A MULTIPATH FAULT-TOLERANT PROTOCOL FOR ROUTING IN PACKET-SWITCHED COMMUNICATION NETWORKS

Anupama Krishnan, B.E..

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

May 2003

APPROVED:

Armin R. Mikler, Major Professor
Azzedine Boukerche, Committee Member
Steve Tate, Committee Member
Krishna Kavi, Chair of the Department of Computer Science
C. Neal Tate, Dean of the Robert B. Toulouse School of Graduate Studies

Krishnan, Anupama, The Multipath Fault-Tolerant Protocol for Routing in Packet-Switched Communication Networks. Master of Science (Computer Science), May 2003, 76 pp., 16 figures, 43 titles.

In order to provide improved service quality to applications, networks need to address the need for reliability of data delivery. Reliability can be improved by incorporating fault tolerance into network routing, wherein a set of multiple routes are used for routing between a given source and destination. This thesis proposes a new fault-tolerant protocol, called the Multipath Fault Tolerant Protocol for Routing (MFTPR), to improve the reliability of network routing services. The protocol is based on a multipath discovery algorithm, the Quasi-Shortest Multipath (QSMP), and is designed to work in conjunction with the routing protocol employed by the network. MFTPR improves upon the QSMP algorithm by finding more routes than QSMP, and also provides for maintenance of these routes in the event of failure of network components. In order to evaluate the resilience of a pair of paths to failure, this thesis proposes metrics that evaluate the non-disjointness of a pair of paths and measure the probability of simultaneous failure of these paths. The performance of MFTPR to find alternate routes based on these metrics is analyzed through simulation.

ACKNOWLEDGEMENTS

There are several people I would like to acknowledge for their valued contributions in the completion of this work. First, I wish to thank my advisor, Dr. Mikler, for his guidance, understanding and infinite patience. He helped me find a stimulating and challenging subject of work and his support throughout this thesis has been invaluable.

I also wish to thank Dr. Boukerche and Dr. Tate for taking time off their busy schedules at the end of the semester to be part of my thesis committee. I thank Dr. Tate for meticulously reviewing my document and patiently answering all my queries. I thank the Computer Science department at UNT for giving me the opportunity to pursue my master's degree. It has been a wonderful experience to study at UNT.

I am especially thankful to Subhashini, who has spent an enormous amount of time to review my thesis document in great detail. Her technical and emotional support throughout these last few weeks has considerably eased my struggle to complete the task of writing this thesis. I also thank Prasanna for taking the time to provide feedback regarding my work. The moral support rendered by both Prasanna and Sandhya has also been important to me.

I would like to wholeheartedly acknowledge and thank my family members for their unstinting love and encouragement throughout my academic career. I owe a world of thanks to my dad for his support, to my mother for helping me set better standards for myself, and to my sister for being my best friend always.

Above all, I wish to thank God for everything I have received and will ever receive in my life.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS ii

LIST OF FIGURES vi

Chapter

1. INTRODUCTION 1

Quality of Service (QoS) 2

Faults and Fault Tolerance 5

Fault Tolerance and Redundancy 9

Fault Tolerance in Routing 11

Road Map 13

2. SOLUTIONS FOR ACHIEVING FAULT TOLERANCE

FOR ROUTING SERVICES 15

Application Embedded Fault Tolerance 16

Fault Tolerance in Client-Server Systems 16

Fault Tolerance in Distributed Systems 17

Solutions for Offering Fault Tolerance to Applications 19

Hot Standby Routing Protocol (HSRP) 19

Dynamic Routing System (DRS) 20
Transmission Control Protocol (TCP) 21
Fault Tolerance in Network Routing 22
The *K*- Shortest Paths Problem 23
Disjoint Multipath Algorithms 24
Non-Disjoint Multipath Algorithms 27
3. THE MULTIPATH FAULT-TOLERANT PROTOCOL
FOR ROUTING 32
Quasi-Shortest Multipath (QSMP) 33
The Multipath Fault-Tolerant Protocol for Routing (MFTPR) 38
Route Discovery Mechanisms 39
Route Maintenance 44
Route Selection Policies and Resilience to Failures 48
Simulation and Experiments 52
Experiments and Analysis 53
4. SUMMARY AND FUTURE WORK 63
Future Work 68
BIBLIOGRAPHY 70

LIST OF FIGURES

2.1 Node Disjoint Paths 24

2.2 Link Disjoint Paths 25

3.1 Quasi-Shortest Path 34

3.2 Graph of a Network 35

3.3 Calculating the Quasi-Shortest Path 35

3.4 Possible Alternate Path from Source W to Destination Z 37

3.5 Alternate Paths from A to C 41

3.6 Requests Generated by Node A to Find Quasi-Via Paths to Destination C 42

3.7 Maintenance of Backup Routes 45

3.8 Main and Backup Routing Tables for Node A 46

3.9 35 Nodes, Commonality 0%, Threshold 50% 55

3.10 45 Nodes, Commonality 0% , Threshold 40% 55

3.11 65 Nodes, Commonality 0%, Threshold 50% 56

3.12 65 Nodes, Commonality 30%, Threshold 40% 56

3.13 Nodes 45, Commonality 30%, Threshold 30% 57

3.14 Nodes 35, Degree 4 59

3.15 Nodes 55, Degree 4 59

3.16 Nodes 65, Degree 5 60

CHAPTER 1

INTRODUCTION

Over the past few years, networks have grown in size exponentially. Many applications, especially businesses, now operate over the Internet and depend upon it for communication and other services. This underlines the need for network services to be provided in a reliable manner. Currently, the routing model of the Internet offers a best-effort service, whereby the network tries to deliver data packets, but provides no guarantees about the reliability of delivery. Consequently, data packets could be lost or delayed. The best-effort routing model is no longer sufficient to provide reliable services to applications, especially since networks now carry much more traffic than they were ever designed to handle. Networks, therefore, need to address the growing need for reliability in services provided to applications. Since reliability can be considered to be a Quality of Service (QoS) parameter, issues related to Quality of Service in general and reliability in particular must be considered.

Quality of Service (QoS)

The term Quality of Service is ambiguous as there is neither a clear consensus on what "services" are provided under the umbrella of QoS, nor are there clearly defined mechanisms to provide these services. Paul Ferguson and Geoff Huston in their book *Quality of Service* [8] aptly draw a parallel between QoS and the Elusive Elephant, where three blind men come upon an elephant and envision it differently as each is faced with just one small part of the elephant. There are several perspectives on QoS, and this can be attributed to the fact that each perspective addresses only a small, specific portion of QoS-related issues. A variety of QoS definitions exist. QoS may be defined as "the ability of a network element (e.g., an application host or router) to have some level of assurance that its traffic and service requirements can be satisfied" [28].

In order to be able to provide Quality of Service to applications, a network must have a reasonable level of *service quality*. Service Quality may be defined as "delivering consistently predictable service" [9]. The parameters used to measure service quality in networks are delay, packet loss, bandwidth, and jitter [9]. For a network consisting of routers and hosts (or nodes), the service quality parameters are defined as follows:

- **Delay** is the amount of time taken by the network to deliver a packet from source to destination. It is measured as a combination of several factors including transmission delay, propagation delay, and queuing delay at the various routers. Reducing delay increases the network's service quality.

- **Packet Loss** is caused by network congestion and component failures. Ideally, packet loss should be non-existent. Realistically, one of the goals while offering better service quality would be to minimize packet loss.

- **Bandwidth** is the maximum data transfer rate that a communication link between two end points can support. The higher the bandwidth offered, the better is the service quality. However, bandwidth is limited by the physical communication medium of the link, which in turn implies that maximizing efficiency in the use of bandwidth is the actual measure of service quality.

- **Jitter** is the variation in delay experienced by packets belonging to a single connection. Applications related to real-time based audio and video are generally considered to be jitter-sensitive. Reduced jitter implies better service quality.

Optimizing the above-mentioned metrics improves service quality and thereby improves QoS. The QoS proffered by a system may also be characterized by factors such as system availability, reliability, safety, security, and maintainability. *Availability*

refers to the probability that the system is functioning correctly and is able to perform its tasks at any given instant. *Reliability* for a network is defined to be inversely proportional to packet loss, and a reliable system should continuously provide services without failure. A *safe system* should eliminate catastrophic consequences in the event of a failure, whereas a *secure system* must maintain and preserve confidentiality among other things. Finally, *maintainability* refers to the ease with which a system facilitates repair externally or recuperates on its own after failure. A system that exhibits these characteristics is said to be a *dependable* system [11], and increasing dependability can be considered a QoS objective.

The ability to measure and characterize QoS in a system by no means simplifies the task of actually providing the requisite QoS. This is especially true of large networks that are composed of several smaller autonomous networks. The constituent networks may offer varying levels of service quality, making it difficult to provide consistent service quality across the heterogeneous network. They may also employ different protocols for network functions such as routing, and in addition may have varying infrastructures that are dictated by local organizational policies. Consequently, it is difficult to propose architectural changes or implement mechanisms to provide uniform end-to-end QoS guarantees.

However, it is possible to incorporate properties such as fault tolerance into the

network that help improve service quality without having to modify existing infrastructure or protocols. This thesis proposes a protocol for fault tolerance that allows networks to improve the reliability of data transfer and consequently increase system dependability. Fault tolerance advocates addressing faults and provides contingency measures to deal with them so that network services remain available even in the face of failures. Below, we discuss some issues related to faults and fault tolerance.

Faults and Fault Tolerance

A fault is defined as the failure of a component of a system or a subsystem of the system [11]. The occurrence of one or many faults may lead to system failure, where the system cannot deliver the promised services or deviates from expected services. For example, a link or node failure is a fault. Multiple link and node failures may lead to failure of network routing services, which in turn may translate to failure of the entire network.

Faults can be classified on the basis of locality, effect and duration [11]. *Locality-based* faults may be caused by failure of hardware components in the system, failure due to incorrect use of the system or failure due to changes in the operational environment. A failed network component such as a router is an instance of a locality-based fault. *Effect-based* faults cause incorrect computational results or unacceptable delay

in delivered services. For example, a memory module that is faulty may lead to erroneous computation. *Durational* faults are time-based and can be further classified as persistent, transient or intermittent faults. *Persistent* faults remain active unless the cause for the fault is repaired. A failed disk drive is an example of a persistent fault. *Transient* faults remain active for a short duration. For example, a noise spike causes a transient fault that results in garbled data, but transmissions after the spike subsides remain unaffected. *Intermittent* faults are faults that become active periodically. A loose contact on a circuit board will cause an intermittent fault.

In order for a system to be dependable and hence provide good QoS, faults must be controlled. There are several ways of controlling and dealing with faults, which include fault avoidance, fault removal, fault evasion and fault tolerance [11].

Fault Avoidance focuses on eliminating faults at the design stage and consequently, the faults do not have to be dealt with later. This technique is especially effective when faults are known in advance or are uncovered during the design stages. However, it provides no way of dealing with faults that cannot be corrected at design stages and that are discovered when the system is deployed and operational.

Fault Removal techniques are used to test and verify the system after it is designed, but before it becomes completely operational. Fault removal is ineffective when faults remain hidden. Undetected faults go uncorrected and there is no way to

deal with these faults during operational stages.

Fault Evasion techniques involve observation of the system in order to detect abnormal behavior. If the observation yields information that suggests that any component of the system is being subjected to high stress, then measures to reduce the load on that component should be taken. If, however, deviant behavior is being exhibited by the system and no obvious reason is visible, then any action that might compensate for the resulting faults is recommended. Fault evasion does not provide any guarantees for system safety and integrity. The effectiveness of this technique depends entirely on the accuracy of observation of the end user, or the effectiveness of the fault detection system and the evasive maneuvers employed.

Fault Tolerance, on the other hand, ensures that a system is available for use especially in the presence of faults. Fault tolerance increases the reliability of a system by trying to deliver the promised services even when the system is experiencing faults. In the event of multiple faults, services may still be provided, albeit at a degraded level. This ensures that the system remains operational at all times. Fault-tolerant capabilities can be viewed as a practical solution that increase reliability, availability, and consequently dependability of the system. In the absence of faults, the fault-tolerant mechanism does not interfere with the normal behavior of the system, but in the presence of faults, it attempts to make the system appear normal to the end

user.

Fault-tolerant systems should be capable of detecting, locating, diagnosing, containing, masking faults, ultimately recovering from faults. *Fault detection* is the process of learning that a fault has occurred. Once fault detection techniques have been employed, *fault location* techniques determine the region where the fault occurred. *Fault diagnosis* is then employed to determine the reason for the occurred fault, after which *fault containment* techniques are used to prevent the fault from propagating to other regions and thus prevent further faults. *Fault masking* is a desirable feature of fault-tolerant systems, which ensures that system services are not affected by the fault, and that the end user remains completely unaware of the occurrence of the fault. Finally, *fault recovery* is the phase where the actual fault repair takes place.

Systems can be classified based on their capability to deal with faults. A fault detection system can only detect faults [37], a fault diagnosing system can determine the cause of the fault [13]. A *fault-secure* system provides an error indication in the event that a fault occurs and prevents propagation of the fault [23]. This is suited to systems in which transactions and operations can be retried after the error has been corrected. A *fault masking* system, on the other hand, delivers services correctly even when faults occur and hides the effects of faults from the end user [4]. Fault masking abilities are especially desirable when faults need to be hidden from applications.

Lastly, a system can be *fail-safe*, in which case the system deals with a fault by first attempting to mask it. If that fails, the system is returned to a safe state, so that catastrophic events are avoided [15]. This might lead to only a fraction of the system's capabilities being operational until the faults are corrected, enabling the system to return to its fault free state.

Fault tolerance is implemented by employing resources that would be unnecessary in a fault-free system. The following section discusses the use of redundant resources in fault tolerance.

Fault Tolerance and Redundancy

Fault tolerance is sometimes called redundancy management because of the extensive use of redundant components in the system [11]. Depending on the kind of faults addressed, the redundancy used may be spatial, information, or temporal [42].

Spatial (physical) redundancy involves the replication of resources and is commonly used to provide fault tolerance in hardware. For instance, the failure of a hardware component can be masked by using an identical component to perform the tasks of the failed component. This technique is very effective when a component experiences a persistent fault or malfunctions and causes computational errors. One such example of spatial redundancy is Triple Modular Redundancy (TMR) [42]. A

process added to a system to increase its fault tolerant capabilities can also be viewed as a form of spatial redundancy, as the process would not be required in a fault-free system. Augmenting a system with additional processes is very effective in providing system fault tolerance and fault tolerance in software. Self-healing rings [41] use procedures to automatically re-establish communication after component failures.

Information redundancy implies that extra information is used to detect and recover from a fault. Parity bits appended to data blocks to enable error detection could be considered as an instance of information redundancy.

Temporal(time) redundancy deals with transient or periodic faults in an effective manner [42]. In time redundancy, an operation that is performed but fails is retried after a time interval. The assumption here is that if an operation is hampered by a transient fault, repeating it at a time sufficiently past the first try would yield correct results, since it is highly probable that the fault will not affect both operations. For example, a temporary spike in noise might render a transmission incoherent, but if the transmission is retried after an interval, the noise spike would most likely have subsided and the transmission would be successful. Temporal redundancy is commonly used to provide fault tolerance in software. Applications whose state can be rolled back to the stage before the fault occurred are typically suited to fault tolerance mechanisms that employ temporal redundancy [2]. Time-out and retry mechanisms

used by several communication protocols like Transmission Control Protocol [29] can also be classified as temporal redundancies.

Depending on the kind of fault tolerance required and the system under consideration, one or many types of redundant resources can be used in conjunction to achieve fault tolerance. In the following section we outline the necessity of fault tolerance in routing and the nature of the redundancy required to achieve this fault tolerance.

Fault Tolerance in Routing

Routing is defined as "the process of determining the path taken by data packets between source and destination" [14]. Routing algorithms can be classified into two broad categories - global and decentralized [14]. Global routing algorithms require each router in the network to have complete knowledge of the network topology at all times. The routing algorithms in such a scheme determine the path to a destination using this knowledge of the network. On the other hand, routers in decentralized algorithms possess knowledge of their immediate neighbors only. Neighboring routers exchange routing information to calculate the path costs to the destinations known to their neighbors. This process is repeated iteratively until each router calculates the least cost to all the destinations on the network. Open Shortest Path First

(OSPF) [24] is an instance of a global routing algorithm whereas the Routing Information Protocol (RIP) [17] is a decentralized routing algorithm.

Existing global and decentralized routing algorithms take a reactive approach to network component failures. Thus, a router or link failure in the network triggers recovery measures to reconfigure router information to reflect the change in topology. However, while the network is re-configuring itself, routing loops could form, leading to lost or undeliverable packets [22]. This is especially true in RIP, which reacts slowly to topological changes. Some applications however may be intolerant to such packet loss. Examples are real-time applications such as process monitoring and control, data acquisition etc, that place great emphasis on reliability. Consider an assembly line monitoring system that acquires data regarding the state of the assembly line process at regular intervals of time. This data is then transmitted to a control center that monitors the process. Since the state of the system changes with time, the data acquired at any instant is unique and non-replicable. This implies that data lost during transmission is irretrievable. Therefore, time-out and retry mechanisms employed by end-to-end protocols such as Transmission Control Protocol (TCP) cannot be used to recover from packet loss. Clearly, these real time applications are unable to sustain packet-loss and are heavily dependent on the underlying network to provide reliable packet delivery. Network failures should be transparent to these applications and

the services promised by the network should be delivered despite component failures. This would require network routing to be fault-tolerant.

Fault tolerance in routing will help increase network reliability and provide better services and is thus a much needed feature. This thesis focuses on providing a fault-tolerant technique to aid routing and minimize packet loss.

Road Map

Fault tolerance in routing has been an active area of research and several solutions have been proposed for the same. These solutions include adding with redundant resources to networks and finding several disjoint and non-disjoint paths between source and destination, among others. Chapter 2 describes prior work in fault-tolerant routing and places an emphasis on the algorithms for finding disjoint paths and non-disjoint paths in a network.

This thesis focuses on providing alternative paths to the primary communication path between a source and destination. Fault tolerance is provided by computing multiple paths between source and destination so that the failure of one path does not disrupt communication. Additional information and processes are required to implement this fault-tolerant technique. The uniqueness of the thesis lies in the approach it takes to evaluating the appropriateness of a pair of non-disjoint paths

between a source-destination pair. A 'risk' factor is defined for every pair of paths between source and destination and this factor denotes the probability that a network component failure renders both paths invalid and thus tries to serve as a probabilistic fault-tolerant metric. Chapter 3 details this technique and the simulation used to evaluate it. The nature and amount of redundancy used to provide the fault tolerance is also analyzed.

Lastly, chapter 4 summarizes the thesis and suggests further work to be done in this area.

CHAPTER 2

SOLUTIONS FOR ACHIEVING FAULT TOLERANCE FOR ROUTING SERVICES

Failures at the network level adversely affect applications that rely on network services for effective performance. Many solutions have been proposed to ensure recovery of applications after network related failures. We classify these solutions as reactive or pro-active, based on their approach to dealing with failures. *Reactive* solutions initiate recovery measures after the occurrence of a failure, whereas *pro-active* solutions take measures to prevent failure when a fault occurs. Fault tolerance can be viewed as a pro-active approach to failures from this perspective. This chapter delves into various fault-tolerant schemes that try to prevent network failures from having a negative impact on application performance.

Some fault-tolerant solutions need to be embedded into the application itself. Other solutions take existing network services and work to make these services reliable for the application. Lastly, there have been solutions proposed that advocate fault tolerance to be integrated into network services as a basic requirement. This chapter takes a brief look at each of these scenarios.

Application Embedded Fault Tolerance

Distributed and client-server applications depend heavily upon network services. Their implementations build fault tolerance into the application itself and thus enable the application to independently recover from non-transparent network failures. For example, online banking systems and distributed file storage are built to provide high availability while maintaining data integrity and consistency, and their implementations incorporate the necessary fault tolerance required to achieve these goals. Below, we briefly outline how such applications try to achieve fault tolerance.

Fault Tolerance in Client-Server Systems

The client-server architecture refers to the relationship between two computer programs in which the client program makes a service request to the server program, which fulfills the request [25] . Applications using the client-server architecture across a network are susceptible to network component failures. For example, consider a database server that responds to update requests from a client program. If a network failure causes communication between the server and client to be terminated, the integrity and consistency of data in the server's database might be violated. Moreover, the node on which the server resides could fail, making the server unavailable to the client. In order to maintain consistency of data in the event of a failure, a rollback

and recovery technique in conjunction with message logging is generally used [36]. In addition, to recover from server crashes in a transparent manner and increase availability of database services, a group of replicated servers may be executed on different nodes [1]. One of the servers in this group services client requests and is designated as the primary server, whereas the other servers act as backup. The primary server periodically updates the backup servers about the state of the system to maintain database consistency across the group. In the event that the primary server fails, the application selects one of the backup servers to perform the tasks of the failed server. In this manner, the client is unaware of any server crashes and the application provides transparent recovery from failures.

Fault Tolerance in Distributed Systems

In distributed systems, a set of autonomous computers (nodes) linked together by a network is supported by software that enables the entire set of computers to operate as a single entity. Applications that operate on distributed systems are generally unaware of the distributed nature of the underlying services. Therefore, it is imperative to make underlying node and link failures of the network transparent to the application.

Fault tolerance in distributed systems is a well-researched domain that can be

used to achieve this objective. The most prevalent approach makes use of fault-tolerant process pairs, where the primary process is provided with a back-up process performing the same task, but generally on a different node [16]. When a node crash causes a primary process to fail, the back-up process takes over and resumes execution from the point where the primary process stopped. As an example, consider Coda, a distributed file system built for high availability [34]. Coda uses server replication and stores copies of a file at multiple servers. Thus if one server fails, the file can be accessed at another server. This is completely invisible to the user who is only aware of the file and not its storage management. Coda also uses a disconnected operation technique in the event that no server is available. Files accessed by a client are cached at the client site. If another client wishes to access these files when a server is not available, the files are provided from the cached site. When a server does become available, the modified files are uploaded to the server. This ensures a limited availability when complete server failure takes place.

Fault-tolerant applications thus ensure that network level failures do not translate to application failures. However, the design of these applications and their implementations are made complicated by the inclusion of considerations for fault tolerance. In order to provide generalized fault tolerance for all applications, solutions have been developed that take existing network services and work to make these services seem

reliable to the applications. Some such solutions are detailed below.

Solutions for Offering Fault Tolerance to Applications

Several protocols and algorithms have been developed with the goal of providing applications with reliable network services. These protocols bridge the gap between applications that require reliable services but do not wish to implement fault tolerance themselves, and networks that cannot provide the required reliability for these applications.

Hot Standby Routing Protocol (HSRP)

HSRP is an application level protocol developed by Cisco Systems for the Internet Protocol (IP) [39]. When a router fails, protocols like Enhanced IGRP and OSPF converge to allow other routers to take over the functions of the failed router. However this still does not restore communication for the hosts that had been configured statically with the IP address of the failed router. Only re-configuration of the default router address on these hosts restores communication. HSRP eliminates the need for this manual re-configuration while providing reliable router back up. To facilitate this, two or more routers use the MAC address and IP network address of a virtual (non-existent) router. This virtual router represents the group of routers that are

meant to backup each other. Each of these routers serves as the default active router for a network. Hosts on a network are configured with the address of the virtual router that represents the group in which the default router of this network participates, and route all their packets to the virtual router. The default router performs the routing functions for its network. In the event that it fails, one of the other routers in the group takes over the functions of the failed router and begins to act as the default active router for this network. HSRP is a very efficient solution as it employs the services of existing routers to backup a failed router and thereby eliminates the need for redundant routers. However, HSRP has been developed exclusively for IP and is thus restricted in its scope of applicability.

Dynamic Routing System (DRS)

DRS is a fault-tolerant pro-active routing algorithm that has been developed specifically for small networks having not more than 254 hosts [4]. DRS has been designed for mission-critical applications, and improves fault tolerance by employing redundant resources for all network components. A pair of links is defined for every pair of hosts in the network. One link serves as the primary communication channel while the other acts as a backup in case the primary link fails. A DRS daemon executes on each host, and continuously polls hosts using ICMP echo requests with

the goal of detecting network link failures. If a node cannot be reached due to failure of both the connecting links, a new route is computed that bypasses the malfunctioning network links to reach the node. Thus, if link failure is discovered then the recovery process ensures that a connection to the node is restored before the need to communicate. Consequently, application performance is not affected. This is of great importance in mission-critical applications where non-transparent network failures could imply application failure. However, as already stated, DRS has been designed for smaller networks and makes use of a completely redundant network. For larger networks, such a redundancy comes at astronomically high costs. Thus DRS, is limited in its scope of usability.

Transmission Control Protocol (TCP)

TCP is a transport layer protocol that is designed to provide end-to-end reliable services for data delivery [29]. TCP makes use of acknowledgements, and time-out and retry techniques to achieve reliable delivery of data. When TCP transmits a packet to a destination, it starts a timer to maintain a window of time for receipt of acknowledgement from the receiver. If the timer expires and no acknowledgement has been received, TCP assumes that the packet is lost, retransmits the packet, and restarts the timer. It repeats this operation a predetermined number of times or until

an acknowledgement arrives. Thus, TCP tries to shield the application from dealing with unreliable network services. However, TCP mechanisms of time-out and retry are not suited to all classes of applications as explained in the example of the process monitoring system in chapter 1.

Solutions such as HSRP, DRS, and TCP outlined above that provide fault tolerance to applications shield applications from network unreliabilities. However, they do not actually improve network reliability, and consequently in protocols such as TCP, reliable packet delivery may be provided at the cost of higher network delay. If network failures are handled at the network layer itself, the overall performance of network services such as routing can be improved. The following section examines some solutions to provide fault-tolerant network routing.

Fault Tolerance in Network Routing

Perspectives on fault tolerance in routing have evolved from the days of the initial ARPANET to the present day Internet. Early routing algorithms were very slow in reacting to changes in network topology caused by failure of network components [20]. Thus resulted in lost or delayed packets, and consequently degraded routing performance. For these algorithms, therefore, quick adaptability to failures was construed as a fault-tolerant feature [21]. Today, most popular routing algorithms such as OSPF

can adapt very well to changes in network topology. Consequently, fault tolerance in routing is now focused on providing continuous network services and making network failures transparent to applications.

One of the most popular approaches to fault tolerance in routing determines a set of multiple paths between source and destination, known as a *multipath*. A multipath provides an immediate alternate means of communication in the event of failure of the primary route between source and destination. If the paths in a multipath have the least possible cost, then fault tolerance can be provided in conjunction with optimal routing. The problem of calculating multiple shortest paths for a source-destination pair is known as the *K-shortest paths problem* and is summarized below.

The K- Shortest Paths Problem

Listing the K shortest paths (for a given integer $K > 1$) between a source and destination in a graph is the known as the K-shortest paths problem [7]. Constraints can be imposed on this problem by requiring the shortest paths to satisfy conditions like being loop-free, disjoint, etc. [19]. When no restrictions are considered in the definition of the path, the problem remains completely unconstrained [19]. Both the unconstrained and constrained problems have been studied extensively and many solutions have been proposed [6]. However, since the applicability of the K-shortest

paths problem to routing is relevant only if the paths are loopless, we focus solely on the constrained problem. Additional constraints such as disjointness can be imposed on a set of loopless paths yielding a *disjoint multipath*. Both disjoint and non-disjoint multipaths can be used to provide fault-tolerant routing as outlined below.

Disjoint Multipath Algorithms

Paths in a disjoint multipath may be classified as node-disjoint or link-disjoint. Two paths between a source-destination pair are said to be *node-disjoint* if they do not have any nodes in common excluding the source and the destination. Similarly, two paths are *link-disjoint* if they do not have any links in common.

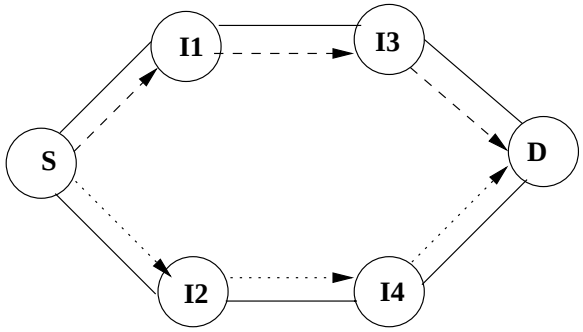

Figure 2.1: Node Disjoint Paths

Consider the set of paths in figure 2.1 with node S as source and node D as the destination. The path S—I1—I3—D is node disjoint from the path S—I2—I4—D. In figure 2.2, the paths S—I1—I3—I4—D and S—I2—I3—D are link disjoint. A pair of

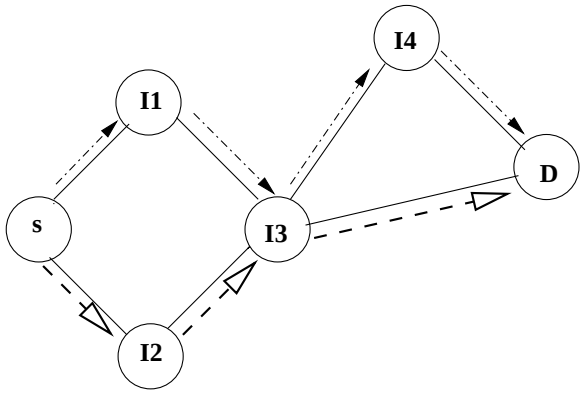

Figure 2.2: Link Disjoint Paths

node-disjoint paths is also link-disjoint. However, the converse is not necessarily true as illustrated by the above example.

When a pair of disjoint paths is used for communication where one path acts as the primary path and the other as a backup, failure of any one path will not disrupt communication, as the other path remains unaffected. This property makes disjoint paths very suitable for fault-tolerant routing. Node-disjoint paths are resilient to both node and link failures whereas link-disjoint paths may be resilient to link-failures alone. This makes node-disjoint paths preferable to link-disjoint paths for achieving better fault tolerance. From this point on, with no loss of generality, disjoint paths will refer to node-disjoint paths unless explicitly stated otherwise.

Several solutions have been proposed to find disjoint paths in a network. One solution [12] takes a global, distributed approach to finding the K link-disjoint paths

from source to destination and requires that all nodes in the network have complete knowledge of the network topology. The algorithm computes K spanning trees with the source as the root such that for every vertex, the paths from the vertex to the root are link disjoint. The algorithm, however, does not attempt to find the shortest paths.

An algorithm that does calculate the pair of disjoint paths with the least possible cost from every source to a single destination is presented by Ogier et al. in [27]. In this distributed, asynchronous algorithm, each node is provided with sufficient information so that it can make routing decisions incrementally to forward packets over disjoint paths. In the event of topology changes that cause paths to become invalid, this algorithm converges in a finite amount of time to provide nodes with information for forwarding packets over a new pair of disjoint paths. The algorithm has both link-disjoint and node-disjoint versions and has been extended by Chenig et al. [3] to find the K shortest disjoint paths between source and destination. Another distributed algorithm to find the shortest node-disjoint paths has been presented by Sidhu et al. [35]. This algorithm is shown to find more paths than that of Ogier et al. [27], and also incurs less messaging overhead. It requires the flooding of path identification information as well as topology related information to various routers to discover the disjoint paths. Once the paths have been discovered, the algorithm

terminates and data packets can be routed along the disjoint paths. Dong et al. [5] present a distributed algorithm that improves on the above algorithms by eliminating the need to propagate path identification information and topology related data in the network. This algorithm forms the basis for this thesis and is explained in chapter 3.

Although disjoint routes provide high resilience to failure, they can incur a great deal of computational overhead. For many applications, a high degree of resilience is not warranted if it comes at the cost of heavy overhead. Other classes of applications place a greater priority on least-cost routing than disjointness of routes. Therefore, relaxing the disjoint requirement for shortest multipaths yields solutions that are more suited to such scenarios.

Non-Disjoint Multipath Algorithms

Several algorithms exist to calculate a multipath for a source-destination pair. The MAximally Disjoint Shortest and WIdest Paths (MADSWIP) algorithm described by Ogier et al. [26] computes a pair of paths from a source to every destination such that the paths are as disjoint as possible and either minimize total path cost or maximize bandwidth. This algorithm is very useful in cases where disjoint paths do not exist between source and destination. First, a tree (with the source as root) is constructed

containing the maximum bandwidth paths from the source to all other nodes. Then, variables required to construct the disjoint paths are calculated for each node in the tree, and finally, disjoint paths from the source to all the nodes are computed. This algorithm also has an application in congestion routing, where traffic is distributed over two different paths to avoid congestion, packet loss, and delay [40]. However, this is a centralized algorithm that is an extension to the Surballe and Tarjan centralized algorithm [38] to find disjoint paths.

An algorithm to compute reliable alternate paths in OSPF was proposed by Pu et al. [31]. OSPF is a global routing algorithm where every router in the network has an identical view of the network topology. Each router uses Dijkstra's algorithm to construct a minimum spanning tree with itself as the root and thereby finds the shortest paths to all destinations. Frequent topology changes cause this procedure to be repeated for each reported failure, thereby degrading the performance of OSPF. To remedy this, Reliable OSPF (ROSPF) was proposed. ROSPF uses three paths for communication where one serves as the primary path and the other two as alternate paths. The algorithm used to compute these paths is based on the logarithmic edge-weight increment procedure [33]. Once the shortest paths from source to destination have been calculated, the weights of all the edges in the shortest path are increased by a large value. This decreases the probability of the same edges being chosen when the

shortest path is recalculated, and consequently alternate routes are found. ROSPF optimizes this procedure to find the shortest paths and also chooses the incremental value to minimize number of edges shared between the primary and alternate routes. In addition to calculating multipaths, methods to find the resilience of multipaths to failure have also been proposed. A method to evaluate the reliability of the alternate paths found by ROSPF is described by Pu et al. [30]. It first calculates the number of edges in each path and also calculates the number of edges shared by each path with the other paths. Using this information, the probability of independent failures of each path and probability of paths failing simultaneously are calculated. This method takes into account only link failures but not node failures and therefore does not provide a complete reliability analysis. However, such an evaluation clearly represents the resilience of a routing scheme based on multipaths.

Similar work has been conducted by Ganesan et al. [10], where multipath routing has been used to increase the resilience of a wireless sensor network to node failures. For each node on the primary path from a source to destination, an alternate path that does not include that node is found. The set of all such paths for a primary route forms the multipath. The paper also presents a simulation-based analysis of the resilience of the multipaths wherein the availability of an alternate path in the multipath is investigated. For this, each node in the network is caused to fail with

a given probability. When a node fails, all the multipaths whose primary route is affected by the node failure are tested to find if an alternate route is available. The number of complete failures is recorded, and this represents the resilience of the multipaths in the network to node failure.

Multipaths have received special attention in wireless ad hoc networks, and several routing protocols based on multipath routing have been proposed. Ad hoc wireless networks are a collection of mobile nodes with no central administration used to operate the network. These networks are characterized by high packet error rates, and dynamic topology caused by the ability of nodes to enter or leave a network at any time. Consequently, routing protocols that calculate multiple routes between source and destination and route packets over these paths simultaneously are able to achieve better reliability of data transfer and also minimize end-to-end delays, in addition to achieving fault tolerance in routing. One such multipath routing scheme is the Ad hoc On-demand Multipath Distance Vector Protocol (AOMDV) [18]. This protocol extends an existing Ad hoc On-demand Distance Vector Protocol (AODV) with the addition of comparatively small overhead. By calculating multiple paths during route discovery, AOMDV reduces the need to discover a new path every time a route failure occurs. In AOMDV, route discovery needs to be initiated only after all the multiple routes fail, and thus a route failure does not necessarily mean communication failure.

In this chapter, several approaches to providing fault tolerance for applications as well as network services have been examined. One of the algorithms for multipath routing, proposed by Dong et al. [5] is used in this thesis to develop a new protocol for fault tolerance in network routing. The following chapter explains the proposed protocol, along with a risk analysis of the chosen route.

CHAPTER 3

THE MULTIPATH FAULT-TOLERANT PROTOCOL FOR ROUTING

Several approaches to building resilience to network level failures were presented in the previous chapter. Among them, fault-tolerant routing can be considered to be the most desirable approach as it mitigates the need for applications to implement complete fault tolerance within themselves. Applications can instead rely on the underlying network to provide continuous services, and their implementations can be streamlined to achieve application service goals and optimizing performance. To provide fault-tolerant routing, several algorithms to calculate multipaths for source-destination pairs in a network were presented in Chapter 2. This thesis develops a new distributed multipath protocol called the Multipath Fault-Tolerant Protocol for Routing (MFTPR), that operates in conjunction with a routing protocol at the level of the network layer to increase reliability of routing services. The protocol is based on a distributed algorithm presented by Dong, Varaiya, and Puri [5] to calculate the shortest multipath, called the Quasi- Shortest Multipath. Using notation similar to that in [5], an overview of the algorithm is presented below.

Quasi-Shortest Multipath (QSMP)

Consider a network represented as a weighted undirected graph $G = \{V, E, c\}$ where V is the set of nodes or vertices in the network, $E \subset V \times V$ is the set of weighted edges or links in the network, and cost function $c : E \rightarrow R_+$ assigns positive costs to the edges. Additionally, we assume that no node in the graph has any self-edges. Lastly, all paths are assumed to be simple (loop-free) paths.

A path p from source s to destination d is represented as $p =< s, v_1, v_2, ..., v_k, d >$ where v_1 to v_k are the k intermediate nodes from s to d, and links (s, v_1), (v_k, d) and (v_i, v_{i+1}) $(1 \leq i < k)$ belong to the set E. The node v_k that is visited just before reaching the destination is called the *predecessor node* of path p. The cost of p is represented as $c(p) = c(s, v_1) + \sum_i c(v_i, v_{i+1}) + c(v_k, d)$, i.e., the sum of the costs of all the links in the path. When p is the shortest path from s to d, no other path from s to d has smaller cost. A multipath from s to d is represented as P_{sd} and is a collection of simple paths from s to d. When all the paths in P_{sd} are node-disjoint from one another, the multipath is a node-disjoint multipath.

A *quasi-shortest* path $p =< s, r_1, r_2, ..., r_i, d >$ is a path, where the subset $p' =< r_1, r_2, ..., r_i, d >$ is the shortest path from r_1 to d but p may not be the shortest path from s to d. To illustrate this concept, consider the graph shown in figure 3.1.

The shortest path from node A to node D (indicated by the dotted lines) is $p =< A, E, D >$, whereas the shortest path from node B to node D (indicated by the dashed lines) is $q =< B, C, D >$. The path $r =< A, B, C, D >$ from A to D is a quasi-shortest path as it contains the shortest path from B to D, but is not the shortest path from A to D. A multipath $P_{sd} = \{p_1, p_2, ..., p_k\}$ is a *Quasi Shortest Multipath* (QSMP) if the shortest path q from s to d is in P_{sd}, and every path $p \in P_{sd}(p \neq q)$ is a quasi-shortest path. When all the paths in a quasi-shortest multipath are disjoint from each other, it is known as a *Quasi Disjoint Multipath* (QDMP).

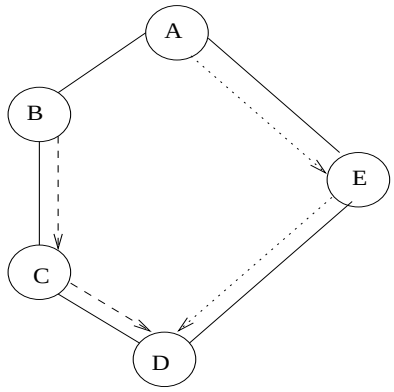

Figure 3.1: Quasi-Shortest Path

The discovery of quasi-shortest paths requires information about the predecessor of a path. This information is assumed to be computed by the routing protocol employed by the network, and stored in a node's routing table. To illustrate the discovery of alternate paths, consider the graph shown in figure 3.2.

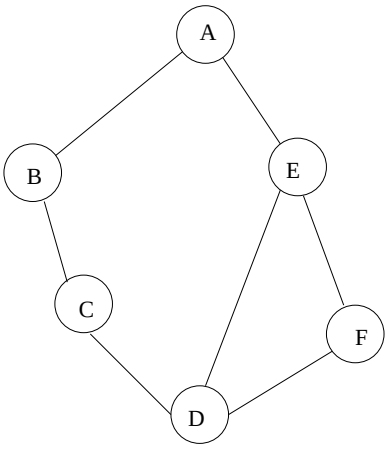

Figure 3.2: Graph of a Network

Routing Table for Node B

Dest	Next Hop	Pred	Cost
A	A	B	1
C	C	B	1
D	C	C	2
E	A	A	2
F	C	D	3

Determining the shortest path from B to F using predecessor information

Step	Pred	Path
1: B to F	D	{B ,....., D, F}
2: B to D	C	{B, ..., C, D, F}
3: B to C	B	{B, C, D, F}

Figure 3.3: Calculating the Quasi-Shortest Path

As seen from the graph, the shortest path from A to F is $< A, E, F >$. Let us assume that A wishes to find alternate routes to destination F that are disjoint from its primary (shortest) path. When the routing protocol causes the two neighbors A and B to exchange routing tables, node A can trace the complete shortest path from B to F using the predecessor information in B's routing table as shown in figure 3.3. This path is checked to see if A is present in it and if so, node A realizes that B cannot offer a quasi-shortest path. However, if A is not present in the traced path, then node A has discovered a quasi-shortest path to destination F. In this case, the quasi-shortest path from A to F is computed to be $< A, B, C, D, F >$. In order to check for disjointness, A simply traces its shortest path to F using its own routing table and compares the two routes to find if any nodes are in common. In this manner, when two neighbors exchange routing tables, each node discovers if its neighbor can offer any quasi-shortest paths, and can test for disjoint paths.

The primary advantage of this algorithm is that it is completely localized and does not require topology information to calculate alternate routes. The procedure to route a packet along a quasi-shortest path is also simple, as only the source needs to decide the neighbor to which it wishes to forward the packet. Once the packet has been forwarded, it follows the shortest route between the neighbor and the destination, thereby requiring no change in the routing decision beyond the source.

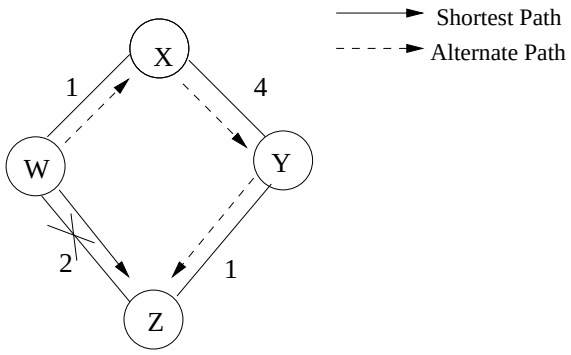

Figure 3.4: Possible Alternate Path from Source W to Destination Z

Although QSMP allows a node to discover alternate disjoint paths to a destination, it cannot be used in situations where a node lies on the shortest path from its neighbor to a destination. As illustrated by the graph in figure 3.4, the primary(shortest) path from W to Z is $< W, Z >$. When W tries to find the quasi-shortest path through the node X, it receives a negative reply as the shortest path from X to Z passes through W. If link (W, Z) fails, W cannot communicate with Z instantly even though the route $< W, X, Y, Z >$ exists.

Another drawback of QSMP is that it does not address issues related to maintenance of backup routes and propagation of route failures. Once a backup route to a destination has been discovered, it is necessary to have a mechanism that can select the backup route for routing in the event of failure of the primary route. Also, the route discovery mechanism must be invoked to find a new alternate route to the

destination.

This thesis proposes to extend the above QSMP algorithm to build a protocol that will find more alternate routes than QSMP. Another advantage of using the protocol is that the fault-tolerant mechanism becomes independent of the routing algorithm. Therefore, the QSMP algorithm can be applied to all routing platforms and does not require platform-dependent implementations. The thesis also provides for maintenance of backup routes through the use of various procedures and messages. In addition, it presents metrics to evaluate the alternate routes in terms of disjointness from the main route, and measures the resilience of the discovered multipath to failure. The following section describes the protocol in detail.

The Multipath Fault-Tolerant Protocol for Routing (MFTPR)

The Multipath Fault-Tolerant Protocol for Routing (MFTPR) seeks to improve the route discovery mechanism of the QSMP algorithm and provide for maintenance of backup routes. For this, it is assumed that each node in the network has two tables: a main routing table and a backup routing table that it uses for routing and forwarding purposes. We also assume that predecessor information required by QSMP exists in the main routing table and is provided by the routing algorithm. Furthermore, we assume that the routing algorithm is a distributed localized algorithm like RIP so as

to assume the availability of the least amount of topology information.

MFTPR has also been developed with the goal of separating the route discovery mechanism from the route selection criteria. Consequently, each node can implement route selection depending on the criteria that suit it best. In this section, route discovery mechanisms and route maintenance procedures for this protocol are described. This is followed by a discussion of multipath evaluation techniques and path selection criteria.

Route Discovery Mechanisms

MFTPR discovers alternate paths when two neighbors exchange their routing tables by following the QSMP algorithm. However, several routing protocols such as RIP can be configured for triggered updates so that neighbors are sent update messages only when there is a change in the routing database, thus making the updates asynchronous. This means that a node does not know when the next update from its neighbors will take place. In such a case, if a node wishes to calculate alternate paths, it has to request the appropriate neighbor for its routing table. This is done using an `UPDATE_REQUEST` message. The node that receives an `UPDATE_REQUEST` replies by sending its routing table in an `UPDATE_REPLY` message. The message overhead incurred due to the `UPDATE_REQUEST` message is equivalent to the size of the routing

table. Since the size of a routing table is proportional to the number of nodes in the network i.e., V, the overhead complexity of this update message is equivalent to $\Theta(V)$.

However, when a node wishes to calculate alternate paths for just a single destination, it is inefficient for the neighbors to send their entire routing tables. In this case, the source sends a `PATH_REQUEST` message to the neighbor. This message asks a neighbor for its shortest path to a particular destination. On receiving a `PATH_REQUEST` message, the neighbor traces its shortest path to the required destination and checks to see if the source lies on this path. If the path contains the source, then it sends the source a negative reply in the `PATH_REPLY` message. Otherwise, it sends the path back to the source. Thus, every path that the source receives is a quasi-shortest path. The overhead incurred due to this message is proportional to the length of the path and is equivalent to $O(V)$, although usually much less.

In addition to discovering quasi-shortest paths from a source to a destination, MFTPR also allows a node to examine all possible quasi-shortest paths of its neighbors to the destination. This allows for the discovery of routes that would otherwise not be discovered by the QSMP algorithm as illustrated by the following example (figure 3.5).

Assume that node A wishes to discover all paths to C that are disjoint from

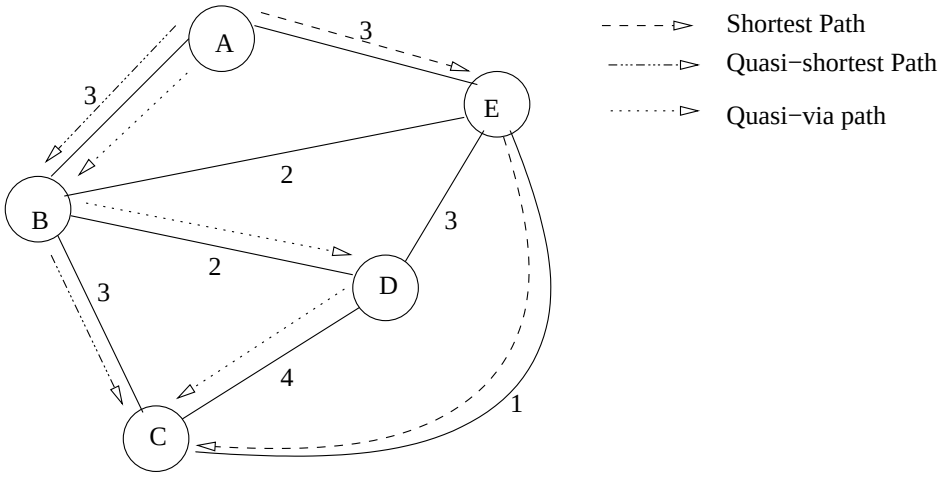

Figure 3.5: Alternate Paths from A to C

its primary route. A routes to C using the primary route $p = \{A, E, C\}$. When trying to find alternate paths that are disjoint from p, A only discovers the quasi-shortest path $q =< A, B, C >$ and does not discover the route $r =< A, B, D, C >$, even though r is completely disjoint from p and can be used as an alternate path to C. This is because path discovery by the QSMP algorithm is limited to finding the primary paths of the neighbors to the required destinations. In order to discover r in MFTPR, node A must request all its neighbors for their possible quasi-shortest paths using a `QUASI_VIA_PATH_REQUEST`. When a node receives this request, it sends a `PATH_REQUEST` to all its neighbors. On receiving their shortest paths, it calculates all the quasi-shortest paths that do not contain the source node and sends these paths back to the source. Thus, on receiving a `QUASI_VIA_PATH_REQUEST` from A,

node B sends not only its shortest path but also the paths $r' =< B, D, C >$ and $s =< B, E, C >$ back to A. This allows A to discover the path r. We define paths like r to be *quasi-via paths*, and a multipath that contains the primary path p, quasi-shortest path q and the quasi-via path r is called a *Quasi Multipath*. Figure 3.6 indicates the messaging that takes place during discovery of quasi-via paths from node A to destination C.

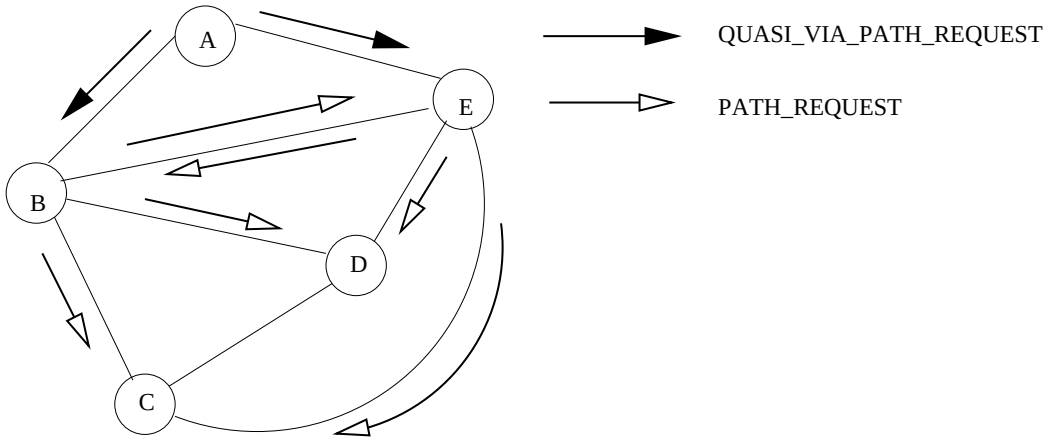

Figure 3.6: Requests Generated by Node A to Find Quasi-Via Paths to Destination C

For the graph of figure 3.4, the alternate route $< W, X, Y, Z >$ is discovered when node W sends a `QUASI_VIA_PATH_REQUEST` to its neighbor X, and X sends back its quasi-shortest path $p =< X, Y, Z >$. Therefore, failure of link (W, Z) does not disrupt communication between nodes W and Z.

Thus, quasi-via paths of a node are essentially the quasi-shortest paths of its neighbors. The inclusion of quasi-via paths in the route discovery procedure allows a node to find more alternate routes to a destination and thereby provides more options for routing. The size of a quasi-via path is bounded by the number of nodes V. The messaging overhead incurred by quasi-via path messages depends upon the number of neighbors m of the source's neighbor, and its complexity is $O(V \times m)$.

It might be argued that to reduce overhead while calculating the quasi-via path, it is enough for a node to request for its neighbor's quasi-shortest path to a given destination. However, each node may select a quasi-shortest path based on its own requirements for an alternate route such as disjointness from the main route, shortest cost, etc. Therefore, using a neighbor's chosen quasi-shortest path to obtain a quasi-via path makes the obtained path dependent upon the path selection criteria of its neighbor. To prevent route discovery mechanisms from being influenced by route selection policies of neighboring nodes, every node tries to discover all its quasi-via paths before choosing one as an alternate path.

The disadvantage of using quasi-via paths for alternate routing is that the choice of path taken by a packet is no longer limited to the source node as the path is not the next-hop neighbor's primary path to a destination. Thus, the neighbor needs to know the quasi-shortest path on which to forward the packet. This can be achieved using

source routing wherein the neighbor is informed of the next hop to forward the data packet. However, source routing requires extra processing at the neighboring router, which may not always be acceptable. Therefore, MFTPR initiates route discovery for quasi-via paths only when alternate routes that use quasi-shortest paths are not existent or not viable.

Once the various routes have been discovered, a node selects the paths it wishes to use as alternate routes and makes entries for these in the backup routing table. The backup routing table is updated and maintained by MFTPR as explained below.

Route Maintenance

QSMP always calculates backup routes when two neighbors exchange routing tables. However, when routing algorithms try to adapt to failures and changes in network topology, routing loops may result and it could be a while before the algorithm converges to correct these looping paths. For this period of time, the backup paths computed by QSMP are subject to loops too. To avoid loops, in the event of failure of a route, MFTPR propagates information about invalid backup routes. In order to illustrate the message propagation, we consider the following graph of figure 3.7. The primary routing table for node A is shown in figure 3.8 along with its backup routing table, which is obtained using MFTPR. The backup routes are selected to be

completely disjoint from the primary route with the least possible cost. The backup routes for nodes B, F, G, and H are quasi-shortest paths, while those for nodes C, D, and E are quasi-via paths.

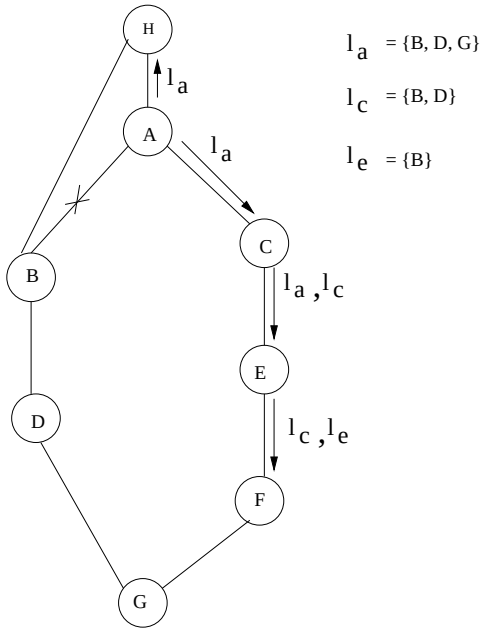

Figure 3.7: Maintenance of Backup Routes

Assume that link (A, B) fails as shown in the figure. On failure of the link (A, B), node A looks into its main routing table and determines all the primary routes that are invalidated because of loss of the link (A, B). It computes a list of the destinations of these primary routes, $l_A = B, D, G$. The backup routes in node A's backup table that have node B as their next hop are also invalidated. This invalidates backup routes

Main Routing Table of Node A

Dest	Next Hop	Pred	Cost
B	B	A	1
C	C	A	1
D	B	B	2
E	C	C	2
F	C	E	3
G	B	D	3
H	H	A	1

Back−up Table

Dest	Next Hop	Cost	Via
B	H	2	−
C	B	6	D
D	C	5	E
E	B	5	D
F	B	4	−
G	C	4	−
H	B	2	−

Figure 3.8: Main and Backup Routing Tables for Node A

for the destinations C, E, F, and H. Then, valid backup routes to the affected destinations B, D, and G are used for routing. The list l affected by failure of primary routes is then broadcast to all the neighbors of node A. For this purpose an `INVALID_ROUTE` message is defined. For A the `INVALID_ROUTE` message contains the list l_A.

When a neighbor such as C receives an `INVALID_ROUTE` message from A, it follows the same procedure as A to determine the list of primary routes l_C that were affected by failure of A's primary route to destinations in list l_A. It looks into its backup routing table and invalidates all the entries in the backup table that contain A as the next hop and have the via field empty and all those paths that have A as the enxt hop and B as the via entry. It then, initiates the use of valid backup routes to the affected destinations. Lastly, it sends an `INVALID_ROUTE` message to all its neighbors.

This message contains the list of destinations sent by A, i.e., l_A as well as its own list of destinations with failed primary routes (l_C).

When any neighbor k receives this message containing l_A, it invalidates all the Via routes through A and follows the same procedure as above, and computes the list l_k. It then sends the `INVALID_ROUTE` message to its neighbors containing lists l_C and l_k and discards the list l_A. Thus destination lists are propagated twice before being discarded. This ensures that information about the failed quasi-via routes are also propagated in addition to the information about failed primary and quasi-shortest paths.

MFTPR aims to propagate news about failed routes as fast as possible so that nodes can immediately try to use backup routes where available. Since this message propagation is independent of the routing protocol's convergence characteristics, MFTPR achieves a degree of autonomy with respect to maintaining and invalidating backup routes. However, MFTPR fails to provide a means of communication when both the primary and back-up routes to a destination fail. In such a case, MFTPR waits for the routing algorithm to converge and find a new primary route to the destination.

An alternate route to a destination that is not completely disjoint from the primary may fail simultaneously along with the primary route. Therefore, some means must

be provided to evaluate the resilience of a pair of paths to failure so that this can be taken into account when a node selects an alternate path to a destination. A selected pair of paths can be evaluated for resilience as discussed below.

Route Selection Policies and Resilience to Failures

MFTPR allows each node to choose alternate paths to destinations based on the node's requirements for cost of the path, disjointness, etc. If a node chooses an alternate path to a destination that is completely disjoint from the primary route, it is assured of a means of communication to the destination even if a node failure causes the primary path to fail. However, when the paths chosen are not completely disjoint from one another, the failure of even one of the non-disjoint components in these paths can cause both paths to fail at the same time.

In order to evaluate the non-disjointness of two paths, we define a metric called the Commonality. To measure commonality of two paths, consider a pair of paths p_1 and p_2 for a given source-destination pair. Let l_1 and l_2 be the lengths of p_1 and p_2 respectively, where we define the length of a path to equal to the number of intermediate nodes between the source and destination. Assuming k nodes to be in common between p_1 and p_2, we define a commonality factor between p_1 and p_2 as

$$C_{p_1p_2} = \frac{k}{(l_1 + l_2 - k)} \times 100$$

$C_{p_1p_2}$ represents the percentage of the two paths p_1 and p_2 that are in common. A commonality of zero implies that p_1 and p_2 are completely disjoint, and commonality of 100 implies that p_1 and p_2 are the same path. As the commonality between p_1 and p_2 increases from zero to 100, the number of network components in common between them increases, thus increasing the likelihood of simultaneous failure of both paths.

In order to choose a pair of paths, a node can set a threshold for commonality, and require that no pair of chosen paths have a commonality factor greater than this threshold. The threshold, in reality, represents the extent to which a node is willing to compromise on disjointness for a pair of paths, and thereby resilience to failures, to acquire alternate routes or satisfy other criteria such as minimum cost etc. Thus using commonality, a tradeoff can be achieved between path cost and the resilience of the pair of paths to failure.

Although commonality does indicate the likelihood of failure of a pair of paths, it does not quantify the resilience of a pair of non-disjoint paths to failure. In order to measure the resilience, we define a *risk* factor $R_{p_1p_2}$ as the probability that paths

p_1 and p_2 fail simultaneously. This risk is dependent on the probability of failure of any one or more of the k nodes that are in common between paths p_1 and p_2. Each of these k nodes is assumed to have failure probability f_i, $1 \leq i \leq k$. It is assumed that the probability function for f has an exponential distribution. By the memoryless property of the exponential distribution function [32], the lifetime of a node is independent of the time at which the probability of failure is being calculated. We also assume that the failure of any node is an independent event.

When k nodes are common between the paths p_1 and p_2, the failure of any one of these nodes can cause both paths to fail. Thus the probability that source can communicate with the destination is equivalent to the probability that none of these k nodes fail.

Therefore,

$$P(\text{successful communication}) = ((1 - f_1) \times (1 - f_2) \times .. \times (1 - f_k))$$

Risk is defined as the probability of failure of communication and hence,

$$R_{p_1 p_2} = 1 - P(\text{successful communication})$$

$$= \quad 1 - ((1 - f_1) \times (1 - f_2) \times .. \times (1 - f_k))$$

A risk of zero signifies completely disjoint paths. When $p_1 = p_2$, the main and backup route are the same, and hence the risk indicates the probability of failure of the path itself. A low risk implies greater resilience and as the risk increases, the resilience of the pair of paths to failure decreases. When a pair of paths has a high commonality factor, the risk measures the actual probability of failure of this pair of paths. Thus if a node wishes to have a risk of at most x for a pair of paths, it can select any pair of paths with complete disregard to their commonality as long as the risk for this pair of paths satisfies x. A tradeoff can be achieved between resilience and cost of the paths by selecting an alternate path that has the least cost and satisfies x. From a node's perspective, risk denotes the probability that communication will fail.

Although risk represents a realistic measure of failure, its computation requires each node to have knowledge of the failure probabilities of all the nodes in a network. This may not be possible in very large networks. In the event that a risk computation cannot be facilitated, commonality can be used as a criterion for an indirect evaluation of the resilience of a pair of paths. Using risk and commonality, nodes can choose to achieve a tradeoff between resilience and other factors such as costs, delays, etc.

MFTPR provides modules for calculating these factors and can be configured to choose backup routes that satisfy the fault tolerance requirements of the node.

In the following section, we present a simulation model of MFTPR and evaluate its performance.

Simulation and Experiments

To measure the improvement of MFTPR's route discovery scheme over that of QSMP, the protocol was simulated using an object-oriented event-based simulator engine (NRLSIM). In order to implement the protocol, the network is modeled so as to facilitate fault tolerance wherein each node in the network requires $n-1$ number of alternate paths to all destinations in addition to the primary route. Thus the network system is viewed as a Fault Tolerance-n (FT-n) system.

The network is simulated using a randomized graph generation algorithm, and the resulting graph is guaranteed to be connected. At any instant of time, the network is assumed to have a particular *level* of fault tolerance, where the level is equivalent to the average number of routes in the network computed over all source-destination pairs. The initial fault tolerance level of the system is FT -1, where only the primary route between source and destination exists. The primary route may be provided

by a distance vector routing algorithm, which implies that each node in the network contains only local topological information. The simulation actually initializes routing tables at the nodes using Dijkstra's shortest path algorithm to emulate the distance vector algorithm. The routing tables are generated to be consistent with those obtained using the distance vector routing algorithm.

Once the routing databases have been initialized, MFTPR messages are used to calculate quasi-shortest paths and quasi-via paths from every source to every destination. For propagation of MFTPR messages, the propagation delay is proportional to the cost of the link between the sender and receiver of the message. A unit of link delay is assumed to 0.003 milliseconds and the propagation delay of any link is equivalent to the product of link cost and the unit of link delay (0.003 ms.). The processing delay of the node is assumed to be constant and ignored in this calculation. Moreover, the message sent over any link is assumed never to be lost, or damaged. It is also assumed that no topology changes take place during route discovery. The MFTPR algorithm was tested for networks of varying size and densities.

Experiments and Analysis

The first set of experiments determines the improvement in the number of alternate routes discovered by using quasi-via paths as compared to quasi-shortest paths.

Each node is the network is required to find $n-1$ alternate routes to each destination with a maximum commonality of $c\%$ with the primary route. Initially, all the nodes in the network find only the quasi-shortest paths that can be as used as alternate routes to destinations. If a node is unable to find the required number of alternate paths to a destination, then it initiates the discovery of quasi-via paths to that destination. At the end of the simulation, the number of nodes that are able to find $T\%$ of extra alternate paths using quasi-via paths is computed, where $T\%$ is the threshold for determining the improvement of MFTPR over QSMP. The threshold is the minimum percentage increase of number of alternate quasi-via paths as compared to quasi-shortest paths. The experiments were conducted on graphs of different sizes by varying the number of nodes from 35 to 65, in steps of 10. For each graph size, the average degree was varied from 3 to 9 in steps of unity and the commonality was varied from 0% to 30%. The threshold for testing the performance improvement of MFTPR was varied from 30% to 50%. The percentage of the nodes in the graph that experienced the threshold amount of improvement has been plotted against varying densities for a given graph size.

We can see in the graph of Figure 3.9, as the density of a 35 node graph increases from 4 to 6, there is an increase in the percentage of nodes in the graph that are able to find 50% more alternate routes that are disjoint from the primary route, using

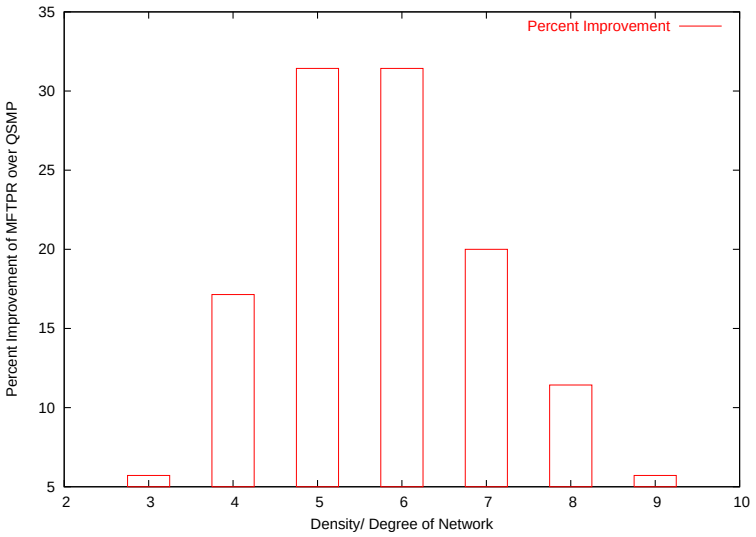

Figure 3.9: 35 Nodes, Commonality 0%, Threshold 50%

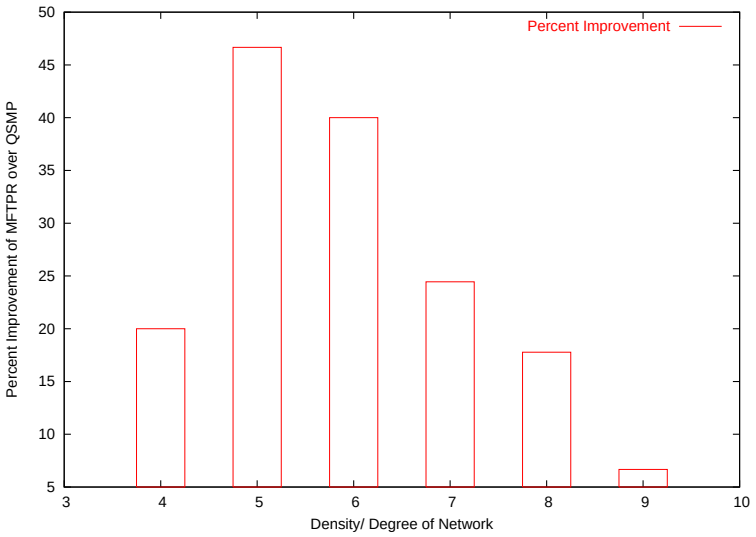

Figure 3.10: 45 Nodes, Commonality 0% , Threshold 40%

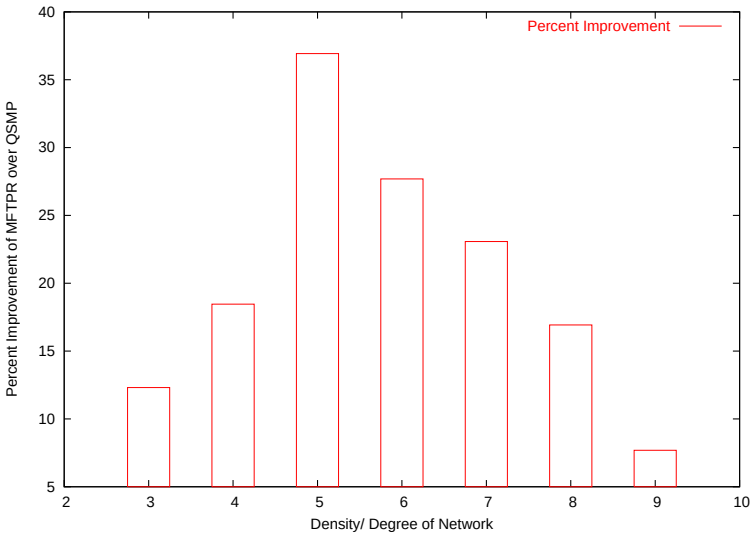

Figure 3.11: 65 Nodes, Commonality 0%, Threshold 50%

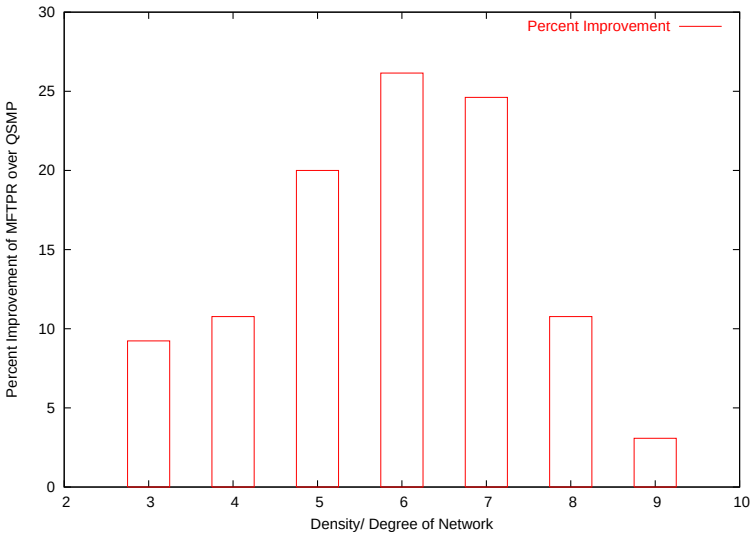

Figure 3.12: 65 Nodes, Commonality 30%, Threshold 40%

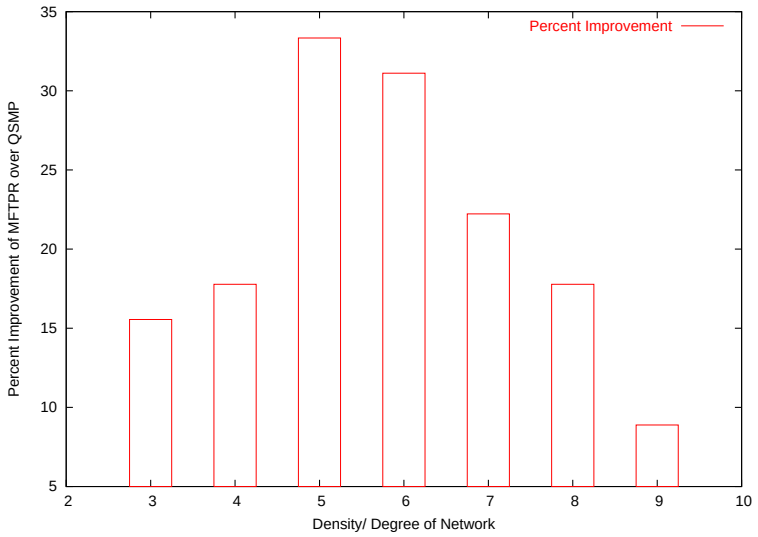

Figure 3.13: Nodes 45, Commonality 30%, Threshold 30%

quasi-via paths. However, as the density of the graph increases, the nodes are able to find the required number of alternate routes using quasi-shortest paths itself and do not need to use quasi-via paths to find more routes. A 35 node graph of degree 3 can be classified as a sparse graph. The reason that quasi-via paths are unable to bring a marked improvement in the discovery of alternate routes can be attributed to the fact that most of the network does not contain alternate paths that are disjoint from the main route. Larger networks that need to find alternate routes that are completely disjoint from the primary route exhibit similar behavior. This leads us to conclude that quasi-via paths are more effective when used in a graph that is not dense or sparse, to find alternate paths that are disjoint from the main route. However, if

the alternate paths do not have to be disjoint from the main route, then quasi-via paths do contribute to finding more alternate routes than quasi-via paths even in sparse graphs as seen in the graph of Figure 3.13. In the same graph we can also see that networks with higher density are hardly aided by quasi-via paths to find more alternate routes than quasi-shortest paths. This can be explained by considering the commonality factor in addition to the density of the network. As the density of a graph increases, the number of available paths increase. Quasi-shortest paths are able to discover a large number of these paths. Further, relaxing the requirement for disjoint routes yields the required routes using quasi-shortest paths themselves. As a result, nodes do not try to discover quasi-via paths. Thus, we conclude that quasi-via paths are more effective in discovering alternate routes that are not disjoint from the main route in networks that are not dense.

To clearly understand the effect of commonality on the ability of MFTPR to find more routes, a second set of experiments was conducted that tested the performance of MFTPR with varying the maximum commonality allowed for a pair paths.

We see in the graph of Figure 3.14, that increasing the allowed commonality for routes aids MFTPR in finding more paths and thereby, the fault-tolerance level of the system is increased. It is seen that a commonality of 10% or less has little or no effect in improving the level of fault tolerance of the system. This can be explained

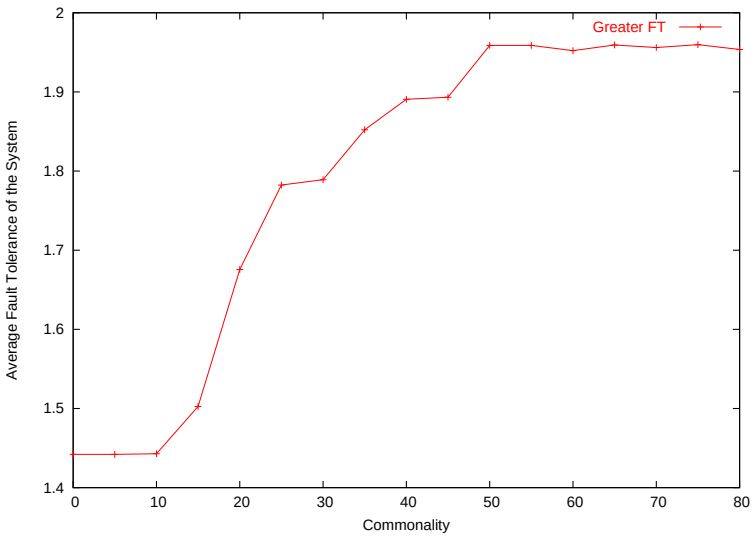

Figure 3.14: Nodes 35, Degree 4

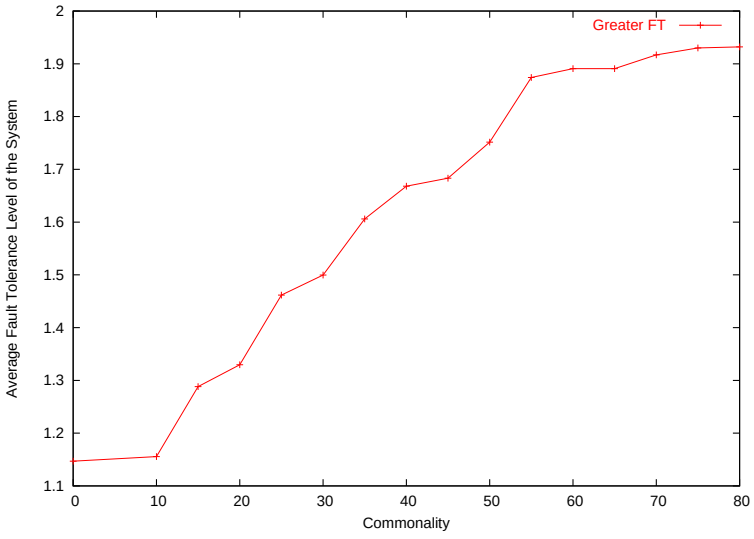

Figure 3.15: Nodes 55, Degree 4

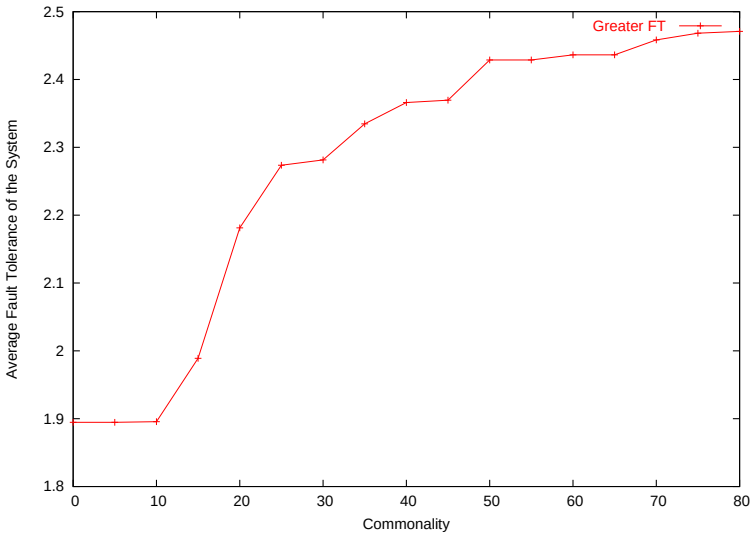

Figure 3.16: Nodes 65, Degree 5

by considering the effect of commonality on a pair of relatively short paths. For example, if the number of intermediate nodes of the a pair of paths p_1 and p_2 were 3 and 5 respectively, then even a single node in common for this pair of paths yields a commonality of 12.5% and so they do not satisfy the commonality requirement. For longer pairs of paths, a commonality of 10% still translates to nearly complete disjoint routes. Thus the increase in commonality from 0% to 10% does very little to aid MFTPR to find more routes.

However, when the commonality is greater than 10% both shorter and longer pairs of paths satisfy the criteria for commonality. Thus MFTPR is able to find more routes that satisfy the commonality requirement and consequently the level of fault

tolerance of the network is increases. However, after commonality increases beyond a particular value, MFTPR is unable to greatly enhance the fault tolerance of the network system. This is because the longer paths in the network contain too many nodes and are unable to satisfy most commonality requirements. For instance, in the extreme case, the longest path in a network visits each and every node in the network before reaching the destination. Such a path has a 100% commonality with all paths in the network. For long paths in the network, it is difficult to find routes which visit nodes that are not in common with the long path. Thus MFTPR reaches a saturation point beyond which it is unable to find eligible alternate routes in the network even if the allowed commonality increases. In the graphs of Figure 3.14 and Figure 3.16, this saturation point is reached at a commonality of 50%. Thus MFTPR is very effective in finding alternate routes to destinations when the commonality factor is greater than 10% and lesser than the saturation point for the particular network. However,increased commonality indicates a higher likelihood of simultaneous failure of the pair of paths. Consequently, we advocate the use of lower values of commonality, i.e., between 10% and 30% to achieve a good tradeoff between resilience and more alternate paths.

We can conclude from the above results that the performance of MFTPR can be optimized by choosing the appropriate commonality depending on the density of the

network.

CHAPTER 4

SUMMARY AND FUTURE WORK

An increase in the size of networks along with the advent of the Internet has resulted in a large number of applications that use the Internet for communication and other services. However, the Internet does not provide any service quality guarantees as it currently uses a best-effort model for routing. Hence, it cannot provide any guarantees applications that have stringent requirements such as reliable data delivery. Reliability can be considered a QoS parameter, and increasing reliability of data delivery can help improve the QoS offered to applications.

Reliability of data delivery in network routing can be improved by incorporating fault tolerance into the network, thereby increasing its resilience. One of the fault-tolerant approaches for routing is to use a multipath scheme wherein a set of multiple routes are calculated between a given source and destination. One of these routes can be used as the primary route and the others as backup. Upon failure of the primary path, one of the backup paths can be used for routing, thereby ensuring uninterrupted communication.

Based on the multipath scheme, this thesis has proposed the use of a fault tolerant protocol, called the Multipath Fault Tolerant Protocol for Routing (MFTPR), to improve the reliability of network routing services. The protocol is designed to work in conjunction with the routing protocol employed by the network and performs functions related to providing fault tolerance in routing. It is based on a multipath discovery algorithm, the Quasi-Shortest Multipath (QSMP) [5]. MFTPR extends the QSMP algorithm and improves on its route discovery scheme while retaining the localized nature of the algorithm. MFTPR tries to find alternate routes using QSMP's route discovery scheme, wherein a node calculates its neighbors' shortest paths to a destination and tests these routes for use as an alternate route. An alternate route resulting from this calculation is called the *quasi-shortest* path. In the event that quasi-shortest paths are not found suitable, MFTPR allows the node to request its neighbors for all their possible quasi-shortest routes. The quasi-shortest paths of a node's neighbors to a destination are the node's *quasi-via* paths to that destination. The discovery of quasi-via paths requires extra messaging and additional router processing while forwarding a packet along the quasi-via route. However, the very nature of fault tolerance implies the use of redundancy and overhead. Since MFTPR's primary goal is to provide alternate routes for communication, this extra messaging

overhead is justifiable when a node is unable to find a required alternate route using quasi-shortest paths. Simulation of MFTPR has shown a marked increase in the number of alternate routes as compared to those found by QSMP alone.

MFTPR also provides for maintenance of backup routes in the event of failure of network components. When a failure occurs, MFTPR propagates information about the effect of this failure on the backup routes of neighbors. The propagation of failure is done independently of the routing protocol employed by the network. The basis for this separation lies in the fact that routing loops can be formed while network re-configuration takes place through the routing protocol. Thus, any alternate routes that are calculated or assumed to be available by examining the data in the routing tables while the routing protocol is trying to converge, may themselves have loops or may no longer exist due to the failure. By dealing with network failures and providing for checking the validity of alternate routes, MFTPR ensures that propagation of route failure takes place immediately, thereby eliminating looping or routing using incorrect information. This route maintenance comes at an extra cost but is well justified as it provides the most recent information for a route, and the valid backup routes computed are loop free paths. In this process, if a node discovers that both its primary and backup routes to a destination have failed, it waits for the routing protocol to converge in order to communicate. The route maintenance however, does

not try to gather information for backup routes or initiate any route discovery for the time required for the routing protocol to converge, and thereby does not incur extra messaging when the network is being flooded with exchange of routing tables. The compartmentalization of route maintenance also has the advantage that route maintenance is independent of the routing protocol being used in the network.

MFTPR separates the route discovery mechanism for backup routes from being influenced by the criteria that decide the choice of an alternate route over other available alternate routes. For this reason, no node is forced to use its neighbor's chosen alternate path. Each node is allowed to discover all the routes to a destination that can be found using MFTPR, and the node decides which route(s) it wishes to use for a backup path to a destination.

The choice of alternate routes is generally governed by the need for resilience to failure of the primary route. Thus, paths that are disjoint from the main route are generally preferred. However, disjoint paths may not always be available and moreover, the available disjoint paths may have path costs that are much higher as compared to the cost associated with the primary route. Some nodes may require alternate paths to have lower costs and may be willing to tradeoff some resilience in return for minimized costs. In such cases, non-disjoint paths can be used. However, while using a pair of non-disjoint paths, it is important to evaluate their resilience

to each other's failures. This thesis proposes a pair of metrics that can be used to measure the possibility of failure of a pair of paths simultaneously.

The first of the proposed pair of metrics is called the *commonality* and is a measure of the non-disjointness of a pair of paths. A non-zero commonality signifies that the two paths have network components in common. As commonality increases, the likelihood that the single failed network component leads to simultaneous failure of both paths increases. Thus commonality is an indication of the resilience of a pair of paths to simultaneous failure. When a node wishes to select an alternate path from a number of possible paths, it can choose the path that yields the least commonality when compared to the main route. The second proposed metric is known as *risk* and it measures the probability of simultaneous failure of a pair of paths from a source to destination. Since risk is calculated using the failure probabilities of each of the individual nodes that are in common for the pair of path, it can be viewed as a more direct measure of the resilience of a pair of paths. As the risk factor for the pair of paths increases, the resilience of this pair to simultaneous failure decreases. However, the computation of risk requires knowledge of the failure probabilities of the common nodes of the two paths. If this knowledge is not available, commonality can instead be used to estimate resilience. MFTPR can be configured to use either of the metrics to facilitate route selection at the various nodes.

In summary, MFTPR tries to find more alternate routes than the QSMP route discovery algorithm. It also manages information regarding backup routes and tries to inform other nodes about failed backup routes. Lastly, it separates the route selection policy from the route discovery mechanism, allowing each node to choose paths independent of any influence from its neighbors.

Future Work

Since, MFTPR extends the QSMP algorithm, it assumes the availability of information required by QSMP to calculate alternate routes. However, the routing protocol would need to be augmented with procedures to calculate the required information. In keeping with the principle of separating fault tolerant mechanisms from routing procedures, it is suggested that a module to calculate the required information be incorporated into MFTPR.

The applicability of multipath routing is not limited to fault tolerance alone. Multipath routing is widely used to provide load-balancing for traffic in communication networks and wireless ad hoc networks. A possible area of future work in MFTPR would be to sensitize this protocol to dynamic changes in traffic patterns so that alternate paths can be chosen to avoid congested areas of the network and thereby minimize end-to-end delays.

This thesis has also presented means to analyze the resilience of a pair of non-disjoint paths. This analysis could be extended to measure the resilience to failure of a multipath containing more than 2 paths. Theoretical and empirical models for such analysis is a possible area of future work.

BIBLIOGRAPHY

[1] B. Kemme and G. Alonso. Database replication based on group communication, February 1998. Technical report, Department of Computer Science, ETH Zurich, No. 289.

[2] B. V. Caenegem, W. V. Parys, F. D. Turck, and P. M. Demeester. Dimensioning survivable WDM networks. *IEEE Journal on Selected Areas in Communications*, 16:1146–1157, 1998.

[3] C. Chenig, S. P. R. Kumar, and J. J. Garcia-Luna-Aceves. A distributed algorithm for finding the k disjoint paths of minimum total length. In *28th Annual Allerton Conference on Communication, Control, and Computing*, October 1990.

[4] A. Chowdhury, O. Frieder, E. Burger, D. Grossman, and K. Makki. Dynamic Routing System (DRS): fault tolerance in network routing. *Computer Networks (Amsterdam, Netherlands)*, 31(1–2):89–99, 1999. citeseer.nj.nec.com/chowdhury99dynamic.html.

[5] Xuanming Dong, Pravin Varaiya, and Anuj Puri. Quasi-shortest Paths for Multipath Routing in Packet-switched Networks. In *Proceedings of the International Conference on Internet Computing (IC 2002)*, pages 125–131, June 2002.

[6] David Eppstein. Bibliography on algorithms for k shortest paths: http://liinwww.ira.uka.de/bibliography/Theory/k-path.html.

[7] David Eppstein. Finding the k shortest paths. In *Proc. 35th IEEE Symposium on Foundations of Computer Science*, pages 154–165, 1994. citeseer.nj.nec.com/eppstein97finding.html.

[8] Paul Ferguson and Geoff Huston. *Quality of Service: Delivering QoS on the Internet and in Corporate Networks*. John Wiley & Sons, Inc., New York, NY, 1998.

[9] Paul Ferguson and Geoff Huston. Quality of Service in the Internet: Fact, Fiction or Compromise? In *INET'98*, Switzerland, Geneva, July 1998.

[10] Deepak Ganesan, Ramesh Govindan, Scott Shenker, and Deborah Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. In *In Mobile Computing and Communications Review (MC2R)*, volume 1, 2002. citeseer.nj.nec.com/ganesan01highlyresilient.html.

[11] W. Heimerdinger and C.Weinstock. A Conceptual Framework for System fault Tolerance. Technical report, CMU/SEI-92-TR33 ESC-TR-92-033, October 1992. http://www.sei.cmu.edu/pub/documents/92.reports/pdf/tr33.92.pdf.

[12] Alon Itai and Michael Rodeh. The multi-tree approach to reliability in distributed networks. In *Proc. 25th IEEE Symposium on Foundations of Computer Science*, pages 137–147, 1984.

[13] Irene Katzela and Mischa Schwartz. Schemes for fault identification in communication networks. *IEEETNWKG: IEEE/ACM Transactions on Networking IEEE Communications Society, IEEE Computer Society and the ACM with its Special Interest Group on Data Communication (SIGCOMM), ACM Press*, 3, 1995.

[14] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison Wesley, 2000.

[15] Juan Leon, Allan L. Fisher, and Peter Steenkiste. Fail-safe PVM: A Portable Package for Distributed Programming with Transparent Recovery. Technical Report CMU-CS-93-124, Feb 93.

[16] David Lomet and Gerhard Weikum. Efficient transparent application recovery in client-server information systems. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 460–471. ACM Press, 1998.

[17] G. Malkin. RFC 2453: RIP version 2, November 1998. ftp://ftp.internic.net/rfc/rfc1388.txt.

[18] M. Marina and S. Das. On-demand Multipath Distance Vector Routing in Ad Hoc Networks. In *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, pages 14–23, 2001. citeseer.nj.nec.com/marina01demand.html.

[19] E. Martins, M. Pascoal, and J. Santos. The k shortest paths problem, June 1998. Research Report, CISUC.

[20] J. M. McQuillan, I. Richer, and E.Rosen. Arpanet routing study - final report. BBN Report No. 3641, September 1977.

[21] Philip M. Merlin and Adrian Segall. A failsafe distributed routing algorithm. *IEEE Transactions on Communications*, 27(9):1280–1287, September 1979.

[22] Microsoft Corp. http://www.microsoft.com.

[23] M.Nicolaidis, S.Manich, and J.Figueras. Achieving Fault Secureness in Parity Prediction Arithmetic Operators : General Conditions and Implementation. In *Proceedings of the European Design and Test Conference*, pages 186–193, 1996.

[24] J. Moy. RFC 2328: OSPF version 2, April 1998. ftp://ftp.internic.net/rfc/rfc2178.txt.

[25] The TechTarget Network. http://whatis.techtarget.com.

[26] R. Ogier, B. Bellur, and N. Taft-Plotkin. An efficient algorithm for computing shortest and widest maximally disjoint paths. Technical report, SRI International Technical Report, 1998.

[27] Richard G. Ogier, Vlad Rutenburg, and Nachum Shacham. Distributed algorithms for computing shortest pairs of disjoint paths. *IEEE Transactions on Information Theory*, 39(2):443–455, March 1993.

[28] The QoS Forum White Paper. The Need for QoS, 1999. Stardust.com Inc.

[29] J. Postel. RFC 793: Transmission control protocol, September 1981. ftp://ftp.internic.net/rfc/rfc793.txt.

[30] Jian Pu, Eric Manning, Gholamali, and C. Shoja. Routing reliability analysis of partially disjoint paths. In *IEEE Pacific Rim Conference on Communications, Computers and Signal processing (PACRIM' 01)*, volume 1, pages 79–82, August 2001.

[31] Jian Pu, Eric Manning, Gholamali C. Shoja, and Anand Srinivasan. A new algorithm to compute alternate paths in reliable ospf (rospf). In *Proceedings of*

PDPTA '2001 (the 2001 International Conference on Parallel and Distributed Processing Techniques and Applications), pages 299–304, June 2001.

[32] Sheldon Ross. *Introduction to Probability models, Fourth Edition.* Academic Press, San Diego, California, 1989.

[33] N. M. Rouphail, S. Ranjithan, W. El Dessouki, T. Smith, and E. D. Brill Jr. A decision support system for dynamic pre-trip route planning. In *Fourth International Conference on Application of Advanced Technologies in Transportation*, pages 325–329, June 1995.

[34] M. Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, 1990. citeseer.nj.nec.com/satyanarayanan90coda.html.

[35] Deepinder Sidhu, Raj Nair, and Shukri Abdallah. Finding disjoint paths in networks. In *Proceedings of the conference on Communications architecture & protocols*, pages 43–51. ACM Press, 1991.

[36] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Conceptsi, Fourth Edition.* McGraw Hill, 2001.

[37] Paul Stelling, Cheryl DeMatteis, Ian T. Foster, Carl Kesselman, Craig A. Lee, and Gregor von Laszewski. A fault detection service for wide area distributed computations. *Cluster Computing*, 2(2):117–128, 1999.

[38] J. W. Surballe and R. E. Tarjan. A quick method for finding shortest pairs of disjoint paths. *networks*, 14:325–336, 1984.

[39] Cisco Systems. http://www.cisco.com.

[40] Taft-Plotkin, B. Bellur, and R. Ogler. Qualityof-service routing using maximally disjoint paths. In the Seventh International Workshop on Quality of Service, June 1999.

[41] Andrew S. Tanenbaum. *Computer Networks, Third Edition*. Prentice-Hall, 1996.

[42] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems : Principles and Paradigms*. Pearson Education, 2001.