MULTI-CONSTRAINED NODE-DISJOINT MULTIPATH QoS ROUTING ALGORITHMS FOR STATUS DISSEMINATION NETWORKS

By

SUPREETH KOUSHIK SHESHADRI

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

MAY 2004

To the faculty of Washington State University:

The members of the Committee appointed to examine the thesis of SUPREETH KOUSHIK SHESHADRI find it satisfactory and recommend that it be accepted.

_____

Chair

_____

_____

Acknowledgements

The satisfaction and euphoria that accompany the successful completion of any project would be incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crowned my efforts with success.

First, I would like to express my sincere gratitude and thanks to Prof. Carl Hauser, for constantly motivating and guiding me throughout this thesis. I sincerely believe that without the able guidance of Prof. Carl Hauser the completion of this thesis would not have been possible.

I would like to thank Prof. Dave Bakken and Prof. Curtis Dyreson for their valuable inputs through out the entire period of the project.

I am grateful to all the staff members of the School of Electrical Engineering and Computer Science for all the help and information they have provided for the fulfillment of the thesis.

I would like to thank Ioanna Dionysiou for helping me in the early phase of thesis. Finally I would like to thank all the members of the Concurrent and Distributed Systems Research group for their constant suggestions and inputs.

MULTI-CONSTRAINED NODE-DISJOINT MULTIPATH QoS ROUTING ALGORITHMS FOR STATUS DISSEMINATION NETWORKS

Abstract

by Supreeth Koushik Sheshadri, M.S
Washington State University
May 2004

Chair: Carl H. Hauser

Status dissemination networks (SDN) are geographically distributed critical infrastructure networks wherein, the status information which includes data and/or control information needs to be gathered and distributed to the registered clients through status variables according to their QoS specification in a reliable and timely manner. In order to achieve this, SDN require routing algorithms to find multiple disjoint paths, which satisfy multiple QoS constraints simultaneously.

In this thesis, we present a heuristic algorithm for QoS routing, which ensures reliability and timeliness by computing a pair of node-disjoint paths that satisfy multiple QoS constraints. The multiple QoS constraints addressed by the heuristic algorithm are delay, delay-jitter, bandwidth and buffer space. We show that the performance of our heuristic algorithm is better than the remove-find method in terms of committing QoS subscriptions, utilization of network resources and finding disjoint pairs of paths.

Key Words: QoS Routing, Node Disjoint Paths, Reliable Routing.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Subsection 1.1 gives a brief introduction to this thesis. Subsection 1.2 describes the thesis problem in brief and its significance. Subsection 1.3 talks about the motivation behind this research and also discusses the GridStat architecture. Subsection 1.4 presents a few definitions and terms commonly used in networking community. Subsection 1.5 highlights the organization of the thesis.

1.1 Introduction

Status dissemination networks (SDN) are geographically distributed critical infrastructure networks. In SDN, the status information which includes data and/or control information is gathered and distributed to the registered clients through status variables according to their QoS specification in a reliable, timely and secure manner. The registered clients include publishers, subscribers or both. Publishers are intelligent devices or programs, which post information through status variables without knowing the details about consumers. Subscribers are programs, which request information about status variables with specific QoS constraints of redundant path. An underlying communication infrastructure routes packets between registered entities. One way to achieve reliability is through redundancy. Our approach to redundancy is to provision multiple disjoint paths between each source and destination. Disjoint paths may be either link-disjoint or node-disjoint. Since SDN are critical in nature, node-disjoint paths are favored, if they exist for the additional reliability they provide.

A QoS routing algorithm is intended to deliver several different service qualities such as latency (delay), bandwidth (rate), delay-jitter and buffer space. It has been shown that finding paths subject to multiple independent QoS constraints is NP-Complete [3].

However, Ma and Steenkiste [5], building on results of Zhang [23], have shown that when routers use proportional allocation, such as WFQ, latency, bandwidth, delay-jitter and buffer space are not independent and the multi-constrained QoS routing problem for single paths becomes tractable.

In this thesis, we present a heuristic algorithm for QoS routing, which computes a pair of node-disjoint paths that satisfy the independent, multiple QoS constraints in a reliable and timely manner. Henceforth, we will refer to this as Multi-Constrained Multipath (MCMP) routing algorithm. We show that MCMP heuristic algorithm performs better than simple remove-find method in terms of committing QoS subscriptions, utilization of network resources and in finding disjoint pairs of paths when implemented in real networks.

1.2 Thesis Problem

Consider a network that is represented by a directed graph, G=(V, E), where V is a set of vertices and E is a set of edges. Each link (u, v) is represented by two non-negative measures namely: a delay $\delta(u, v)$ and a residual bandwidth $\mathbf{R}(u, v)$.

The problem of MCMP is defined as follows: Given a delay bound $\boldsymbol{d}$, a delay-jitter bound $\boldsymbol{j}$, and buffer space bound $\boldsymbol{b_m}$ for all $\boldsymbol{m}$ nodes in the network, a leaky bucket $< \sigma, \mathbf{b} >$, where σ is the average token rate and b is the maximal burst rate in bytes (token bucket size) and a bandwidth to reserve $\boldsymbol{r}$ ($r \geq \sigma$), find a pair of node-disjoint paths namely $\boldsymbol{p'}$ and $\boldsymbol{p''}$ such that $D(p, r, b) \leq \boldsymbol{d}$, $J(p, r, b) \leq \boldsymbol{j}$, $B(p, r, h) \leq \boldsymbol{b_h}$ for all nodes along the path, where $\boldsymbol{b_h}$ is the buffer space constraint for the node with h hops from the source and $r \leq R_j$ ($\forall R_j \in p$) where R_j is the link residual bandwidth and $p = \boldsymbol{p'}$ in first iteration and $p = \boldsymbol{p''}$ in second iteration. For a path $\mathbf{p}$, the provable end-to-end delay $\boldsymbol{D}$,

end-to-end delay-jitter ***J*** and buffer space requirement at the h-th hop ***B*** are as defined in Zhang [23, 28].

In other words, devise a path allocation mechanism which includes computing pair of node disjoint paths each satisfying the multiple independent QoS properties such as delay, delay-jitter, bandwidth and buffer space and committing if such feasible paths exists. A path is said to be a feasible path if it meets all the given QoS constraints.

From a real systems perspective what we are interested is not merely finding paths for a single subscription but efficiently using network resources so that many subscriptions can be supported. Applying efficient utilization of network resources tends to support more connections. For single non-redundant paths Ma and Steenkiste [5] found that choosing a widest shortest path meeting the QoS constraints was, in most cases preferable on the grounds of efficient utilization of network resources. Widest shortest path is defined in subsection 1.4.The heuristic algorithm proposed in chapter 3 finds widest shortest path in its second pass to efficiently utilize the given network resources.

1.3 Motivation

Recent deregulation of the electric power grid resulted in the need for delivering status information to legitimate parties in a timely, secure and reliable manner. The current communication infrastructure of electric power grid consists of a fixed hardwired and sequential data acquisition infrastructure, which is not able to meet the new trends in the power industry, [1]. In order to accommodate the needs of recent deregulation, we have designed and built a novel status dissemination and control infrastructure for electric power grids called *GridStat, [2],* which is capable of gathering and distributing status

information to any registered entities in the network according to their QoS specification in a reliable, timely and secure manner.

The primary challenge in delivering status information (data and control) with QoS guarantees in a SDN is to find a path that satisfies the given QoS requirements and the secondary challenge is to ensure reliability by finding a pair of such disjoint paths. In order to achieve the above-mentioned goals, an efficient QoS routing algorithm ensuring reliability is needed.

Lack of such an efficient QoS routing algorithm ensuring reliability, for GridStat has been the motivating factor for this thesis. This thesis addresses the problem by proposing a heuristic algorithm for Multi-Constrained Multipath (MCMP) routing. This thesis assumes that there exists a baseline Status dissemination mechanism for GridStat and this research is focused towards an efficient heuristic algorithm for MCMP routing in order to provide a reliable QoS guaranteeing mechanism for GridStat.

GridStat is a status dissemination middleware framework. Status dissemination middleware is a variant of message-oriented middleware, which has the publish-subscribe model as its underlying technology. This technology offers decoupling in time and space among the participating applications. Applications can be publishers, subscribers or both. The publisher and subscriber interaction is handled by GridStat middleware, which acts as transparent storage for status posted by publishers and as a forwarding agent to disseminate the status to the interested subscribers. Figure 1.1 shows the status dissemination middleware logical view.

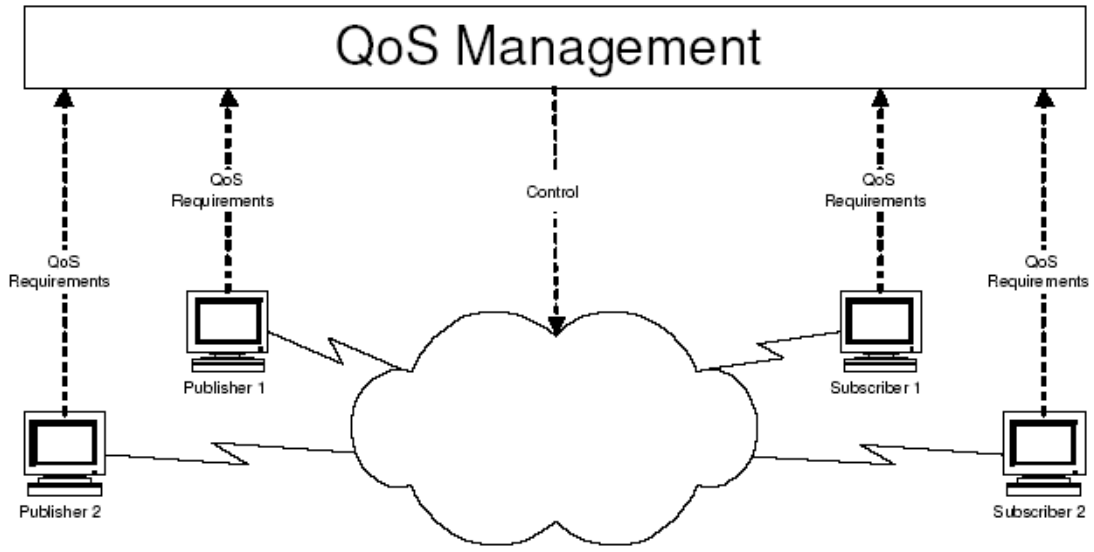

Figure 1.1: Status Dissemination Middleware Logical View [1, 2]

GridStat consists of a data and management planes. The data plane is made up of clouds of status routers while the management plane is made up of a hierarchy of QoS brokers. Status routers are similar to IP routers, but they also provide middleware level routing, aggregation, rate filtering and QoS management mechanisms. A group of status routers are interconnected through point-to-point links to form clouds. Two special kinds of status routers called *Edge status routers* and *Border status routers* are identified. Edge status routers are access points where publishers and subscribers connect to the communication infrastructure. Border status routers connect different clouds. Clouds are autonomous system like the Internet.

The hierarchical management plane consisting of QoS brokers controls these clouds by making path allocation, access control and fault tolerance decisions. The lowest level QoS brokers are called *Leaf QoS brokers*. Each leaf QoS broker manages a single cloud that involves status routers, publishers and subscribers. Each Leaf QoS broker maintains a view of the current cloud topology and currently available network resources.

It is the responsibility of a status router to inform its leaf QoS broker whenever it joins/leaves the cloud so as to enable the leaf QoS broker to always maintain the correct view of the network topology.

The goal of the GridStat architecture is that the topology information and current state information pertaining to each cloud be known only to its Leaf QoS broker and be abstract to its upper-level QoS brokers. The upper-level QoS brokers manage multiple clouds and are responsible for the allocation and de-allocation of inter-cloud subscriptions.

GridStat is intended to deliver several different service qualities: latency (delay), bandwidth (rate) and redundancy. In this thesis we present a heuristic algorithm for finding and allocating paths in the GridStat network that meet multiple quality of service constraints. The GridStat approach to redundancy is to provision multiple paths from each publisher to each subscriber. These paths are chosen to be node-disjoint when possible.

Finding node-disjoint paths satisfying multiple QoS constraints in a flat network (a network without hierarchical parts) is a previously unsolved problem, which we address here. The hierarchical structure of GridStat further complicates the problem and so a solution that exploits the hierarchy remains for future work.

In order to achieve the two goals of GridStat routing, namely satisfying multiple QoS constraints and ensuring redundancy, the current heuristic algorithm requires that the topology information and current network state be known all the way up the hierarchy. In this way the problem of inter domain routing gets reduced to that of intra domain routing. Here routing problems are solved at a common ancestor of the source and destination by

utilizing topology and state information for all the clouds that are to be involved. A detailed architecture of GridStat is shown in figure 1.2 below.

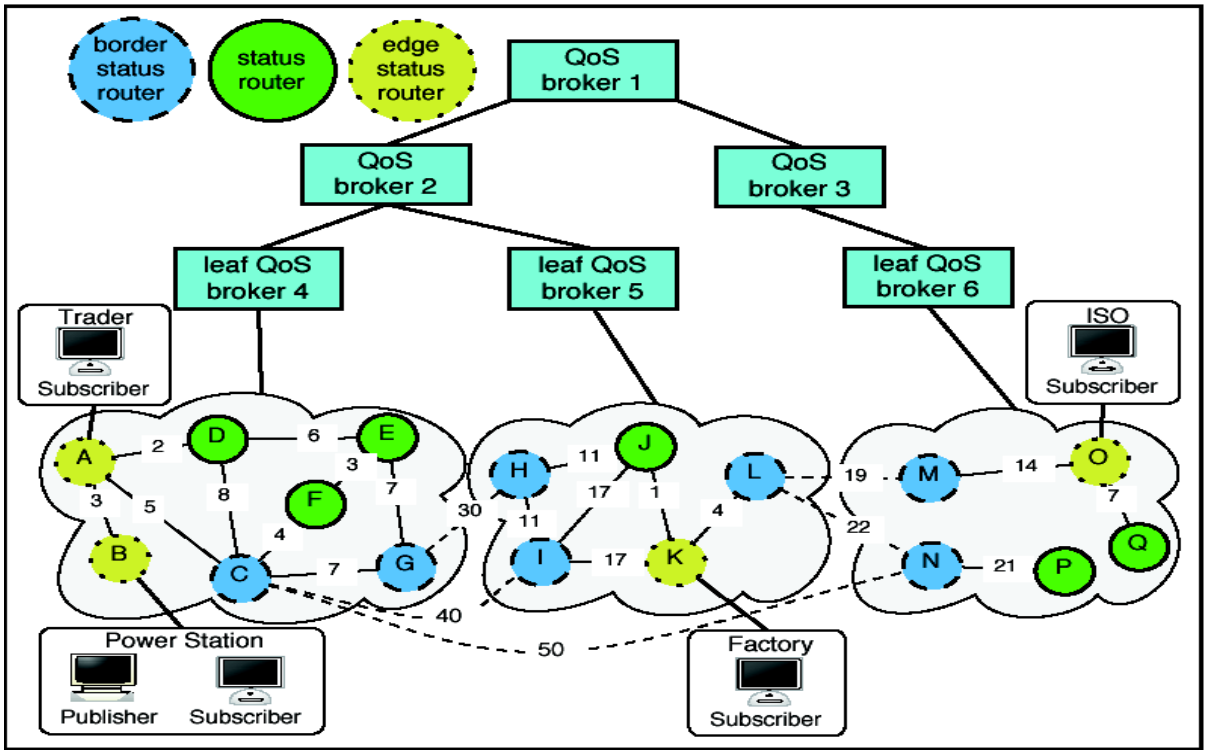

Figure 1.2: Detailed Architecture of GridStat [1]

1.4 Definitions and Explanation of terms

- **Delay:** Delay between the node (u, v) is defined as the time taken by the packet to reach from node u (source) to node v (destination).
- **Bandwidth:** The amount of data that can be sent over a network. It is measured in Kilobytes or Megabytes per second. It also represents the capacity of the connection.
- **Link-disjointness:** A pair of paths is considered link-disjoint if they have no links in common.
- **Node-disjointness:** A pair of paths is considered node-disjoint if they have no nodes in common.

- **Multi-Constrained Path (MCP) Definition**: MCP is defined as follows in [4]: Consider a network that is represented by a directed graph. G= (V, E). Each link (u, v) in E is specified by a link weight vector with as components m additive QoS weights $w_i(u, v) \geq 0, i=1,\ldots\ldots, m$. Given m constraints L_i $i=1,\ldots\ldots, m$, the problem is to find a path P from the source node s to a destination node d such that $w_i(P) = \sum_{(u,v) \text{ in } P} w_i(u,v) \leq L_i$ for i = 1,……..,m . A path that satisfies all m constraints is referred to as a feasible path.
- **Performance:** The qualitative level at which a network fulfills its function.
- **Widest-Shortest path:** Using modified BFS algorithm a feasible path with minimum hop count is found. If several such paths exist then the path with maximum reservable bandwidth is chosen.

1.5 Thesis Outline

Chapter 2 reviews the related literature and discusses the work done towards MCMP to date. Chapter 3 presents the foundation for MCMP algorithm. Chapter 4 discusses the MCMP heuristic algorithm in detail. Chapter 5 analyses the performance of MCMP heuristic algorithm in realistic networks. The thesis ends with a conclusion and also highlights the future work in Chapter 6.

Chapter 2: Related Work

This section is divided into three areas. Subsection 2.1 discusses the work done on computing multi-constrained paths. In this section we discuss the work done by Wang-Crowcroft, Jaffe, Iwata, Chen, Korkmaz-Krunz and Ma and Steenkiste. Subsection 2.2 discusses the work done on finding disjoint paths. In this section we discuss the work done by Suurballe, Sidhu and Bhandari in computing a pair of disjoint paths. Subsection 2.3 discusses the work done on determining multi-constrained disjoint paths. In this section we discuss the work done towards DIMCRA and work done by Orda and Sprintson in finding multi-constrained disjoint paths.

2.1 Multi-Constrained Paths

The need for QoS routing arose as lot of new applications such as video conferencing; IP telephony and multimedia streaming required QoS guarantees, which the existing normal routing algorithms could not provide. QoS Routing involves two components namely routing protocol and routing algorithm. Routing protocol keeps track of the current state of the network topology and its available network resources. Routing algorithm finds a path that satisfies the given QoS constraint and commits if such a path exists. Here after, we assume that an efficient QoS routing protocol exists and discuss only the work related to computation of a feasible path. An efficient resource reservation protocol like RSVP can be used to commit the path.

It has been shown that QoS routing subject to multiple independent additive constraints is know to be NP-Complete [29,3]. According to [6], metrics for path selection can be grouped into additive and non-additive constraints depending on *"whether or not the path metric is obtained by adding the metric value for all links in the*

path". Examples of additive constraints include delay, delay-jitter, and hop count. Example of non-additive constraints includes bandwidth.

Multi-Constrained Path (MCP) Problem Definition

MCP is defined as follows in [4]: Consider a network that is represented by a directed graph, G= (V, E). Each link (u, v) in E is specified by a link weight vector with as components m additive QoS weights $w_i(u, v) \geq 0, i=1,\ldots\ldots, m$. Given m constraints, L_i $i=1,\ldots\ldots, m,$ the problem is to find a path P from the source node *s* to a destination node *d* such that $w_i(P) = \sum_{(u, v) \text{ in } P} w_i(u, v) \leq L_i$ for i = 1,……..,m . A path that satisfies all m constraints is referred to as a feasible path.

2.1.1 Wang-Crowcroft Algorithm

Wang and Crowcroft in [3] proved that the problem of finding a path (just a feasible one and not an optimal one) subject to any two of the metrics of delay, loss probability, cost and jitter is NP-Complete. Wang-Crowcroft algorithm presented a bandwidth-delay-constrained path finding algorithm. The problem of bandwidth-delay constrained path is formally described as "Given nodes *s* and *d* of the graph, and two constraints *B* and *D*, the routing problem is to find a path p* between *s* and *d* so that width $(p^*) \geq B$ and length $(p^*) \leq D$". Width (p*) is referred to as the bottleneck bandwidth of the path p* and length (p*) is referred to as the total delay of all links in path p*. The algorithm first simply prunes all those links that do not satisfy the bandwidth constraint and then finds the shortest path meeting the delay bound in the pruned graph using Dijkstra shortest path algorithm if one such exists. They showed that path subject to a non-additive constraint

such as delay can be found using algorithms with polynomial time complexity. Wang and Crowcroft algorithm cannot be used to solve multi-path constrained problems whose metrics are additive. But the work done by Wang and Crowcroft can be considered as a foundation work towards MCP.

2.1.2 Jaffe's Algorithm

Jaffe [4] presented an algorithm to deal with MCP problem under two constraints. Jaffe's proposal is to use a well-known shortest path algorithm on a linear combination of the two link weights as given by the equation 1, where *d1* and *d2* are positive multipliers.

$$w(u, v) = d1 * w1(u, v) + d2 * w2(u, v) \text{ --------(1)}$$

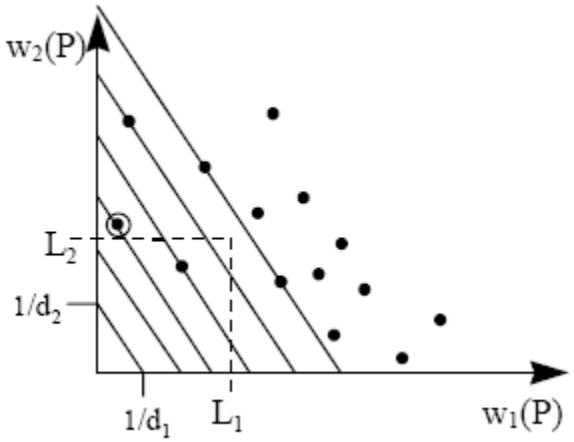

Figure 2.1:Representation of the link weight vector w(P) in two dimensions

Figure 2.1 is a reproduction of the original figure presented in the paper by Van Mieghem et al in [7] which shows a representation of the link weight vector *w*(P) in two dimensions. The algorithm begins its search from the origin along the parallel lines specified by *w*(P) = *c*. Each diagonal line represents a path obtained by varying the positive multipliers namely d1 and d2 in equation 1. As soon as the line hits a path, the algorithm returns this path as the shortest path with respect to equation 1. If the path

found is within the dashed rectangle then it is called as feasible path as it satisfies the constraints. Jaffe observed that the shortest path based on linear combination of weights does not necessarily reside within the constraints, which are illustrated, in figure 2.1.

2.1.3 Iwata's Algorithm

Iwata et al in [8] chose a different approach in solving the MCP problem in polynomial time. The algorithm involves a two-step approach. In the first step the shortest path is computed based on one QoS metric. In the second step a check is performed over the computed shortest path to determine if it meets all the other QoS metrics. If the computed shortest path does not meet one metric, the procedure is repeated with a different QoS metric until a feasible path is found or all the QoS metrics are explored. The drawback with this algorithm is that there is no guarantee that any of the shortest paths w.r.t each individual measure is close to a path within the constraints.

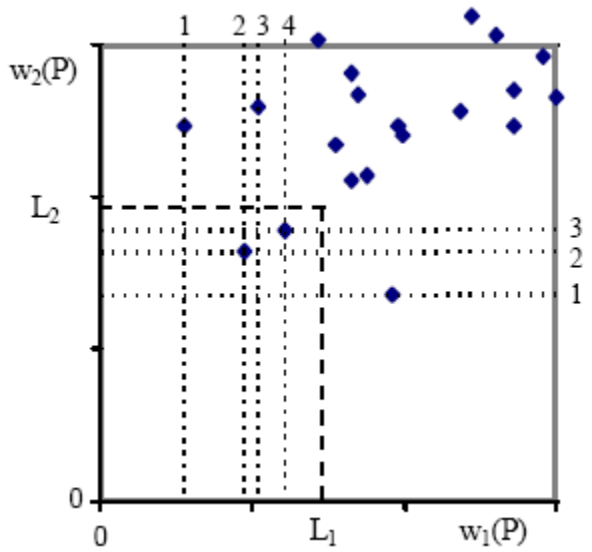

Figure 2.2: Simulation results of Iwata's algorithm as presented in [7]

Figure 2.2 is a reproduction of the original figure presented in the paper by Van Mieghem et al in [7]. Van Mieghem et al in [7] discuss the simulation results obtained by running Iwata's algorithm. Figure 2.2, which shows the twenty shortest paths of a two-

constraint problem applied to a random graph with 100 nodes. Each path is represented as a dot and the co-ordinates of each dot are its path-length for each measure individually. Only the second and third shortest path for measure 1 and second and fourth shortest path for measure 2 lie within the constraints.

2.1.4 Chen's Algorithm

Chen et al in [9] presented a heuristic solution for MCP problem with a polynomial time complexity. Multi-constrained routing problem with two or more additive constraints is considered as NP-Complete [29,3]. Given a set of *m* metrics, Chen et al have shown that the MCP problem can be solved in polynomial time if all metrics or except one take bounded integer values, i.e., m-1 metrics should take bounded integer values. The idea is to change the link weights of m-1 metrics from real to integer as follows. Given a MCP Problem with two additive constraints *MCP (Graph, source, destination, weight constraint w1, weight constraint w2, constant c1, constant c2).* The idea is to simplify this MCP problem which is NP-Complete to a simpler one which can be solved in polynomial time either by an extended Dijkstra's algorithm or by Bellman-Ford algorithm where the simpler one is defined as *MCP (Graph, source, destination, weight constraint w1, new weight constraint w2', constant integer c1, integer x) where w2' (u, v) = (w2 (u, v) * x)/c2.*

The problem of delay-cost-constrained routing is a NP-Complete problem as delay and cost are both additive constraints. Their proposed algorithm for solving the delay-cost path problem *MCP (Graph, source, destination, delay, cost, constant D, constant C)* converts the original MCP problem into two simpler ones; namely *MCP (Graph, source, destination, new delay, cost, integer x, Constant C)* and *MCP (Graph,*

source, destination, delay, new cost, Constant D, integer x). A solution to either simpler ones is a solution to the original MCP problem. The value of x determines the performance and the overhead of the algorithm. This algorithm can be extended to multiple constraints.

2.1.5 Korkmaz and Krunz Randomized Algorithm

Korkmaz et al in [10] proposed a randomized heuristic for the MCP problem. The key behind the randomized search is to make random decisions during the course of execution of an algorithm so that unpredicted traps or pitfalls could be avoided during the course of search for a feasible path. The algorithm consists of two phases namely an initialization phase and a randomized search phase. In the initialization phase the algorithm computes the shortest path from every node u to destination d with respect to each metric. This information will provide lower bounds for the path weight vectors of the paths from u to d. The information obtained from the initialization phase is examined to determine whether there is a chance of finding the feasible path or not. The algorithm terminates if a feasible path does not exist. If a feasible path might exist it starts its randomized search phase. The algorithm starts the randomized breadth first search from the source node s and discovers nodes from which there is a good chance to reach the destination d. The effect of this algorithm when executed multiple times between a given source and given destination may return different paths. Randomized BFS uses the information obtained in the initialization phase and checks whether this chance exists before discovering a node or not. If there is no chance of reaching the destination, the algorithm stops exploring such nodes.

The authors try to improve the randomized search phase, by selecting a path with minimum hop count among the available feasible paths. This improved scheme called as MinHop-MC results is efficient utilization of network resources. Their proposed MinHop-MC tries to find a path *p* from source node *s* to destination node *d* such that the number of nodes in p is minimum, while satisfying the bounds imposed on the multiple path constraints.

Randomized algorithm may return different paths between the same source and destination pair in order to provide load balancing when executed multiple times under the same network conditions. The paths can be cached and can be returned to the application if requested repeatedly. *The heuristic algorithm presented in chapter 3 makes use of the MinHop concept.*

2.1.6 Ma-Steenkiste Algorithm

Ma and Steenkiste in [5] observed two basic assumptions in the existing study of QoS routing algorithm. First that multiple QoS constraints are unrelated and hence need to be constrained independently and second that delay and delay jitter of a link are known *a priori.*

They took a different approach by observing constraints are not independent and the delay, delay-jitter of a link are not known *a priori.* Propagation delay, transmission delay and queuing delay constitute link delay. Propagation delay and transmission delay are related to link distance and capacity and hence can be known or determined in advance. Queuing delay is determined by the service disciplines deployed in the network, amount of bandwidth reserved for the flow as well as the burstiness of the traffic sources.

Therefore, scheduling algorithms deployed in the network and traffic source specification must be taken into account while making a routing decision.

Building on the work of Zhang [23], Ma and Steenkiste observed that end-to-end delay, delay-jitter and buffer space constraints are determined by bandwidth allocation when rate proportional service disciplines such as Weighted Fair Queuing are used. Rate-proportional service disciplines provide a guaranteed rate for each flow and mathematically proven bounds exist for delay, delay-jitter and buffer space if the traffic source confines to the traffic specification governed by the leaky bucket.

Therefore, with a rate-proportional service discipline deployed in the network, by taking into consideration that QoS constraints are function of bandwidth and not independent and queuing delay is determined by taking into consideration the service discipline and traffic specifications, Ma and Steenkiste have proved that MCP problem becomes tractable and can be solved in polynomial time using an iterative Bellman-Ford algorithm. Using an iterative Bellman-Ford algorithm with hop-count constraints, a feasible path with delay, delay-jitter, buffer space and bandwidth constraints can be found if one such exists. *This thesis favors the approach chosen by Ma and Steenkiste in solving MCP problem and adapts this solution in the heuristic algorithm presented in chapter 3.*

2.2 Disjoint Paths

Multiple disjoint paths can increase *reliable delivery of data; provide resilience to failure; reduce congestion in the network; reduce the probability of dropped packets and helps in load balancing in a network.* Reliable delivery of data can be achieved by sending the information between a given source and destination along the multiple disjoint paths. Load balancing in a network can be achieved by splitting the traffic

between a given source and destination across multiple disjoint paths. Load balancing reduces congestion in the network and also reduces the probability of dropped packets. A pair of disjoint paths between a given source and destination provide resilience to failure by considering first path as primary path and the second path as restoration path. In the event of a failure routing algorithm has the ability to switch promptly from one path to another.

Disjoint paths may be either *link-disjoint* or node-*disjoint*. A pair of paths is link-disjoint if they have no links in common. A pair of paths is node-disjoint if they have no common nodes. Node-disjoint paths provide reliability than to link-disjoint paths.

Definition of the basic Node-Disjoint paths problem

Consider a network that is represented by a directed graph, G=(V, E), where V is a set of vertices and E is a set of edges. Each link (u, v) is represented by two non-negative measures namely: a delay δ(u, v) and a residual bandwidth R(u, v).Given a source *s*, destination *d* and delay bound ***D***; find a pair of disjoint paths namely p′ and p″, such that p′ and p″ do not share any common nodes between them and path length of p′ and p″ is less than or equal to the delay bound ***D***. If such a pair of paths exists then they are called as feasible paths.

2.2.1 Suurballe's Algorithm

Suurballe's algorithm [12] finds the shortest pairs of disjoint paths and attempts to solve the k-disjoint paths. Suuraballe's algorithm requires a graph transformation before applying Dijkstra's shortest path algorithm in a modified graph. This transformation ensures that the modified graph contains no negative arcs as Dijkstra algorithm works

only for graphs with non-negative arcs. This algorithm when executed over k iterations discovers k node-disjoint paths with minimum length. The time taken by the Suurballe's algorithm in computing disjoint paths is high as the shortest paths from the source vertex to all other vertices in the given graph needs to be determined first and new lengths of the arcs for the transformed graph needs to be computed. The algorithm involves two runs of Dijkstra algorithm and graph transformations. The graph transformation is done prior to each run of Dijkstra algorithm. Suurballe's graph transformation ensures that the resultant modified graph is non-negative, facilitating thereby the use of Dijkstra algorithm to find the shortest paths.

2.2.2 Sidhu's Algorithm

Sidhu et al in [11] use a different way to find disjoint paths in network. Their algorithm does not perform any graph transformation. They presented a distributed distance-vector algorithm for finding multiple disjoint paths by assuming that each node knows its preferred neighbor and its up-tree neighbors. The distributed algorithm constructs a set, S_x, of minimum node disjoint paths from every node x to a destination node and retains the shortest path in set S_x. Disjoint paths are constructed by the exchange of two types of messages between the nodes namely *PID* and *ALT* over the links of the graph G. PID messages propagate information about shortest path costs, path identifiers and branch identification lists to all nodes. ALT messages propagate information about the alternate paths through down-tree, up-tree or horizontal neighbors. The labeling scheme within the distributed algorithm labels each node with an identifier called path identifier which helps in identifying disjoint paths. This algorithm includes shortest path

as one of the disjoint paths and introduces overhead due to exchange of messages and routing table for large-scale networks.

2.2.3 Bhandari's Algorithm

Bhandari et al in [18, 22, 30] presented an algorithm to find a shortest pair of disjoint paths. The algorithm involves two runs of a shortest path algorithm such as modified Dijkstra or the BFS algorithm and uses a graph transformation on the first shortest path to engender disjointness. Dijkstra's algorithm can be applied only to non-negative graphs with no cycles. After the graph transformation the resultant graph becomes a graph with negative weights but no cycles. Bhandari extended the Dijkstra algorithm to modified Dijkstra algorithm, which can be applied to graphs with negative weights but no negative cycles. The graph transformation varies depending on whether link disjoint or node disjoint paths are needed. In order to achieve link disjoint paths the graph transformation includes replacing each edge of the shortest path by a single arc directed towards the source vertex and by making the length of each of the above arcs negative. In order to achieve node disjoint paths the graph transformation includes splitting of vertices on the first shortest path. The difference between Suurballe's algorithm Bhandari's algorithms is in the way graph transformation is done. Suurballe performs a graph transformation, which ensures that the resultant graph is non-negative so that Dijkstra algorithm can be applied over the resultant graph to get shortest pair of paths. Bhandari's graph transformation results with a negative graph but no cycles. As a result of which Bhandari uses modified Dijkstra algorithm over the resultant modified graph to get shortest pair of paths.

2.3 Multi-constrained Node-disjoint paths

Multi-constrained Node-disjoint paths find significance in delivering real time information to geographically distributed critical infrastructure applications; survivable design of telecommunication networks and in layout design of VLSI integrated chips. The main aim is to find multiple node-disjoint paths, which satisfy the multiple independent QoS constraints. Most of the research is currently focused towards solving MCP problem or finding multiple disjoint paths independently. Disjointness and QoS guarantees need to be addressed together in multi-constrained node-disjoint paths algorithm in order to ensure reliability and timeliness.

Multi-constrained Node-disjoint path Definition

Consider a network that is represented by a directed graph, G= (V, E). Each link (u, v) in E is specified by a link weight vector with as components m additive QoS weights $w_i(u, v) \geq 0, i=1,\ldots..., m.$ Given m constraints, $L_i\ i=1,\ldots..., m,$ the problem is to find a pair of disjoint paths namely P′ and P″, from the source node *s* to a destination node *d* such that $w_i(P') = \sum_{(u, v)\ in\ P'} w_i(u, v) \leq L_i$ for i = 1,……,m and $w_i(P'') = \sum_{(u, v)\ in\ P''} w_i(u, v) \leq L_i$ for i = 1,……,m , where P′ and P″ do not share any common nodes between them. Pair of paths that satisfies all m constraints is referred to as a feasible path pair.

2.3.1 DIMCRA

Guo et al in [14] explored in the area of multi-constrained, multiple disjoint paths. They propose a heuristic algorithm called DIMCRA, which finds a link-disjoint multiple constrained paths between a given source and destination nodes. DIMCRA is based on multi-constrained routing algorithm called SIMCRA proposed by Van Mieghem et al in [15]. SIMCRA is based on three concepts namely nonlinear path length, k-shortest path

routing and non-dominance. The link-disjoint pair of paths returned from DIMCRA always obeys the multiple constraints. The authors say that the pair of link-disjoint paths returned from DIMCRA need not be necessarily optimal in terms of minimizing the total length of the returned paths. DIMCRA does not always guarantee to find a pair of feasible paths. The authors claim DIMCRA performs better than simple remove-find method but the authors do not present the simulation or experimental results to justify this claim. The authors do not present the performance and results of DIMCRA over deployed communication networks. *This thesis presents a heuristic algorithm for solving multi-constrained node-disjoint path problem in chapter 3.*

2.3.2 Orda and Sprintson Algorithm

Orda and Sprintson in [16] took a totally different approach and presented a comprehensive analysis of the problem by using the framework of network flows. The authors have studied the fundamental problem of finding two disjoint paths that satisfy the QoS constraints at minimum cost and presented approximation algorithms with provable performance guarantees. The authors also show that any polynomial algorithm used to solve this problem violates the delay constraint by a factor of two. They propose four approximation algorithms out of which the first algorithm tries to identify a solution that violates the delay constraints by a factor of 1.5 and 3 for primary path and restoration path respectively. The other three algorithms reduce the delay violation along the primary and restoration path at the expense of higher cost and computational complexity. The cycle cancellation approach method proposed in the paper could be used to solve h-disjoint path problems for any h>2.

Chapter 3: Foundation for Heuristic MCMP Algorithm

Section 3.1 lists the assumptions made in this thesis. Section 3.2 discusses the shortest path algorithms and its shortcomings. The shortest path algorithms dealt in this section includes Dijkstra, Modified-Dijkstra and BFS algorithm. Section 3.3 discusses the multi-constrained single path QoS algorithms. Subsection 3.3.1 talks about the key factors behind choosing a feasible path. Subsection 3.3.2 talks about selecting efficient feasible paths. Section 3.4 presents multiple disjoint algorithms in detail. Subsection 3.4.1 presents Remove-Find algorithm, which is commonly used to find node-disjoint paths. Subsection 3.4.2 discusses in detail the vertex splitting based node-disjoint shortest pair algorithm. Subsection 3.4.3 highlights the need for splitting vertices with degree greater than 4.

3.1 ASSUMPTIONS

The following assumptions are made throughout the thesis.

- Since most of the communication networks are bi-directional in nature the given graph is undirected graph. Consider a network that is represented by a graph, G= (V, E), is a set of V vertices and a set of E Edges. The given undirected graph can be represented as a directed graph by replacing each edge with a pair of oppositely directed arcs of length equal to the edge length
- The given graph is devoid of any loops and multiple edges. Loops and multiple edges can be eliminated by insertion of dummy vertices with an edge length of zero between the dummy vertex and the non-dummy vertex.
- The given graph is devoid of negative arcs.

- There exists a path for every pair of vertices in the given graph. Such a graph is said to be connected graph.
- During disjoint path construction the given non-negative graph gets converted into a graph with negative arcs but no negative cycles.

3. 2 Shortest Path Algorithms

Dijkstra [17] in 1959 first presented an algorithm for finding the shortest path between a given pair of vertices. Dijkstra's shortest path Algorithm is valid for nonnegative graphs and has a complexity of O ($|V|^2$). During the computation of disjoint pair of paths the given nonnegative graph gets converted into a graph with negative arcs that cannot be handled by Dijkstra shortest path algorithm. Bhandari in [18] presented a modified version of Dijkstra shortest path algorithm for disjoint path computation. This modified version of Dijkstra algorithm, called the ***modified Dijkstra algorithm*** is valid for nonnegative graphs with no negative cycles. ***Breadth First Search (BFS)*** method [19,20] can be used as an alternative to the modified Dijkstra algorithm. In BFS the search for the shortest path proceeds from all the vertices relabeled in the previous iteration while the search for the shortest path in modified Dijkstra algorithm proceeds from a single "permanently" labeled vertex.

3.2.1 Dijkstra Shortest path Algorithm

Consider a network that is represented by a graph, G= (V, E), is a set of V vertices and set of E Edges. Let *w(u, v)* denote the non-negative edge weight between node *u* and node *v*. Let the source vertex be denoted by *source* and the destination vertex be denoted by *dest*. Let d[i] denote the distance of vertex i from source vertex A. Let P[i] denote its predecessor. The implementation assumes that the graph G is represented by adjacency

lists which is denoted by *Adj*. Dijkstra algorithm uses greedy approach in choosing the optimal node at each step. The algorithm is in figure 3.1 [17, 21, 22].

```
Dijkstra (GRAPH G, NODE source, NODE dest)
Begin
        Initialize-Single-Source (G, source)
        S ← ∅
        Q ← V- {source}
        while Q is not an empty set do
           u ← Extract_Min (Q)
          S ← S ∪ u
          If u = dest then
             break
          Else
              for each vertex v ∈ Adj [u] and v ∈ Q do
                  Relax (u, v, w)
End

Initialize-Single-Source (GRAPH G, NODE source)
Begin
        for each vertex v ∈ V [G] do
            d[v]← ∞
            P[v]←source
        for each vertex v ∈ Adj[source] do
            d[v]← l(source, v)   /* l(source, v) = the length of arc from vertex source to vertex v*/
        d[source]← 0
        P[source]←NONE
End

Relax (NODE u, NODE v, weight pair w)
Begin
       If d[v] > d[u] +w(u, v) then
          d[v]← d[u] +w(u, v)
          P[v]← u
End
```

Figure 3.1: Dijkstra Shortest Path Algorithm

3.2.2 Modified Dijkstra Algorithm

Modified Dijkstra algorithm presented by Bhandari in [18] is a slight modification of the original Dijkstra algorithm. It is well known fact that the Dijkstra algorithm fails when the given graph contains negative arcs. In the construction of disjoint paths the given nonnegative graph gets modified into a graph with negative arcs, but no negative

cycle. The modified Dijkstra algorithm reduces to original Dijkstra algorithm if the graph is non-negative. The algorithm is in figure 3.2 [18].

```
Modified_Dijkstra (GRAPH G, NODE source, NODE dest)
Begin
        Initialize-Single-Source (G, source)
        S ← ∅
        Q ← V- {source}
        while Q is not an empty set do
            u ← Extract_Min (Q)
                S ← S ∪ u
            If u = dest then
                break
            Else
                for each vertex v ∈ Adj [u] do
                    Modified_Relax (u, v, w, Q)
End

Initialize-Single-Source (GRAPH G, NODE source)
Begin
    for each vertex v ∈ V [G] do
        d[v]← ∞
        P[v]←source
    for each vertex v ∈ Adj[source] do
        d[v]← l(source, v)    /* l(source, v) = the length of arc from vertex source to vertex*/.
    d[source]← 0
    P[source]←NONE
End

Modified_RELAX (NODE u, NODE v, weight pair w, Set Q)
Begin

    If d[v] > d[u] +w(u, v) then
        d[v]← d[u] +w(u, v)
        P[v]← u
        Q ← Q ∪ v
End
```

Figure 3.2: Modified Dijkstra Algorithm

Modified Dijkstra algorithm has a variant of *RELAX* function called *Modified_RELAX* function and the *for loop* which invokes the Modified_RELAX function has a different Boolean test condition. In Dijkstra algorithm only those neighbors belonging to Set Q are scanned, where as the modified Dijkstra algorithm scans all the neighbors of the vertex *u.* This extended scanning is necessary, since a previously "permanently" labeled vertex using EXTRACT_MIN function can be

relabeled, i.e., receive a lower label upon further scanning in graphs with negative arcs but no negative cycles. Furthermore, the modified Dijkstra algorithm, via $Q \leftarrow Q \cup v$, ensures the reentry into set Q of any previously "permanently" labeled vertex that was relabeled in Modified_RELAX function.

The fact that all the neighbors of the selected vertex (which include previously permanently labeled vertices) are scanned implies redundancy, since rescanning a previously permanently labeled vertex does not alter its label. However, in the construction of disjoint paths a given nonnegative graphs gets modified such that a part of it becomes negative. In such a situation, rescanning can update the label of a previously selected or "permanently" labeled vertex. This updated label is placed back in the set Q in Modified_RELAX function.

3.2.3 Breadth First Search Algorithm

BFS [19,20] is an alternative algorithm to the modified Dijkstra algorithm. It finds the shortest path in a graph with negative arcs but no negative cycles. BFS computes a path from given source *s* to the given destination *d* using fewest number of edges. Since BFS tries to find a path with fewest numbers of edges, it helps in efficient utilization of network resources. BFS search proceeds in each iteration from all the vertices labeled in the previous iteration whereas modified Dijkstra algorithm search proceeds from a single permanently labeled vertex.

BFS algorithm executes until the set Q is empty. BFS finds path in graphs with negative arcs but no negative cycles. If more than one shortest path exists, BFS finds the shortest path with least number of hops. Lesser number of hops means less consumption of network resources. In QoS routing it would be worthwhile to efficiently utilize the

network resources so that more QoS subscriptions can be handled by the network. Since efficient utilization of network resources can be achieved by using BFS algorithm we propose a modified BFS algorithm to handle QoS. The algorithm is in figure3.3 [22, 18].

```
BFS (GRAPH G, NODE source, NODE dest)
Begin
        Initialize-Single-Source (G, source)
        Q ← {source}
        while Q is not an empty set do
              S ← ∅
             u ← DELETE_FRONT (Q)
             for each vertex v ∈ Adj [u] do
                  Relax (u, v, w, dest, S)
        Q ← S – (S ∩ dest)
 End

Initialize-Single-Source (GRAPH G, NODE source)
Begin
        for each vertex v ∈ V [G] do
            d[v]← ∞
            P[v]←source
        d[source]← 0
        P[source]←NOONE
End

Relax (NODE u, NODE v, weight pair w, NODE dest, Set S)
Begin

        If  d[v] > d[u] +w(u, v)  and  d[dest] > d[u] +w(u, v) then
            d[v]← d[u] +w(u, v)
            P[v]← u
            S ← S ∪  v
End
```

Figure 3.3: BFS ALGORITHM

3. 3 Multi-Constrained Single Path QoS Algorithms

Building on the work of Zhang [23], Ma and Steenkiste observed that end-to-end delay, delay-jitter and buffer space constraints are determined by bandwidth allocation when rate proportional service disciplines such as Weighted Fair Queuing are used. They also showed that with WFQ scheduling algorithm being deployed over the network, finding a path that satisfies end-to-end delay, delay-jitter and buffer space constraints is solvable in polynomial time when the relationship between constraints in taken into account. We adapt the results proved by Ma and Steenkiste in our heuristic algorithm.

Examples of rate proportional service disciplines include Virtual Clock [24], Worst-Case Weighted Fair Queuing [25], WFQ [26] and Self Clocked Fair Queuing [27]. The rate proportional service disciplines based scheduling algorithms ensures a guaranteed share of link resources by isolating each guaranteed session from other sessions [5]. Bandwidth being reserved and burstiness of the traffic source are the two factors that determine the end-to-end queuing delay of the flow.

For the above mentioned rate proportional service disciplines, if the traffic source is constrained by a leaky bucket < σ, b >, where σ is the average token rate and b is the maximal burst rate in bytes (token bucket size) then for a path **p** with *n* hops and link capacity C_i at hop i, the *provable end-to-end delay* as given by Zhang [23,28] is:

$$D(p, r, b) = \frac{b}{r} + \frac{n * \mathrm{Lmax}}{r} + \sum_{i=1}^{n} \frac{L\max}{Ci} + \sum_{i=1}^{n} propi \qquad (3.1)$$

Where L_{max} = Maximum packet size in the network

$prop_i$ = Propagation delay of link *i*.

$(r \geq \sigma)$ = Bandwidth reserved.

The end-to-end delay-jitter is given by

$$J(p, r, b) = \frac{b}{r} + \frac{n * Lmax}{r} \quad (3.2)$$

The buffer space requirement at the h-th hop is given by

$$B(p, r, b) = b + h * L_{max} \quad (3.3)$$

A path is said to be *feasible path* for traffic with delay guarantees if it meets the delay, delay-jitter and buffer space requirements as given by equation 3.1, 3.2 and 3.3 respectively. We consider two cases where the bandwidth to be reserved (r) is known *a priori.*

3.3.1 Selecting Feasible Paths

Problem Definition: Consider a network that is represented by a directed graph, G=(V, E), where V is a set of vertices and E is a set of edges. Each link (u, v) is represented by two non-negative measures namely: a delay δ(u, v) and a residual bandwidth R(u, v). Given a delay bound ***d***, a delay-jitter bound ***j***, and buffer space bound b_m for all ***m*** nodes in the network, a leaky bucket $< \sigma, b >$, where σ is the average token rate and b is the maximal burst rate in bytes (token bucket size) and a bandwidth to reserve ***r*** ($r \geq \sigma$) to reserve, find a path ***p*** such that $D(p, r, b) \leq d$, $J(p, r, b) \leq j$, $B(p, r, h) \leq b_h$ for all nodes along the path, where b_h is the buffer space constraint for the node with *h* hops from the source and $r \leq R_j$ ($\forall R_j \in p$) where R_j is the link residual bandwidth.

Proposition: QoS routing problem of finding a path with delay, delay-jitter, buffer space and bandwidth can be solved by modified Dijkstra algorithm or modified BFS algorithm in polynomial time if the bandwidth to reserve is known *a priori*.

Proof: Since we have assumed that the bandwidth to be reserved is known a priori, Ma and Steenkiste have proposed an additive distance function by using the link cost *l*(i) and length function *l*(P) where,

$$l(\mathrm{i}) = \frac{\mathrm{L_{max}}}{r} + \frac{\mathrm{L_{max}}}{\mathrm{Ci}} + \mathrm{prop_i} \quad \& \quad l(\mathrm{p}) = \frac{b}{r} + \sum_{j \in P} l(j) \qquad (3.4)$$

The goal now is to find a path ***p,*** such that *l(p)* < ***d*** using modified Dijkstra algorithm or modified BFS algorithm. If *l(p)* > ***d*** then there is no path that meets the given delay bound ***d.*** The delay-jitter bound as defined in equation 3.2 is given by J (p, r, b) = $\frac{b}{r}$ + $\frac{n * \mathrm{L_{max}}}{r}$. In equation 3.2 the burst rate in bytes (b), bandwidth to reserve(r), max packet size in the network $\mathrm{L_{max}}$ are constant parameters and only the hop count is a variable parameter. So the delay-jitter bound is met or not is determined by the hop count (n) iff

$$\mathrm{n} \leq \lfloor \frac{(r * J - b)}{L_{\max}} \rfloor \qquad (3.5)$$

The delay-jitter bound for any path can be satisfied as long as the hop count is less than or equal to the value defined by equation 3.5. The buffer space requirement at the h-th hop as defined in equation 3.3 is given by B (p, r, b) = b + h ∗ $\mathrm{L_{max}}$. In equation 3.3 the burst rate in bytes (b), max packet size in the network. $\mathrm{L_{max}}$ are constant parameters and only the hop count h, is a variable parameter. For each node *u* in the network, the buffer space bound $\mathrm{B_u}$ defines a bound on the hop count $\mathrm{N_u}$ as defined in equation 3.6. During step h, we consider only those nodes with $\mathrm{N_u} \geq$ h.

$$\mathrm{N_u} \leq \lfloor \frac{(B_{\mathrm{u}} - b)}{L_{\max}} \rfloor \qquad (3.6)$$

A path ***p*** satisfying the delay, delay-jitter and buffer space requirement is said to exist, iff the length of path p as defined by equation 3.4 is less than or equal to delay bound ***d*** (i.e., $l(P) < d$) and as long as the hop count is no more than the value defined by equation 3.5 and each node along the path p meet the hop count bound as defined in equation 3.6. If such a path exists then the path ***p*** is said to be a feasible path.

3.3.2 Selecting Efficient Paths

Modified Dijkstra algorithm and modified BFS algorithm finds a feasible path if one such exists. However, there might be more than one feasible path available in the network. In order to efficiently utilize the network resources we should select the feasible path, which consumes less network resources among the available multiple feasible paths. Therefore we need some optimality criteria in choosing a feasible path among multiple feasible paths. The optimality criteria that can be considered are minimum hop count, minimum delay, minimum bandwidth and maximum bandwidth. A feasible path can be selected using one of the four optimality criteria mentioned above independently or using a combination of them with priorities. A few combinations of optimality criteria that can be used to achieve efficient utilization of network resources are *shortest-delay path, widest-shortest path, shortest-widest path and shortest-minimum-bandwidth path.* We present the shortest-delay path algorithm using modified Dijkstra algorithm in *appendix A*.

3. 4 Multiple Disjoint Paths

Problem Definition: Consider a network that is represented by a directed graph, G=(V, E), where V is a set of vertices and E is a set of edges. Each link (u, v) is

represented by two non-negative measures namely: a delay δ(u, v) and a residual bandwidth R(u, v).

Given a source *s*, destination *d* and delay bound ***D***; find a pair of disjoint paths namely p′ and p″, such that p′ and p″ do not share any common nodes between them and path length of p′ and p″ is less than or equal to the delay bound ***D***. If such a pair of paths exists then they are called as feasible paths. This problem definition can be extended to k-paths. One way to achieve node-disjoint paths is through graph transformations.

3.4.1 Remove-Find Method (RF)

RF method is a naive approach to achieve node-disjoint pair of paths. RF method takes a two-step approach in finding a pair of node-disjoint paths. The algorithm is in figure 3.4.

STEP 1: Find the shortest path p′ in the given graph between the given source and destination using any of the shortest path algorithms such that path length of p′ ≤ D, where D is the given delay bound.

STEP 2: Remove all edges incident on the nodes along the shortest path p′ except the source and destination. Removal of incident edges along the shortest path ensures that the second path will be node-disjoint. Run any of the shortest path algorithms over the modified graph between the given source and destination to find a shortest path p″, such that path length of p″ ≤ D, where D is the given delay bound.

Figure 3.4: RF Algorithm

Now the paths p′ and p″are said to be the shortest pair of node-disjoint paths. The above-mentioned RF algorithm sometimes fails to find a pair of node-disjoint paths when such paths actually exist. This drawback of RF algorithm is generally called as false alarms [22]. The RF algorithm fails to find a pair of node-disjoint paths when executed over the following network topology. The topology 3.4.1.1 is shown in figure 3.5.

TOPOLOGY 3.4.1.1

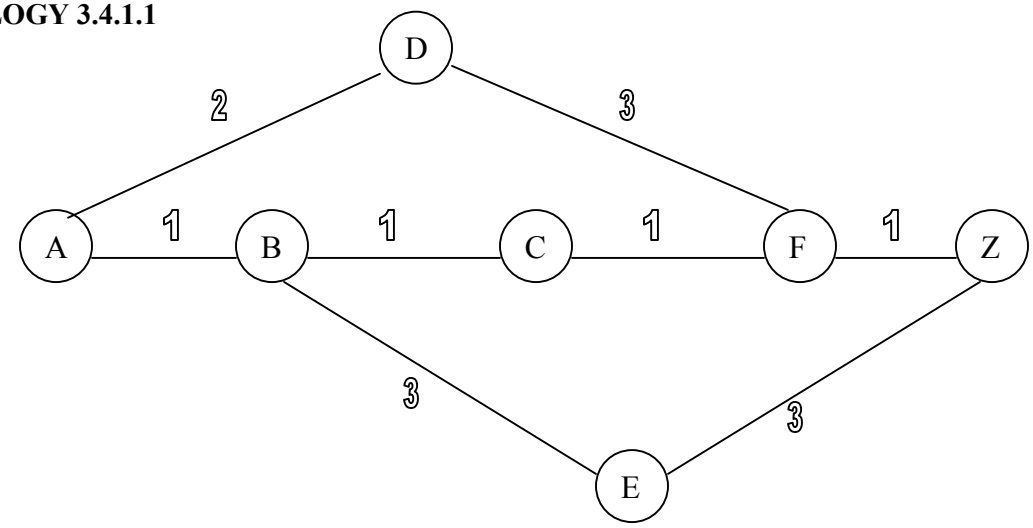

Figure 3.5: Topology 3.4.1.1

In the first step the shortest path found between source *A* and destination *Z* is ABCFZ of path length 4. As per the RF algorithm if we remove all the incident edges along the shortest path ABCFZ (except the source and destination) the resultant graph looks as shown below.

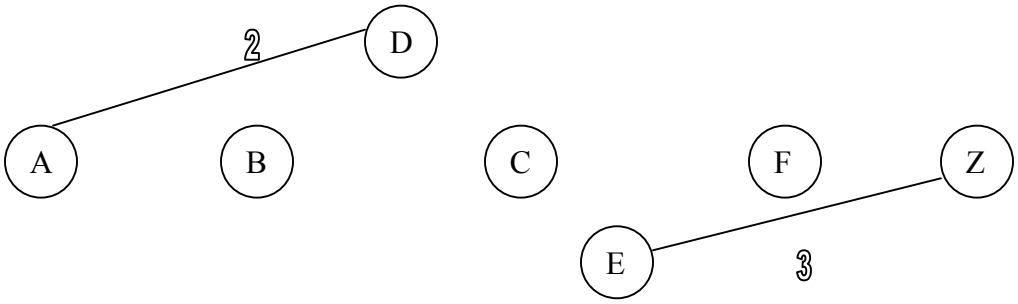

Figure 3.6: Execution of RF algorithm over Topology 3.4.1.1

The resultant graph becomes an unconnected graph and the shortest path algorithm cannot find the shortest path and hence the RF algorithm fails to find the shortest pair of node-disjoint paths when such paths actually exist. The resultant graph is shown in figure 3.6. In the above-mentioned topology a pair of node disjoint paths namely *ADFZ* of path length 6 and *ABEZ* of path length 7 exist between the source *A* and

destination *Z* that the RF algorithm failed to find. Hence RF algorithm is not a good choice in finding a pair of node-disjoint paths.

3.4.2 Vertex splitting based Node-Disjoint Shortest Pair Algorithm

This subsection deals with finding a shortest pair of node-disjoint paths over a given graph, G=(V, E), where V is a set of vertices and E is a set of edges. Vertex disjointness or node disjointness is achieved by splitting the vertices along the first shortest path. We adapt the *vertex splitting technique Method 2* as given by Bhandari in the book [22]. The algorithm is in figure 3.7.

STEP 1: Find the shortest path p′ in the given graph between the given source and destination such that path length of p′ ≤ D, where D is the given delay bound using the modified Dijkstra algorithm or BFS algorithm as given in section 3.2. If such a path doesn't exist then return failure.

STEP 2: Use the ***Check_and_SplitVertex algorithm*** to obtain the resultant modified graph G′. Based on the degree of nodes along the shortest path p′, apart from source and destination the need for vertex split is determined and the given graph G is modified to G′ accordingly.

STEP 3: Find the shortest path p″ in the modified graph G′ between the given source and destination using the modified Dijkstra algorithm or BFS algorithm such that path length of p″ ≤ D, where D is the given delay bound. If such a path doesn't exist then return failure.

STEP 4: Use the ***Collace_Interlace*** algorithm to obtain the shortest pair of node-disjoint paths namely P1 and P2.

Figure 3.7: Vertex splitting based node-disjoint algorithm

The Check_and_SplitVertex algorithm is in figure 3.8.

STEP 1: Check the degree of each node along the shortest path p′ except the source node and destination node.

STEP 2: If the degree of all the nodes along the shortest path p′ apart from the source node and destination node is less than 4 goto **STEP 4.**

STEP 3: If the degree of one of the nodes along the shortest path p′ apart from the source node and destination node is greater than 3 goto **STEP 5.**

STEP 4:

- **i)** Make a copy of given graph G and lets call it as G′.
- **ii)** Replace each edge along the shortest path p′ by a single arc directed towards the source vertex in the graph G′.
- **iii)** Set the length of the each arc in the graph G′, along the shortest path equal to $-l_i$, where l_i is the length of each edge in the original graph G.
- **iv)** Return G′(V, E′), where V is a set of vertices and E′is a new set of edges.

STEP 5:

- **i)** Make a copy of given graph G and lets call it as G′.
- **ii)** Replace each edge along the shortest path p′ by a single arc directed towards the destination vertex in the graph G′.
- **iii)** In graph G′, split the vertices along the shortest path p′ apart from the source vertex and the destination vertex into two collocated subvertices joined by an arc directed towards

Figure 3.8: Check and Split Algorithm

The Collace_Interlace algorithm is given in figure 3.9. The idea to use Collace_Interlace algorithm was originally given by Bhandari in [22]. We provide the implementation details in this thesis.

STEP 1: For path p″

i) Remove the zero length arcs.

ii) Coalesce the subvertices into their parent vertices and replace the single arcs of the shortest path p″ with their original edges as given in graph G. Lets call this new shortest path as P″.

STEP 2:

i) For each node along the paths p′ and P″ build a table with three entries namely node name, successor name and boolean_marked with value initially set to 0. For the destination node set the successor name as –1.

ii) For a given node say x, along the path p′ search for successor (x) say y, along the path P″. If y is found along the path P″, then check if x = successor (y). If x = successor (y), boolean_marked is set to 1 against the node x along path p′ and against the node y along path P″. This process is repeated for entire paths p′ and P″.

iii) For each node along the path p′, keep traversing along the path and appending the node to path P1 until a node with boolean_marked set to one is encountered. If such a node is encountered search for that node value in second path namely P″ and keep traversing along the path P″, appending each node to the path P1 until a node with boolean_marked set to one is encountered or until destination node is reached. This process of switching to other path is repeated whenever a node whose boolean_marked is set to one is encountered. The same set of rules is applied to path P″, appending the nodes to path P2.

iv) Return the node-disjoint pair of paths P1 and P2.

Consider the network topology 3.4.1.1. Let us find the shortest pair of path from source vertex A to destination vertex Z. Using modified Dijkstra algorithm the shortest path found between source vertex A and destination vertex is ABCFZ of path length 4. Topology 3.4.3.1 shows the edges along the shortest path being replaced by an arc directed towards the source vertex A. Topology 3.4.3.1 is given in figure 3.10.

Now let us start the search for the second path using modified Dijkstra algorithm from source vertex A. First, Node D gets permanently marked and node F gets permanently marked through node D. Node F is the first node encountered with degree equal to 3. The only node allowed for scanning from node F to engender node disjointness is node C. Thus vertex disjointness is ensured automatically if degree of node is equal to 3 and vertex splitting is not needed. Therefore node C gets permanently labeled through node F.

Topology 3.4.3.1

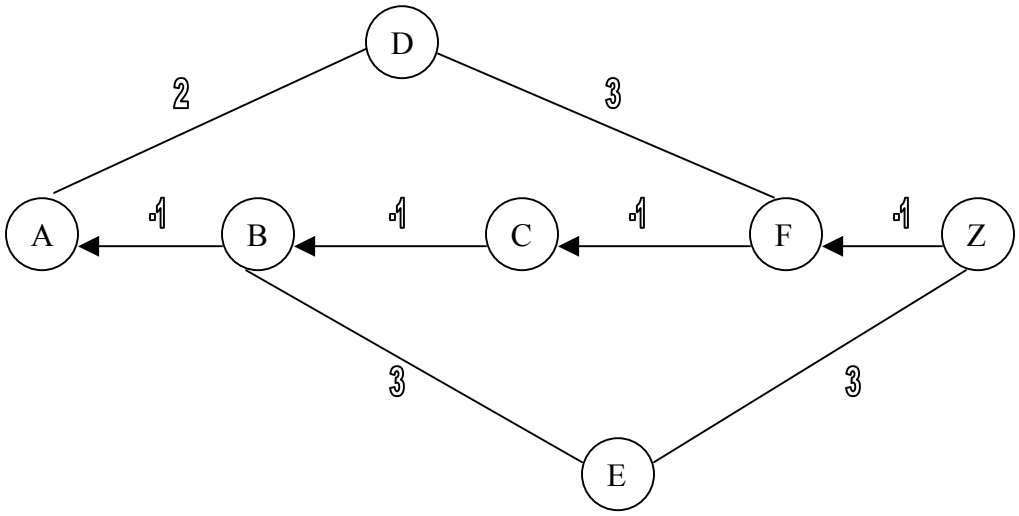

Figure 3.10: Topology 3.4.3.1

Node C has a degree of two with one incoming degree and one outgoing degree. So there is only one outgoing path. Therefore node B gets labeled through node C. As there are no external edges connected to node C, the only way node C can get marked is through node F and hence node-disjointness is ensured, as this node can exist only in one path.

Let us assume that there exists a link between node F and node E in the topology 3.4.1.1 as shown in figure 3.5 with link weight =3. Let us find the shortest pair of path from source vertex A to destination vertex Z. Using modified Dijkstra algorithm the shortest path found between source vertex A and destination vertex is ABCFZ of path length 4. Topology 3.4.3.2 shows the edges along the shortest path being replaced by an arc directed towards the source vertex A. Topology 3.4.3.2 is given in figure 3.11

Topology 3.4.3.2

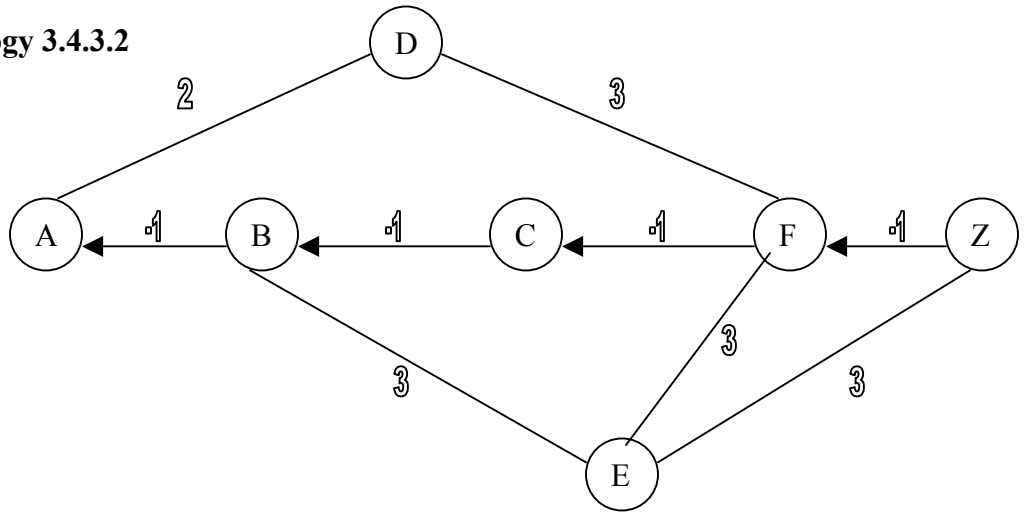

Figure 3.11: Topology 3.4.3.2

Let us start the search for the second path using modified Dijkstra algorithm from source vertex A. First, Node D gets permanently marked and node F gets permanently marked through node D. Node F is the first node encountered with degree greater than 3. Since the degree of node F is greater than 3 now node F can scan node E as well as node C and can choose either of them. If node E gets permanently marked from node F, then node F will be present in both the paths and it will be impossible to ensure node-disjointness. In order to ensure node disjointness the only node that should be reachable from node F should be node C.

Node-disjointness can be achieved by splitting all those nodes whose degree is greater than 3 along the shortest path apart from source and destination into two collocated subvertices joined by an arc of length zero directed towards the destination and replacing each external edge connected to a vertex along the shortest path by two component arcs, with one arc terminating on one subvertex and other arc emanating from the other subvertex such that along with the zero-length arc, a cycle is formed. Also, reverse the direction of arcs along the shortest path towards the source vertex.

Topology 3.4.2.3

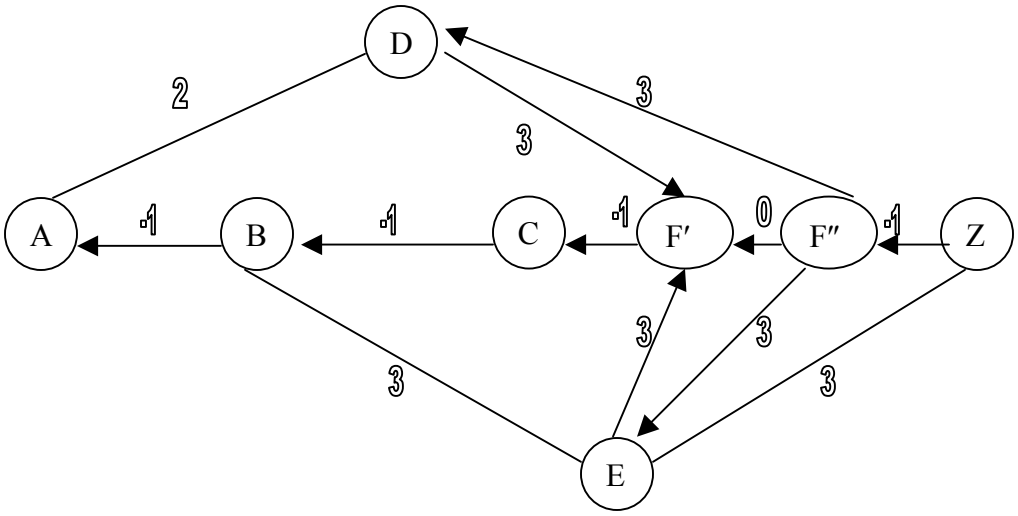

Figure 3.12: Topology 3.4.2.3

Applying the above set of rules to node F the resulting topology 3.4.2.3 is as shown in figure 3.12. For the above-mentioned topology the only node allowed for scanning from node F′ to engender node disjointness is node C. Thus vertex disjointness is achieved. Hence splitting of vertices with degree greater than 3 ensures vertex-disjointness. Finding node-disjoint pair of paths for topology 3.4.1.1 and its variants using vertex splitting based node-disjoint shortest pair algorithm is presented in *appendix A*.

Chapter 4: Heuristic MCMP Algorithm

Section 4.1 presents the heuristic multi-constrained node-disjoint QoS algorithm in detail. Section 4.2 highlights the need for choosing minimum delay path for the first path. Subsection 4.2.1 presents a proposition for existence/non-existence of disjoint pair of paths. Section 4.3 presents the shortest-delay path algorithm and section 4.4 presents widest-shortest path algorithm. Section 4.5 presents the effect of cycles caused by negative weights.

4.1 Heuristic Multi-constrained Node-disjoint QoS Algorithm

We presented multi-constrained single path QoS algorithms and vertex splitting based node-disjoint shortest pair algorithms in chapter 3. In this chapter we present a heuristic multi-constrained node-disjoint QoS algorithm which finds a pair of node-disjoint paths which meet the multiple QoS constraints namely end-to-end delay, delay jitter, buffer space and bandwidth.

Problem Definition: Consider a network that is represented by a directed graph, G=(V, E), where V is a set of vertices and E is a set of edges. Each link (u, v) is represented by two non-negative measures namely: a delay $\delta(u, v)$ and a residual bandwidth R(u, v).Given a delay bound $\boldsymbol{d}$, a delay-jitter bound $\boldsymbol{j}$, and buffer space bound $\boldsymbol{b_m}$ for all $\boldsymbol{m}$ nodes in the network, a leaky bucket $< \sigma, b >$, where σ is the average token rate and b is the maximal burst rate in bytes (token bucket size) and a bandwidth to reserve $\boldsymbol{r}$ ($r \geq \sigma$) to reserve, find a pair of node-disjoint paths namely $\boldsymbol{p'}$ and $\boldsymbol{p''}$ such that $D(p, r, b) \leq \boldsymbol{d}$, $J(p, r, b) \leq \boldsymbol{j}$, $B(p, r, h) \leq \boldsymbol{b_h}$ for all nodes along the path, where $\boldsymbol{b_h}$ is the buffer space constraint for the node with *h* hops from the source and $r \leq R_j$ ($\forall R_j \in p$) where R_j is the link residual bandwidth and $p = \boldsymbol{p'}$ in first iteration and $p = \boldsymbol{p''}$ in second iteration.

STEP 1: Find the shortest path p′ in terms of delay over the given graph G between source vertex A and destination vertex Z using the ***minimum-delay path algorithm*** as presented in section 4.3. The shortest delay algorithm is invoked by the following function call: ***Delay1=Minimum_Delay (G, A, Z, delay, bandwidth, jitter, buffer_space)*** where G is the given graph, A is source vertex, Z is destination vertex, delay is the given delay bound, bandwidth is the given bandwidth bound, jitter is the given end-to-end jitter bound, buffer_space is an array which has the given buffer space bound for each node. If Delay1 value is equal to INFINITY then stop the execution of the algorithm as the shortest path cant meet the given delay bound.

STEP 2: If Delay1 ≤ delay, find the deviation of Delay1 w.r.t delay. Calculate Deviation = delay - Delay1.

STEP3: Use the ***Check_and_SplitVertex algorithm*** to obtain the resultant modified graph G′. Based on the degree of nodes along the shortest path p′, apart from source and destination the need for vertex split is determined and the given graph G is modified to G′ accordingly.

STEP 4: Find the shortest path p″ in terms of fewest hops over the modified graph G′ between source vertex A and destination vertex Z using the ***widest-shortest path algorithm*** as presented in section 4.4. The widest shortest path algorithm is invoked by the following function call: **Boolean** ***Widest_shortest (G′, A, Z, delay, bandwidth, jitter, buffer_space, deviation)*** where G′ is the modified graph, A is source vertex, Z is destination vertex, delay is the given delay bound, bandwidth is the given bandwidth bound, jitter is the given end-to-end jitter bound, buffer_space is an array which has the given buffer space bound for each node and deviation is the difference between the given delay bound and Delay1.

a) If destination is reached and (delay [destination] ≤ delay) or (delay [destination] ≤ delay +deviation) invoke ***Check_feasibility algorithm***.
b) If Check_feasibility algorithm returns true, display message " *Disjoint Pair of paths meeting QoS exists*". Access the Global variable used to store ***P1*** and ***P2*** and display them as they have been already computed using ***Collace_Interlace algorithm*** in Check_feasibility algorithm. Commit paths ***P1*** and ***P2.*** Return *TRUE* from Widest-Shortest path algorithm.
c) If Check_feasibility algorithm returns false or (delay [destination] > delay +deviation) continue processing until a destination node whose delay value is either (delay [destination] ≤ delay) or (delay [destination] ≤ delay +deviation) is reached *or* queue becomes empty. If such a destination node is found goto step a. If queue is empty and delay [destination] > delay, display message "*Disjoint Pair of paths meeting QoS do not exist*". Return *FALSE* from Widest-Shortest path algorithm.

Figure 4.1: Heuristic MCMP Algorithm

The main goal of this thesis is to find a pair of node-disjoint paths, which satisfy the multiple QoS constraints namely delay, delay-jitter, buffer space and bandwidth. The algorithm is given in figure 4.1

Time Complexity of heuristic MCMP Algorithm:

Step 1: Minimum-delay path algorithm is an extension to modified Dijkstra algorithm to handle QoS. For a graph with no negative arcs modified Dijkstra algorithm performs the same as Dijkstra algorithm. Minimum-delay path algorithm is implemented using an adjacency list and has a running time of $O((|V| + |E|) * m)$, where m is the different values of link residual bandwidth. In the worst case m is equal to E and the running time will be $O((|V| + |E|) * |E|)$.

Step 2: The running time of Check_and_Split algorithm is $O(p * e)$, where p is the path length of the minimum-delay path and e is the number of edges incident on these paths. In the worst case p is equal to |V| , total number of vertices and e is equal to E and the running time will be $O(|V| + |E|)$.

Step 3: Widest-shortest path algorithm is an extension to BFS algorithm to handle QoS. Widest-shortest path algorithm is implemented using an adjacency list and the nodes may be visited more than once. Thus, widest shortest path has a running time of $O((|V| * |E|) * m)$, where m is the different values of link residual bandwidth. In the worst case m is equal to E and the running time will be $O((|V| * |E|) * |E|)$.

4.2 Need for choosing the minimum delay path for the first path

Minimum delay is required for the first path both in order to avoid negative cycles in the modified graph and to allow combining of paths to work. Consider a network that is represented by a directed graph, G=(V, E), where V is a set of vertices and E is a set of

edges. Each link (u, v) is represented by two non-negative measures namely: a delay δ(u, v) and a residual bandwidth R(u, v). Let the delay bound that needs to be met be denoted by **D.**

Theorem: Combining Paths

Assume the min-delay path has delay *A* and a second path has been found in the modified graph whose delay *B* is less than the delay bound **D**. So we have A<=**D**, B<=**D**. When we combine them we get two new paths whose delays are A'=A+C and B'=B-C for some non-negative value C (since the sum of the path delays of the new paths is the same as the sum of the delays of the original paths). From the fact that ***A is the minimum delay*** we have that B'=B-C>=A. So A'=A+C<=B. But we already knew that B<=**D**, so A'<=**D** and clearly B'<=B<=**D**. Notice that the second path (length B) need not be the shortest path in the modified graph.

Theorem: Avoid Negative Cycles

Presence of negative cycles results in non-convergence of a search for the path by the shortest path algorithm and hence needs to be avoided. The proof has been given by Bhandari in [22].

4.2.1 Proposition for Existence/Non-existence of disjoint pair of paths

Paths in the modified graph with each length no more than **D** will combine with the min path to form a pair of disjoint paths each meeting the delay constraint. This has been proved in earlier section.

Proposition: If there are no path in the modified graph with length less than ***D****+(****D*** *– min path length) then disjoint paths meeting the delay constraints do not exist.*

Let the given delay bound be **D**. Assume the minimum delay path has delay A whose delay is less than the delay bound **D**. So we have A≤ D. Lets denote Deviation = difference between the delay bound D and the minimum delay path whose delay being A (Deviation = **D** - A). Let us assume that the second path found in the modified graph has delay B whose delay is greater than delay bound **D** as well as sum of delay bound and deviation.

So we have B > (**D** + Deviation)

⇒ B > (**D** + **D**-A)

⇒ B > (2**D**-A)

⇒ B +A > 2**D.**

Hence, disjoint pair of paths don't exist.

*Proposition: Paths in the modified graph with length between **D** and **D**+(**D**-min path length) may or may not meet the delay constraint when combined with the minimum-delay path.*

Proof by example: Consider the undirected graph, V={A, B, C, Z, E, F}, E={(A, B)[1], (B, C)[1], (C, Z)[1], (A, E)[2], (E, C)[2], (B, F)[2], (F, Z)[2]}, where the weights are in []. ***Let the delay bound be 5.***

The min-delay path is ABCZ with delay 3, which is less than given delay bound D. The path in the modified graph is AECBFZ with cost 7, which is greater than given delay bound D. Since the minimum delay path has less delay than the constraint we will add the difference to D in running the second step. In this instance D becomes original D + difference = 5+(5-3) = 7, the length of the path in the modified graph. After removing

the interlacing edges between the two paths the final disjoint paths are AECZ and ABFZ with delay 5.

However if we look at the slightly different original graph with edge weights {(A, E)[3]}, {(B, F)[1]} now we have a graph that does not have disjoint paths, each with delay less than 5. In the first pass the shortest path chosen is ABCZ with delay 3. The path in the modified graph is AECBFZ with cost 7, which is greater than given delay bound D. If we add the difference to D in running the second step, the delay bound for second step will be 5+(5-3) = 7, the length of the path in the modified graph. After removing the interlacing edges between the two paths the final disjoint paths are AECZ with delay 6 and ABFZ with delay 4.

*Hence paths in the modified graph with length between **D** and **D**+(**D**-min path length) may or may not meet the delay constraint when combined with the minimum-delay path.*

4.3 Minimum-Delay Path Algorithm

The algorithm is given in figure 4.2. We extend the modified Dijkstra algorithm proposed by Bhandari in [22] to find a minimum delay path meeting multiple QoS constraints. The algorithm begins by sorting all the link residual bandwidth in decreasing order and eliminating all the duplicates. In line 2 we delete links whose bandwidth capacity is less than the required bandwidth constraint. In line 3 the number of unique link residual bandwidth whose capacity is greater than the required bandwidth is computed. In Line 5 the hopcount bound for each node v as set by buffer space is determined. The maximum hop count can be no bigger than the max hops of the nodes with the biggest Max_hop count and the Hop_count_iterations is determined by choosing the

minimum of the Max_hop and number of nodes in graph G. In line 6 the initialization function in invoked where in, for each possible residual bandwidth the number of hops in the feasible path as determined by jitter is computed and the source is initialized. In line 7 all the nodes in the graph except the source is enqueued into queue Q. The following steps are repeated until the queue is empty or until a destination node is reached. The *Extract_Min* function extracts the node with the minimum path length. Lets call the extracted node as *u*. We iterate over hopcount and different values of link residual bandwidth. If the hopcount meets the bound set by the jitter, we explore those nodes (*v*) adjacent to node *u,* whose hopcount meets the hopcount bound set by buffer space by invoking the modified relax function. In the *modified relax function* we update node *v* only if the path length to reach node *v* from node *u* is less than its current path length. If node *v* is updated, it is enqueued into queue Q. Path length is returned if the path length to reach destination is less than or equal to delay bound. An error message is returned if the queue becomes empty and/or the path length to reach destination is greater than the delay bound.

4.4 Widest-Shortest Path Algorithm

The algorithm is given in figure 4.3. We extend the BFS algorithm to find a path with fewest numbers of hops satisfying multiple QoS constraints. A feasible path with fewest numbers of hops consumes less network resources. These results in an efficient utilization of network resources and more subscriptions can be committed. The shortest – delay algorithm finds a path with minimum delay satisfying multiple QoS constraints. In widest-shortest path algorithm every time the destination node is reached, a feasibility check is performed *w.r.t* the first path computed using shortest-delay path algorithm to

see if disjoint pair of paths exists. If disjoint pair of paths exists the algorithm commits those paths in the graph *G* and returns *true*. If the feasibility check fails the algorithm continues processing until the queue becomes empty or disjoint pair of paths is found. The algorithm returns *false* if the queue becomes empty and/or path length to reach the destination is either greater than delay or (delay +deviation).

4.5 Effect of Cycles Caused By Negative Link Weights

Vertex splitting based node-disjoint shortest pair algorithm presented in chapter 3 requires that the modified graph be devoid of negative cycles in order to find disjoint pair of paths. The graph transformation is done by *check and split vertex algorithm*. In order to avoid negative cycles the shortest path having minimum delay needs to be chosen as the first path. The proof has been given by Bhandari in [22]. Because of which, the MCMP algorithm presented earlier also tries to find the minimum delay path meeting multiple QoS constraints in step 1. If we just consider *delay* as the only constraint, then the shortest path with minimum delay might not be the same as minimum delay path meeting multiple constraints chosen by MCMP algorithm in step 1. As a result of which the modified graph might have negative cycles.

In MCMP algorithm each node carries the path to reach it. The node under consideration gets updated only if the new delay to reach the node is less than the earlier delay and if the node under consideration is not already in the path. If the node under consideration is already in the path, it means that a loop is about to occur and hence the node is not updated. In this way negative cycles are avoided. In our experience running MCMP algorithm we have never noticed existence of negative cycles.

```
Mnimum_Delay (GRAPH G, Node source, Node dest, int delay, int bandwidth, real jitter, b_space: Node->int)
Begin
    1.  Unique_resbw ←Sort _ and _Eliminate_duplicates (list_resbw)
    2.  Link_resbw ← Delete (Unique_resbw)
    3.  Count_unique_bw ← | Link_resbw |
    4.  Max_hop = NULL;
    5.  for  v ∈ V do
                BF_v= ⌊(b_space (v) – burst_rate) /L_max⌋
                 Max_hop = max (Max_hop, BF_v )
                 Hop_count_iterations ← min (Max_hop, |V|)
    6.  Initialize-Single-Source (G, source, Count_unique_bw, Link_resbw, jitter)
    7.  S ← ∅
    8.  Q ← V- {source}
    9.   while Q is not an empty set do
            for each k ← 1 to Count_unique_bw do
            u ← Extract_Min (Q)
            S ← S ∪ u
            h ← hop count (u) +1
          If  u = dest then
                 Break.
           Else
                 If (JIT_k ≥ h) then
                        for each vertex v ∈ Adj [u] do
                            If  (h ≤ BF_v )then
                                Modified_RELAX (u, v, w_k, Q)
    10. If d_k[dest] ≤ delay then
                 Print, "Path meets delay, delay-jitter, bandwidth, buffer space "
                 return d_k[dest]
    11. Else
                 Print, "Path doesn't meet QoS"
                 return INFINITY
End

Modified_RELAX (Node u, Node v, weight pair w, Set Q)
Begin
    1.  If  d_k[v] > d_k [u] + w_k (u, v) then
          d_k [v]← d_k [u] + w_k (u, v)
          P_k [v]← u
          Path_vector_k[v] ← Path_vector_k [u]
          Path_vector_k [v] ← Append (u)
          Q ← Q ∪ v
End

Initialize-Single-Source (GRAPH G, Node source, int Count_unique_bw, int Link_resbw[ ], real jitter)
Begin
    1.   for k ← 1 to Count_unique_bw do
           JIT_k ←⌊(Link_resbw[k] * jitter- burst_rate)/ L_max⌋
    2.    for each vertex v ∈ V [G] do
                 d_k [v]← ∞
                 P_k [v]←source
                  Path_vector_k [v] ← NULL
    3.    for each vertex v ∈ Adj[source] do
                 d_k [v]← l(source, v),
                 Path_vector_k [v] ← source
    4.    d_k [source]← 0
    5.    P_k [source]←NONE

End
```

Figure 4.2:Minimum-Delay Path Algorithm

```
Boolean Widest_shortest (GRAPH G, Node source, Node dest, int delay, int bandwidth, real jitter, b_space: Node->int)
Begin
    1.  Unique_resbw ←Eliminate_duplicates (list_resbw)
    2.  Link_resbw ←Sort_and_Delete (Unique_resbw)
    3.  Count_unique_bw ← | Link_resbw |
    4.  Max_hop = NULL;
    5.  for  v ∈ V do
                BF_v = ⌊(b_space (v) – burst_rate) /L_max⌋
                Max_hop = max (Max_hop, BF_v )
                Hop_count_iterations ← min (Max_hop, |V|)
    6.  Initialize-Single-Source (G, source, Count_unique_bw, Link_resbw, jitter)
    7.  Q ← {source}
    8.   while Q is not an empty set do
            for each k ← 1 to Count_unique_bw do
            u ← Dequeue (Q)
            If ( u = dest)  and ((d_k[dest] ≤ delay ) or (d_k [dest] ≤ delay+ deviation))  then
              If (Check_feasibility (path_p′, Path_vector [dest])) then
                 Commit_Disjoint_Paths ()
                 return TRUE
            Else
                If (JIT_k  ≥ Hopcount_k [u]) then
                        for each vertex v ∈ Adj [u] do
                            If  (Hopcount_k [u] ≤  BF_v )then
                                RELAX (u, v, w_k, dest, S)
    9.  If ( Q← empty) and (d_k [dest] >  delay  or d_k [dest] >  delay + deviation) then
                Print, "Disjoint pair of paths meeting QoS doesn't exist"
                return FALSE
End

RELAX (Node u, Node v, weight pair w, Set Q)
Begin
    1.  If  d_k [v] > d_k [u] +w_k(u, v)  and d_k [dest] > d_k [u] + w_k (u, v)  then
          d_k [v]← d_k [u] + w_k (u, v)
          P_k [v]← u
          Path_vector_k [v] ← Path_vector_k [u]
          Path_vector_k [v] ← Append (u)
          Hopcount_k[v] ← Hopcount_k [u] +1
          Enqueue(Q, v)
End

Initialize-Single-Source (GRAPH G, Node source, int Count_unique_bw, int Link_resbw[ ], real jitter)
Begin
    1.  for k ← 1 to Count_unique_bw do
            JIT_k ←⌊(Link_resbw[k] * jitter- burst_rate)/ L_max⌋
    2.      for each vertex v ∈ V [G] do
                d_k [v]← ∞
                P_k [v]←source
                Path_vector_k [v] ← NULL
                Hopcount_k [v] ← 0
    3.      d_k [source]← 0
    4.      P_k [source]←NONE
End
Boolean Check_feasibility (Path_vector p′, Path_vector p″)
Begin
        {P1,P2} = Collace_and_Interlace(p′,  p″)
        return   (P1 ≤ delay and P2 ≤ delay)? TRUE: FALSE

End
```

Figure 4.3: Widest-Shortest Path Algorithm

Chapter 5: Comparative Performance Studies

Section 5.1 defines performance metrics for MCMP and describes the experimental procedure used to conduct performance studies. Section 5.2 presents the results of performance studies of a small network topology. Section 5.3 presents the results of performance studies of a grid network topology. Section 5.4 presents the results of performance studies conducted over a deployed real network. Section 5.5 presents the execution times of MCMP and RF over 300-bus system topology.

5.1 Experimental Procedure

Setup: Performance is defined as the qualitative level at which a network fulfills its function. The performance analysis is conducted by evaluating the performance of MCMP algorithm over commonly used RF algorithm in terms of committing QoS subscriptions. RF algorithm is presented in chapter 3 and MCMP algorithm is presented in chapter 4.

The RF algorithm presented in chapter 3 uses either Dijkstra algorithm or BFS algorithm to find the shortest path in step1 and step2. In order to conduct a fair performance analysis, we use *shortest-delay path algorithm* presented in chapter 4 in step1 of RF algorithm and use *widest-shortest path algorithm* presented in chapter 4 in step 2 of RF algorithm. The main difference lies is in the way graph transformations are done by MCMP algorithm and RF algorithm.

Experimental Procedure: We submit a series of QoS subscription requests to MCMP algorithm and RF algorithm and tabulate the number of QoS requests committed by each algorithm and analyze the commit to request ratio for MCMP algorithm and RF algorithm. Each subscription request is submitted to a graph modified by the previous

request. The experiment is stopped when all the resources are exhausted or when both the algorithms fail to find disjoint pair of paths simultaneously. The sample set of QoS subscription requests look as follows:

- QoS subscription requests between random source and random destination with fixed bandwidth requests or varying bandwidth requests.
- QoS subscription requests between a given source and given destination with fixed bandwidth requests or varying bandwidth requests.

The above experimental procedure is conducted over several topologies. We conduct the above experiment over the topology given by Bhandari in [22], AT&T IP backbone network 2Q2000 [31] and over a network with grid topology. The experimental results are tabulated and presented in next three sections.

5.2 Simple Topology

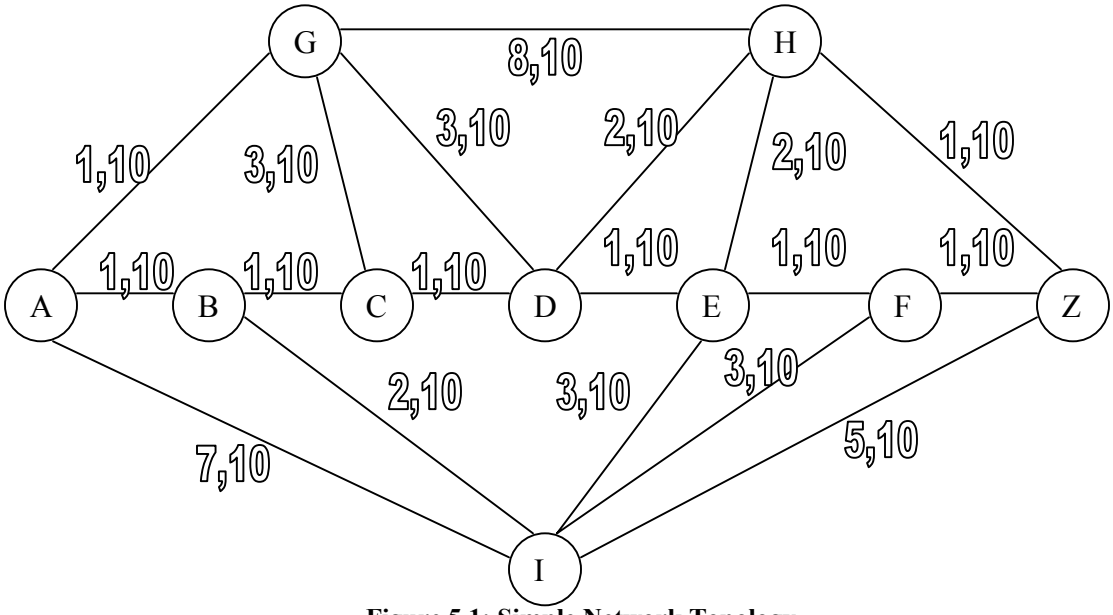

Figure 5.1: Simple Network Topology

The topology is taken from [22]. Each link is represented by two non-negative measures. The first measure is the delay associated between the link and second measure

is the link residual bandwidth. For this topology we chose random source node and random destination node and for each subscription a bandwidth of 1Mbps was requested.

Experiment 1: RF ALGORITHM

Delay Bound = 10 sec, Bandwidth per request = 1Mbps.

Sl#	Source	Dest	Disjoint Pair of Paths
1	A	Z	P1= ABCDEFZ (6) P2 = AGHZ (10)
2	B	H	P1=BCDH (4) P2=BIEH (7)
3	D	I	P1=DEI (4) P2= DCBI (4)
4	C	Z	P1=CDEFZ (4) P2= CBIZ (8)
5	G	Z	P1=GCDEFZ (6) P2= GABIZ (9)
6	F	G	P1=FEDCG (5) P2=FIBAG (7)
7	I	G	P1=IBAG (4) P1=IEDG (7
8	Z	A	P1= ABCDEFZ (6) P2 = AGHZ (10)
9	A	E	P1=ABIE(6) P2=AGCDE(6)
10	Z	B	P1=ZHDCB(5) P2=ZFIB(6)
11	A	Z	Fails
12	Z	D	P1=ZHD(3) P2=ZFED(3)
13	Z	B	P1=ZHDCB(5) P2=ZFIB(6)

Table 1: QoS requests committed by RF algorithm for simple topology (Fig 5.1)

Number of QoS Subscription requests = 13

Number of QoS requests Committed = 12

Commit Ratio = Num of QoS requests committed/ Num of QoS subscription requests

= 12/13= 0.923

Percentage commit ratio for the above topology using RF algorithm = $0.923 * 100$

= 92.3%

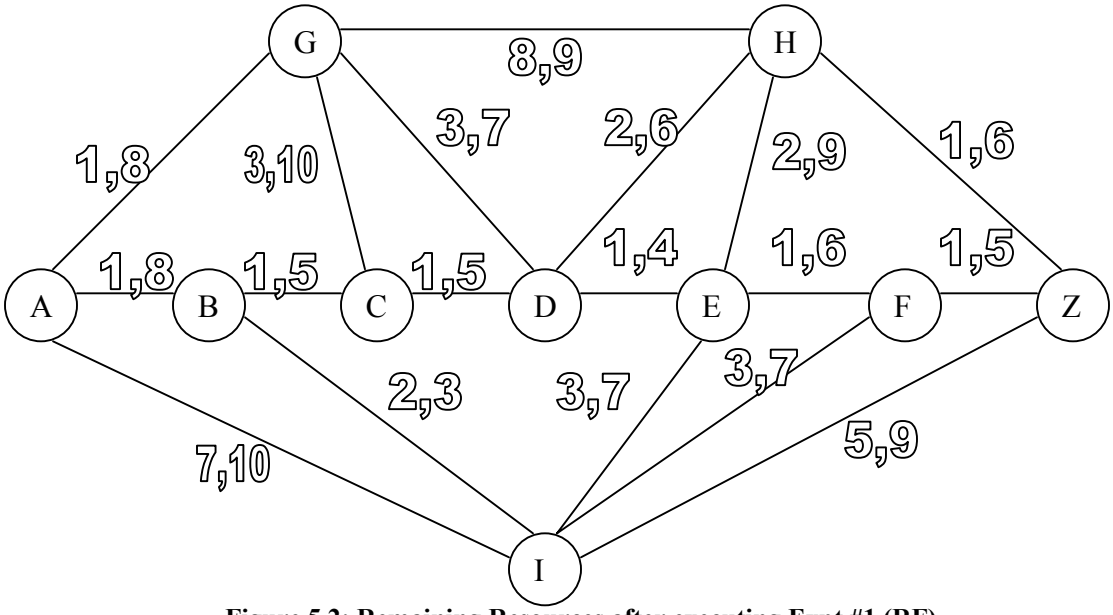

Figure 5.2: Remaining Resources after executing Expt #1 (RF)

Experiment 1: MCMP ALGORITHM

Delay Bound = 10 sec, Bandwidth per request = 1Mbps.

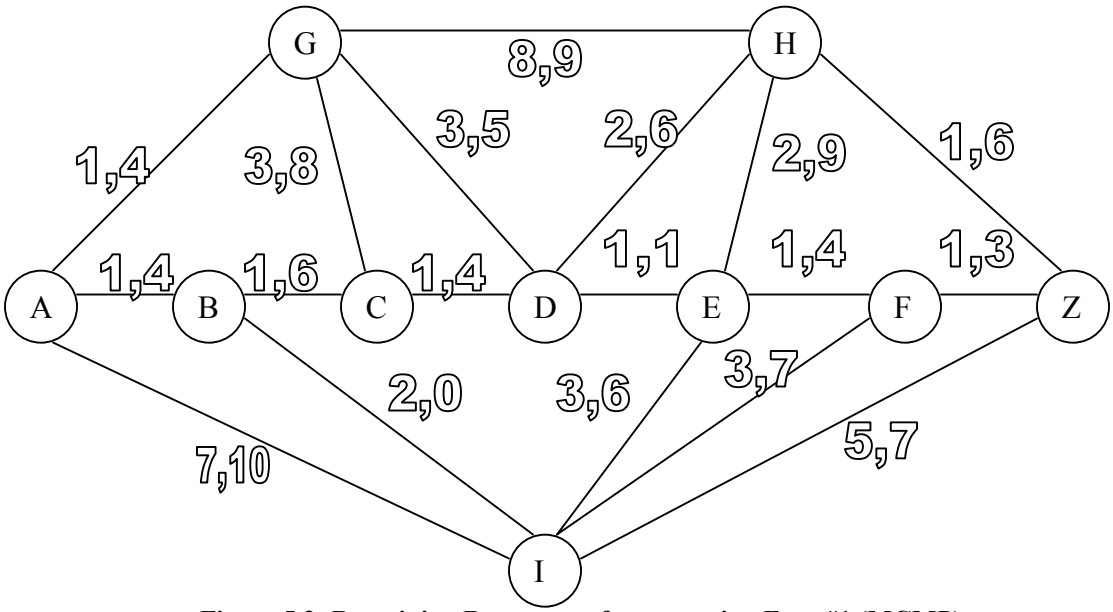

Figure 5.3: Remaining Resources after executing Expt #1 (MCMP)

SL #	Source	Dest	Disjoint Pair of Paths
1	A	Z	P1= ABIZ (8) P2 = AGCDEFZ (7)
2	B	H	P1=BCDH (4) P2=BIEH (7)
3	D	I	P1=DEI (4) P2= DCBI (4)
4	C	Z	P1=CDEFZ (4) P2= CBIZ (8)
5	G	Z	P1=GCDEFZ (6) P2= GABIZ (9)
6	F	G	P1=FEDCG (5) P2=FIBAG (7)
7	I	G	P1=IBAG (4) P1=IEDCG (7)
8	Z	A	P1=ZHDCGA(8) P2=ZFIBA(7)
9	A	E	P1=ABIE(6) P2=AGCDE(6)
10	Z	B	P1=ZHDCB(5) P2=ZFIB(6)
11	A	Z	P1= ABIZ (8) P2 = AGCDEFZ (7)
12	Z	D	P1=ZHD(3) P2=ZFED(3)
13	Z	B	P1=ZHDCB(5) P2=ZFIB(6)

Table 2: QoS requests committed by MCMP algorithm for simple topology (Fig 5.1)

Number of QoS Subscription requests = 13

Number of QoS requests Committed = 13

Commit Ratio = Num of QoS requests committed/ Num of QoS subscription requests

= 13/13= 1.0

Percentage commit ratio for the above topology using MCMP algorithm = 1* 100

= 100 %

The experiment is repeated over RF algorithm and MCMP algorithm for different source destination pair with different delay bounds and the results are tabulated as follows. For each request, bandwidth of 1Mbps was requested. The algorithms are executed over the simple network topology shown in figure 5.1.

Experimental results of RF ALGORITHM –Simple Topology

Expt #	Source-Destination Pairs	Num of requests submitted	Num of requests committed	Delay in sec	Commit ratio in %
1	{(A, Z), (B, H), (D, I), (C, Z), (G, Z), (F, G), (I, G), (Z, A), (A, E), (Z, B), (A, Z), (Z, D), (Z, B)}	13	12	10	92.3
2	{(A, Z), (B, H), (D, I), (C, Z), (G, Z), (F, G), (I, G), (Z, A), (A, E), (Z, B), (A, Z), (Z, D), (Z, B)}	13	9	9	69.2
3	{(A, Z), (B, H), (D, I), (C, Z), (G, Z), (F, G), (I, G), (Z, A), (A, E), (Z, B), (A, Z), (Z, D), (Z, B)}	13	9	8	69.2
4	{(A,Z),(C,F),(F,I),(G,I),(F,A), (C,Z),(A,I),(H,Z),(F,B),(G,B), (B,G), (A,Z),(Z,A),(D,B),(A,D), (Z,B),(H,G),(C,B)}	18	16	10	88.8
5	{(A,Z),(C,F),(F,I),(G,I),(F,A), (C,Z),(A,I),(H,Z),(F,B),(G,B), (B,G), (A,Z),(Z,A),(D,B),(A,D), (Z,B),(H,G),(C,B)}	18	14	9	77.7
6	{(A,Z),(C,F),(F,I),(G,I),(F,A), (C,Z),(A,I),(H,Z),(F,B),(G,B), (B,G), (A,Z),(Z,A),(D,B),(A,D), (Z,B),(H,G),(C,B)}	18	14	8	77.7
7	{(A,Z),(I,D),(E,F),(E,C),(C,A), (A,D),(H,E),(Z,A),(Z,C),(F,I), (A,B), (A,F),(G,F),(I.G),(I,E), (C,G),(E,Z),(E,H),(Z,E)}	19	18	10	94.7
8	{(A,Z),(I,D),(E,F),(E,C),(C,A), (A,D),(H,E),(Z,A),(Z,C),(F,I), (A,B), (A,F),(G,F),(I.G),(I,E), (C,G),(E,Z),(E,H),(Z,E)}	19	16	9	84.2
9	{(A,Z),(I,D),(E,F),(E,C),(C,A), (A,D),(H,E),(Z,A),(Z,C),(F,I), (A,B), (A,F),(G,F),(I.G),(I,E), (C,G),(E,Z),(E,H),(Z,E)}	19	16	8	84.2
10	{(A,Z),(D,B),(H,G),(G,B),(G,F), (G,Z),(A,F),(B,I),(F,C),(B,I), (A,D), (C,G),(E,G),(B.H),(E,F) (F,H),(D,B),(H,Z),(F,E)}	19	19	10	100
11	{(A,Z),(D,B),(H,G),(G,B),(G,F), (G,Z),(A,F),(B,I),(F,C),(B,I), (A,D), (C,G),(E,G),(B.H),(E,F) (F,H),(D,B),(H,Z),(F,E)}	19	17	9	89.5
12	{(A,Z),(D,B),(H,G),(G,B),(G,F), (G,Z),(A,F),(B,I),(F,C),(B,I), (A,D), (C,G),(E,G),(B.H),(E,F) (F,H),(D,B),(H,Z),(F,E)}	19	16	8	84.2
AVERAGE COMMIT RATIO IN %					*84.34*

Table 3: Experimental results of RF algorithm-Simple Topology

Experimental results of MCMP ALGORITHM –Simple Topology

Expt #	Source-Destination Pairs	Num of requests submitted	Num of requests committed	Delay in sec	Commit ratio in %
1	{(A, Z), (B, H), (D, I), (C, Z), (G, Z), (F, G), (I, G), (Z, A), (A, E), (Z, B), (A, Z), (Z, D), (Z, B)}	13	13	10	100
2	{(A, Z), (B, H), (D, I), (C, Z), (G, Z), (F, G), (I, G), (Z, A), (A, E), (Z, B), (A, Z), (Z, D), (Z, B)}	13	12	9	92.3
3	{(A, Z), (B, H), (D, I), (C, Z), (G, Z), (F, G), (I, G), (Z, A), (A, E), (Z, B), (A, Z), (Z, D), (Z, B)}	13	10	8	76.9
4	{(A,Z),(C,F),(F,I),(G,I),(F,A), (C,Z),(A,I),(H,Z),(F,B),(G,B), (B,G), (A,Z),(Z,A),(D,B),(A,D), (Z,B),(H,G),(C,B)}	18	18	10	100
5	{(A,Z),(C,F),(F,I),(G,I),(F,A), (C,Z),(A,I),(H,Z),(F,B),(G,B), (B,G), (A,Z),(Z,A),(D,B),(A,D), (Z,B),(H,G),(C,B)}	18	17	9	94.4
6	{(A,Z),(C,F),(F,I),(G,I),(F,A), (C,Z),(A,I),(H,Z),(F,B),(G,B), (B,G), (A,Z),(Z,A),(D,B),(A,D), (Z,B),(H,G),(C,B)}	18	17	8	94.4
7	{(A,Z),(I,D),(E,F),(E,C),(C,A), (A,D),(H,E),(Z,A),(Z,C),(F,I), (A,B), (A,F),(G,F),(I.G),(I,E), (C,G),(E,Z),(E,H),(Z,E)}	19	19	10	100
8	{(A,Z),(I,D),(E,F),(E,C),(C,A), (A,D),(H,E),(Z,A),(Z,C),(F,I), (A,B), (A,F),(G,F),(I.G),(I,E), (C,G),(E,Z),(E,H),(Z,E)}	19	19	9	100
9	{(A,Z),(I,D),(E,F),(E,C),(C,A), (A,D),(H,E),(Z,A),(Z,C),(F,I), (A,B), (A,F),(G,F),(I.G),(I,E), (C,G),(E,Z),(E,H),(Z,E)}	19	19	8	100
10	{(A,Z),(D,B),(H,G),(G,B),(G,F), (G,Z),(A,F),(B,I),(F,C),(B,I), (A,D), (C,G),(E,G),(B.H),(E,F) (F,H),(D,B),(H,Z),(F,E)}	19	19	10	100
11	{(A,Z),(D,B),(H,G),(G,B),(G,F), (G,Z),(A,F),(B,I),(F,C),(B,I), (A,D), (C,G),(E,G),(B.H),(E,F) (F,H),(D,B),(H,Z),(F,E)}	19	18	9	94.7
12	{(A,Z),(D,B),(H,G),(G,B),(G,F), (G,Z),(A,F),(B,I),(F,C),(B,I), (A,D), (C,G),(E,G),(B.H),(E,F) (F,H),(D,B),(H,Z),(F,E)}	19	17	8	89.5
AVERAGE COMMIT RATIO IN %					*95.19*

Table 4: Experimental results of MCMP algorithm-Simple Topology

MCMP and RF algorithms performed same in terms of average commit ratio when the delay bound was either less than 8 sec or more than 10 sec. We could find significant difference in the commit ratio when the delay bound was varied from 8 sec to 10 sec. Hence we chose the delay bound values as 8, 9 and 10 seconds respectively for the simple topology.

Inference: Performance of heuristic MCMP algorithm is better than the remove-find method in terms of committed QoS subscriptions using disjoint paths and utilization of network resources.

5.2 Grid Topology

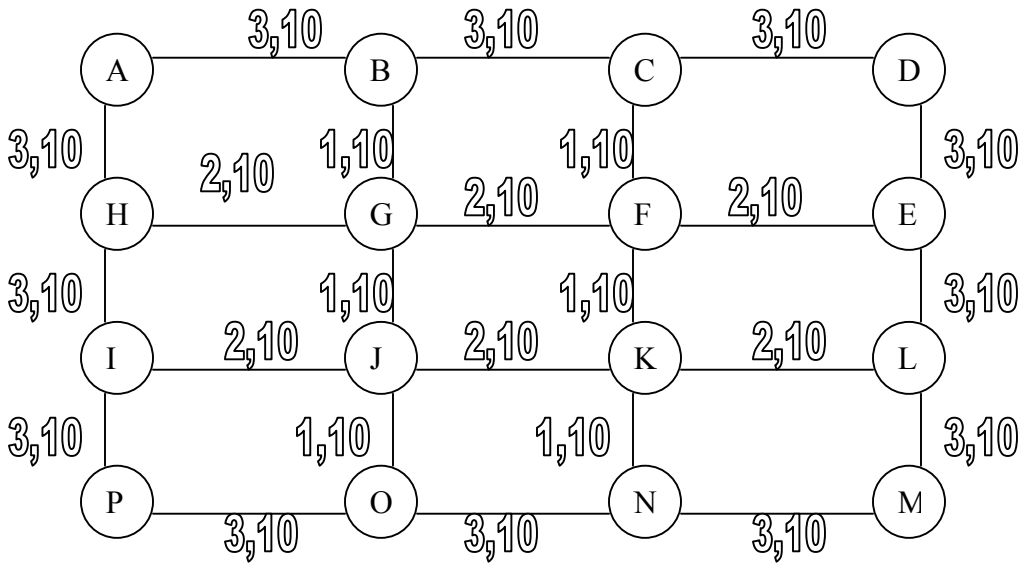

Figure 5.4: Grid Network Topology

Each link is represented by two non-negative measures. The first measure is the delay associated between the link and second measure is the link residual bandwidth. The experiment is repeated over RF algorithm and MCMP algorithm for different source destination pair with different delay bounds and the results are tabulated as follows. For each request, bandwidth of 1Mbps was requested. The algorithms are executed over the grid network topology shown in figure 5.4.

Experimental results of RF ALGORITHM –Grid Topology

Expt #	Source-Destination Pairs	Num of requests submitted	Num of requests committed	Delay in sec	Commit ratio in %
1	{(A, M), (D, P), (M, A), (P, D), (N, C), (D, A), (O, E), (C, A), (O, K), (H, C), (N, C), (H, B), (J, P), (J, P), (I, F) (N,P), (M,H),(N,M),(P,D),(P,I)}	20	12	20	60
2	{(A, M), (D, P), (M, A), (P, D), (N, C), (D, A), (O, E), (C, A), (O, K), (H, C), (N, C), (H, B), (J, P), (J, P), (I, F) (N,P), (M,H),(N,M),(P,D),(P,I)}	20	12	15	60
3	{(A, M), (D, P), (M, A), (P, D), (N, C), (D, A), (O, E), (C, A), (O, K), (H, C), (N, C), (H, B), (J, P), (J, P), (I, F) (N,P), (M,H),(N,M),(P,D),(P,I)}	20	11	12	55
4	{(H, E), (H, F), (O,N), (B, H), (H, F), (E, A), (B, I), (E, M), (D, K), (B, F), (O, H), (C, K), (G, E), (K, J), (D, B) (N,G), (C,F),(L,E),(M,P),(A,M)}	20	19	20	95
5	{(H, E), (H, F), (O,N), (B, H), (H, F), (E, A), (B, I), (E, M), (D, K), (B, F), (O, H), (C, K), (G, E), (K, J), (D, B) (N,G), (C,F),(L,E),(M,P),(A,M)}	20	19	15	95
6	{(H, E), (H, F), (O,N), (B, H), (H, F), (E, A), (B, I), (E, M), (D, K), (B, F), (O, H), (C, K), (G, E), (K, J), (D, B) (N,G), (C,F),(L,E),(M,P),(A,M)}	20	18	12	90
7	{(G,E),(I,H),(B,P),(E,A),(A,C), (D,A),(F,A),(E,P),(O,K),(J,H), (H,E), (P,K),(O,B),(G,I),(N,A), (A,N),(M,F),(M,A),(D,P),(A,M) (J,M),(E,J)}	22	19	20	86.4
8	{(G,E),(I,H),(B,P),(E,A),(A,C), (D,A),(F,A),(E,P),(O,K),(J,H), (H,E), (P,K),(O,B),(G,I),(N,A), (A,N),(M,F),(M,A),(D,P),(A,M) (J,M),(E,J)}	22	18	15	81.8
9	{(G,E),(I,H),(B,P),(E,A),(A,C), (D,A),(F,A),(E,P),(O,K),(J,H), (H,E), (P,K),(O,B),(G,I),(N,A), (A,N),(M,F),(M,A),(D,P),(A,M) (J,M),(E,J)}	22	15	12	68.2
10	{(L,O),(I,G),(D,C),(N,D),(E,P), (I,N),(B,I),(L,M),(C,A),(I,N), (F,C), (K,O),(J,P),(D,B),(D,K), (B,O),(G,P),(D,F),(E,I),(H,F) (P,D),(D,P)}	22	20	20	90.9
11	{(L,O),(I,G),(D,C),(N,D),(E,P), (I,N),(B,I),(L,M),(C,A),(I,N), (F,C), (K,O),(J,P),(D,B),(D,K), (B,O),(G,P),(D,F),(E,I),(H,F) (P,D),(D,P)}	22	19	15	86.3
12	{(L,O),(I,G),(D,C),(N,D),(E,P), (I,N),(B,I),(L,M),(C,A),(I,N), (F,C), (K,O),(J,P),(D,B),(D,K), (B,O),(G,P),(D,F),(E,I),(H,F) (P,D),(D,P)}	22	17	12	77.3
AVERAGE COMMIT RATIO IN %					*78.83*

Table 5: Experimental results of RF algorithm-Grid Topology

Experimental results of MCMP ALGORITHM –Grid Topology

Expt #	Source-Destination Pairs	Num of requests submitted	Num of requests Committed	Delay in sec	Commit ratio in %
1	{(A, M), (D, P), (M, A), (P, D), (N, C), (D, A), (O, E), (C, A), (O, K), (H, C), (N, C), (H, B), (J, P), (J, P), (I, F) (N,P), (M,H),(N,M),(P,D),(P,I)}	20	20	20	100
2	{(A, M), (D, P), (M, A), (P, D), (N, C), (D, A), (O, E), (C, A), (O, K), (H, C), (N, C), (H, B), (J, P), (J, P), (I, F) (N,P), (M,H),(N,M),(P,D),(P,I)}	20	19	15	95
3	{(A, M), (D, P), (M, A), (P, D), (N, C), (D, A), (O, E), (C, A), (O, K), (H, C), (N, C), (H, B), (J, P), (J, P), (I, F) (N,P), (M,H),(N,M),(P,D),(P,I)}	20	15	12	75
4	{(H, E), (H, F), (O,N), (B, H), (H, F), (E, A), (B, I), (E, M), (D, K), (B, F), (O, H), (C, K), (G, E), (K, J), (D, B) (N,G), (C,F),(L,E),(M,P),(A,M)}	20	20	20	100
5	{(H, E), (H, F), (O,N), (B, H), (H, F), (E, A), (B, I), (E, M), (D, K), (B, F), (O, H), (C, K), (G, E), (K, J), (D, B) (N,G), (C,F),(L,E),(M,P),(A,M)}	20	20	15	100
6	{(H, E), (H, F), (O,N), (B, H), (H, F), (E, A), (B, I), (E, M), (D, K), (B, F), (O, H), (C, K), (G, E), (K, J), (D, B) (N,G), (C,F),(L,E),(M,P),(A,M)}	20	19	12	95
7	{(G,E),(I,H),(B,P),(E,A),(A,C), (D,A),(F,A),(E,P),(O,K),(J,H), (H,E), (P,K),(O,B),(G,I),(N,A), (A,N),(M,F),(M,A),(D,P),(A,M) (J,M),(E,J)}	22	22	20	100
8	{(G,E),(I,H),(B,P),(E,A),(A,C), (D,A),(F,A),(E,P),(O,K),(J,H), (H,E), (P,K),(O,B),(G,I),(N,A), (A,N),(M,F),(M,A),(D,P),(A,M) (J,M),(E,J)}	22	22	15	100
9	{(G,E),(I,H),(B,P),(E,A),(A,C), (D,A),(F,A),(E,P),(O,K),(J,H), (H,E), (P,K),(O,B),(G,I),(N,A), (A,N),(M,F),(M,A),(D,P),(A,M) (J,M),(E,J)}	22	19	12	86.3
10	{(L,O),(I,G),(D,C),(N,D),(E,P), (I,N),(B,I),(L,M),(C,A),(I,N), (F,C), (K,O),(J,P),(D,B),(D,K), (B,O),(G,P),(D,F),(E,I),(H,F) (P,D),(D,P)}	22	22	20	100
11	{(L,O),(I,G),(D,C),(N,D),(E,P), (I,N),(B,I),(L,M),(C,A),(I,N), (F,C), (K,O),(J,P),(D,B),(D,K), (B,O),(G,P),(D,F),(E,I),(H,F) (P,D),(D,P)}	22	22	15	100
12	{(L,O),(I,G),(D,C),(N,D),(E,P), (I,N),(B,I),(L,M),(C,A),(I,N), (F,C), (K,O),(J,P),(D,B),(D,K), (B,O),(G,P),(D,F),(E,I),(H,F) (P,D),(D,P)}	22	20	12	90.9
AVERAGE COMMIT RATIO IN %					*95.18*

Table 6: Experimental results of MCMP algorithm-Grid Topology

Inference: Performance of heuristic MCMP algorithm is better than the remove-find method in terms of committed QoS subscriptions using disjoint paths and utilization of network resources.

5.4 Deployed Network – AT&T IP Backbone

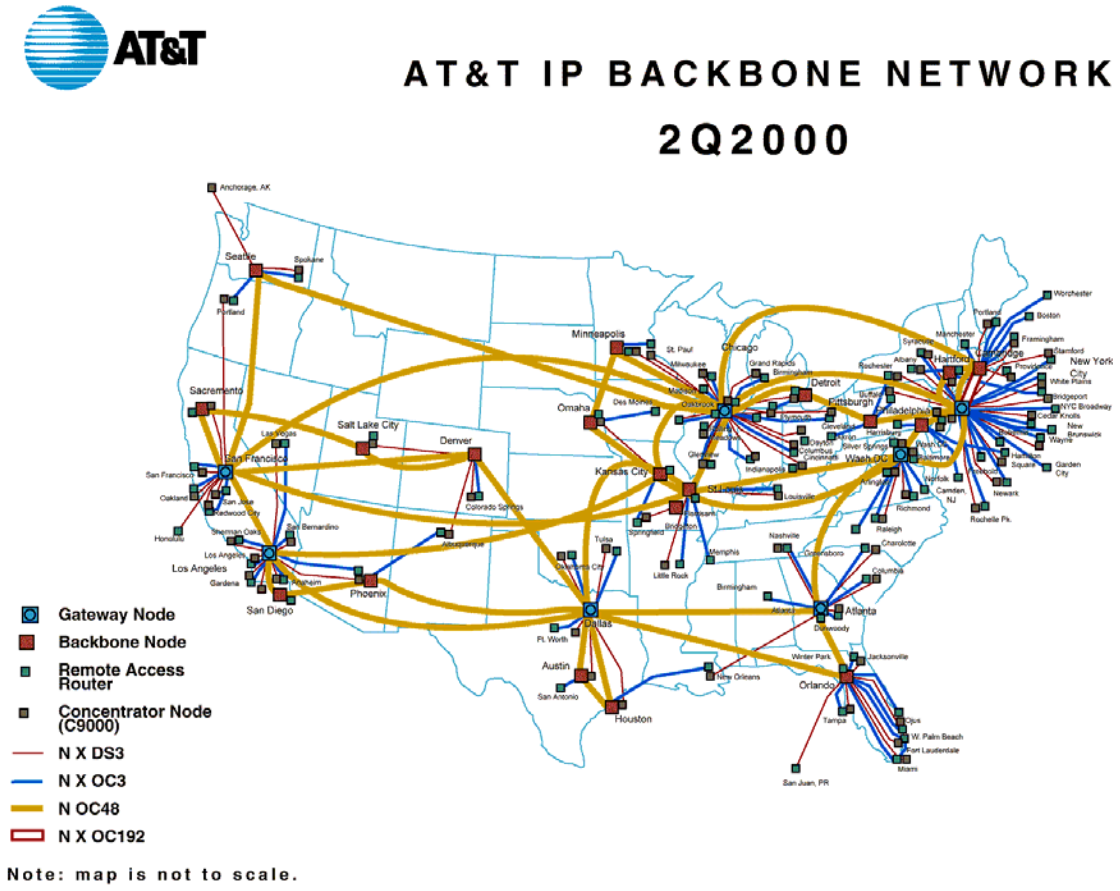

Figure 5.5: AT&T IP Backbone Network 2Q 2000 [31]

We chose AT&T IP backbone network 2Q2000 [31] to conduct performance study of RF algorithm and MCMP algorithm over deployed networks. The network is shown in figure 5.5. For the experiment purpose we chose *Gateway nodes* and *Backbone nodes* only. Gateway nodes are marked in blue color and backbone nodes are marked in maroon color in the above figure. We assumed that bandwidth capacity of cable NOC48 marked in yellow color is 100Mbps. The delay between two nodes was calculated using the formula

D(u, v) = (Physical distance between node *u* and node *v*) / (Speed of Light), where the delay between two nodes *u* and *v* be denoted by D(u,v).

The experiment was conducted as per the experimental procedure given in section 5.1 over AT&T network for different source destination pair with different delay bounds and different bandwidth requests.

Results: *RF algorithm can find feasible paths if such paths exist along the outer part of the network. MCMP algorithm can search deep inside the network and find feasible paths in addition to the paths found using RF algorithm MCMP algorithms can handle more subscription when such feasible paths are deep inside the network. The experimental results showed that RF algorithm and MCMP algorithm could make equal number of subscriptions in each experiment. This is because of the fact that AT&T IP backbone network is designed in such a way that one of the paths get selected along the top part of the network and the second path gets selected along the bottom path of the network. In other words AT&T IP backbone network 2Q2000 is designed with reliability taken into consideration.*

This thesis was focused in achieving a reliable node-disjoint QoS routing algorithm for GridStat. After running MCMP algorithm and RF algorithm over AT&T network we suggest that GridStat network be designed along the lines of AT&T IP backbone network 2Q2000 to ensure more reliability to the infrastructure itself.

5.5 Comparison of Execution times of MCMP and RF algorithms

The goal of this experiment was to measure the execution time of MCMP and RF algorithm over 300-bus topology. The experiment as described in section 5.1 was conducted over a standard topology commonly used in power systems research. It is popularly known as the 300-bus system. This topology was chosen because the future electric power grid communications

network might be designed similar to this topology. After eliminating the spines (singly connected nodes) the system had 213 nodes. Delays were random in the range of [1, 10] and bandwidths were assigned random in the range of [1, 100]. For each trial a series of 25 random source and destination pair requests were submitted to MCMP algorithm and RF algorithm and the execution time of each algorithm were measured based on CPU clock. Each trial of the experiment was repeated 25 times. The delay bound chosen was high so that both the algorithms could find disjoint pair of paths. 10 such trials were conducted and the results are tabulated as follows. The experiment was run on P4 1.8GHz Intel processor, with 512MB RAM.

Sl #	Total Number of Requests	Execution Time of MCMP in sec	Execution time of RF in sec
1	25	0.841	0.731
2	25	0.961	0.868
3	25	0.911	0.795
4	25	1.031	0.891
5	25	0.831	0.751
6	25	0.951	0.791
7	25	0.931	0.781
8	25	0.971	0.831
9	25	0.910	0.801
10	25	0.790	0.730
Average Execution times in seconds		0.912	0.796

Table 7: Execution times of MCMP and RF algorithm over 300-bus system topology

Inference: RF algorithm has an average lower execution time when compared to MCMP algorithm.

Chapter 6: Conclusion and Future Work

Subsection 6.1 presents the conclusion in brief. Subsection 6.2 points towards future work that needs to be accomplished

6.1 Conclusion

In SDN, the status information, which includes data and/or control information, needs to be gathered and distributed to the registered clients through status variables according to their QoS specification in a reliable and timely manner. In order to achieve the above, an efficient QoS routing algorithm ensuring reliability is needed.

Such an efficient QoS routing algorithm for GridStat has been the topic of this thesis. We presented a heuristic algorithm for QoS routing which computes a pair of node-disjoint paths that satisfy the independent, multiple QoS constraints, which we refer to as *Multi-constrained Multipath (MCMP)* routing algorithm. Based on the simulation results we showed that the performance of our heuristic algorithm is better than the remove-find method in terms of committed QoS subscriptions using disjoint paths and utilization of network resources. This thesis was focused in achieving a reliable node-disjoint QoS routing algorithm for GridStat. After running MCMP algorithm and RF algorithm over AT&T network we suggest that GridStat network be designed along the lines of AT&T IP backbone network 2Q2000 to ensure more reliability to the infrastructure itself in addition to having MCMP routing algorithm.

6.2 Future Work

In this thesis we presented a heuristic algorithm, which finds node-disjoint paths satisfying multiple QoS constraints in a flat network (a network without hierarchical parts). GridStat has a hierarchical structure and as a first step towards finding an efficient

MCMP algorithm we assumed that the topology information and current network state are known all the way up the hierarchy. As a result of this assumption, the problem of inter-domain routing gets reduced to that of intra-domain routing. Routing problems are currently solved at a common ancestor of the source and destination by utilizing topology and state information for all the clouds that are involved. *A solution that exploits the hierarchy remains for future work.*

In addition to exploiting hierarchy, MCMP should be extended to handle multicast subscriptions and MCMP should be fine tuned to accommodate reallocation of paths whenever better resources become available in the network pool. More research needs to done towards exploring possibilities in choosing a different initial path if step 3 of MCMP algorithm fails.

APPENDIX A: Execution of Vertex Splitting Algorithm

A.1 Finding node-disjoint pair of paths for Topology 3.4.1.1 using vertex splitting based node-disjoint shortest pair algorithm.

STEP 1: The shortest path obtained using modified Dijkstra algorithm between the source vertex A and destination vertex Z over graph G is ABCFZ of path length 4. Let's call this path p′.

STEP 2: Invoking Check_and_SplitVertex algorithm

i) Checking degree of all nodes along path p′

NODE	A	B	C	F	Z
DEGREE	-	3	2	3	-

ii) Degree of all nodes apart from source and destination is less than 4. So go to STEP 4 of Check_and_SplitVertex algorithm.

iii) Resulting Graph G′ returned has the following topology A.1.1.

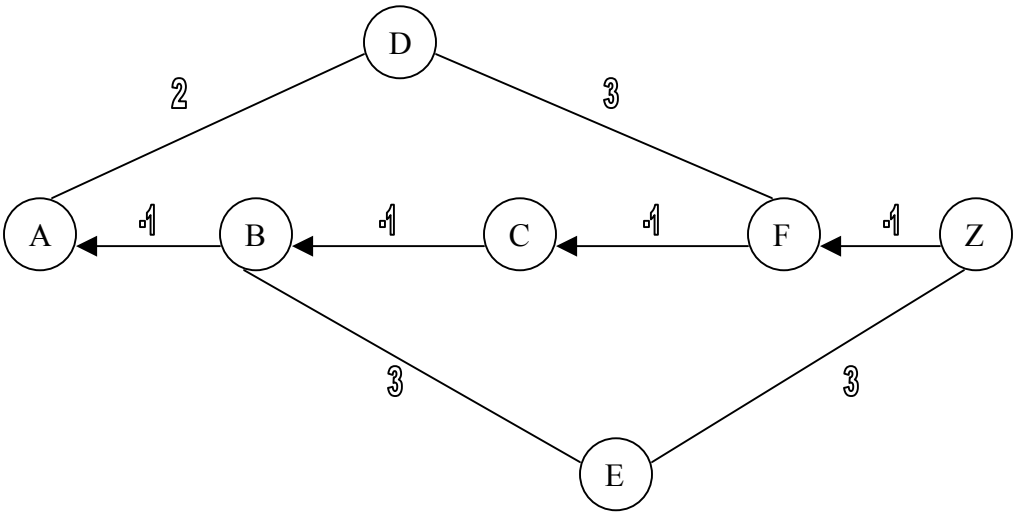

Figure A.1: Topology A.1.1

STEP 3: The shortest path obtained using modified Dijkstra algorithm between the source vertex A and destination vertex Z over graph G′ is ADFCBEZ of path length 9. Let's call this path p″.

STEP 4: Invoking Collace_Interlace algorithm

i) As there was no vertex splitting in STEP 2, the path P″ will be identical to path p″.

ii)

Node	A	D	F	C	B	E	Z
Successor	D	F	C	B	E	Z	-1
Boolean_marked	0	0	1	1	0	0	0

Node	A	D	F	C	B	E	Z
Successor	D	F	C	B	E	Z	-1
Boolean_marked	0	0	1	1	0	0	0

Interlacing Edges are BC and FC. Removing the interlacing edges in path p′ and P″ and grouping the remaining edges results in a node-disjoint pair of paths.

iii) Node disjoint pair of paths are:

P1 = A $\rightarrow$B$\rightarrow$E$\rightarrow$Z, path length of P1 =7

P2 = A$\rightarrow$D$\rightarrow$F$\rightarrow$Z, path length of P2 = 6

iv) The shortest pair of node disjoint paths is P1 and P2 with path lengths 7 and 6 respectively.

A.2 Finding node-disjoint pair of paths for Topology 3.4.1.1 with an additional link between E and F of weight 3 using vertex splitting based node-disjoint shortest pair algorithm.

STEP 1: The shortest path obtained using modified Dijkstra algorithm between the source vertex A and destination vertex Z over graph G is ABCFZ of path length 4. Let's call this path p′.

STEP 2: Invoking Check_and_SplitVertex algorithm

i) Checking degree of all nodes along path p′

NODE	A	B	C	F	Z
DEGREE	-	3	2	4	-

ii) Degree of one of the nodes apart from source and destination is greater than 3. So go to STEP 5 of Check_and_SplitVertex algorithm.

iii) Resulting Graph G′ returned has the following topology A.2.1.

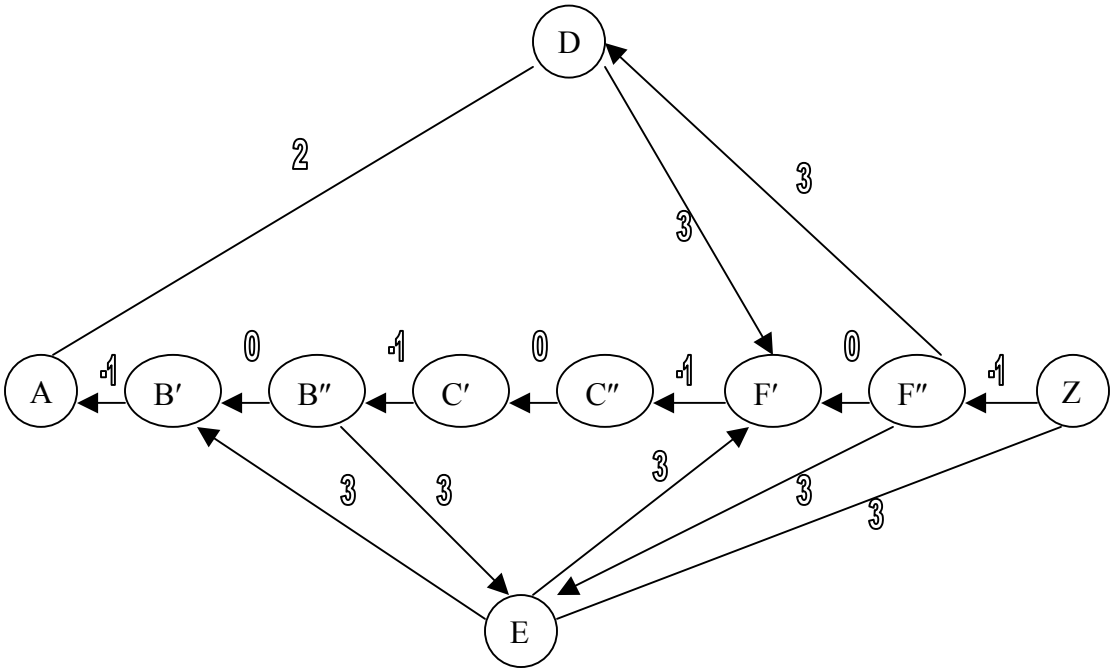

Figure A.2: Topology A.2.1

STEP 3: The shortest path obtained using modified Dijkstra algorithm between the source vertex A and destination vertex Z over graph G′ is ADF′C″C′B″EZ of path length 9. Let's call this path p″.

STEP 4: Invoking Collace_Interlace algorithm

i) As there is vertex splitting in STEP 2, the zero length arcs are removed and the co-located subvertices are coalesced into their parent vertices and the single arcs of the shortest path are replaced with original edges in path p″ to form P″ = ADFCBEZ.

Node	A	B	C	F	Z
Successor	B	C	F	Z	-1
Boolean_marked	0	1	1	0	0

Node	A	D	F	C	B	E	Z
Successor	D	F	C	B	E	Z	-1
Boolean_marked	0	0	1	1	0	0	0

ii) Interlacing Edges are BC and FC. Removing the interlacing edges in path p′ and P″ and grouping the remaining edges results in a node-disjoint pair of paths.

iii) Node disjoint pair of paths are:

P1 = A →B→E→Z, path length of P1 =7

P2 = A→D→F→Z, path length of P2 = 6

iv) The shortest pair of node disjoint paths is P1 and P2 with path lengths 7 and 6 respectively.

BIBLIOGRAPHY

1. J.C.R Bennett and H.Zhang. "WF^2Q: Worst-case Fair Weighted Fair Queueing". *In Proceedings of IEEE INFOCOM'96*, May 1996.
2. D.P.Bertsekas. *Linear Network Optimization: Algorithms and Codes*. The MIT Press, 1991.
3. R.Bhandari. *Survivable Networks: Algorithms for Diverse Routing*. Kluwer Academic Publishers, 1999.
4. R.Bhandari. "A Shortest Pair of Physically-Disjoint Paths in Telecommunication Fiber Networks". *Proceedings of the Sixth International Symposium on Network Planning*. Pages 100-106, September 1994.
5. R.Bhandari. "Optimal Diverse Routing in Telecommunication Fiber Networks". *Proceedings of IEEE Infocom*, pages 1498-1508, June 1994.
6. S. Chen and K. Nahrstedt. "On finding multi-constrained paths". *In Proceedings of the IEEE International Conference on Communications*, pages 874–879, Atlanta, USA, June 1998.
7. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press, 1989.
8. A.Demers, S.Keshav, and S.Shenker. "Analysis and Simulation of a Fair Queueing Algorithm". *ACM SIGCOMM 89,* 19(4): 2-12, August 19-22,1989.
9. E.W.Dijkstra. "A Note on Two Problems in Connexion with Networks". *Numer.Math.1*, 269-271 (1959).

10. I.Dionysiou. “Secure Management Mechanisms and Policies for Status Dissemination Middleware”. Ph.D Preliminary Exam Material, School of EECS, WSU, Pullman-99164.
11. D. Fedyk and A. Ghanwani. “Metrics and resource classes for traffic engineering”. *Traffic Engineering Working Group - Internet Draft*, October 1999.
12. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979. (Page 214).
13. H.Gjermundrod, I. Dionysiou, D.Bakken, C.Hauser and A.Bose. “Flexible and Robust Status Dissemination Middleware for the Electric Power Grid”. Technical Report EECS-GS-03, School of EECS, WSU, Pullman-99164.
14. S.Golestani. “A Self-Clocked Fair Queueing Scheme for Broadband Applications”. *In Proceedings of IEEE INFOCOM’94*, pages 636-646, Toronto, Canada, June 1994.
15. Y. Guo, F.A. Kuipers and P. Van Mieghem. “Link-Disjoint Paths for Reliable QoS Routing”. *International Journal of Communication Systems*, 2003,16:779-798.
16. R. Haynal’s ISP Page. http://navigators.com/isp.html
17. K.Ishida, Y.Kakuda, T.Kikuno. “A Routing Protocol for Finding Two Node-Disjoint Paths in Computer Networks”. *IEEE International conference on Network protocols*, 1995.
18. A.Iwata, R. Izmailov, D.-S. Lee, B. Sengupta, G. Ramamurthy and H. Suzuki. “ATM routing algorithms with multiple QoS requirements for multimedia

internetworking". *IEICE Transactions and Communications E79-B*, no. 8, pp. 999-1006,1996.

19. J. Jaffe. "Algorithms for Finding Paths with Multiple Constraints". *Networks, volume 14*, pages 95-116, 1984.
20. T. Korkmaz and M. Krunz. "A randomized algorithm for finding a path subject to multiple qos requirements". *Computer Networks*, 36(2): 251–268,2001.
21. Q.Ma and P.Steenkiste. "Quality-of-Service Routing for Traffic with Performance Guarantees". *In IFIP Fifth International Workshop on Quality of Service*, pages 115-126, NY, May 1997.
22. Q.Ma. "Quality-of-Service Routing in Integrated Services Networks". PhD Thesis, CMU-CS-98-138, 1998.
23. E.F.Moore. "The Shortest Path through the Maze". *Proceedings of the International Symposium on the Theory of Switching*, 1957, Part II, Harvard University Press, pages 285-292 (1959).
24. A.Orda, A.Sprintson. "Efficient Algorithms for Computing Disjoint QoS Paths". *Infocomm* 2004.
25. D. Sidhu, R. Nair, and S. Abdallah. "Finding disjoint paths in networks". *ACM SIGCOMM Computer Communication Review*, 21(4): 43–51, 1991.
26. J.W. Suurballe. "Disjoint paths in a network". *Networks*, (4): 125–145, 1974.
27. P. Van Mieghem, H. De Neve and F.A. Kuipers. "Hop-by-hop Quality of Service Routing". *Computer Networks*, vol. 37. No 3-4, pp. 407-423, 2001.
28. P. Van Mieghem (ed.), F.A. Kuipers, T. Korkmaz, M. Krunz, M. Curado, E. Monteiro, X. Masip-Bruin, J. Solé-Pareta, and S. Sánchez-López. *Quality of*

Service Routing, Chapter 3 in *Quality of Future Internet Services, EU-COST 263 Final Report*, edited by Smirnov et al. in Springer LNCS 2856, pp. 80-117, 2003.

29. Z. Wang and J. Crowcroft. "Bandwidth-delay based routing algorithms". *In Proceedings of the IEEE Global Telecommunications Conference*, volume 3, pages 2129–2133, Singapore, November 1995.
30. H.Zhang. "Service Disciplines For Guaranteed Performance Service in Packet-Switching Networks". *In Proceedings of IEEE*, 83(10): 1374--96, Oct. 1995.
31. L.Zhang. " Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks". *SIGCOMM 90*, pages 19-29, 1990.