. *Internet Draft (work in progress)*, expired in Jun. 2013.

[10] S. Bryant, C. Filsfils, S. Previdi, and M. Shand. IP fast reroute using tunnels. *IEFT Internet Draft*, expired in May 2008.

[11] H. J. Chao. Next generation routers. *Proceedings of the IEEE*, 90:1518–1558, 2002.

[12] C. Chekuri, A. Gupta, A. Kumar, J. S. Naor, and D. Raz. Building edge-failure resilient networks. *Integer Programming and Combinatorial Optimization (IPCO 2002)*, pages 439–456, May 2002.

[13] S. Cho, T. Elhourani, and S. Ramasubramanian. Independent directed acyclic graphs for resilient multipath routing. *IEEE/ACM Transaction on Networking*, 20:153–162, February 2012.

[14] Cisco. OSPF incremental SPF. *http://www.cisco.com*.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to algorithm (2th edition). *MIT press*.

[16] L. Csikor and G. Retvari. IP fast reroute with remote loop-free alternates: The unit link cost case. *4th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT 2012)*, pages 663–669, October 2012.

[17] J. C. de Oliveira, C. Scoglio, l. F. Akyildiz, and G. Uhl. New preemption policies for Diffserv-aware traffic engineering to minimize rerouting in MPLS networks. *IEEE/ACM Transactions on Networking*, 12:733–745, August 2004.

[18] R. Dechter. Constraint processing. *Morgan Kaufmann*, 2003.

[19] T. Elhourani, S. Ramasubramanian, and A. Kvalbein. Enhancing shortest path routing for resilience and load balancing. *2011 IEEE International Conference on Communications (ICC 2011)*, pages 1–6, June 2011.

[20] A. Elwalid, C. Jin, S. Low, and I. Widjaja. MATE: MPLS adaptive traffic engineering. *The 20th IEEE Conference on Computer and Communications Societies (INFOCOM 2001)*, 3:1300–1309, April 2001.

[21] G. Enyedi and G. Rétvári. A loop-free interface-based fast reroute technique. *Next Generation Internet Networks*, pages 39–44, April 2008.

[22] G. Enyedi, P. Szilagyi, G. Retvari, and A. Csaszar. IP fast reroute: Lightweight not-via without additional addresses. *2009 IEEE International Conference on Computer Communications Societies (INFOCOM 2009)*, pages 2771–2775, April 2009.

[23] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and K. van der Merwe. The case for separating routing from routers. *ACM SIGCOMM Workshop on Future Directions in Network Architecture*, 6, September 2004.

[24] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. *The 19th IEEE Conference on Computer and Communications Societies (INFOCOM 2000)*, pages 519–528, March 2000.

[25] P. Francois, S. Bryant, B. Decraene, and M. Horneffer. Loop-free alternate (LFA) applicability in service provider (SP) networks. *RFC 6571*, June 2012.

[26] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure. Achieving sub-second IGP convergence in large IP networks. *SIGCOMM Comput. Commun. Rev.*, pages 35–44, July 2005.

[27] M. Gjoka, V. Ram, and X. Yang. Evaluation of IP fast reroute proposals. *International Conference on Communication Systems Software and Middleware 2007 (COMSWARE 2007)*, pages 1–8, January 2007.

[28] A. Greenberg, G. Hjalmtysson, D. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A clean slate 4D approach to network control and management. *ACM SIGCOMM Computer Communication Review*, 35:41–54, October 2005.

[29] R. Guerin, A. Orda, and D. Williams. QoS routing mechanisms and OSPF extensions. *1997 IEEE Global Telecommunications Conference (GLOBECOM 1997)*, 1997.

[30] A. Gupta, B. N. Jain, and S. Tripathi. QoS aware path protection schemes for MPLS networks. *The 15th international conference on Computer communication (ICCC 2002)*, pages 103–118, 2002.

[31] S. Hanks, T. Li, D. Farinacci, and P. Traina. Generic routing encapsulation. *RFC 1701*, October 1994.

[32] A. F. Hansen, O. Lysne, T. Cicic, and S. Gjessing. Fast proactive recovery from concurrent failures. *IEEE International Conference on Communications (ICC 2007)*, pages 115–122, June 2007.

[33] M. Hara and T. Yoshihiro. Adaptive load balancing based on IP fast reroute to avoid congestion hot-spots. *2011 IEEE International Conference on Communications (ICC 2011)*, pages 1–5, June 2011.

[34] R. L. Hemminger and L. W. Beineke. Line graphs and line digraphs. *Selected Topics in Graph Theory*, pages 271–305, 1978.

[35] M. Herzberg, S. J. Bye, and A. Utano. The hop-limit approach for spare-capacity assignment in survivable networks. *IEEE/ACM Transaction on Networking*, 3:775–784, December 1995.

[36] S. Kandula, D. Katabi, B. Davie, and A. Charny. Walking the tightrope: responsive yet stable traffic engineering. *ACM SIGCOMM Computer Communication Review*, 35:253–264, August 2005.

[37] M. Kodialam and T. V. Lakshman. Minimum interference routing with applications to MPLS traffic engineering. *The 19th IEEE Conference on Computer and Communications Societies (INFOCOM 2000)*, pages 884–893, March 2000.

[38] A. Kvalbein, C. Dovrolis, and C. Muthu. Multipath load-adaptive routing: putting the emphasis on robustness and simplicity. *17th IEEE International Conference on Network Protocols (ICNP 2009)*, October 2009.

[39] A. Kvalbein, A. F. Hansen, T. Čičic, S. Gjessing, and O. Lysne. Multiple routing configurations for fast IP network recovery. *IEEE/ACM Transaction on Networking*, 17:473–486, April 2009.

[40] A. Kvalbein, T. Čičic, and S. Gjessing. Post-failure routing performance with multiple routing configurations. *The 26th IEEE International Conference on Computer Communications Societies (INFOCOM 2007)*, pages 98–106, May 2007.

[41] K.-W. Kwong, R. G. L. Gao, and Z.-L. Zhang. On the feasibility and efficacy of protection routing in IP networks. *IEEE/ACM Transactions on Networking*, 19(5):1543–1556, October 2011.

[42] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. S., and I. Stoica. Achieving convergence-free routing using failure-carrying packets. *ACM SIGCOMM Computer Communication Review*, pages 241–252, August 2007.

[43] J. Lau, M. Townsley, and I. Goyret. Layer two tunneling protocol - version 3 (L2TPv3). *RFC 3931*, March 2005.

[44] S. Lee, Y. Yu, S. Nelakuditi, Z.-L. Zhang, and C.-N. Chuah. Proactive vs reactive approaches to failure resilient routing. *The 23rd IEEE Conference on IEEE Computer and Communications Societies (INFOCOM 2004)*, pages 178–186, March 2004.

[45] S. S. W. Lee, P.-K. Tseng, A. Chen, and C.-S. Wu. Non-weighted interface specific routing for load-balanced fast local protection in IP networks. *IEEE International Conference on Communications (ICC 2011)*, pages 1–6, June 2011.

[46] S. S. W. Lee, P.-K. Tseng, K.-Y. Li, W.-Y. Chang, and A. Chen. Interface specific fast failure rerouting for load balanced IP networks. *2011 IEEE Symposium on Computers and Communications (ISCC 2011)*, pages 316–319, July 2011.

[47] A. Li, X. Yang, and D. Wetherall. Safeguard: safe forwarding during route changes. *The 5th international conference on Emerging networking experiments and technologies (CoNEXT 2009)*, pages 301–312, December 2009.

[48] L. Li, M. Buddhikot, C. Chekuri, and K. Guo. Routing bandwidth guaranteed paths with local restoration in label switched networks. *IEEE Journal on Selected Areas in Communications*, 23:437–449, February 2005.

[49] Y. Liu, D. Tipper, and P. Siripongwutikorn. Approximating optimal spare capacity allocation by successive survivable routing. *IEEE/ACM Transaction on Networking*, 13:198–211, February 2005.

[50] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot. Characterization of failures in an IP backbone. *23rd IEEE International Conference on Computer Communications (INFOCOM 2004)*, pages 2307–2317, October 2004.

[51] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matrix estimation: existing techniques and new directions. *ACM SIGCOMM Computer Communication Review (SIGCOMM 2002)*, pages 161–174, October 2002.

[52] P. Merindol, J.-J. Pansiot, and S. Cateloin. Improving load balancing with multipath routing. *17th International Conference on Computer Communications and Networks (ICCCN 2008)*, pages 1–8, August 2008.

[53] D. Mitra and K. G. Ramakrishnan. A case study of multiservice, multipriority traffic engineering design for data networks. *1999 IEEE Global Communications Conference (GLOBECOM 1999)*, pages 1077–1083, December 1999.

[54] P. Mrindol, P. Francois, O. Bonaventure, S. Cateloin, and J.-J. Pansiot. An efficient algorithm to enable path diversity in link state routing networks. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 55:1132–1149, April 2011.

[55] P. Narvaez, K.-Y. Siu, and H.-Y. Tzeng. Efficient algorithms for multi-path link state routing. *ISCOM*, 1999.

[56] A. Nccui, N. T. S. Bhattacharyya, and C. Diot. IGP link weight assignment for operational tier-1 backbones. *IEEE/ACM Transactions on Networking*, 15(4):789–802, August 2007.

[57] S. Nelakuditi, S. Lee, Y. Yu, and Z.-L. Zhang. Failure insensitive routing for ensuring service availability. *The 11th international conference on Quality of service (IWQoS 2003)*, pages 287–304, 2003.

[58] S. Nelakuditi, S. Lee, Y. Yu, Z.-L. Zhang, and C.-N. Chuah. Fast local rerouting for handling transient link failures. *IEEE/ACM Transactions on Networking*, 15:359–372, April 2007.

[59] S. Nelakuditi, Z. Zhong, J. Wang, R. Keralapura, and C.-N. Chuah. Mitigating transient loops through interface-specific forwarding. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, pages 593–609, February 2008.

[60] K. Nemeth and G. Retvari. Traffic splitting algorithms in multipath networks: Is the present practice good enough? *XVth International Networks,Telecommunications Network Strategy, and Planning Symposium (NETWORKS 2012)*, pages 1–6, October 2012.

[61] Y. Ohara, S. Imahori, and R. V. Meter. MARA: Maximum alternative routing algorithm. *2009 IEEE International Conference on Computer Communications (INFOCOM 2009)*, pages 298–306, April 2009.

[62] M. Peyravian and A. D. Kshemkalyani. Connection preemption: Issues, algorithms, and a simulation study. *The 16th IEEE Conference on Computer and Communications Societies (INFOCOM 1997)*, pages 143 – 151, April 1997.

[63] R. Prasad and C. Dovrolis. Beyond the model of persistent TCP flows: Open-loop vs closed-loop arrivals of non-persistent flows. *41st Annual Simulation Symposium (ANSS 2008)*, pages 121 – 130, October 2008.

[64] C. Reichert and Y. Glickmann. Two routing algorithms for failure protection in IP networks. *the 10th IEEE Symposium on Computers and Communications (ISCC 2005)*, pages 97–102, 2005.

[65] G. Rétvári, J. J. Bíró, and T. Cinkler. On shortest path representation. *IEEE/ACM Transaction Networking*, 15(6):1293–1306, December 2007.

[66] G. Retvari, J. Tapolcai, G. Enyedi, and A. Csaszar. IP fast reroute: Loop Free Alternates revisited. *The 30th IEEE Conference on Computer and Communications Societies (INFOCOM 2011)*, pages 2948–2956, April 2011.

[67] G. Robertson and S. Nelakuditi. Handling multiple failures in IP networks through localized on-demand link state routing. *IEEE Transactions on Network and Service Management*, 9:293 – 305, September 2012.

[68] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. *RFC 3031*, January 2001.

[69] M. Roughan. Simplifying the synthesis of internet traffic matrices. *ACM SIGCOMM Computer Communication Review (SIGCOMM 2005)*, pages 93–96, October 2005.

[70] M. Roughan, A. Greenberg, C. Kalmanek, M. Rumsewicz, J. Yates, and Y. Zhang. Experience in measuring backbone traffic variability: models, metrics, measurements and meaning. *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement*, pages 91–92, 2002.

[71] H. Saito, Y. Miyao, and M. Yoshida. Traffic engineering using multiple multipoint-to-point LSPs. *The 19th IEEE Conference on Computer and Communications Societies (INFOCOM 2000)*, 2:894–901, March 2000.

[72] G. Schollmeier, J. Charzinski, and A. Kirstadter. Improving the resilience in IP networks. *High Performance Switching and Routing 2003 HPSR Workshop*, pages 91–96, June 2003.

[73] B. S. W. Schröder. Ordered sets: An introduction. *Birkhäuser, Boston*.

[74] M. Shand and S. Bryant. A framework for loop-free convergence. *RFC 5715*, January 2010.

[75] M. Shand and S. Bryant. IP fast reroute framework. *RFC 5714*, January 2010.

[76] M. Shand, S. Bryant, and S. Previdi. IP fast reroute using not-via addresses. *Internet Draft (work in progress)*, expired in Jun. 2012.

[77] V. Sharma and F. Hellstrand. Framework for multi-protocol label switching MPLS-based recovery. *RFC 3469*, February 2003.

[78] C.-C. Shyur, T.-C. Lu, and U.-P. Wen. Applying tabu search to spare capacity planning for network restoration. *Computers and Operations Research*, 26:1175–1194, October 1999.

[79] W. Simpson and Daydreamer. IP in IP tunneling. *RFC 1853*, October 1995.

[80] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with rocketfuel. *IEEE/ACM Transactions on Networking*, 12:2–16, February 2002.

[81] A. Sridharan, R. Guérin, and C. Diot. Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks. *IEEE/ACM Transaction on Networking*, 13:234–247, April 2005.

[82] D. Stamatelakis and W. D. Grover. IP layer restoration and network planning based on virtual protection cycles. *IEEE Journal on Selected Areas in Communications*, 18:1938–1949, September 2006.

[83] J. Tapolcai. Sufficient conditions for protection routing in IP networks. *Springer Optimization Letters*, 7:723–730, 2013.

[84] E. Tsang. Foundations of constraint satisfaction. *Academic Press*, 1993.

[85] B. Van Caenegem, N. Wauters, and P. Demeester. Spare capacity assignment for different restoration strategies in mesh survivable networks. *1997 IEEE International Conference on Communications (ICC 1997)*, pages 288–292, June 1997.

[86] H. Q. Vo, O. Lysne, and A. Kvalbein. Permutation routing for increased robustness in IP networks. *IFIP Networking 2012 (Networking 2012)*, 1:217–231, May 2012.

[87] H. Q. Vo, O. Lysne, and A. Kvalbein. Increased robustness with interface based permutation routing. *2013 IEEE International Conference on Communications (ICC 2013)*, 2013.

[88] H. Q. Vo, O. Lysne, and A. Kvalbein. Routing with joker links for maximized robustness. *IFIP Networking 2013 (Networking 2013)*, May 2013.

[89] S. Vutukury and J. J. Garcia-Luna-Aceves. A simple approximation to minimum-delay routing. *ACM SIGCOMM Computer Communication Review (SIGCOMM 1999)*, pages 227–238, October 1999.

[90] Y. Wang, H. Wang, A. Mahimkar, R. Alimi, Y. Zhang, L. Qiu, and Y. R. Yang. R3: resilient routing reconfiguration. *ACM SIGCOMM Computer Communication Review (SIGCOMM 2010)*, pages 291–302, October 2010.

[91] Y. Wang and Z. Wang. Explicit routing algorithms for Internet traffic engineering. *Eight International Conference on Computer Communications and Networks (ICCCN 1999)*, pages 582–588, October 1999.

[92] Z. Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14(7):1228–1234, September 2006.

[93] K. Xi and H. J. Chao. IP fast rerouting for single-link/node failure recovery. *Fourth International Conference on Broadband Communications, Networks and Systems (BROADNETS 2007)*, pages 142–151, September 2007.

[94] K. Xi and H. J. Chao. IP fast reroute for double-link failure recovery. *IEEE Global Telecommunications Conference (GLOBECOM 2009)*, pages 1–8, December 2009.

[95] Y. Xiong, D. Xu, and C. Qiao. Achieving fast and bandwidth-efficient shared-path protection. *Journal of Lightwave Technology*, 21:365–371, 2003.

[96] D. Xu, M. Chiang, and J. Rexford. DEFT: Distributed exponentially-weighted flow splitting. *The 26th IEEE International Conference on Computer Communications (INFOCOM 2007)*, pages 71–79, May 2007.

[97] D. Xu, M. Chiang, and J. Rexford. Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering. *IEEE/ACM Transaction on Networking*, 19:1717–1730, December 2011.

[98] Z. Zhong, S. Nelakuditi, Y. Yu, S. Lee, J. Wang, and C.-N. Chuah. Failure inferencing based fast rerouting for handling transient link and node failures. *24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005)*, March 2005.