ABSTRACT

QUALITY-OF-SERVICE PROVISIONING IN HIGH SPEED NETWORKS: ROUTING PERSPECTIVE

by
Gang Cheng

The continuous growth in both commercial and public network traffic with various quality-of-service (QoS) requirements is calling for better service than the current Internet's best effort mechanism. One of the challenging issues is to select feasible paths that satisfy the different requirements of various applications. This problem is known as QoS routing. In general, two issues are related to QoS routing: state distribution and routing strategy. Routing strategy is used to find a feasible path that meets the QoS requirements. State distribution addresses the issue of exchanging the state information throughout the network, and can be further divided into two sub-problems: when to update and how to disseminate the state information.

In this dissertation, the issue of when to update link state information from the perspective of information theory is addressed. Based on the rate-distortion analysis, an efficient scheme, which outperforms the state of the art in terms of both protocol overhead and accuracy of link state information, is presented. Second, a reliable scheme is proposed so that, when a link is broken, link state information is still reachable to all network nodes as long as the network is connected. Meanwhile, the protocol overhead is low enough to be implemented in real networks. Third, QoS routing is NP-complete. Hence, tackling this problem requires heuristics. A common approach is to convert this problem into a shortest path or k-shortest path problem and solve it by using existing algorithms such as Bellman-Ford and Dijkstra algorithms. However, this approach suffers from either high computational complexity or low success ratio in finding the feasible paths. Hence, a new problem, All Hops k-shortest Path (AHKP), is introduced and investigated. Based on the solution to AHKP, an efficient self-adaptive routing algorithm is presented, which

can guarantee in finding feasible paths with fairly low average computational complexity. One of its most distinguished properties is its progressive property, which is very useful in practice: it can self-adaptively minimize its computational complexity without sacrificing its performance. In addition, routing without considering the staleness of link state information may generate a significant percentage of false routing. Our proposed routing algorithm is capable of minimizing the impact of stale link state information without stochastic link state knowledge. Fourth, the computational complexities of existing ε-approximation algorithms are linearly proportional to the adopted linear scaling factors. Therefore, two efficient algorithms are proposed for finding the optimal (the smallest) linear scaling factor such that the computational complexities are reduced. Finally, an efficient algorithm is proposed for finding the least hop(s) multiple additive constrained path for the purpose of saving network resources.

QUALITY-OF-SERVICE PROVISIONING IN HIGH SPEED NETWORKS: ROUTING PERSPECTIVE

by
Gang Cheng

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Electrical Engineering

Department of Electrical Engineering

May 2005

APPROVAL PAGE

QUALITY-OF-SERVICE PROVISIONING IN HIGH SPEED NETWORKS: ROUTING PERSPECTIVE

Gang Cheng

Dr. Nirwan Ansari, Dissertation Advisor — Date
Professor of Electrical and Computer Engineering, NJIT

Dr. Dionissios Karvelas, Committee Member — Date
Special Lecturer of Computer Science, NJIT

Dr. Symeon Papavassiliou, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Indra Widjaja, Committee Member — Date
MTS, Lucent Bell Laboratories

Dr. Mengchu Zhou, Committee Member — Date
Professor of Electrical and Computer Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Gang Cheng

Degree: Doctor of Philosophy

Date: May 2005

Undergraduate and Graduate Education:

- Master of Science in Signal and Information Processing,
  Beijing University of Posts and Telecommunications, 2000
- Bachelor of Science in Information Engineering,
  Beijing University of Posts and Telecommunications, 1997

Major: Electrical and Computer Engineering

Presentations and Publications:

G. Cheng and N. Ansari, "Adaptive QoS provisioning by Pricing Incentive QoS Routing for Next Generation Networks," submitted to *Computer Networks*.

G. Cheng and N. Ansari, "Finding A Least Hop(s) Path Subject to Multiple Additive Constraints," submitted to *Computer Communication*.

L. Zhu, G. Cheng, K. Xu, and N. Ansari, "Edge-based Adaptive Queue Management (EAQM)," submitted to *IEE Proc. Communications*.

G. Cheng and N. Ansari, "An Information Theory Based Framework for Optimal Link State Update," *IEEE Communications Letters*, vol. 8, pp. 692-694, 2005

G. Cheng, L. Zhu, and N. Ansari, "A New Deterministic Traffic Model for Core-stateless Scheduling," submitted to *IEEE Transactions on Communications*.

G. Cheng, L. Zhu, and N. Ansari, "Lower the Computational Complexities of ε-approximation Algorithms With the Optimal Linear Scaling Factor," submitted to *IEEE Transactions on Communications*.

N. Ansari, G. Cheng and, R.N. Krishnan, "Efficient and Reliable Link State Information Dissemination," *IEEE Communications Letters*, vol. 8, pp. 317 - 319, 2004.

G. Cheng and N. Ansari, "Finding All Hops Shortest Paths," *IEEE Communications Letters*, vol. 8, pp. 122-124, 2004.

L. Zhu, G. Cheng and N. Ansari, "Local stable condition for random exponential marking," *IEE Proc. Communications*, vol. 150, pp. 367-370, 2003.

G. Cheng, Y. Tian and N. Ansari, "A New QoS Routing Framework for Solving MCP," *Special Issue on Internet Technology, IEICE Trans. on Communications*, Vol. E86-B, No. 2, pp. 534-541, 2003.

L. Zhu, G. Cheng and N. Ansari, "Delay Bound of Youngest Serve First Aggregated Packet Scheduling," *IEE Proc. Communications*, Vol. 150, No. 1, pp. 6-10, 2003.

G. Cheng and N. Ansari, "On Multiple Additively Constrained Path Selection," *IEE Proc. Communications*, Vol. 149, No. 5, pp.237-241, 2002.

G. Cheng and N. Ansari, "Minimizing the Impact of Stale Link State Information on QoS Routing," submitted to *IEEE GlobeCom'05*.

G. Cheng and N. Ansari, "A Framework for Finding the Optimal Linear Scaling Factor of ε-approximation Solutions," to be presented at *IEEE ICC'05*.

G. Cheng and N. Ansari, "ROSE: A Novel Link State Information Update Scheme for QoS Routing," to be presented at *IEEE HPSR05*.

G. Cheng and N. Ansari, "Achieving 100% Success Ratio In Finding The Delay Constrained Least Cost Path," *IEEE GlobeCom'04*, vol. 3, pp. 1505-1509, 2004.

G. Cheng, Y. Tian, K. Xu, and N. Ansari, "Core-stateless Proportionally Fair Queuing for Assured Forwarding," *IEEE GlobeCom'04*, vol. 2, pp. 732-736, 2004.

G. Cheng and N. Ansari, "A New Heuristics for Finding the Delay Constrained Least Cost Path," *Proc. IEEE GLOBECOM '03*, vol. 7, pp. 3711-3715, 2003.

G. Cheng, L. Zhu, and N. Ansari, "A New Traffic Model for Core-stateless Scheduling," *Proc. IEEE GLOBECOM '03*, vol. 6, pp. 3206-3210, 2003.

G. Cheng and N. Ansari, "Finding All Hops k-Shortest Paths," *Proc. 2003 IEEE PACRIM '03*, vol. 1, pp. 474-477, 2003.

G. Cheng and N. Ansari, "A Theoretical Framework for Selecting the Cost Function for Source Routing," *Proc. IEEE ICC'03*, vol. 1, pp. 631-635, 2003.

G. Cheng, Y. Tian and N. Ansari, "An Efficient Iterative Source Routing Algorithm," *Proc. 2003 37th Conference on Information Sciences and Systems*, 2003.

L. Zhu, G. Cheng, N. Ansari, Z. Sahinoglu, A. Vetro, and H. Sun, "Proxy Caching for Video on Demand Systems in Multicasting Networks," *Proc. 2003 37th Conference on Information Sciences and Systems*, 2003.

N. Ansari, G. Cheng, S. Israel, Y. Luo, et. al., "QoS Provision with Path Protection for Next-Generation SONET," *Proc. 2002 IEEE ICC'02*, vol. 1, pp. 2152 -2156, 2002.

to my wife Fenghua.

ACKNOWLEDGMENT

This dissertation could not have been completed without the support and inspiration of many people. First of all, I have been fortunate to have Dr. Nirwan Ansari as my advisor. Dr. Ansari simultaneously provided me the freedom to work on what I wanted and the guidance that enabled me to succeed in my work. He not only taught me how to become a better researcher, but also helped me to become a better person. His engaging arguments and strong feedback have contributed greatly to this dissertation. I hope and look forward to continued collaboration with him in the future. Next, I would like to thank my wife, Fenghua, for her patience and persistent encouragement. For behind-the-scenes support, I am forever indebted to my parents for their endless love. I am grateful to all the friends and colleagues with whom I spent my time at New Jersey Institute of Technology. Thanks to my friends, Li Zhu, Kai Xu, and Yie Tian, with whom I spent many hours discussing various exciting topics. Finally, I would like to thank my dissertation committee members (Dr. Dionissios Karvelas, Dr. Symeon Papavassiliou, Dr. Indra Widjaja, and Dr. Mengchu Zhou) for spending their valuable time in my Ph.D exams and making constructive comments.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Overview

As the Internet evolves into a globe commercial infrastructure, it is expected to accommodate a variety of applications that require quality-of-service (QoS) such as video conferencing, interact TV, Internet telephony. However, today's internet only provides one simple service: best-effort datagram delivery, in which data packets may experience unpredictable delay and packet loss rate [1], and arrive at the destination out of order. Hence, the best effort service architecture is not enough to meet the demands of the emerging applications, and more sophisticated mechanisms, e.g., the integrated service architectures [2]- [5] and the differentiated service architecture [6], are urgent to provide less oscillatory and more predictable services to applications. Over the past two decades, the subject of providing QoS guarantees in packet-switched networks has been a major research focus in the Internet research community. In this dissertation, we focus on the QoS routing problem, and propose high performance solutions to both link state information update and routing strategies.

The basic function of QoS routing is to find paths [7]- [8] (for unicast application) or trees [9]- [12] (for multicast applications) that satisfy the given sets of performance requirements according to link state information in a network. There are many reasons that make QoS routing difficult. First of all, different applications such as Interactive TV and video conferencing have different QoS requirements. In the meantime, it has been proved that multiple constrained path selection is NP-complete [13], i.e., it is intractable to select a path subject to multiple constraints in a network. Therefore, tackling this problem needs heuristics. Many solutions have already been proposed in the literature. However, they suffer either the high computational complexity or low success ratio in finding the

feasible paths. Second, due to the highly dynamic nature of link QoS metrics [14]- [16], it is hard, if not impossible, to maintain accurate link state information of the whole network. Otherwise, intolerable amount of network resources is consumed on distributing link state information which, in turn, increases the difficulty of designing high performance routing schemes. Therefore, for the purpose of minimizing protocol overhead, current link-state routing protocols such as OSPF [17] recommend that the link state is updated periodically with large intervals. For instance, a link updates its state information every 30 minutes and disseminates it simply by flooding in OSPF. However, from the perspective of QoS routing, there are two major drawbacks of adopting OSPF as the link state update scheme [17] in the Internet. First, the link state information is often outdated, which may greatly degrade the effectiveness of the QoS routing algorithms. Second, since link state information is distributed by flooding, it is not scalable, i.e., the protocol overhead increases with the number of links. Hence, in order to implement QoS routing in high speed networks, we need to investigate the issue of how to update link state information efficiently and effectively.

As mentioned above, there are two critical issues to QoS routing: state distribution and routing strategy. Routing strategy is used to find a feasible paths (trees) meeting the QoS requirements. According to how the state information is maintained and how the search of paths is carried out, routing strategies can be classified into three categories [18]: source routing [19], distributed routing [20]- [25], and hierarchical routing [26]. We mainly focus on source routing in this dissertation and assume that accurate link state information is available. State distribution addresses the issue of exchanging the state information throughout the network and can be further decomposed into two subproblems: when to update and how to disseminate the link state information.

1.2 Contributions and Organization

Among the contributions of this dissertation are a highly efficient rate-distortion analysis based link state update scheme, and two routing strategies respectively for the Multiple Additive Constrained Path (MACP) selection and Delay Constrained Least Cost (DCLC) path selection. In addition, along with an optimal comparison based solution, we introduce and investigate a new problem: All Hops Shortest Path (AHSP) selection, and provide a tight lower bound on the worst-case computational complexity on its comparison based solutions. Through extensive simulations and theoretical analysis, we show that our proposed link state update scheme greatly outperforms the state-of-the-art solutions both in terms of the update rate and accuracy of the link state information, where the update rate is defined as the average number of link updates in an unit time, and the accuracy of link state information as the probability of routing failures introduced by the staleness of link state information. We also present a simple but efficient and reliable scheme for disseminating link state information to all nodes over a Tree-based Reliable Topology (TRT), that guarantees that in the case of a single link failure, link state information is still reachable to all nodes as long as the link is not an edge cut of the network. We experimentally show that our proposed routing strategies greatly outperform their contenders in terms of the worst-case computational complexity and the success ratio in finding the feasible paths or the optimal feasible paths. Having observed that the computational complexities of the ε-approximation algorithms using the linear scaling technique are linearly proportional to the linear scaling factors, we investigate the issue of finding the optimal (the smallest) linear scaling factor to reduce the computational complexities, Two algorithms, Optimal Linear Scaling Algorithm (OLSA) and Transformed Optimal Linear Scaling Algorithm (T-OLSA), are proposed. We theoretically show that OLSA and T-OLSA are both highly efficient in reducing the computational complexities of linear scaling based ε-approximation algorithms. QoS routing is NP-complete. Hence, tackling this problem requires heuristics. A common approach is to convert this problem into a shortest path

or a k-shortest path problem and solve it with existing algorithms, e.g., Bellman-Ford and Dijkstra algorithms. However, this approach suffers either high computational complexity or low success ratio in finding the feasible paths. Hence, we introduce and investigate a new problem, All Hops k-shortest Path (AHKP). Based on the solution to AHKP, we propose an efficient self-adaptive routing algorithm, which can guarantee in finding feasible paths with fairly low average computational complexity. One of its most distinguished properties is its progressive property, which is very useful in practice: it can self-adaptively minimize its computational complexity without sacrificing its performance. In addition, we show that routing without considering the staleness of link state information may generate significant percentage of false routing. Our proposed routing algorithm is capable of minimizing the impact of stale link state information without stochastic link state knowledge. Finally, we propose an efficient algorithm for finding the least hop(s) multiple additive constrained path for the purpose of saving network resources.

The rest of the dissertation is organized as follows. In Chapter 2, we address the issue of when to update link state information and propose a rate-distortion analysis based update scheme. A simple but reliable link state information dissemination scheme is presented in Chapter 3. The problem of AHSP is introduced and investigated in Chapter 4. Based on an optimal comparison-based solution to AHSP, we develop two high performance routing algorithms for MACP and DCLC, respectively, in Chapter 5. Aiming at reducing the computational complexities of ε-approximation algorithms that use linear scaling techniques, we introduce OLSA and T-OLSA in Chapter 6. We address the issue of minimizing the impact of stale link state information in Chapter 7. In Chapter 8, we introduce and investigate the issue of finding the Least Hop(s) Multiple Additive Constrained Path (LHMACP). Finally, the concluding remarks are given in Chapter 9.

CHAPTER 2

A RATE-DISTORTION ANALYSIS BASED LINK STATE INFORMATION UPDATE SCHEME

2.1 Introduction

Many proposed QoS routing solutions assume that accurate link state information is available to each node. However, this is impossible in real networks. Moreover, to facilitate accurate enough link state information would impose a significant bandwidth and processing overhead on the network resource, i.e., the network resource will be greatly consumed if an update of the link state information is triggered whenever a minor change to the QoS parameters occurs. Current link-state routing protocols such as OSPF [17] recommend that the link state is updated periodically with large intervals. For instance, a link disseminates its state information every 30 minutes in OSPF. Consequently, because of the highly dynamic nature of link state parameters, the link state information known to a node is often outdated. As a result, the effectiveness of the QoS routing algorithms may be degraded significantly. To overcome this problem, several link state update policies (threshold, equal class, and exponential class based update policies) have been proposed in [27]. Given a predefined threshold value (τ), an update is triggered in the threshold based update policy if $|b_c - b_o|/b_o > \tau$, where b_o is the last advertised value of the available bandwidth, and b_c is the current available bandwidth. However, in equal class and exponential class based update policies, the bandwidth is partitioned into classes and an update is triggered whenever the available bandwidth crosses a class boundary. The only difference between them is that the bandwidth is partitioned into classes of equal size ($(0, B)$, $(B, 2B)$,...) in equal class based update policy, while it is partitioned into unequal classes, whose sizes ($(0, B)$, $(B, (f+1)B)$, $((f+1)B, (f^2+f+1)B)$,...) grow geometrically by a factor f, in the exponential class based update policy, where

B is a predefined constant. Recently, many works considering the effects of the stale or coarse-grained information on the performance of QoS routing algorithms were reported in the literature. In [28], extensive simulations were made to uncover the effects of the stale link state information and random fluctuations in the traffic load on the routing and setup overheads. In [29]- [30], the effects of the stale link state information on QoS routing algorithms were demonstrated through simulations by varying the link state update interval. A combination of the periodic and triggered link state update is considered in [31]. Instead of using the link capacities or instantaneous available bandwidth values, Li *et al.* [32] used a stochastic metric, Available Bandwidth Index (ABI), and extended BGP to perform the bandwidth advertising.

As reviewed above, although many link state update policies have been proposed, there is still a lack of a rigorous theoretical foundation. As a result, they may not be efficient enough and may waste the network resource. In this paper, we will provide a theoretical framework for the link state update, from which we will propose a high performance link state update mechanism. We theoretically demonstrate that from the perspective of QoS routing, our proposed link state policy outperforms its contenders in terms of the update rate and false blocking probability of incoming connections. The rest of the paper is organized as follows. The problem is formulated in Section II. In Section III, we provide an insight on the link state update based on information theory. We propose an efficient link state update policy in Section IV, and present the simulation results in Section V. Finally, concluding remarks are given in Section VI.

2.2 Problem Formulation

A key challenge for a network simulation is the selection of the network topology and the traffic patterns. Owing to the constantly changing and decentralized nature of current networks, it is rather difficult to define a typical network topology applicable for exploring any protocols [15]. Moreover, the results of a simulation over different network topologies

with different traffic patterns may vary dramatically. In this paper, the problem of designing an efficient link state update policy in a network is translated into that of finding an update policy for a single link. We further discover that the performance of a link state update policy in a network can be evaluated over a single link. As a result, we successfully avoid the above problems. Moreover, our proposed link state policy is designed without any assumption on the traffic pattern of the network.

Without loss of generality, we only focus on one link state metric (here, we adopt bandwidth) and assume there exists an optimal QoS routing algorithm that is, if the exact link state information is available to each node, always capable of locating a feasible path (a path that meets QoS requirements) as long as it exists. If a route is successfully found for a connection, the corresponding portion of the bandwidth (the requested bandwidth of the connection) on each link it traverses is reserved. In this paper, we ignore the delay of disseminating the link state information, i.e., we assume that all nodes can instantly receive the link state information when an update occurs.

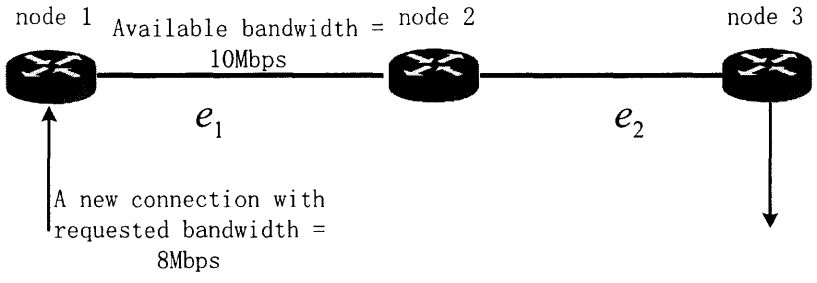

Figure 2.1 A 3-node network consists of nodes 1, 2, and 3. The available bandwidth of link e_1 is 10Mbps. A new connection arrives at node 1, whose requested bandwidth and desitination are 8Mbps and node 3, respectively.

Define a routing failure as the case that the optimal routing algorithm fails to find a feasible path (due to the staleness of the link state information) when, in fact, a feasible path exists, and a setup failure as the case that the algorithm finds a false feasible path (a path

computed by the algorithm as a feasible path but is not feasible in reality). Collectively, both failures are referred to as the false blockings of connections since both failures result from the stale link state information. Note that setup failures cause network resources unnecessarily consumed for reserving the bandwidth over the links (along the false routes), and routing failures cause unnecessary rejections of the connection requests. Ideally, both failures should be minimized. As shown in Fig. 2.1, assume a connection going to node 3 with the requested bandwidth of 8Mbps arrives at node 1, a setup failure occurs if, from the perspective of node 1, the available bandwidth of link e_2 is 10Mbps while the actual one is only 5Mbps (because node 1 thinks that enough network resource is available to accommodate the connection, and accepts the connection), i.e., due to the staleness of the link state information, a setup failure occurs when the available bandwidth of link e_2 known by node 1 (10Mbps) is larger than the actual one (5Mbps), and the amount of the required bandwidth (8Mbps) is just between them (less than the available bandwidth known to a node but larger than the actual one); on the other hand, if the actual available bandwidth of link e_2 is 10Mbps and the one known by node 1 is 5Mbps, a routing failure occurs. Hence, it is impossible to minimize both the number of setup and routing failures at the same time, i.e., there is a trade off. In this case, define c_1 as the cost of a routing failure and c_2 as the cost of a setup failure; our objective is to minimize the total cost of routing and setup failures under a given upper bound (network) on the average bandwidth used for the link state dissemination, which in turn yields an upper bound on the average bandwidth (r) consumed by every link for the link state update. Assume the link state metric of a link is a time independent random process (with memory). Therefore, each link can be viewed as a signal source and all nodes that do not directly connect to this link are receivers. From the perspective of information theory, our task is to minimize the source distortion (the sum of the routing and setup failure costs) for a given transmission rate r (or to minimize the transmission rate r for a given source distortion).

Given a link, assume its available bandwidth known to each node is b_o, and the actual one is b. Further assume the amount of the requested bandwidth of a connection is a random variable x with probability density function $p(x)$. Therefore, define the cost (source distortion) as:

$$\begin{aligned} D &= \frac{c_1}{2}\int_{\min\{b_o,\,b\}}^{\max\{b_o,\,b\}} p(x)dx\left[1-\frac{(b_o-b)}{|b_o-b|}\right] \\ &\quad +\frac{c_2}{2}\int_{\min\{b_o,\,b\}}^{\max\{b_o,\,b\}} p(x)dx\left[1-\frac{(b-b_0)}{|b_o-b|}\right] \\ &= \int_{\min\{b_o,\,b\}}^{\max\{b_o,\,b\}} p(x)dx\{\frac{c_1}{2}\left[1-\frac{(b_o-b)}{|b_o-b|}\right] \\ &\quad +\frac{c_2}{2}\left[1+\frac{(b_o-b)}{|b_o-b|}\right]\}. \end{aligned} \tag{2.1}$$

For simplicity, we only consider the case that $c_1 = c_2 = c$ in this paper, which follows that

$$D = c\int_{\min\{b_o,\,b\}}^{\max\{b_o,\,b\}} p(x)dx. \tag{2.2}$$

Note that the source distortion is linearly proportional to $\int_{\min\{b_o,\,b\}}^{\max\{b_o,\,b\}} p(x)dx$. Next, we provide two **interchangeable** definitions for the optimal link state update policy.

Definition 1 *For a given source distortion to be less than or equal to D, the optimal link state update policy is the one that minimizes the transmission rate r of each link.*

Definition 2 *For a given transmission rate to be less than or equal to r, the optimal link state update policy is the one that minimizes the source distortion.*

Based on the above definitions of the optimal link state update policy, current proposed update policies are not efficient enough. Consider the ones proposed in [27] as an example. Assume $p(x)$ is shown in Fig. 2.2, where $B_{\min}$ and $B_{\max}$ are the lower bound and

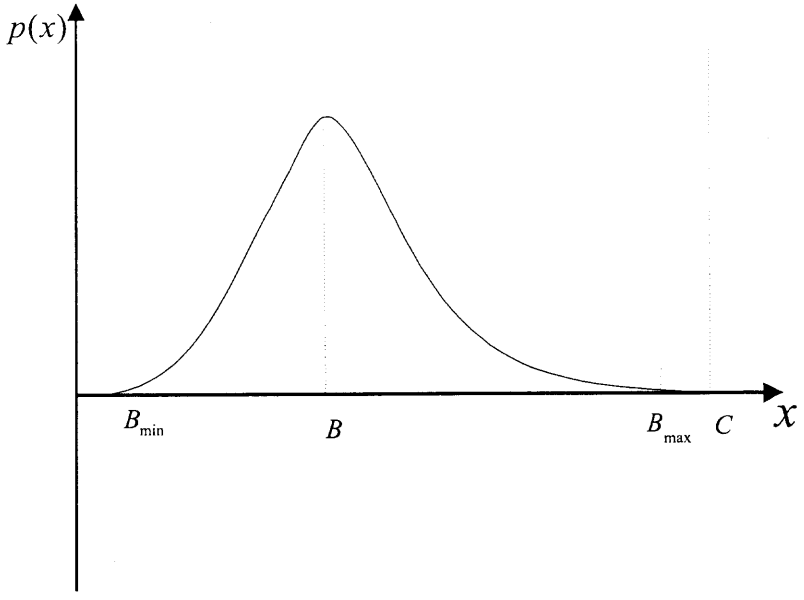

Figure 2.2 The probability density function of the requested bandwidth of connections.

upper bound on the requested bandwidth of connections, respectively ($\int_{B_{\min}}^{B_{\max}} p(x)dx = 1$), and C is the link capacity. Therefore, it can be observed that, when the current (exact) available bandwidth of the link is between 0 and $B_{\min}$ or $B_{\max}$ and C, except for the first time that the available bandwidth falls into these regions, it is not necessary to update the link state information because, no matter how the available bandwidth changes in these two regions, it does not affect the setups of the connections in the network, i.e., the probabilities of setup failures and routing failures are both zero. Moreover, the probability density function of the requested bandwidth attains the maximum value at B; it is intuitive that more accurate link state information is necessary when the available bandwidth is around B so that the source distortion (cost) is minimized than that when the available bandwidth is around 0 or C, i.e., since the source distortion is defined as $c \int_{\min\{b_o, b\}}^{\max\{b_o, b\}} p(x)dx$ which is related to the probability density function (pdf) of x, the requested bandwidth of connections, it is intuitive that, when $p(x)$ is high, small $|b_o - b|$ is preferred (accurate link

state information is preferred), and when $p(x)$ is low, $|b_o - b|$ can be relatively large (coarse link state information is allowed). However, since the three proposed link state update policies are designed without considering the requested QoS parameters of connections, bandwidth is inevitably wasted on updating the link state information because of some unnecessary updates. For example, using the threshold based update policy with a given threshold τ, when the available bandwidth changes from $\frac{B_{\min}}{2(1+\tau)}$ to $\frac{B_{\min}}{2}$, an update is triggered. However, since it is impossible that a connection requests a bandwidth less than $B_{\min}$, it is unnecessary to make such update, i.e., such update is useless with the respect to the requested bandwidth of connections. As a result, the network resource is wasted. By the above example, a link state update policy cannot be universally optimal if it only relies on the information of the available bandwidth, i.e., the knowledge of both the available bandwidth and connection requests is needed for designing an optimal link state update policy. On the other hand, since only the available bandwidth information is considered in most existing works such as [32] and [27], they cannot be optimal and efficient enough.

Note that, in [28], a conclusion was made by simulations that the threshold based update policy does not significantly affect the overall blocking probability of connections. We need to point out that this may be only applicable to the special simulation environment (the inefficient threshold based update policy, large average available bandwidth, and small average bandwidth requested by connections) deployed in [28]: the traffic loads (occupied bandwidth) of links are around $0.75C$, and the bandwidths requested by new connections are all less than $0.1C$ (the average bandwidth requested by connections are no larger than $0.05C$). Note that updating the link state information is necessary only when the available bandwidth is less than $0.1C$. Moreover, in some simulations, the upper bound on the requested bandwidth of connections is only $0.04C$, which further decreases the threshold to update the link state information. Consequently, the threshold based update policy defined in [27] (set τ around 0.5) is rather inefficient with respect to the simulation environment and has little effect on the blocking probability of connections.

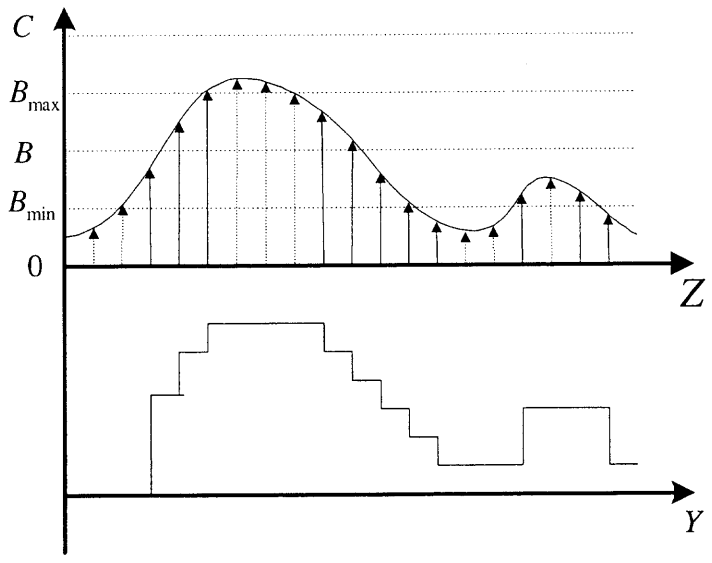

Figure 2.3 A demonstration of z_1, z_2, ... and y_1, y_2,.... The solide arrows represent the corresponding samples that are used for updates while the dash arrows represent the samples that are not used.

In this paper, we will propose an efficient link state update policy that takes into account of the requested QoS parameters of connections.

2.3 An Insight from Information Theory

Many studies have been done to characterize the Internet traffic, revealing interesting facts such as Long-Range Dependence or multi-fractal behaviors [33]-[35]. Therefore, we cannot assume that the source signal (available bandwidth) is memoryless. Hence, each QoS parameter of a link is viewed as a random process with memory that is independent of time in this paper. By applying the sampling theorem, each continuous random process can be converted into an equivalent discrete-time sequence of samples. Therefore, each QoS parameter on a link can be treated as a time independent continuous random sequence with memory. As shown in Fig. 2.3, denote z_1, z_2,..., and y_1, y_2,... as the exact sequence of the available bandwidth and the sequence of the available bandwidth known to a node that is

not directly connected to the link, respectively, where C, $B_{\max}$, B, and $B_{\min}$ are the same as those in Fig. 2.2.

Note that the definition of an optimal link state update policy is the one that minimizes the transmission rate r of each link while limiting the source distortion to be less than or equal to a given constant D. Hence, we can apply the rate-distortion function in information theory to the problem of link state update. Denote the entropy of sequence y_1, y_2,...,y_n as $\mathbf{H}(Y^n)$, and $\mathbf{H}(Y^n|Z^n)$ as the conditional entropy between the sequences of z_1, z_2,..., z_n and y_1, y_2,...,y_n. Therefore,

$$\mathbf{H}(Y^n) = -\int_{Y^n} \cdots \int p(Y^n) \log p(Y^n) dY^n, \tag{2.3}$$

and

$$\mathbf{H}(Y^n|Z^n) = -\int_{Z^n} \cdots \int \int_{Y^n} \cdots \int p(Z^n, Y^n) \log p(Y^n|Z^n) dY^n dZ^n. \tag{2.4}$$

The optimal link state update policy satisfies

$$\begin{aligned} R(D) &= \min_{\xi_D} \left\{ \lim_{n\to\infty} \left\{ \frac{1}{n} \mathbf{I}(Z^n, Y^n) \right\} \right\} \\ &= \min_{\xi_D} \left\{ \lim_{n\to\infty} \left\{ \frac{1}{n} \left[\mathbf{H}(Y^n) - \mathbf{H}(Y^n|Z^n) \right] \right\} \right\} \end{aligned} \tag{2.5}$$

where $R(D)$ is the bandwidth (a single link) used by the optimal link state update policy, $\mathbf{I}(\cdot,\cdot)$ is the mutual information, and ξ_D is the set of transition probabilities from Z^n to Y^n subject to the distortion D, i.e., ξ_D satisfies the following condition

$$\lim_{n\to\infty} \left\{ \frac{c}{n} \sum_{i=1}^{n} \int_{\min\{y_i,z_i\}}^{\max\{y_i,z_i\}} p(x) dx \right\} \leq D. \tag{2.6}$$

By equations 2.5 and 2.6, a lower bound on the transmission rate of the link state update under the constraint of a given source distortion can be computed. In this paper, since no assumption on the distribution of $p(Z^n)$ and $p(x)$ is made, we cannot compute $R(D)$ (in

reality, we can assume $p(Z^n)$ and $p(x)$ are known. For instance, we can assume Z^n is self-similar distributed such as Pareto distribution).

2.4 The Proposed Link State Update Policy

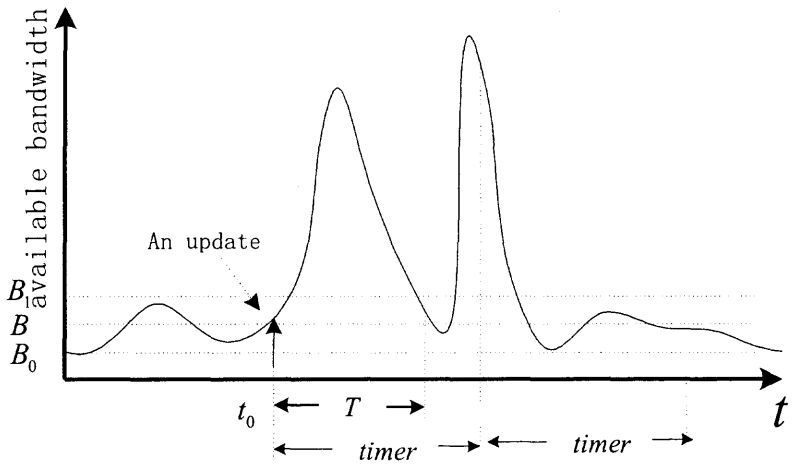

Figure 2.4 The bursty traffic of the Internet.

The key difference between our proposed link state policy and those in [27] is that instead of partitioning the bandwidth into classes of equal sizes or exponentially growing sizes, we divide the bandwidth into classes by taking into account of the requested bandwidth of the connections and the available bandwidth, for the purpose of minimizing the source distortion under a bandwidth constraint for disseminating the link state information. As a result, our proposed link state update policy can avoid the unnecessary updates from the QoS routing perspective, and therefore, performs better than those in [27] in terms of the average bandwidth for disseminating the link state information and the average costs due to routing and setup failures. Our proposed link state update policy consists of the following three steps:

1. Sample the available bandwidth. Different sampling strategies can be adopted. For instance, we can sample the available bandwidth every T_s, where T_s is the sampling interval. We can also compute the available bandwidth upon the acceptance and the end of a connection (only after a route with enough bandwidth is found for a connection, it is accepted).

2. Quantize the samples.

3. If the current quantized value is different from the previous updated one, use the current one for update.

Since data traffic in the current Internet is inherently bursty, a QoS parameter may fluctuate dramatically in a very short time. As a result, some unnecessary updates may occur. For example, as shown in Fig. 2.4, the available bandwidth has been changed substantially in a very short time T, and many updates are triggered. Therefore, for the purpose of saving the bandwidth used for link state update, similar to [27], we can set a clamp down timer (the minimum interval between two consecutive updates) to make a link state update policy less sensitive to the fluctuation of the available bandwidth. However, it should be noted that setting a clamp down timer also has negative impacts on the accuracy of the link state information. As shown in Fig. 2.4, assume the average available bandwidth is B, and in most of the time, the available bandwidth is less than B_1 and larger than B_0. At t_0, an update of link state information is triggered, and at the end of the corresponding timer, another link state information update is triggered because the available bandwidth is very large at that time. Observe that in most of the time of the second timer duration, the available bandwidth is between B_1 and B_0. However, because of the clamp down timer, no link state information update is allowed. Hence, the link state information during this timer period is rather inaccurate. Therefore, the clamp down timer is not recommended here.

Next, we propose how to partition the bandwidth with a given number of classes in order to minimize the source distortion. Assume the number of classes is n, and classes

are $[0, b_1)$, $[b_1, b_2)$, ..., $[b_{n-1}, C)$ ($b_0 = 0$ and $b_n = C$). Further assume B_0, B_1, ..., B_{n-1} are quantized values (we update the link state information with B_i if the current available bandwidth is in $[b_i, b_{i+1})$). Therefore, we need to minimize the distortion

$$d = \int_0^C f(x)e(\widetilde{x}, x)dx, \tag{2.7}$$

where $e(\widetilde{x}, x) = \int_{\min\{\widetilde{x},x\}}^{\max\{\widetilde{x},x\}} p(x)dx$, $\widetilde{x}$ is the corresponding quantized value of x, and $f(x)$ is the probability density function of the available bandwidth. Hence,

$$\begin{aligned} d &= \int_0^C f(x)e(\widetilde{x}, x)dx \\ &= \sum_{i=0}^{n-1} \int_{b_i}^{b_{i+1}} \left(\int_{\min\{B_i,x\}}^{\max\{B_i,x\}} p(y)dy \right) f(x)dx \\ &= \sum_{i=0}^{n-1} \int_{b_i}^{B_i} \left(\int_x^{B_i} p(y)dy \right) f(x)dx + \sum_{i=0}^{n-1} \int_{B_i}^{b_{i+1}} \left(\int_{B_i}^{x} p(y)dy \right) f(x)dx \end{aligned} \tag{2.8}$$

Note that

$$\int_0^C f(x)dx = 1, \tag{2.9}$$

and

$$\int_0^C p(x)dx = 1. \tag{2.10}$$

Therefore, we can form a Lagrangian Relaxation by incorporating the constraints, Eqs. 2.9 and 2.10, into Eq. 2.8:

$$L(B_0, B_1, ..., B_{n-1}, b_1, b_2, ..., b_{n-1}) = d - \lambda_1 \left(\int_0^C f(x)dx - 1 \right) - \lambda_2 \left(\int_0^C p(x)dx - 1 \right). \tag{2.11}$$

So,

$$\frac{\partial L}{\partial b_i} = f(b_i) \left[\int_{B_{i-1}}^{b_i} p(x)dx - \int_{b_i}^{B_i} p(x)dx \right], \tag{2.12}$$

and

$$\frac{\partial^2 L}{\partial b_i^2} = f^{'}(b_i)\left[\int_{B_{i-1}}^{b_i} p(x)dx - \int_{b_i}^{B_i} p(x)dx\right] + 2f(b_i)p(b_i).$$

Note that

$$\int_{B_{i-1}}^{b_i} p(x)dx - \int_{b_i}^{B_i} p(x)dx = 0 \Rightarrow \frac{\partial L}{\partial b_i} = 0, \tag{2.13}$$

and

$$\int_{B_{i-1}}^{b_i} p(x)dx - \int_{b_i}^{B_i} p(x)dx = 0 \Rightarrow \frac{\partial^2 L}{\partial b_i^2} = 2f(b_i)p(b_i) \geq 0. \tag{2.14}$$

Hence, $\forall i$, by letting

$$\int_{B_{i-1}}^{b_i} p(x)dx - \int_{b_i}^{B_i} p(x)dx = 0, \tag{2.15}$$

$L(B_0, B_1, ..., B_{n-1}, b_1, b_2, ..., b_{n-1})$ is minimized, i.e., the source distortion is minimized. Moreover,

$$\frac{\partial L}{\partial B_i} = p(B_i)\left(\int_{b_i}^{B_i} f(x)dx - \int_{B_i}^{b_{i+1}} f(x)dx\right), \tag{2.16}$$

and

$$\frac{\partial^2 L}{\partial B_i^2} = p^{'}(B_i)\left(\int_{b_i}^{B_i} f(x)dx - \int_{B_i}^{b_{i+1}} f(x)dx\right) + 2p(B_i)f(B_i). \tag{2.17}$$

By letting

$$\int_{b_i}^{B_i} f(x)dx - \int_{B_i}^{b_{i+1}} f(x)dx = 0, \tag{2.18}$$

$$\begin{aligned}\frac{\partial^2 L}{\partial B_i^2} &= p^{'}(B_i)\left(\int_{b_i}^{B_i} f(x)dx - \int_{B_i}^{b_{i+1}} f(x)dx\right) + 2p(B_i)f(B_i) \\ &= 2p(B_i)f(B_i) \geq 0,\end{aligned} \tag{2.19}$$

and

$$\frac{\partial L}{\partial B_i} = p(B_i)\left(\int_{b_i}^{B_i} f(x)dx - \int_{B_i}^{b_{i+1}} f(x)dx\right) = 0, \tag{2.20}$$

$L(B_0, B_1, ..., B_{n-1}, b_1, b_2, ..., b_{n-1})$ is minimized. So, the optimal partition of the bandwidth can be achieved by solving Eqs. (2.15) and (2.18). Observe that, if $f(x) = p(x)$, from Eqs. (2.15) and (2.18), we have

$$\int_{B_{i-1}}^{b_i} f(x)dx - \int_{b_i}^{B_i} f(x)dx = 0 \tag{2.21}$$

and

$$\int_{b_i}^{B_i} f(x)dx - \int_{B_i}^{b_{i+1}} f(x)dx = 0. \tag{2.22}$$

It follows that $\forall i, j < n$ $(i, j > 1)$

$$\int_{b_{i-1}}^{b_i} f(x)dx = \int_{b_{j-1}}^{b_j} f(x)dx. \tag{2.23}$$

Then,

$$\int_0^C f(x)dx = 1 \Rightarrow \forall i, \int_{b_{i-1}}^{b_i} f(x)dx = \frac{1}{n}. \tag{2.24}$$

Therefore, when both the requested bandwidth and the available bandwidth of connections are uniformly distributed from 0 to C, the partition of the bandwidth of the equal class based update policy proposed in [27] is optimal. With Eq. (2.8), we can further derive the minimized source distortion when the number of classes of bandwidth is given. Assume the available bandwidth is uniformly distributed, i.e., $f(x) = \frac{1}{C}$. Therefore, by Eq. (2.18), $B_i - b_i = b_{i+1} - B_i$, for all $0 \leq i < n$. Hence, by Cauchy Mean-value Theorem and Eq.

(2.15),

$$\begin{aligned} & \int_{B_{i-1}}^{b_i} p(x)dx = \int_{b_i}^{B_i} p(x)dx \\ \Rightarrow \quad & p(\xi_{i-1})(b_i - B_{i-1}) = p(\delta_i)(B_i - b_i) \\ \Rightarrow \quad & p(\xi_{i-1})(b_i - b_{i-1}) = p(\delta_i)(b_{i+1} - b_i), \end{aligned} \tag{2.25}$$

where $\forall i$, $B_{i-1} \leq \xi_{i-1} \leq b_i$ and $b_i \leq \delta_i \leq B_i$. Hence, if $p(\delta_i) > p(\xi_{i-1})$, then $b_{i+1} - b_i < b_i - b_{i-1}$, i.e., the size of the ith class is less than that of the $(i-1)$th class, which conforms to our claim that relatively accurate link state information is preferred at the point (available bandwidth) where the value of the probability density function of the requested bandwidth of connections is large.

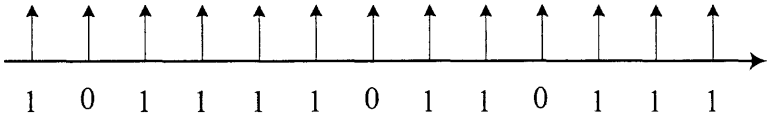

Figure 2.5 n example of the link state update with only two classes.

Intuitively, the larger the number of classes, the more sensitive our link state update policy is to the fluctuation of the available bandwidth is. As a result, the bandwidth required for updating the link state information is larger. Therefore, given an upper bound on the bandwidth for link state update, there accordingly exists an upper bound on the number of classes. In this paper, we approximately compute the bandwidth used for link state update under the condition that the number of classes is n. Denote $p(B_i)$ as the probability that the quantized value of a sample is B_i; $q(B_i)$ as the probability that the link state information is updated with B_i, $i = 1, 2, ..., n-1$; $p(B_j|B_i)$ as the transition probability of two consecutive (quantized value of) samples from B_i to B_j, and T_s as the average interval between two consecutive samples. Note that $q(B_i)$ is different from $p(B_i)$ because B_i is used for an update only when the previous update (not the previous sample) B_j is different from B_i.

For example, as shown in Fig. 2.5, assume there are only two classes and two quantized values B_0 and B_1, $p(B_0) = 0.2, p(B_1) = 0.8$, and $p(B_0|B_0) = 0$ (the transition probability that the current quantized value is B_0 under the condition that the previous one is also B_0). Observe that before and after each B_0, there must be a sample with value B_1 (because $p(B_0|B_0) = 0$). Since an update can occur only when two consecutive samples are different from each other, $q(B_0) = 0.5$, and $q(B_1) = 0.5$. As mentioned before, the sample sequence is a random sequence with memory. Note that a sample is used for update only when it differs from the latest update, i.e., whether a sample is used for update only depends on the previous update. Therefore, for simplicity, we approximately assume that the sample sequence has the Markov memory structure, i.e., $p(B_i|B_{i-1}, B_{i-2}, ..., B_0) = p(B_i|B_{i-1})$, where $p(B_i|B_{i-1}, B_{i-2}, ..., B_0)$ is the conditional probability that, under the condition that the previous samples are $B_{i-1}, B_{i-2}, ..., B_0$, the ith sample is B_i. Therefore, we have

$$\mathbf{p} \begin{pmatrix} p(B_0|B_0) & p(B_1|B_0) & \cdots & p(B_{n-1}|B_0) \\ p(B_0|B_1) & p(B_1|B_1) & \cdots & p(B_{n-1}|B_1) \\ \vdots & & \ddots & \\ p(B_0|B_{n-1}) & p(B_1|B_{n-1}) & \cdots & p(B_{n-1}|B_{n-1}) \end{pmatrix} = \mathbf{p}, \tag{2.26}$$

where $\mathbf{p} = (p(B_0), p(B_1), ..., p(B_{n-1}))$. Accordingly, the sequence of updates exhibits the Markov memory structure. Hence, assuming the transition probability of updates from B_i to B_j is $q(B_j|B_i)$,

$$\mathbf{q} \begin{pmatrix} q(B_0|B_0) & q(B_1|B_0) & \cdots & q(B_{n-1}|B_0) \\ q(B_0|B_1) & q(B_1|B_1) & \cdots & q(B_{n-1}|B_1) \\ \vdots & & \ddots & \\ q(B_0|B_{n-1}) & q(B_1|B_{n-1}) & \cdots & q(B_{n-1}|B_{n-1}) \end{pmatrix} = \mathbf{q}, \tag{2.27}$$

where $\mathbf{q} = (q(B_0), q(B_1), ..., q(B_{n-1}))$. Since a sample is used for update only when it is different from the previous update, $q(B_i|B_i) = 0$ for all $0 \leq i < n-1$. Therefore,

$$q(B_j|B_i) = \frac{p(B_j|B_i)}{1 - p(B_i|B_i)}, \; i \neq j. \tag{2.28}$$

Hence, we can compute $\mathbf{q}$ by solving Eq. (2.27).

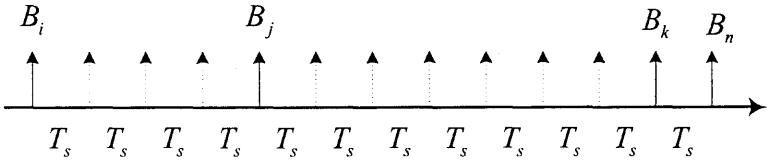

Figure 2.6 Solid arrows represent the samples that are used for link state update and dash arrows are not.

As shown in Fig. 2.6, assume that the average number of samples between any two consecutive updates is m. Accordingly, the average interval between any two consecutive updates is $(m+1)T_s$, and the average number of updates in a unit time, or the update rate, is $\frac{1}{(m+1)T_s}$. Assume that the size of the packets used for link state update is L_u, then the bandwidth used for link state update (a single link) is $\frac{L_u}{(m+1)T_s}$, and the number of link state information packets received by each node of a network in a unit time is $\frac{E}{(m+1)T_s}$, where E is the number of links in the network. Next, we compute m under the assumption that $p(B_j|B_i)$ and $q(B_i)$ $(0 \leq i, j < n)$ are known. Note that an update occurs if the current sample is in a different class from the last update. Therefore, if the current update value is B_i, the average number of samples till the next update is

$$\sum_{k=0}^{\infty} k p^k(B_i|B_i)(1 - p(B_i|B_i)). \tag{2.29}$$

Hence, the average number of samples between any two consecutive updates is

$$m = \sum_{i=0}^{n-1} q(B_i) \left(\sum_{k=0}^{\infty} k p^k(B_i|B_i)(1 - p(B_i|B_i)) \right), \tag{2.30}$$

and thus the average bandwidth used for link state update can be computed.

2.5 Simulations

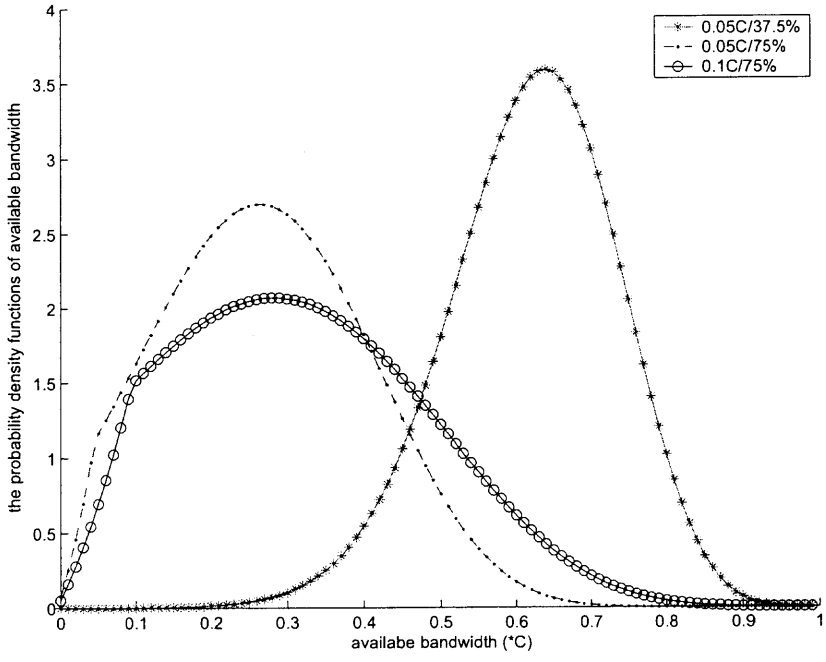

Figure 2.7 The distributions of the available bandwidth.

Note that given a blocking probability p of a connection on every single link and without considering the relation among links, its overall blocking probability to traverse through an $h-$hop path is: $1-(1-p)^h$. Moreover, the number of link state updates of a network is simply the sum of the updates of all the links in the network because each link updates its own link state information independently. Therefore, the performance of a link state update policy in a network can be reflected by its performance on a single link. Hence, instead of conducting simulations in a network, we choose an easier alternative: evaluating the performance of our proposed link state update by comparing it with those in [27] (except the threshold based update policy because, as mentioned earlier, it is rather

ineffective) on a single link. For completeness, we briefly review the equal class based and exponential class based update policies.

Definition 3 *Equal class based update policy [27] is characterized by a constant B which is used to partition the available bandwidth operating region of a link into multiple equal size classes:* $(0, B)$, $(B, 2B)$,..., *etc. An update is triggered when the available bandwidth on an interface changes to a class that is different from the one at the time of the previous update.*

Definition 4 *Exponential class based update policy [27] is characterized by two constants B and f (*$f > 1$*) which are used to define unequal size classes:* $(0, B)$, $(B, (f+1)B)$, $((f+1)B, (f^2+f+1)B)$,..., *etc. An update is triggered when a class boundary is crossed.*

Note that when the available bandwidth fluctuates around a class boundary, many meaningless updates may be triggered. In order to dampen such oscillatory behavior, the two class based policies are augmented by a hysteresis mechanism in [27]: the generation of an update is suppressed until the available bandwidth reduces sufficiently to cross beyond the middle value of the new classes. Such rule is not applied when the available bandwidth increases and crosses a class boundary.

We adopt two performance indices for the purpose of comparison: the update rate (average number of updates in a unit time) and the false blocking probability of connections, which are respectively defined below:

$$\frac{\text{The total number of updates}}{\text{Total simulation time}}, \tag{2.31}$$

$$\frac{\text{Total number of false blockings of connections}}{\text{Total number of connection requests}}, \tag{2.32}$$

where the total simulation time is the total number of unit times simulated. Similar to [28], the arrivals of connection requests are generated with a Poisson process with rate $\lambda = 1$

and the duration of each connection is of a standard Pareto distribution with $\alpha = 2.5$ (cumulative distribution of the standard Pareto distribution is $F_x(x) = 1 - (\frac{\beta}{x})^{\alpha}$, where α is the shape parameter and β is the scale parameter). Hence, the average duration of a connection is $l = \frac{\alpha\beta}{\alpha-1}$ (the mean of the standard Pareto distribution). By [28], the average traffic load of the link is $\rho = \lambda lb/R$, where b is the average requested bandwidth of a connection. For the purpose of comparison, we do not set the clamp down timer in our simulations. Upon the acceptance and the end of a connection, the available bandwidth is re-computed. The bandwidth requested by each connection is uniformly distributed in $[0, B_{\max}]$. Note that for a single class based link state update policy, the larger number of the classes the bandwidth is partitioned into, the more accurate the link state information is, implying the lower false blocking probability of connections, while the more sensitive it is to the fluctuation of the available bandwidth, thus resulting in a larger update rate. Hence, we can claim that policy 1 outperforms policy 2 if and only if, for any given number of classes to policy 2, an appropriate number of classes can always be found for policy 1 such that it achieves better performance on both the update rate and false blocking probability of connections. By extensive simulations, we found that our proposed link state update policy outperforms the equal and exponential class based link state update policies for any given number of classes. In this paper, due to the page limit, we only selectively present the simulation results of the cases that the numbers of classes of the equal class based update policy are 5 and 10 ($B = 0.1C$ and $0.2C$), and $B = 0.05C$ and $f = 2$ for the exponential class based update policy (the number of classes is 4). The number of classes we used for comparison in our proposed link state update policy are 2 and 3.

As the first step of our proposed link state update policy, we compute the probability density function of the available bandwidth as shown in Fig. 2.7, where the traffic loads are 37.5% and 75%, and $B_{\max} = 0.05C$ and $0.1C$, respectively (we denote $0.1C/75\%$ as the case that $B_{\max} = 0.1C$ and the traffic load is 75%). Note that the probability that the available bandwidth is less than $B_{\max}$ is very small (the integration of the probability

density function of the available bandwidth from 0 to $B_{\max}$, i.e., the probability that the available bandwidth is no larger than $B_{\max}$, is less than 5%). Hence, the update rate of our proposed link state update policy is low.

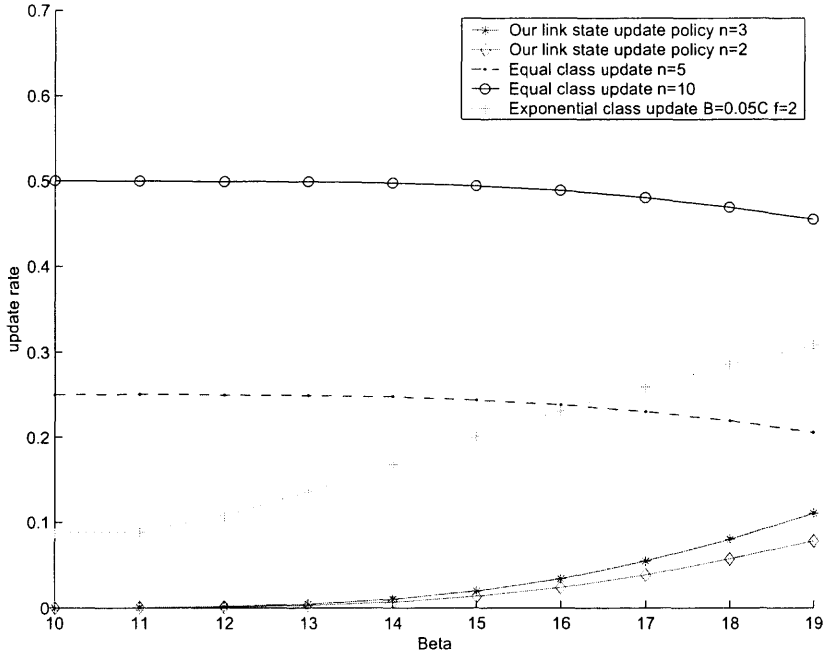

Figure 2.8 Update rates of policies when B is 0.05C.

Figs. 2.8-2.11 illustrate our simulation results, in which n denotes the number of classes and $Beta$ denotes the parameter β. In both simulations, our proposed link state update policy can achieve much better performance than others, i.e., our proposed link state update policy achieves lower blocking probabilities with lower update rates than others, implying that our proposed link state update is more practical than the equal and exponential class based link state update policies in terms of the update rate and false blocking probability of connections.

Finally, it should be noted again that although our simulations are conducted on a single link, as long as no cooperation between links is used for updating the link state

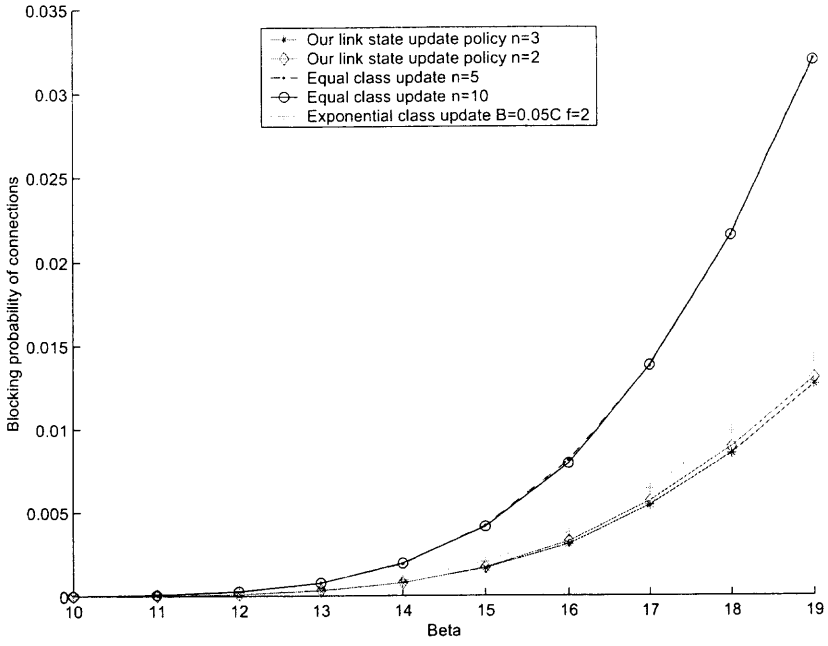

Figure 2.9 False blocking probabilities of policies when B is 0.05C.

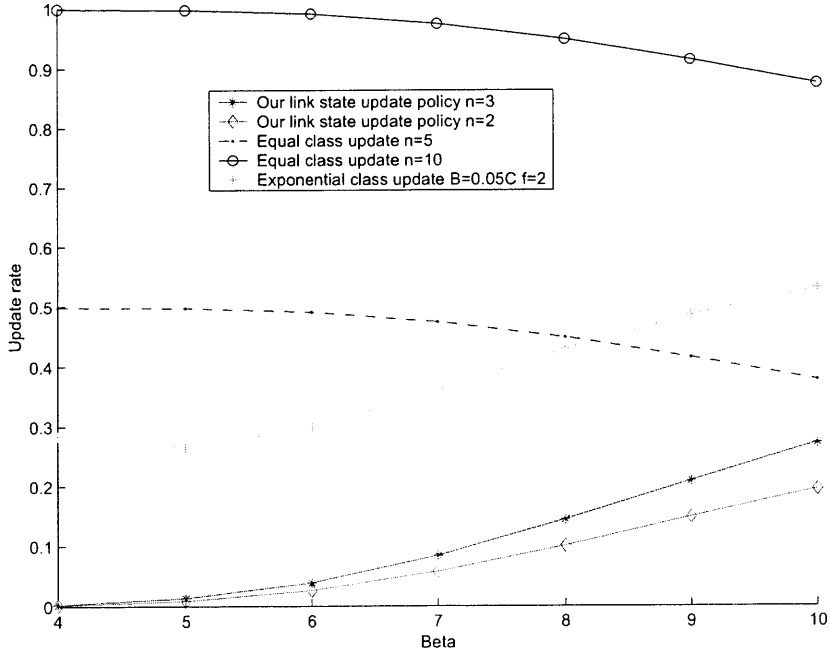

Figure 2.10 Update rates of policies when B is 0.1C.

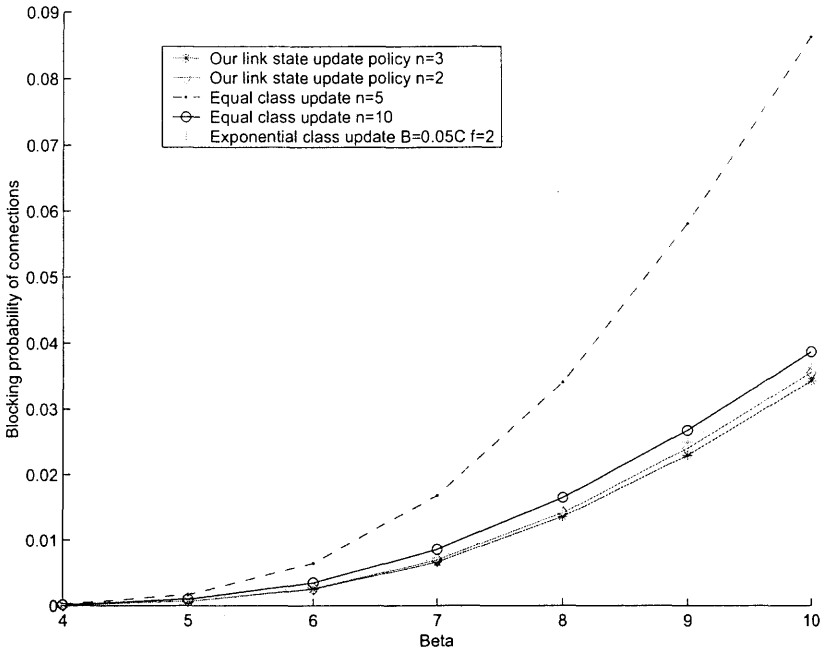

Figure 2.11 False blocking probability of connections when B is 0.1C.

information, we can equally claim that our proposed link state information update scheme outperforms others from the perspective of a whole network.

2.6 Summary

In this chapter, we have proposed an efficient link state update policy. Through theoretical analysis and extensive simulations, we have shown that it greatly outperforms its contenders, i.e., it achieves a much lower false blocking probability with a very low update rate. As a result, we can increase the performance of QoS routing using the proposed link state update policy.

CHAPTER 3

A RELIABLE LINK STATE INFORMATION DISSEMINATION SCHEME

3.1 Introduction

Distributing link state information may introduce a significant protocol overhead on the network resource. Many existing link-state routing protocols recommend that link state information is disseminated by simply flooding or flooding-like approaches [36], [17]. These kinds of approaches ensure that all routers within a link state domain converge on the same link state information within a finite period of time; they are also robust, i.e., in the case of link failures and node failures, link state information is still reachable to all nodes as long as the network is connected. On the other hand, because of the poor scalability of flooding, a large update interval has to be adopted in order to reduce the protocol overhead on the network resource. For instance, a link disseminates its state information every 30 minutes in OSPF. Consequently, because of the highly dynamic nature of link state parameters, the link state information known to a node is often outdated. Hence, the effectiveness of the QoS routing algorithms may be degraded significantly.

To overcome the problems of disseminating link state information by flooding or flooding-like approaches, many tree based link state dissemination schemes have been proposed. Moy [37] has proposed to distribute link state information over a subset of the network topology. Specifically, link state information is distributed over a spanning tree, instead of flooding over the whole network. In order to reduce the communication cost for maintaining the trees, a protocol [38], called Topology Broadcast based on Reverse Path Forwarding (TBRPF), uses the concept of Reverse Path Forwarding (RPF) to broadcast link-state updates in the reverse direction along the spanning tree formed by the minimum-hop paths from all nodes to the source of the update. As reviewed above, it can be observed that the protocol overhead of the tree based approaches is far less than

that of flooding. However, they suffer the reliability problem, i.e., in the case of a single link failure, the tree (subnet) is splitted into two parts and link state information is not reachable to some nodes anymore, even though the network is still connected. Hence, existing proposals have one or more of the drawbacks of poor scalability, poor reliability, and slow convergence.

In this chapter, we propose a new scheme for distributing link state information that possesses the advantages of fast convergence, reliability, and scalability.

3.2 Proposed Scheme

In this section, a reliable, fast convergent, and scalable link state information dissemination scheme is proposed. We first define the reliable link state information dissemination below:

Definition 5 *Given a network topology $G(N, E)$, where N is the set of nodes and E is the set of links, a link state information dissemination scheme is reliable if for any link e, as long as $\{e\}$ is not a minimum edge cut of $G(N, E)$, link state information is still reachable to all nodes in the case that e is broken.*

By the above definition, it can be observed that the scheme of simply flooding link state information over the whole network is reliable, while distributing link state information over a spanning tree is not when the network is not a tree. However, flooding link state information over a network introduces heavy burden on the network resource and is rather inefficient. Hence, one of the key issues of designing a reliable and efficient link state information dissemination scheme is to find an appropriate subnet such that the scheme is reliable and the protocol overhead is minimized. In this chapter, we introduce a new term, Reliable Topology (RT), over which the scheme of simply flooding link state information is reliable.

Definition 6 *Given a network topology $G(N, E)$, a subnet, $G(N, \widetilde{E})$, is a Reliable Topology (RT) if for any link $e \in \widetilde{E}$, $\{e\}$ is not a minimum edge cut of $G(N, \widetilde{E})$ as long as $\{e\}$ is not a minimum edge of $G(N, E)$.*

Note that a RT must exist for any network topology by Definition 6. It can also be observed that a link state information dissemination scheme is reliable if the minimum edge cut of the subnet topology over which it floods link state information has at least two members (edges). However, such topology (the RT without a one-link minimum edge cut) does not exist if there exists a one-member minimum edge cut of G. On the other hand, if the network does not have a one-link minimum edge cut, any of its RTs does not have a one-link minimum edge cut either, i.e., any link in a RT has at least one protection. Hence, the key point to provide reliable link state information dissemination is to find an appropriate subnet topology or a RT. An intuitive solution is the Hamiltonian Cycle [39], especially the least cost Hamiltonian Cycle. However, the major disadvantage of this approach is that finding a Hamiltonian Cycle in a given graph is NP-complete and would not be practical to use for fast convergence in real time networks. Therefore, alternatively, we provide an effective and efficient solution in this chapter, Tree-based Reliable Topology (TRT), which is proposed based on Theorem 1 (below) and built upon the combination of multiple spanning trees.

Definition 7 *Tree-based Reliable Topology (TRT): Given a connected network topology $G(N, E)$, and one of its spanning trees $T(N, \widehat{E})$, assume $G(N, E - \widehat{E})$ is the topology constructed by removing all the links in $\widehat{E}$ from $G(N, E)$, and consists of n ($n \geq 1$) connected sub-networks, $G_1(N_1, E_1)$, $G_2(N_2, E_2)$, ..., $G_n(N_n, E_n)$. Further assume $T_1(N_1, \widehat{E}_1)$, $T_2(N_2, \widehat{E}_2)$, ..., $T_n(N_n, \widehat{E}_n)$ are the spanning trees of $G_1(N_1, E_1)$, $G_2(N_2, E_2)$, ..., $G_n(N_n, E_n)$, respectively. The topology $D(N, \widetilde{E})$ constructed by combining $T(N, \widehat{E})$, $T_1(N_1, \widehat{E}_1)$, $T_2(N_2, \widehat{E}_2)$, ..., and $T_n(N_n, \widehat{E}_n)$ is referred to as a Tree-based Reliable Topology (TRT).*

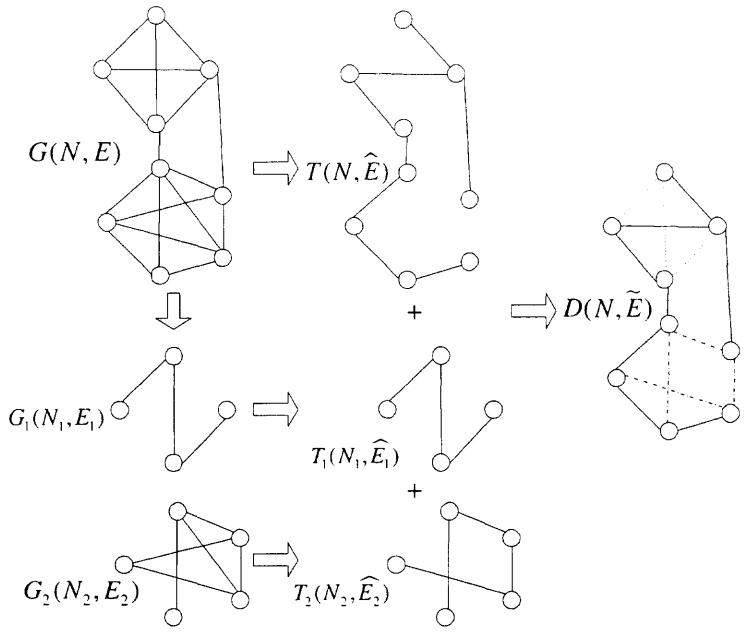

Figure 3.1 An illustration of the construction procedure of a DST.

The construction procedure of TRT is illustrated by an example shown in Fig. 3.1. Given a network $G(N, E)$ as shown in Fig. 3.1, we can construct one of its TRTs, $D(N, \widetilde{E})$, by combining $T(N, \widehat{E})$, $T_1(N_1, \widehat{E_1})$, and $T_2(N_2, \widehat{E_2})$, where $T(N, \widehat{E})$ is one of the spanning trees of $G(N, E)$, and $T_1(N_1, \widehat{E}_1)$, and $T_2(N_2, \widehat{E}_2)$ are the spanning trees of $G_1(N_1, E_1)$ and $G_2(N_2, E_2)$, respectively, which are the remaining networks after removing the links in $T(N, \widehat{E})$ from $G(N, E)$. By Definition 7, it can be observed that if $n = 1$, i.e., $G(N, E - \widehat{E})$ is still a connected network; TRT is actually constructed by combining the three spanning trees of $G(N, E)$.

Theorem 1 *Any TRT, $D(N, \widetilde{E})$, is also a RT of the corresponding topology $G(N, E)$.*

Proof: By contradiction. Assume $\exists e \in \widetilde{E}$ such that $\{e\}$ is a minimum edge cut of $D(N, \widetilde{E})$ while $\{e\}$ is not a minimum edge cut of $G(N, E)$. Since $\{e\}$ is a minimum edge cut of $D(N, \widetilde{E})$, further assume $D(N, \widetilde{E})$ is divided into two parts, $\widetilde{G}_1(\widetilde{N}_1, \widetilde{E}_1)$ and $\widetilde{G}_2(\widetilde{N}_2, \widetilde{E}_2)$, by removing e from $D(N, \widetilde{E})$. Moreover, since $\{e\}$ is not an edge cut of

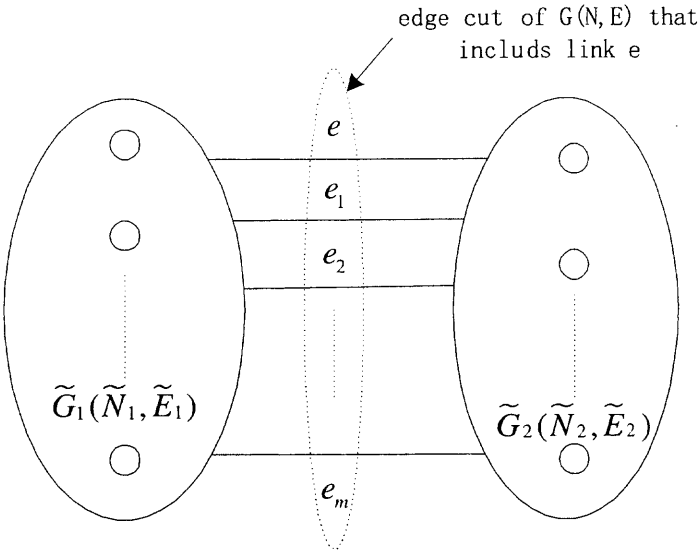

Figure 3.2 A minimum edge cut of $G(N, E)$ that includes link e, which divides network into two parts, $G_1(N_1, E_1)$ and $G_2(N_2, E_2)$.

$G(N, E)$, assume a minimum edge cut of $G(N, E)$ that includes link e and divides nodes $\widetilde{N}_1$ from $\widetilde{N}_2$ is $\{e, e_1, e_2, ..., e_m\}$, as shown in Fig. 3.2. Note that $e_1, e_2, ..., e_m$ are not links of $D(N, \widetilde{E})$. Otherwise, $\{e\}$ cannot be a minimum edge cut of $D(N, \widetilde{E})$ because after removing $\{e\}$ from $D(N, \widetilde{E})$, $\widetilde{G}_1(\widetilde{N}_1, \widetilde{E}_1)$ and $\widetilde{G}_2(\widetilde{N}_2, \widetilde{E}_2)$ are both connected and there still exists at least a link connecting them, i.e., after removing link e from $D(N, \widetilde{E})$, it is still connected, thus contradicting the assumption. By the definition of TRT, assume $D(N, \widetilde{E})$ consists of $T(N, \widehat{E})$, $T_1(N_1, \widehat{E}_1)$, $T_2(N_2, \widehat{E}_2)$, ..., $T_n(N_n, \widehat{E}_n)$, where $T(N, \widehat{E})$ is a spanning tree of $G(N, E)$, and $T_1(N_1, \widehat{E}_1)$, $T_2(N_2, \widehat{E}_2)$, ..., $T_n(N_n, \widehat{E}_n)$ are spanning trees of $G_1(N_1, E_1)$, $G_2(N_2, E_2)$, ..., $G_n(N_n, E_n)$, respectively, which are the remaining connected networks by removing $\widehat{E}$ from $G(N, E)$. Assume e_1 is the link connecting node u and v. Since $e_1 \notin \widetilde{E}$ and $\widehat{E} \subset \widetilde{E}$, u and v are directly connected by e_1 after removing $T(N, \widehat{E})$ from $G(N, E)$, and thus belong to a single connected network $G_k(N_k, E_k)$, $1 \leq k \leq n$. Hence, u is still reachable to v after removing $T(N, \widehat{E})$ from $D(N, \widetilde{E})$. Since $\{e\}$ is a a minimum edge cut of $D(N, \widetilde{E})$ and $T(N, \widehat{E})$ is a spanning tree of $D(N, \widetilde{E})$, $e \in \widehat{E}$.

Hence, by removing link e from $D(N, \widetilde{E})$, node u ($u \in \widetilde{N}_1$) is still reachable to v ($v \in \widetilde{N}_2$), which contradicts to the assumption that $\widetilde{G}_1(\widetilde{N}_1, \widetilde{E}_1)$ and $\widetilde{G}_2(\widetilde{N}_2, \widetilde{E}_2)$ are separated by removing link e from $D(N, \widetilde{E})$. Hence, any TRT is also a RT of the corresponding network. ■

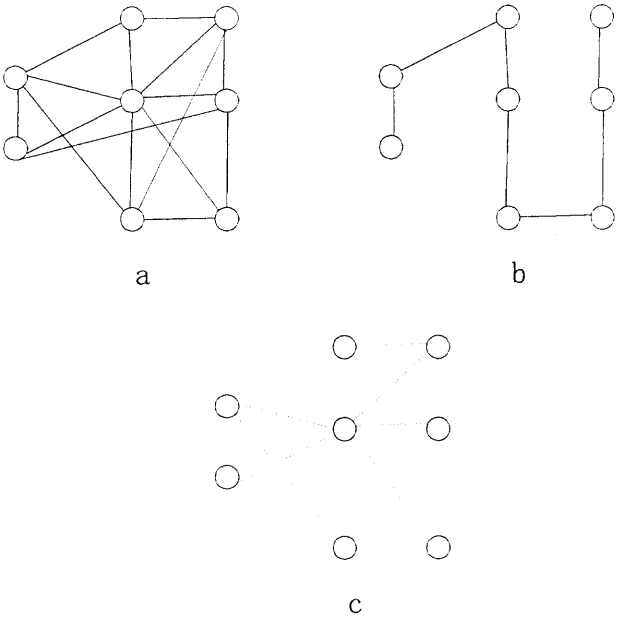

Figure 3.3 A network and the spanning trees of its DST.

By Theorem 1, we know that link state information can be reliably disseminated over a TRT, i.e., we can design a reliable link state information scheme by constructing a TRT over which link state information is distributed. Note that a TRT is the combination of several spanning trees. Hence, the remaining problem is to select appropriate spanning trees in order to minimize the protocol overhead and convergence time. Here, we simply deploy Minimum Spanning Tree (MST) to construct TRT, i.e., in the process of constructing the TRT, all the spanning trees are the minimum spanning trees of the corresponding networks. As the result, we can guarantee that the convergence time of our proposed scheme is low. Furthermore, in order to evaluate the protocol overhead of link state information dissemination schemes, we define the number of Link State Advertisements (LSAs) upon

each link state update as a performance index. Note that a TRT consists of several spanning trees. Instead of treating a TRT as a single subnet but many independent trees with joint nodes, and LSAs are independently flooded over each of them, it can be proved that the number of LSAs is only twice that of the Moy's scheme [37], in which link state information is flooded over a spanning tree of the network. For instance, given a network (Fig. 3.3a) and one of its TRTs, which consists of two spanning trees (Fig. 3.3b&3.3c, respectively). In the case of a link state update, LSAs are flooded independently over two spanning trees (in this example, it can be viewed as the combination of two parallel tree-based link state dissemination schemes), instead of over the whole TRT. It can be observed that the number of LSAs is twice that of LSAs over a single spanning trees.

Computational Complexity: Note that by Prim's algorithm [40], the computational complexity of computing the minimum spanning tree in a network is $O(|E| \log |N|)$, where $|N|$ is the number of nodes and $|E|$ the number of links. From the definition of TRT, the computational complexity is the sum of the computational complexities of computing the spanning trees of $G(N, E)$, and $G_1(N_1, E_1)$, $G_2(N_2, E_2)$, ..., $G_n(N_n, E_n)$. Since $\sum_{i=1}^{n} |N_i| = |N|$ and $\sum_{i=1}^{n} |E_i| \leq |E|$, the computational complexity of computing the TRT that consists of minimum spanning trees is

$$\begin{aligned} & O(|E| \log |N|) + \sum_{i=1}^{n} O(|E_i| \log |N_i|) \\ \leq\ & O(|E| \log |N|) + \sum_{i=1}^{n} O(|E_i| \log |N|) \\ =\ & O(|E| \log |N|), \end{aligned} \tag{3.1}$$

where $|E_i|$ and $|N_i|$ are the numbers of links and nodes of $G_i(N_i, E_i)$, $i = 1, 2, ..., n$, respectively. Hence, we can claim that the computational complexity of computing the TRT for our proposed link state update scheme is fairly low, and it can be deployed practically for real networks.

Note that although TRT is proposed for distributing link state information in this chapter, it can be deployed for other purposes. For example, it can be used in WDM networks for providing link protection: in the case of any single link failure, TRT can guarantee an alternate route between the two corresponding nodes connected by the link as long as the network is still connected. Compared to the approach of using the Hamiltonian Cycle [41], the one using TRT has obviously the advantage of low computational complexity.

3.3 Summary

In this chapter, based on the Tree-based Reliable Topology (TRT), we have proposed an efficient link state dissemination scheme. We show that our proposed scheme possesses the advantages of reliability, low protocol overhead, and fast convergence. By proving that the computational complexity of computing the TRT over which link state information is disseminated is compatible to that of computing the minimum spanning tree, we show that our proposed scheme is practical for a real network from the perspective of the computational complexity.

CHAPTER 4

FINDING ALL HOPS SHORTEST PATHS

4.1 Introduction

Many proposed source routing algorithms tackle QoS routing problem by transforming it into the shortest path selection problem or the k-shortest paths selection problem, which are P-complete, with an integrated cost function that maps the multi-constraints of each link into a single cost. However, since the solutions are computed by finding the shortest path, one of their common problems is that they cannot minimize the number of hops of their solutions. As a result, the network resource is wasted. Given a set of constraints $(\alpha_1, \alpha_2, ..., \alpha_M)$ and a network that is modeled as a directed graph $G(N, E)$, where N is the set of all nodes and E is the set of all links, assume each link connected from node u to v, denoted by $e_{u,v} = (u, v) \in E$, is associated with randomly distributed additive parameters: $w_i(u, v) \geq 0$, $i = 1, 2, ..., M$, and define $P_r\{W_1(p) \leq \alpha_1, W_2(p) \leq \alpha_2 W_M(p) \leq \alpha_M | C(p) = u, H(p) = n\}$ as the probability that a path p is a feasible path with $C(p) = u$, and its hop count, $H(p) = n$, where $C(p)$ is the cost of p, which is a function of the weights of the links on p, and $W_i(p) = \sum_{e_{u,v} \in p} w_i(u, v)$. The probability of the shortest path to be a feasible path may not be the largest in all paths. Therefore, computing a feasible path among all hops shortest paths, instead of only the shortest path, can increase the success ratio of finding a feasible path. In this chapter, we introduce and investigate a new problem referred to as all hops Shortest paths (AHSP) problem, defined below.

Definition 8 *All Hops Shortest Paths (AHSP) Problem: Assume a network is modeled as a directed graph $G(N, E)$, where N is the set of all nodes and E is the set of all links. Each link connected from node u to v, denoted by $e_{u,v} = (u, v) \in E$, is associated with an additive weight $c(u, v)$. Given a source node $s \in N$ and maximal hop count H, $H < n$,*

find, for each hop count value h, $1 \leq h \leq H$, and any other destination node $u \in N$, the shortest, i.e., the least weight, h-hop paths from s to u if an h-hop path exists. In this chapter, we will refer to the length, i.e., cost, of a path as the sum of its link weights.

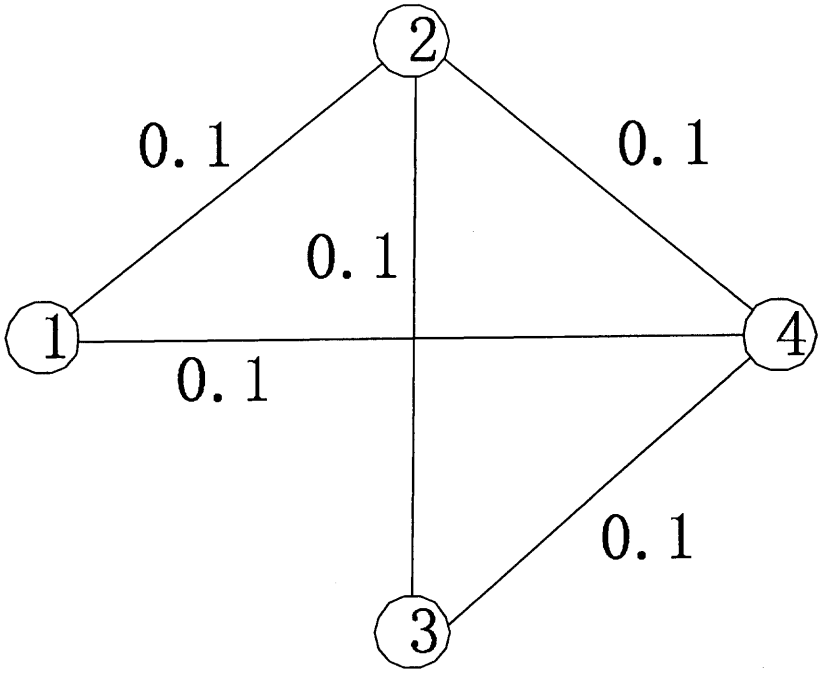

Figure 4.1 A 4-nodes network.

Note that AHSP is different from add-AHOP [42]. For each hop count value h, $1 \leq h \leq H$, add-AHOP is to select the shortest path from a given source to a destination that has a hop count no larger than h; while AHSP is to select the shortest path from the source to the destination that has a hop count equal to h if such a path physically exists (it might be more appropriate to call AHOP as All-Hops-Constrained Optimal Path problem instead of All Hops Optimal Path problem). For example, as shown in Fig. 4.1, the cost of each link in the 4-node network is 0.1. Let $(\alpha_1, \alpha_2, ..., \alpha_h)$ represents an h-hop path from node α_1 to node α_h sequentially traversing nodes $\alpha_1, \alpha_2, ..., \alpha_h$, respectively. Given $H = 3$, since $(1, 4)$ is the shortest path from node 1 to node 4 with a hop count of 1, for each hop

count $h \in \{1, 2, 3\}$, the shortest path selected by add-AHOP with a hop count no larger than h from node 1 to node 4 would always be $(1, 4)$; while the h-hop shortest paths selected by AHSP from node 1 to node 4 are $(1, 4)$, $(1, 2, 4)$, and $(1, 2, 3, 4)$, respectively, when h equals to 1, 2, and 3. Hence, solving add-AHOP does not need to solve AHSP. On the other hand, if AHSP is solved, add-AHOP is also solved. AHSP seems to be more difficult than add-AHOP because in addition to the paths selected by add-AHOP, AHSP involves selecting paths that are not selected by add-AHOP, i.e., since the shortest path from the source to the destination that has a hop count no larger than h must be a $\widetilde{h}$-hop shortest path from the source to the destination, where $\widetilde{h}$ is its hop count and $\widetilde{h} \leq h$; the paths selected by AHSP must include the paths selected by add-AHOP. However, in this chapter, we prove that the comparison-based optimal solutions for both add-AHOP and AHSP have the same order of worst-case computational complexities, where the optimal solutions are referred to as the solutions possessing the minimum worst-case computational complexity.

4.2 A Lower Bound on the Worst-case Computational Complexities of the Comparison-based Solutions

Before we proceed to further analysis, we first present the following definition [43].

Definition 9 *A path-comparison-based shortest-path algorithm Γ, which accepts as input a graph G and a weight function, can perform all standard operations, but the only way it can access the edge weights is to compare the weights of two different paths [43].*

Denote d_i as the degree of node i and $i_1, i_2, \ldots, i_{d_i}$ as its neighboring nodes. If there does not exist a physical link from the source node s to any other node i, assume there exists a virtual link $\widehat{e}(s, i)$ from s to i with a cost of infinity. Denote $p^n(s, i)$ as an n-hop path from the source s to i (see Fig. 4.2). Then,

1. $\forall i \in N$, $p^1(s, i) = e(s, i)$ (if, in reality, no link between the source s and node i exists, $p^1(s, i) = \widehat{e}(s, i)$).

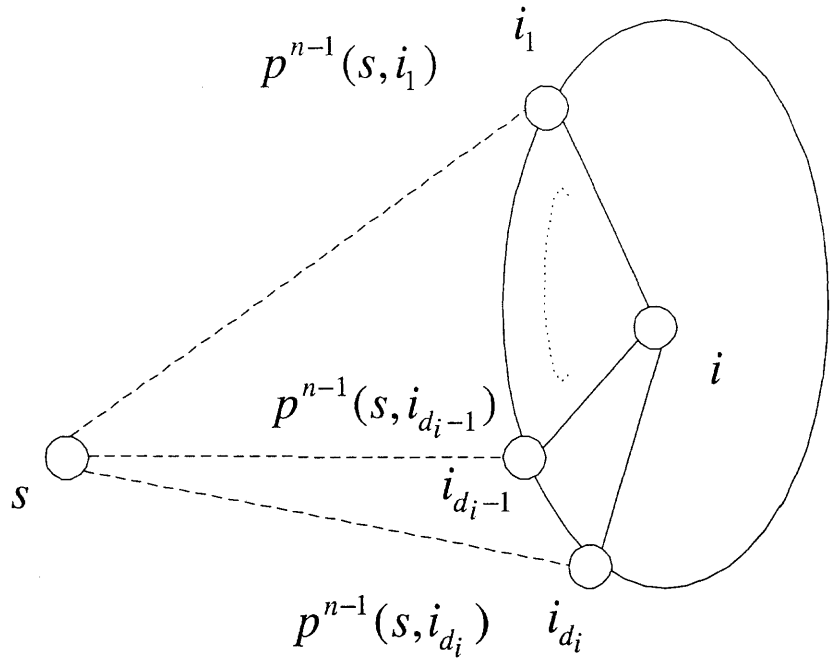

Figure 4.2 The definition of $p^n(s, i)$.

2. $p^h(s, i)$ represents the shortest h-hop path among the paths $p^{h-1}(s, i_d) + e(i_d, i), 1 \leq d \leq d_i$. If, in reality, no h-hop path from s to i exists, we assume that there exists a virtual h-hop path whose weight is infinity.

Denote D_i^h as the weight of $p^h(s, i)$; $\pi^h(i)$ represents the predecessor node of i along the path.

Theorem 2 *$p^h(s, i)$, $1 \leq h \leq H$, are the solutions to AHSP.*

Proof: Proof: When $h = 1$, from the definition of the initial value of D_i^1 ($i \neq s$), $p^1(s, i)$ is the one hop shortest path from s to i, $i \in \{1, 2, ..., N\}$.

We assume that the proposition is correct for $h = k$. We want to prove by deduction that it is true for $h = k + 1$. Assume when $h = k + 1$, $\exists j \neq s$ such that $p^{k+1}(s, j)$ is not the shortest path in all $(k + 1)$-hop paths from s to j (D_j^{k+1} is larger than the cost of the $(k + 1)$-hop shortest path from s to j). Further assume path $\widehat{p}^{k+1}(s, j)$ is the shortest path

in all $(k+1)$-hop paths from s to j, the predecessor node of node j in $\widehat{p}^{k+1}(s,j)$ is d, the path from s to d in $\widehat{p}^{k+1}(s,j)$ is $\widehat{p}^{k}(s,d)$ (note that $\widehat{p}^{k}(s,d)$ may not be the shortest k-hop path from s to d, and by the earlier assumption, since a k-hop path exists, $p^{k}(s,d)$ is the shortest path from s to d), the cost of $\widehat{p}^{k+1}(s,j)$ is c, and the cost of $\widehat{p}^{k}(s,d)$ is c'. Thus,

$$c < D_j^{k+1}. \tag{4.1}$$

Since $\widehat{p}^{k+1}(s,j)$ is resulted by concatenating $\widehat{p}^{k}(s,d)$ with link $e(d,j)$, if $\widehat{p}^{k}(s,d) = p^{k}(s,d)$ or $c' = D_d^k$,

$$c = D_d^k + c(d,j) \geq \min_l\{D_l^k + c(l,i)\} = D_j^{k+1}, \tag{4.2}$$

which contradicts Eq. 4.1. Hence, $\widehat{p}^{k}(s,d)$ is not the k-hop shortest path from s to d, i.e.,

$$\widehat{p}^{k}(s,d) \neq p^{k}(s,d), \tag{4.3}$$

and

$$c' > D_d^k. \tag{4.4}$$

So, the cost of $\widehat{p}^{k+1}(s,j)$ is

$$c = c' + c(d,j) > D_d^k + c(d,j) \geq \min_l\{D_l^k + c(l,i)\} = D_j^{k+1}, \tag{4.5}$$

which contradicts Eq. 4.1, implying that $\widehat{p}^{k+1}(s,j)$ is not the $(k+1)$-hop shortest path from s to j, contradicting to assumption. So, when $h = k+1$, $p^{k+1}(s,i)$, $i \in \{1,2,...,N\}$, is the shortest path among all $(k+1)$-hop path from s to i. Thus, for any node $i \in \{1,2,...,N\}$, if at least one h-hop path from s to i physically exists, the path $p^{h}(s,i)$ must be the shortest path among all h-hop paths from s to i, i.e., $p^{h}(s,i)$, $1 \leq h \leq H$, are the solutions to AHSP. ■

By Theorem 1, we know that the h-hop shortest path from s to i is the shortest path of the paths that are constructed by concatenating the $(h-1)$-hop shortest paths from s to the

neighboring nodes of i with the corresponding links. Next, we provide a tight lower bound on the worst-case computational complexity of the optimal comparison-based solution to AHSP based on Theorem 1.

Theorem 3 *The optimal comparison-based solution to AHSP has the worst-case computational complexity of* $O(H|E|)$.

Proof: Proof: First of all, we prove that the upper bound on the worst-case computational complexities of the optimal comparison-based solutions to AHSP is $O(H|E|)$. Note that the $(h+1)$-hop least weight path from s to i is the least weight path among the paths constructed by concatenating the h-hop least weight paths from s to the neighboring nodes of i with the corresponding links. Hence, in order to compute the $(h+1)$-hop shortest path from s to i, the h-hop least weight paths from s to the neighboring nodes of i should be computed first. Moreover, given a node i, the $(h+1)$-hop least weight path from s to i can be achieved with the computational complexity of $O(d_i)$ by the definition of the comparison-based algorithm and Theorem 4.1. Hence, given the h-hop shortest paths from s to all other nodes, the $(h+1)$-hop shortest paths from s to all other nodes can be computed with the computational complexity of $O(\sum_{i \in N, i \neq s} d_i) = O(|E|)$. Define f_H as the computational complexity bound on the optimal solutions (i.e., the ones with the minimum worst-case computational complexity) to AHSP, where H is the maximum hop. Therefore,

$$f_H \leq f_{H-1} + O(|E|) \Longrightarrow f_H \leq O(H|E|), \tag{4.6}$$

implying that the optimal comparison-based solution to AHSP has a computational complexity no larger than $O(H|E|)$.

On the other hand, since AHSP computes all the paths AHOP computes, the computational complexity of the optimal comparison-based solution to AHSP must be no less than that of the optimal comparison-based solution to add-AHOP. It has been proved in [42] that the tight lower bound on the computational complexity of the optimal comparison-based solution to add-AHOP is $O(|V|^3)$, where $|V|$ is the number of nodes. Therefore,

$$O(|V|^3) \leq f_H \leq O(H|E|), \tag{4.7}$$

However, only when $|E| = O(|V|^2)$ and $H = O(|V|)$, $O(|V|^3) = O(|V||E|)$. Hence, the optimal comparison-based solution to AHSP has the worst-case computational complexity of $O(H|E|)$. ■

By Theorem 3, we know that the worst-case computational complexity of an optimal comparison-based solution to AHSP is the same as that of an optimal comparison-based solution to add-AHOP. Note that link weights are assumed to be additive in AHSP in this chapter. Hence, Theorem 3 may not be applicable to the case that they are not additive, e.g., concave and multiplicative (it may not be appropriate to call the problem as AHSP anymore). However, for the case when the link weights are concave, it can be proved that the n-hop path with the largest weight from the source to node i can be computed by letting $p^h(s,i)$ represent the n-hop path with the largest weight among the paths $p^{h-1}(s,i_d) + e(i_d,i)$, $1 \leq d \leq d_i$ (when the link weights are concave, it has become the problem of finding all hops paths with the largest weights). Moreover, multiplicative link weights can be converted to additive weights by using the logarithm function. Therefore, similar to Theorem 3, we still can prove that $O(H|E|)$ is an upper bound on the worst-case computational complexity of the optimal comparison-based solutions.

4.3 Summary

In this chapter, we have introduced and investigated a new problem referred to as the All Hops Shortest Paths (AHSP) problem. A tight lower bound on the worst-case computational complexity of the optimal comparison-based solution to AHSP has also been derived.

CHAPTER 5

AN OPTIMAL COMPARISON BASED SOLUTION TO AHSP AND ITS APPLICATIONS TO QOS ROUTING

5.1 Introduction

In this chapter, based on an optimal comparison based solution to AHSP, we focus on designing the routing strategies, especially finding the delay constrained least cost paths and multiple additively constrained paths, and assume that accurate network state information is available to each node. A number of research works have also addressed inaccurate information [44]- [47], which is, however, beyond the scope of this proposal.

It has been proved that multiple constrained path seleciton is NP-complete [13]. Hence, tackling this problem requires heuristics. The limited path heuristic [8] proposed by Yuan maintains a limited number of candidate paths, say x, at each hop. The computational complexity is $O(x^2nm)$ for the Extended Bellman-Ford algorithm for two constraints, where n and m are the number of links and nodes, respectively. For the purpose of improving the response time and reducing the computation load on the network, precomputation-based methods [48] have been proposed. Korkmaz and Krunz [7] provided a heuristic with the computational complexity compatible to that of the Dijkstra algorithm to find the least cost path subject to multiple constraints. An algorithm [49], called A*Prune, is capable of locating multiple shortest feasible paths from the maintained heap in which all candidate paths are stored. For the case that only inaccurate link state information is available to nodes, approximate solutions [50] have been proposed for the Most Probable Bandwidth Delay Constrained Path (MP-BDCP) selection problem by decomposing it into two sub-problems: the Most Probable Delay Constrained Path (MP-DCP) and the Most Probable Bandwidth Constrained Path (MP-BCP). A LAgrange Relaxation based Aggregated Cost (LARAC) was proposed in [51] for the Delay Constrained Least Cost

path problem (DCLC). This algorithm is based on a linear cost function $c_\lambda = c + \lambda d$, where c denotes the cost, d the delay, and λ an adjustable parameter. It was shown that the computational complexity of this algorithm is $O(m^2 \log^4 m)$. Many researchers have posed the QoS routing problem as the k-shortest path problem [52]. The authors in [53] proposed an algorithm, called TAMCRA, for Multiple Constrained Path selection (MCP) by using a non-linear cost function and a k-shortest path algorithm. The computational complexity of TAMCRA is $O(kn \log(kn) + k^3 mM)$, where k is the number of shortest paths and M is the number of constraints. To solve the delay-cost-constrained routing problem, Chen and Nahrstedt [54] proposed an algorithm, which maps each constraint from a positive real number to a positive integer. By doing so, the mapping offers a "coarser resolution" of the original problem, and the positive integer is used as an index in the algorithm. The computational complexity is reduced to pseudo-polynomial time, and the performance of the algorithm can be improved by adjusting a parameter, but with a larger overhead. In [55], a heuristic algorithm was proposed based on a linear cost function for two additive constraints; this is a MCP (Multiple Constrained Path Selection) problem with two additive constraints. A binary search strategy for finding the appropriate value of β in the linear cost function $w_1(p) + \beta w_2(p)$ or $\beta w_1(p) + w_2(p)$, where $w_i(p)$ $(i = 1, 2)$ are the two respective weights of the path p, was proposed, and a hierarchical Dijkstra algorithm was introduced to find the path. It was shown that the worst-case complexity of the algorithm is $O(\log B(m+n \log n))$, where B is the upper bound of the parameter β. The authors in [56] simplified the multiple constrained QoS routing problem into the shortest path selection problem, in which the Weighted Fair Queuing (WFQ) service discipline is assumed. Hence, this routing algorithm cannot be applied to networks where other service disciplines are employed. Widyono [57] introduced a Constrained Bellman-Ford (CBF) algorithm, which deploys a breadth-first-search to locate paths of monotonically increasing delay while recording and updating the lowest cost path to the visited nodes. This approach yields the optimal path – being the least cost path among all the paths satisfying the delay

constraint. However, its worst-case computational complexity is exponentially increasing with the network size.

Many ε-approximation algorithms (the solution has a cost within a factor of $(1+\varepsilon)$ of the optimal one) subject to DCLC have been proposed in the literature. Lorenz et al. [59] presented several ε-approximation solutions for both the DCLC and the multicast tree. Among them, the algorithm subject to DCLC possesses the best-known computational complexity of $O\left(nm\log n\log(\log n)+\frac{nm}{\varepsilon}\right)$. Hassin [60] presented two ε-approximations algorithms for the Restricted Shortest Path problem (RSP) with complexities of $O((\frac{nm}{\varepsilon})\log\log U)$ and $O(m\frac{n^2}{\varepsilon}\log(\frac{n}{\varepsilon}))$, where U is the upper bound of the cost of the path computed. Raz and Shavitt [61] proposed an efficient dynamic programming solution for the case in which the QoS parameters are integers, and a sub-linear algorithm for the case in which all link costs use the (same) function of their corresponding delays. Different from [59]- [61], Goel et al. [62] proposed a lower computational complexity solution that can guarantee to find a path having the delay less than a factor of of the delay constraint, while the cost of the path is no larger than that of the optimal one. However, since the path computed by [62] may have a cost larger than the delay constraint, it cannot guarantee 100% success ratio.

Existing algorithms reviewed above may have the following drawbacks.

1. Although the algorithms such as the ε-approximation approaches [59], [60] can achieve 100% or near 100% success ratio, their worst-case computational complexities are too high to be practical (assume ε is very small in ε-approximation algorithms so that their success ratios are close to 1).

2. The algorithms such as [53] have the advantage of having low computational complexities. However, they suffer low success ratio in finding a feasible path when it exists. Moreover, their success ratios in finding a feasible path may decrease sharply with the network size. In order to increase the success ratio, many proposed algorithms deploy k-shortest paths selection solutions (the number

of shortest paths are generally fixed in these algorithms), instead of the shortest path selection solutions. However, the computational complexities of the algorithms increase unnecessarily in the case in which a feasible path can be found with only one shortest path searching algorithm.

In this chapter, we shall first present an optimal comparions based solution to AHSP, based on which two algorithms are proposed respectively for solving the Multiple Additvely Constrained Path (MACP) selection and the delay constrained least cost path selection problems. The two algorithms can overcome the above drawbacks. We denote $p_1 + p_2$ as the concatenation of two paths p_1 and p_2, and $\min\{p_1, p_2\}$ as the shortest path between p_1 and p_2. The least weight path is also referred to as the shortest path in this chapter.

5.2 An Optimal Comparison Based Solution to AHSP

In the previous chapter, we have shown that we can compute the solutions of AHSP as follows:

1. $\forall i \in N$, $p^1(s,i) = e(s,i)$ (if, in reality, no link between the source s and node i exists, $p^1(s,i) = \widehat{e}(s,i)$).

2. $p^h(s,i)$ represents the shortest h-hop path among the paths $p^{h-1}(s,i_d) + e(i_d,i), 1 \leq d \leq d_i$. If, in reality, no h-hop path from s to i exists, we assume that there exists a virtual n-hop path whose weight is infinity.

Therefore,we develop a solution to AHSP, Extended Bellman-Ford (EB) algorithm, based on the standard Bellman-Ford algorithm. We show the pseudo-code of the relaxation procedure of EB as Fig. 5.1, where $\pi^{h+1}(i)$ represents the predecessor node of node i along the path $p^{h+1}(s,i)$. Note that there is no difference between the relaxation procedure of EB and that of the standard Bellman-Ford algorithm, implying that the computational

Relax(j,i)

1 *if* $D_i^{h+1} \geq D_j^h + c(i,j)$ *then*

2 $D_i^{h+1} = D_j^h + c(i,j)$

3 $p^{h+1}(s,i) = p^h(s,j) + e(i,j)$

4 $\pi^{h+1}(i)=j$

5 *end if*

Figure 5.1 The pseudo-code of the relaxation procedure of EB.

complexity is the same as that of the standard Bellman-Ford algorithm. However, there are two distinguished differences between our algorithm and the Bellman-Ford algorithm, that are not showed in Fig. 5.1:

1. D_i^{h+1} is defined as $\min_j[c(i,j) + D_j^h]$, $h = 1, 2, ...$, in our algorithm, while D_i^{h+1} is defined as $\min\{\min_j[c(i,j) + D_j^h], D_i^h\}$ in the Bellman-Ford algorithm. Here, $D_i^1 = c(s,i)$. D_i^{h+1} can be obtained iteratively through the relaxation procedure by simply setting the initial value of D_i^{h+1} as infinity, instead of D_i^h in the Bellman-Ford algorithm.

2. In our algorithm, $D_s^h = \infty$, $h = 1, 2, ...$, while $D_s^h = 0$, $h = 1, 2, ...$, in the Bellman-Ford algorithm.

By the above two modifications, we can compute $p^h(s,i)$, $h = 1, 2, ...$ which is a least cost path among all h-hop paths from s to i. From Fig. 5.1, it can be observed that in order to reconstruct all the paths after the execution of EB, for any node i and hop count h, we keep the predecessor node of i in $p^h(s,i)$, which contributes the memory complexity of

EB as $O(n^2)$ ($h < n$ and the number of nodes is n). For the purpose of avoiding loops, we adopt a simple method: associating paths with indicators. For example, we associate a path traversing nodes s, 1, 5, and 7 with an integer array of size n, in which 1st, 5th, and 7th array elements are set to 1, and the rest to 0. Hence, we can easily find out if a node is in the path by only checking the corresponding array element's value. By this method, we can prevent loops without increasing the worst-case computational complexity. Further notice that we can erase the indicator arrays associated with the shortest h-hop paths when all the shortest $(h+1)$-hop paths are computed. Therefore, the memory cost introduced by indicator arrays is limited by $O(n^2)$ (n nodes and the size of each indicatory array is n): the introduction of indicator arrays does not increase the worst-case memory complexity of EB.

5.3 Proposed Routing Algorithm for Finding a Path Subject to Multiple Additive Constraints

Most works reported in the literature approach the QoS routing problem as a special case of the multi-constrained QoS routing problem, i.e., mostly considering two constraints only. We will propose an algorithm in which there is no limitation on the number of QoS constraints. QoS constraints can be categorized into three types: concave, additive, and multiplicative. Since concave parameters set the upper limits of all the links along a path such as bandwidth, we can simply prune all the links and nodes that do not satisfy the QoS constraints. We can also convert multiplicative parameters into additive parameters by using the logarithm function. For instance, we can take $-\log(1-\alpha)$ as the replacement for loss rate α. Thus, without loss of generality, we only consider additive constraints and formulate the problem as follows:

Definition 10 *Multiple Additively Constrained Path Selection (MACP): Assume a network is modeled as a directed graph $G(N, E)$, where N is the set of all nodes and E is the*

set of all links. Each link connected from node u to v, denoted by $e_{u,v} = (u,v) \in E$, is associated with M additive parameters: $w_i(u,v) \geq 0$, $i = 1,2,...,M$. Given a set of constraints $c_i > 0$, $i = 1,2,...,M$, and a pair of nodes s and t, the objective of MACP is to find a path p from s to t subject to $w_i(p) = \sum_{e_{u,v} \in p} w_i(u,v) < c_i$, $i = 1,2,...,M$.

In this proposal, any path that meets the give set of QoS constraints is referred to as a feasible path.

Note that EB is a optimal solution to AHSP, in which there is only one weight associated with each link. Hence, similar to [51]- [55], we adopt a cost function, with which we map the multiple additive constraints of each link into a single weight. Then, we use EB to find a feasible path that meets the given constraints. We introduce next theorem that is used for designing our proposed routing algorithm.

Theorem 4 *Given a cost function $\xi(\cdot)$ such that for any $1 \leq i \leq M$, $\frac{\partial^2 \xi(x_1,x_2,...,x_M)}{\partial x_i^2} = 0$ and $\frac{\partial \xi(x_1,x_2,...,x_M)}{\partial x_i} \geq 0$, no feasible path exists if the least cost path has the cost larger than $\xi(c_1,c_2,...c_M)$.*

Proof: By contradiction. Assume path $\widehat{p}$ satisfies the constraint $(c_1,c_2,...c_M)$ and the least cost among all paths is larger than or equal to $\xi(c_1,c_2,...c_M)$; that is,

$$c(p) \geq \xi(c_1,c_2,...c_M), \forall p \Rightarrow c(\widehat{p}) \geq \xi(c_1,c_2,...c_M). \tag{5.1}$$

Also, since $\xi(\cdot)$ is linear,

$$\xi(w_1(\widehat{p}, w_2(\widehat{p}), ..., w_M(\widehat{p})) = c(\widehat{p}). \tag{5.2}$$

Thus,

$$\xi(w_1(\widehat{p}, w_2(\widehat{p}), ..., w_M(\widehat{p})) \geq \xi(c_1,c_2,...c_M). \tag{5.3}$$

However, since $\frac{\partial \xi(x_1,x_2,...,x_M)}{\partial x_i} \geq 0$ and path $\widehat{p}$ satisfies the constraint $(c_1,c_2,...c_M)$,

$$w_i(\widehat{p}) < c_i, \forall i \in \{1,2,...,M\}, \Rightarrow \xi(w_1(\widehat{p}, w_2(\widehat{p}), ..., w_M(\widehat{p})) < \xi(c_1,c_2,...c_M), \tag{5.4}$$

which contradicts 5.3, and thus theorem is proved. ■

Lemma 5 *If path p is a feasible path,*

$$\xi(w_1(p, w_2(p), ..., w_M(p)) < \xi(c_1, c_2, ...c_M). \tag{5.5}$$

Proof: The proof is similar to that of above theorem. ■

Lemma 6 *Given a path p, only if*

$$\xi(w_1(p, w_2(p), ..., w_M(p)) < \xi(c_1, c_2, ...c_M), \tag{5.6}$$

it can be a feasible path,

Proof: The proof is similar to that of above theorem. ■

We divide our routing algorithm into two parts: forward EB and backward EB. We search for a feasible path from the source to the destination using the forward EB, and reverse the search by the backward EB. Therefore, the computational complexity of our proposed routing algorithm is just twice that of EB. The cost functions used in both searches are different. We adopt the following simple but effective cost function for the forward EB:

$$f(x_1, x_2, ..., x_M) = \sum_{i=1}^{M} \frac{x_i}{c_i}. \tag{5.7}$$

Assume the shortest path found by the forward EB is not a feasible path and $\exists i \in \{1, 2, ..., M\}$ such that $w_i(p) > c_i$. By Theorem 1, the second search (backward EB) is executed only when the cost of p is less than $f(c_1, c_2, ...c_M)$. Since we already know that the least cost path of the first search (forward EB) is not a feasible path, the cost function should be adjusted for the second search such that

- If a feasible path does exist, p should not be the least cost path computed by the second search;

- If p is the least cost path of the second search, $f'(w_1(p, w_2(p), ..., w_M(p)) \geq f'(c_1, c_2, ...c_M)$ so that Theorem 1 can be invoked, where $f'(\cdot)$ is the cost function for the backward EB.

```
Relax(j,i)
1   if i  is the destination node t
2           if  D_i^h + c(j,t) ≤ f(c_1,c_2,....,c_M)   /*check if this path is feasible path*/
4           if p^h(s,j) + e(j,t)  is feasible, then
5               return SUCCESS   /*feasible path is found*/
6           end if
7           end if
8   end if
9   if  D_i^{h+1} ≥ D_j^h + c(i,j)  then
10      D_i^{h+1} = D_j^h + c(i,j)
11      p^{h+1}(s,i) = p^h(s,j) + e(i,j)
12      π^{h+1}(i)=j
13  end if
```

Figure 5.2 The pseudo-code of the relaxation procedure of the EB in BEB.

Based on Eq. 5.7, the cost function for the backward EB is defined as:

$$\begin{aligned} f'(x_1, x_2, ..., x_M) &= \sum_{\substack{j=1 \\ j \neq i}}^{M} \frac{x_j}{c_j} + \left(\frac{f(c_1, c_2, ..., c_M) - c(p)}{w_i(p) - c_i} + \frac{1}{c_i}\right) x_i \\ &= \sum_{\substack{j=1 \\ j \neq i}}^{M} \frac{x_j}{c_j} + \left(\frac{f(c_1, c_2, ..., c_M) - c(p)}{w_i(p) - c_i}\right) x_i. \end{aligned} \tag{5.8}$$

In order to increase the success ratio in finding a feasible path, by Lemma 5 and 6, we modified the relaxation procedure of EB as shown in Fig. 5.2. It can be observed that with a cost function $f(\cdot)$, whenever we compute a path from the sourece to the destination having a cost less than $f(c_1, c_2, ...c_M)$, we will check if it is a feasible path. As as a result, the success ratio in finding a feasible path can be increased, while the computational

```
Algorithm BEB(G,s,t,c_1,c_2,....,c_M )
1   if EB(G,s,t,  c_1,c_2,....,c_M , LeastCost) = SUCCESS
2    return SUCCESS
3   else
4    if LeastCost ≥  f(c_1,c_2,....,c_M )
5        return No Feasible Path Exists    /*no feasible path existing*/
6   else
7        Compute New Cost Function  f'(•)
8         if EB(G,t,s,  c_1,c_2,....,c_M , LeastCost)= SUCCESS
9             return SUCCESS
10       else
11            if LeastCost≥ f'(c_1,c_2,....,c_M )
12                return No Feasible Path Exists   /*no feasible path existing*/
13            end if
14        end if
```

Figure 5.3 The pseudo-code of BEB.

complexity remains the same. Fig. 5.3 shows the pseudo-code of our proposed QoS routing algorithm, referred to as B-EB (Bi-directional Extended Bellman-Ford).

5.4 Proposed Routing Algorithm for Delay Constrained Least Cost Path Selection

Based on EB, we propose a simple but efficient algorithm for selecting the delay constrained least cost path in this section. For completeness, we present the definition of DCLC below:

Definition 11 *Delay Constrained Least Cost Path Selection (DCLC): Assume a network is modeled as a directed graph $G(N, E)$, where N is the set of all nodes and E is the set of all links. Each link connected from node u to v, denoted by $e_{u,v} = (u, v) \in E$, is associated with a cost $c(u, v)$ and a delay $d(u, v) \geq 0$. Given a delay constraint $d > 0$, and a pair of nodes s and t, the objective of DCLC is to find the path p that has the least cost among the paths from s to t subject to $\sum_{e_{u,v} \in p} d(u, v) < d$.*

Similar to BEB, our proposed routing algorithm for Delay Constrained Least Cost Path Selection (DCLC) also consists of two parts: forward EB and backward. Hence,

in order to differentiate it for BEB, we call it as Dual Extended Bellman-Ford algorithm (DEB). The only difference between the DBE and BEB lies in the fact that the adopted cost functions are different. Therefore, the computational complexity of our proposed routing algorithm is just twice that of EB. The weight functions used in both searches are different. Since we try to find the delay constrained least cost path, we first need to find out if there exists a path satisfying the delay constraint. Hence, the cost function used in the forward EB is

$$f(d, c) = d. \tag{5.9}$$

Since EB is capable of computing all hops least weight path(s) between a source and a destination, no feasible path exists if a feasible path is not found by the forward EB, implying that even the least delay (weight) path has a delay larger than the delay constraint. Therefore, we terminate the search if we fail to find a feasible path in the forward EB. If a feasible path is found in the forward EB, we will try to minimize the cost in the second search. Therefore, the weight function adopted in the backward EB is

$$f(d, c) = c. \tag{5.10}$$

If the least weight path in the backward EB is a feasible path, it is definitely the least cost feasible path because the least weight path in the backward EB is actually the least cost path. The pseudo code of our proposed algorithm is shown in Fig. 5.4, which is called the Dual Extended Bellman-Ford (DEB) algorithm.

5.5 Simulations

We divide our simulations into two parts. We evaluate the performance of BEB and DEB by comparing BEB with Binary Search (BR) [55] and TAMCRA [53] in the first part, and comparing DEB with LARAC [51] and H-MCOP [7] in the second part, respectively.

```
Algorithm DEB(G,s,t, d )
1       set weight function as  f(d,c) = d
2       if EB(G,s,t,  d ,  p_min ) = FAIL
3               return FAIL /*no feasible path is found*/
4       else
5               set weight function as  f(d,c) = c
6               execute the backward EB, EB(G,t,s,  d ,  p_min )
7               if the least weight path is a feasible path,
8                       return SUCCESS /*the optimal path is found*/
9               end if
10      return  p_min
```

Figure 5.4 The pseudo-code of DEB.

In addition to the 32-node network [55], [54], two larger networks with 50 and 100 nodes, respectively, generated by using the Doar's model [63] are used to conduct the simulations for comparison purposes. In all simulations, the link weights are independent and uniformly distributed from 0 to 1, and all data are obtained by running 1,000,000 requests.

5.5.1 Simulation 1

To evaluate the performance, we do not adopt the success ratio defined in [55] and [54] that is defined as:

$$SR = \frac{Total\ number\ of\ success\ request}{Total\ number\ of\ request}. \tag{5.11}$$

Intuitively, when the QoS constraints are tight, the success ratio defined by Eq. 5.11 is low; while when the constraints are loose, the sucess ratio 5.11 must be high. Therefore, Eq. 5.11 cannot truly reflect algorithms' capability in finding a feasible path. Therefore, we propose the following more appropriate success ratio definition as our performance index

$$SR = \frac{Total\ number\ of\ success\ request\ of\ the\ algorithm}{Total\ number\ of\ success\ request\ of\ the\ optimal\ algorithm}. \tag{5.12}$$

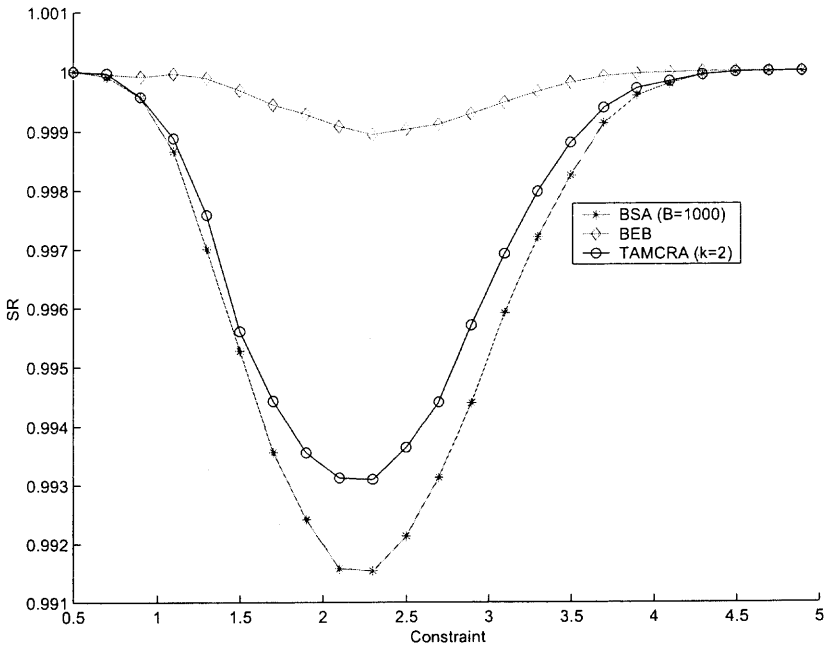

Figure 5.5 SR of the algorithms in the 32-node network.

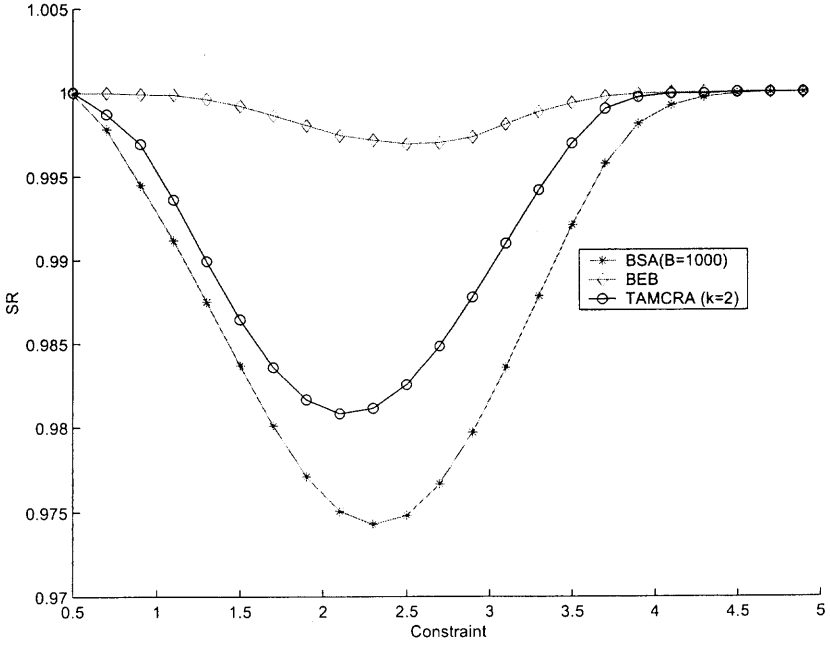

Figure 5.6 SR of the algorithms in the 50-node network.

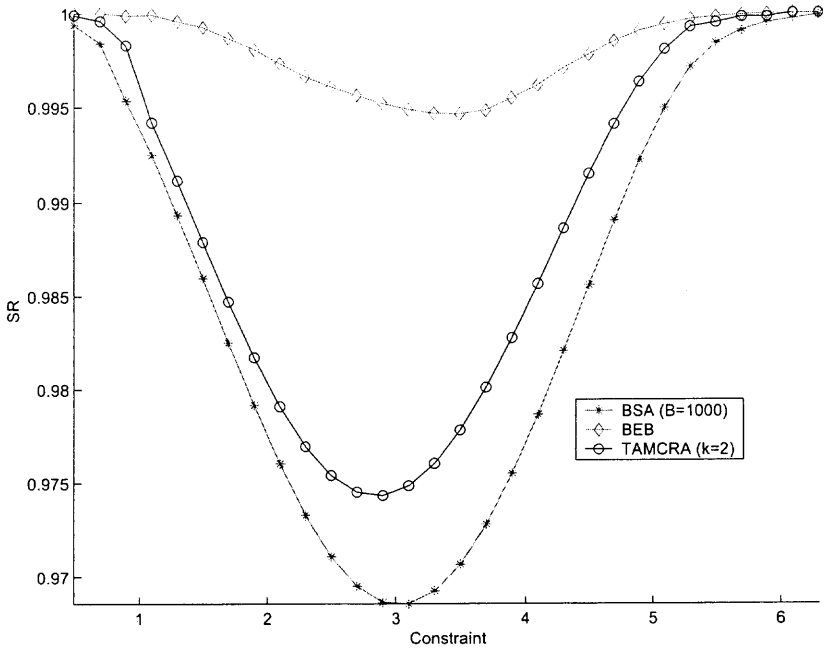

Figure 5.7 SR of the algorithms in the 100-node network.

The algorithm that can always locate a feasible path as long as it exists is refereed to as the optimal algorithm. Here, it is achieved simply by flooding which is rather exhaustive. Note that the success ratios of [55] and [53] in finding a feasible path increases with B and the number of shortest paths k (as mentioned before, [53] is based on k shortest paths search algorithm). Hence, for comparison purpose, we set $B = 1000$ for [55] in the simulations so that the success ratio of [55] is close to its upper bound, which can be achieved by setting B as a larger enough number; while the number of shortest paths k is set to 2 for [53].

Figs. 5.5-5.7 illustrate the success ratios of algorithms. It can be observed that the success ratio of BEB is very close to 100% in all the simulations. Moreover, as network size increases, the success ratios of [55] and [53] decrease much sharper than that of BEB. Hence, BEB is more scalable than [55] and [53].

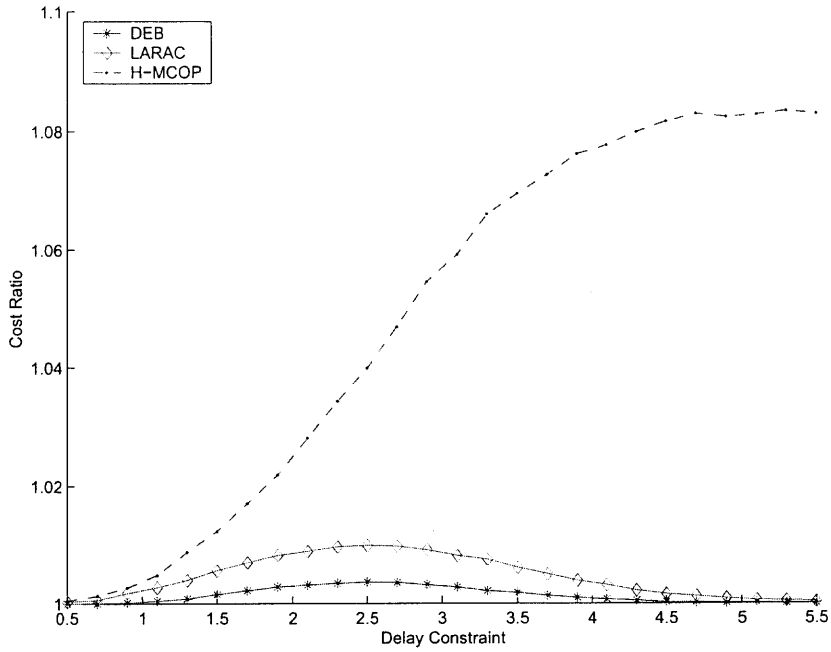

Figure 5.8 CRs of the algorithms in the 32-node network.

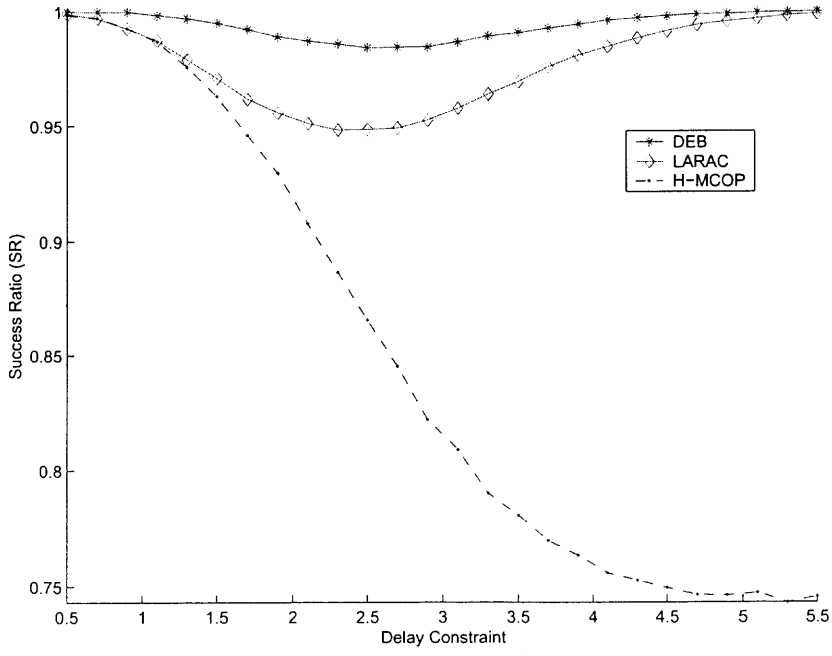

Figure 5.9 SRs of the algorithms in the 32-node network.

5.5.2 Simulation 2

Many proposed DCLC solutions can find a feasible path if a feasible path exists, but cannot guarantee that it is the optimal feasible path. Hence, beside adopting the success ratio in finding the optimal feasible path (the least cost path subject to the delay constraint) as a performance index for the DCLC solutions, here, we propose another performance index, Cost Ratio(CR), defined as follow:

$$CR = \frac{\text{Average cost of the solutions of an algorithm}}{\text{Average cost of the corresponding optimal solutions}}. \tag{5.13}$$

It can be observed that, when the delay constraint is either very tight or very loose, DEB and LARAC have similar performance; they both can locate the optimal path with success ratios close to 1. This is attributed to the fact that when the delay constraint is tight, there are only a small number of feasible paths, and the optimal path can be easily found; when the delay constraint is loose, the possibility that the least cost path is a feasible path is high. Meantime, it can be observed that the performance of H_MCOP decreases with the increment of delay constraint (the least cost path from the source to the destination is not computed in H_MCOP). On the other hand, when the delay constraint is neither tight nor loose, our proposed algorithm, DEB, achieves better performance than both LARAC and H_MCOP in terms of CR and SR in all simulations; the CR of DEB is lower than those of LARAC and H_MCOP, while the SR of DEB is higher than those of LARAC and H_MCOP. Note that the worst-case computational complexity of LARAC is $O(m^2 \log^2 m)$, while it is $O(mn)$ for DEB. Hence, DEB outperforms LARAC in terms of SR, CR, and the worst-case computational complexity.

5.6 Summary

In this section, based on an optimal comparison-based solution to AHSP, Extended Bellman-Ford algorithm (EB), we have proposed two simple but efficient algorithms for

respectively MACP and DCLC. By extensive simulations, we have shown that both BEB and DEB are high performance algorithms.

CHAPTER 6

FINDING THE OPTIMAL LINEAR SCALING FACTOR FOR EPSILON-APPROXIMATION ALGORITHMS

6.1 Introduction

Many ε-approximation algorithms (the solution has a cost within a factor of $(1+\varepsilon)$ of the optimal one) subject to DCLC have been proposed in the literature. However, except those proposed for the case in which link costs are already integers or discrete, most ε-approximation algorithms, if not all, use the linear scaling technique. Moreover, we find that their computational complexities are linearly proportional to the linear scaling factor (note that since the linear scaling factor of all ε-approximation algorithms reported in the literature is linearly proportional to $\frac{1}{\varepsilon}$, their computational complexities are linearly proportional to $\frac{1}{\varepsilon}$). For instance, many ε-approximation algorithms use Bellman-Ford-like algorithms (e.g., RSP [59] and DAD [62]) to find an optimal solution (the least cost path satisfying the delay constraint) in the network where link costs are integers. Since the computational complexities of these Bellman-Ford-like algorithms are linearly proportional to the cost of the optimal feasible path (if it exists), which is, in turn, also linearly proportional to the linear scaling factor, the computational complexities of these algorithms are consequently linearly proportional to the linear scaling factor. Hence, our task in this chapter is to minimize the linear scaling factor so that the computational complexity of ε-approximation algorithms can be reduced. *It should be noted that although our algorithms presented in this chapter are tailored for the DCLC ε-approximation algorithms, they can be readily applied to all the cases where linear ε-approximation techniques are deployed, except those in which $\lfloor\cdot\rfloor$ is used for scaling (not $\lceil\cdot\rceil$).* Furthermore, we analytically show that the computational complexities of our proposed algorithms are very low with respect to those of ε-approximation algorithms. Therefore, incorporating the two algorithms into

ε-approximation algorithms does not increase their computational complexities, but can in fact effectively reduce their computational complexities because the optimal linear scaling factor can always be computed by our proposed algorithms.

It should be noted that this chapter presents, to our best knowledge, the first attempt at minimizing the linear scaling factor of the ε-approximation algorithms. Accordingly, the algorithms, OLSA and T-OLSA, are the first two algorithms proposed specifically to computing the optimal linear scaling factor.

6.2 A Framework of ε-Approximation Approaches for DCLC

In this section, a framework for optimizing linear scaling ε-approximation algorithms is presented. Here, a linear scaling ε-approximation algorithm refers to an algorithm that can provide an ε-approximation solution by first linearly increasing or decreasing the link costs and then quantizing them into integers. In particular, given a non-decreasing quantization function $f_\varepsilon(\cdot) : \mathbb{R}^+ \rightarrow \mathbb{Z}^+$, where $\mathbb{R}^+$ is the set of positive real numbers and $\mathbb{Z}^+$ is the set of positive integers, and a network $G(N, E)$, in which each link $e_{u,v}$ is associated with a cost $c(u, v) \in \mathbb{R}^+$ and a delay $d(u, v) \in \mathbb{R}^+$, and a delay constraint $d > 0$, an algorithm, which yields the optimal solution in $G'(N, E)$ that is an ε-approximation solution in $G(N, E)$, where $G'(N, E)$ is constructed from $G(N, E)$ by mapping the link cost $c(u, v)$ to $c'(u, v) = f_\varepsilon(\lambda c(u, v))$ and λ is the linear scaling factor, is referred to as a linear scaling ε-approximation approach. Observe that in an ε-approximation approach, the non-decreasing quantization function plays the key role. Hence, next definition is provided below to formulae the set of functions that can be used to design an ε-approximation algorithm.

Definition 12 *Given an instance of DCLC and a non-decreasing function* $f_\varepsilon(\cdot)$, $\forall x \in \mathbb{R}^+$, $f_\varepsilon(x) \in \mathbb{Z}^+$, *a delay constraint* $d > 0$, *and a network* $G(N, E)$, *in which each link* $e_{u,v}$ *is associated with a cost* $c(u, v) \in \mathbb{R}^+$ *and a delay* $d(u, v) \in \mathbb{R}^+$, $G'(N, E)$ *is constructed*

from $G(N,E)$ *by mapping the cost of link* $e_{u,v}$ *to* $c'(u,v) = f_\varepsilon(c(u,v))$. *If the optimal feasible path in* $G'(N,E)$ *from the source to the destination has a cost no greater than a factor of* $1+\varepsilon$ *from that of the optimal feasible path between the corresponding pair of nodes in* $G(N,E)$, $f_\varepsilon(\cdot)$ *is called a feasible* ε*-approximation function.*

Based on Definition 12, we present the next Proposition, from which feasible ε-approximation functions may be derived. Note that as long as a function satisfies the next Proposition, it satisfies Definition 12 (for any $G(N,E)$), i.e., the functions satisfying the next Proposition are universally feasible.

Proposition 7 *Given any instance of DCLC, assume there exists a non-decreasing function* $f_\varepsilon(\cdot)$ *such that* $\forall x \in \mathbb{R}^+$, $f_\varepsilon(x) \in \mathbb{Z}^+$, *and for any two sets of positive numbers,* $\{\alpha_i > 0, i = 1,2,...,n\}$ *and* $\{\beta_j > 0, j = 1,2,...,m\}$, *if*

$$\sum_{i=1}^{n} \alpha_i \leq \sum_{j=1}^{m} \beta_j, \tag{6.1}$$

and

$$\sum_{i=1}^{n} f_\varepsilon(\alpha_i) \geq \sum_{j=1}^{m} f_\varepsilon(\beta_j), \tag{6.2}$$

implying that

$$(1+\varepsilon)\sum_{i=1}^{n} \alpha_i \geq \sum_{j=1}^{m} \beta_j, \tag{6.3}$$

then, the function, $f_\varepsilon(\cdot)$, *is a feasible* ε*-approximation function.*

Proof: Given an instance of DCLC, assume an optimal feasible path between nodes s and t is path p, and the optimal feasible path is $\widehat{p}$ after the costs of links have been scaled to integers via the function, $f_\varepsilon(\cdot)$. Therefore,

$$\sum_{e_{u,v}\in p} c(u,v) \leq \sum_{e_{u,v}\in \widehat{p}} c(u,v), \tag{6.4}$$

and

$$\sum_{e_{u,v}\in p} f_\varepsilon(c(u,v)) \geq \sum_{e_{u,v}\in \widehat{p}} f_\varepsilon(c(u,v)). \tag{6.5}$$

By the definition of $f_\varepsilon(\cdot)$,

$$(1+\varepsilon)\sum_{e_{u,v}\in p} c(u,v) \geq \sum_{e_{u,v}\in \widehat{p}} c(u,v) \Leftrightarrow (1+\varepsilon)C(p) \geq C(\widehat{p}). \tag{6.6}$$

Hence, $f_\varepsilon(\cdot)$ is a feasible ε-approximation function. ■

Finding such universally feasible ε-approximation functions for any instance of DCLC solely based on Proposition 7 is difficult. Instead, we focus on deriving a solution for an easier case: find a feasible linear scaling ε-approximation function for a given instance of DCLC. Since the computational complexities of ε-approximation algorithms subject to DCLC are linearly proportional to the linear scaling factor (λ), the smaller the linear scaling factor, the better. Therefore, for the purpose of reducing the computational complexities of ε-approximation algorithms, our objective is to find the smallest linear scaling factor.

Definition 13 *Given a network $G(N,E)$, the feasible linear scaling ε-approximation function $f_\varepsilon(\cdot)$ that has the least scaling factor among all feasible functions is called the optimal linear scaling ε-approximation function.*

6.3 Optimal Linear Scaling Feasible ε-approximation Functions

In this section, we will analytically demonstrate how to find the optimal linear scaling ε-approximation function. We first provide the next Proposition to simplify the search of a feasible ε-approximation function.

Proposition 8 *Given an instance of DCLC and $f_\varepsilon(x)$, $f_\varepsilon(c(u,v)) \in \mathbb{Z}^+$ is a feasible ε-approximation function if there exists a λ such that $\forall e_{u,v} \in E$,*

$$(1+\varepsilon)c(u,v)\lambda \geq f_\varepsilon(c(u,v)) \geq c(u,v)\lambda. \tag{6.7}$$

Proof: Given any two sets of links, $E_1 \subset E$ and $E_2 \subset E$, if

$$\sum_{e_{u,v} \in E_1} c(u,v) < \sum_{e_{u,v} \in E_2} c(u,v), \tag{6.8}$$

and

$$\sum_{e_{u,v} \in E_1} f_\varepsilon(c(u,v)) \geq \sum_{e_{u,v} \in E_2} f_\varepsilon(c(u,v)), \tag{6.9}$$

then,

$$\begin{aligned} & (1+\varepsilon)\lambda \sum_{e_{u,v} \in E_1} c(u,v) \\ \geq & \sum_{e_{u,v} \in E_1} f_\varepsilon(c(u,v)) \\ \geq & \sum_{e_{u,v} \in E_2} f_\varepsilon(c(u,v)) \geq \lambda \sum_{e_{u,v} \in E_2} c(u,v)) \\ \Rightarrow & (1+\varepsilon) \sum_{e_{u,v} \in E_1} c(u,v) \geq \sum_{e_{u,v} \in E_2} c(u,v)). \end{aligned} \tag{6.10}$$

By Proposition 7, $f_\varepsilon(\cdot)$ is a feasible ε-approximation function. ■

Different from the functions satisfying Proposition 7, the ones satisfying Proposition 8 may be only feasible to a given instance of DCLC, in which link costs are already given, i.e., they may not be universally feasible. Since $\forall e_{u,v} \in E$, $f_\varepsilon(c(u,v)) \in \mathbb{Z}^+$ and $f_\varepsilon(c(u,v)) \in [\lambda c(u,v), (1+\varepsilon)\lambda c(u,v)]$, a straightforward solution is $f_\varepsilon(c(u,v)) = \lceil \lambda c(u,v) \rceil$. It should also be noted that for a given λ, $f_\varepsilon(c(u,v)) = \lceil \lambda c(u,v) \rceil$ may not be feasible since there may exist $e_{u,v} \in E$ such that

$$(1+\varepsilon)c(u,v)\lambda < f_\varepsilon(c(u,v)). \tag{6.11}$$

Hence, we will try to compute λ such that $\forall e_{u,v} \in E$, Eq. 6.7 is satisfied. As shown in Fig. 6.1, given a link $e_{u,v}$, because of the discrete nature of $\lceil \lambda c(u,v) \rceil$, there are many intersections between $(1+\varepsilon)\lambda c(u,v)$ and $\lceil \lambda c(u,v) \rceil$, and accordingly, the region for λ

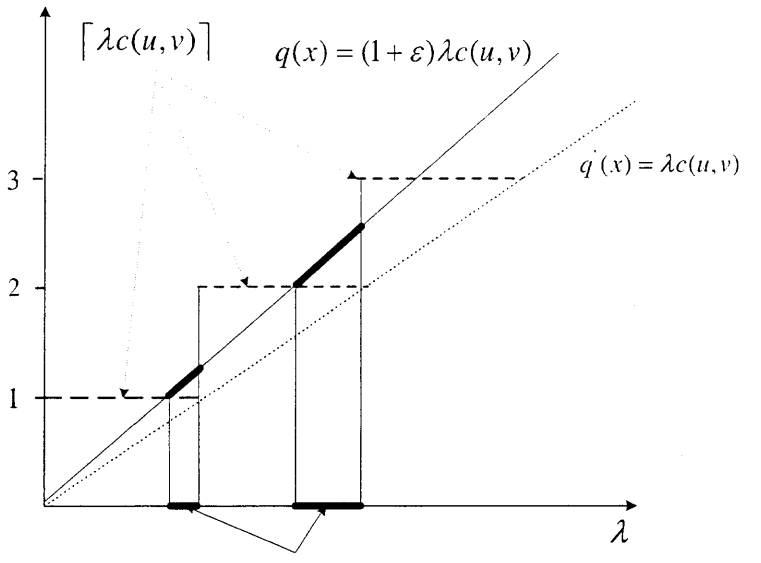

Figure 6.1 The illustration of the region of feasible linear scaling factor for a given link.

satisfying Eq. 6.7 is split into many intervals, which are highlighted with bold lines. We refer to the union of these intervals (of $e_{u,v}$) as the feasible region of $e_{u,v}$, and these intervals as the feasible intervals of $e_{u,v}$. Therefore, our objective is to find a point which exists in all feasible regions of links, i.e., to find λ such that $f_\varepsilon(x) = \lceil \lambda x \rceil$ is a feasible function. In other words, we need to find the λ that is located in the intersection of the feasible regions of all links. From Definition 13, the smaller the λ, the better. Hence, the optimal feasible linear scaling factor is the lower bound of the intersection of the feasible regions of all links. We numerically present the feasible intervals of a link by the next theorem.

Theorem 9 *Given an instance of DCLC, $f_\varepsilon(x) = \lceil \lambda x \rceil$ is a feasible ε-approximation function if $\forall e_{u,v} \in E$, $\exists k_{u,v} \in \overline{\mathbb{Z}^-}$ $(\{0\} \cup \mathbb{Z}^+)$ such that*

$$k_{u,v} + \max\{0, \frac{1-\varepsilon k_{u,v}}{1+\varepsilon}\} \leq \lambda c(u,v) \leq k_{u,v} + 1. \tag{6.12}$$

Proof: $\forall e_{u,v} \in E, \exists k_{u,v} \in \overline{\mathbb{Z}^-}$ such that

$$k_{u,v} + \max\{0, \frac{1-\varepsilon k_{u,v}}{1+\varepsilon}\} \leq \lambda c(u,v) \leq k_{u,v} + 1 \Rightarrow \lceil \lambda c(u,v) \rceil = k_{u,v} + 1 \quad (6.13)$$

Consider the case that $k_{u,v} < \frac{1}{\varepsilon}$,

$$\lceil \lambda c(u,v) \rceil = k_{u,v} + 1 = \left[k_{u,v} + \max\{0, \frac{1-\varepsilon k_{u,v}}{1+\varepsilon}\} \right] (1+\varepsilon) \leq (1+\varepsilon)\lambda c(u,v). \quad (6.14)$$

Consider the other case that $k_{u,v} \geq \frac{1}{\varepsilon}$,

$$k_{u,v} + \max\{0, \frac{1-\varepsilon k_{u,v}}{1+\varepsilon}\} = k_{u,v}. \quad (6.15)$$

Therefore,

$$(1+\varepsilon)\lambda c(u,v) = \lambda c(u,v) + \varepsilon\lambda c(u,v) \geq k_{u,v} + k_{u,v}\varepsilon \geq k_{u,v} + 1 \geq \lambda c(u,v). \quad (6.16)$$

Therefore, $f_\varepsilon(x) = \lceil \lambda x \rceil$ is a feasible ε-approximation function by Proposition 8. ■

Definition 14 *λ is feasible if $f_\varepsilon(x) = \lceil \lambda x \rceil$ is a feasible ε-approximation function, and λ is optimal if $f_\varepsilon(x) = \lceil \lambda x \rceil$ is the optimal feasible linear scaling ε-approximation function. λ is said to be feasible to a link $e_{u,v}$ if $\exists k_{u,v} \in \overline{\mathbb{Z}^-}$ such that*

$$k_{u,v} + \max\{0, \frac{1-\varepsilon k_{u,v}}{1+\varepsilon}\} \leq \lambda c(u,v) \leq k_{u,v} + 1. \quad (6.17)$$

Theorem 9 defines the constraints for $f_\varepsilon(x) = \lceil \lambda x \rceil$ to be a feasible ε-approximation function. Since our final objective is to minimize the computational complexity of ε-approximation algorithms by finding the least feasible linear scaling factor, it is preferable that the computational complexity introduced by computing the optimal λ is trivial, or negligible with respect to the overall computational complexity of ε-approximation algorithms.

Let

$$\alpha_k^{e(u,v)} = \frac{k}{c(u,v)} + \max\{0, \frac{1-k\varepsilon}{(1+\varepsilon)c(u,v)}\} \quad (6.18)$$

and

$$\beta_k^{e(u,v)} = \frac{k+1}{c(u,v)}. \tag{6.19}$$

By Theorem 9, if $f_\varepsilon(x) = \lceil \lambda x \rceil$ is feasible, $\lambda \in \bigcap_{e_{u,v} \in E} \zeta_{u,v}$, where

$$\zeta_{u,v} = \bigcup_{k=0}^{\infty} [\alpha_k^{e(u,v)}, \beta_k^{e(u,v)}]. \tag{6.20}$$

The lower bound of $\bigcap_{e_{u,v} \in E} \zeta_{u,v}$ is the optimal λ. Given a link $e(w,x) \in E$,

$$\begin{aligned} \bigcap_{e_{u,v} \in E} \zeta_{u,v} &= \left(\bigcup_{k=0}^{\infty} [\alpha_k^{e(w,x)}, \beta_k^{e(w,x)}] \right) \bigcap_{e(u,v) \neq e(w,x)} \zeta_{u,v} \\ &= \bigcup_{k=0}^{\infty} \left([\alpha_k^{e(w,x)}, \beta_k^{e(w,x)}] \bigcap_{e(u,v) \neq e(w,x)} \zeta_{u,v} \right). \end{aligned} \tag{6.21}$$

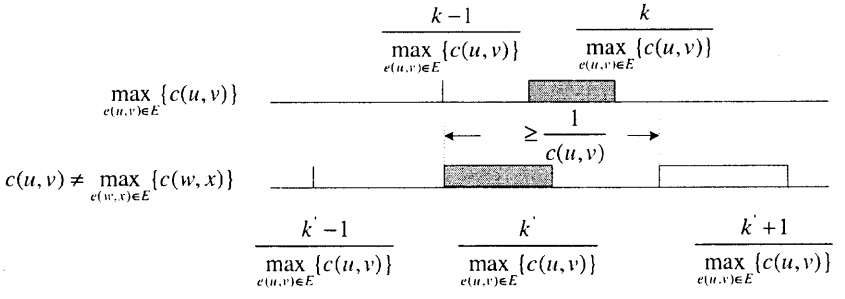

Figure 6.2 The shaded regions are the regions for λ to be feasible to the corresponding links.

We can iteratively search for the feasible λ in $[\alpha_k^{e(w,x)}, \beta_k^{e(w,x)}]$, where k is initialized as zero and increased by one after each iteration. Since $\lambda \in \mathbb{R}^+$ and there is exactly only one interval, $[\alpha_k^{e(u,v)}, \beta_k^{e(u,v)}]$, in $(\frac{k-1}{c(u,v)}, \frac{k}{c(u,v)}]$ for λ to be feasible to link $e(u,v)$, we divide $\mathbb{R}^+$ into an infinite number of periods, $(\frac{k-1}{c(u,v)}, \frac{k}{c(u,v)}]$ (each period has a fixed length of $\frac{1}{c(u,v)}$ and the kth period is from $\frac{k-1}{c(u,v)}$ to $\frac{k}{c(u,v)}$), $k \in \mathbb{Z}^+$, for each link $e(u,v)$. As shown

in Fig. 6.2, since $\frac{1}{c(u,v)} \geq \frac{1}{c_{\max}}$, for all $e(u,v) \in E$, at most two consecutive periods of link $e(u,v)$ would have non-empty intersections with one period of the link(s) having costs $c_{\max}$. Therefore, denoting $\alpha_k = \frac{k+\max\{0,\frac{1-\varepsilon k}{1+\varepsilon}\}}{c_{\max}}$, $\beta_k = \frac{k+1}{c_{\max}}$, $k^-_{u,v} = \lfloor \alpha_k c(u,v) \rfloor$, and $k^+_{u,v} = \lfloor \beta_k c(u,v) \rfloor$, we can prove the next theorem.

Theorem 10 *Given an instance of DCLC and $k \in \overline{\mathbb{Z}^-}$, a feasible ε-approximation function $f_\varepsilon(x) = \lceil \lambda x \rceil$, $\alpha_k \leq \lambda \leq \beta_k$, exists iff $\bigcap_{e(u,v)\in E} \pi_{u,v} \neq \Phi$, where*

$$\pi_{u,v} = \left[\alpha_{k^-}^{e(u,v)}, \beta_{k^-}^{e(u,v)}\right] \cup \left[\alpha_{k^+}^{e(u,v)}, \beta_{k^+}^{e(u,v)}\right] \cup \left[\alpha_k, \frac{k^-_{u,v}}{c(u,v)}\right] \tag{6.22}$$

and Φ is the empty set.

Proof: Note that, given a link $e_{u,v} \in E$, only when $\alpha_k c(u,v) \in \mathbb{Z}^+$, $[\alpha_k, \frac{k^-_{u,v}}{c(u,c)}] \neq \Phi$, i.e., $[\alpha_k, \frac{k^-_{u,v}}{c(u,c)}]$ is a point only when $\alpha_k c(u,v) \in \mathbb{Z}^+$. When $k^-_{u,v} = k^+_{u,v}$,

$$\left[\alpha_{k^-}^{e(u,v)}, \beta_{k^-}^{e(u,v)}\right] = \left[\alpha_{k^+}^{e(u,v)}, \beta_{k^+}^{e(u,v)}\right]. \tag{6.23}$$

Furthermore, $k^+_{u,v} - k^-_{u,v} \in \{0,1\}$ because

$$\begin{aligned} \beta_k c(u,v) - \alpha_k c(u,v) &= c(u,v)(\beta_k - \alpha_k) \\ &\leq 1 - \max\{0, \frac{1-\varepsilon k}{1+\varepsilon}\} \leq 1 \\ &\Rightarrow \lfloor \beta_k c(u,v) \rfloor - \lfloor \alpha_k c(u,v) \rfloor \leq 1. \end{aligned} \tag{6.24}$$

We first prove that if a feasible ε-approximation function $f_\varepsilon(x) = \lceil \lambda x \rceil$, $\alpha_k \leq \lambda \leq \beta_k$, exists, then $\bigcap_{e_{u,v}\in E} \pi_{u,v} \neq \Phi$. Given a link $e_{u,v}$, $c(u,v) \neq \max_{e_{w,x}\in E}\{c(w,x)\}$. Consider the case that $k^-_{u,v} = k^+_{u,v}$. By Theorem 1, if $f_\varepsilon(x) = \lceil \lambda x \rceil$ is a feasible ε-approximation function, $\exists k_{u,v} \in \overline{\mathbb{Z}^-}$ such that

$$k_{u,v} + \max\{0, \frac{1-\varepsilon k_{u,v}}{1+\varepsilon}\} \leq \lambda c(u,v) \leq k_{u,v} + 1$$

$$\Rightarrow \frac{k_{u,v}}{c(u,v)} + \max\{0, \frac{1-\varepsilon k_{u,v}}{(1+\varepsilon)c(u,v)}\} \leq \lambda \leq \frac{k_{u,v}+1}{c(u,v)}. \tag{6.25}$$

If $k_{u,v} < \lfloor \alpha_k c(u,v) \rfloor$,

$$\frac{k_{u,v}+1}{c(u,v)} \leq \frac{\lfloor \alpha_k c(u,v) \rfloor}{c(u,v)} \leq \frac{\alpha_k c(u,v)}{c(u,v)} = \alpha_k. \tag{6.26}$$

The equation is held only when $k_{u,v} = \lfloor \alpha_k c(u,v) \rfloor - 1$ and $\alpha_k c(u,v) \in \mathbb{Z}^+$ (it is impossible that $\alpha_k c(u,v) = 0$ because $\alpha_k > 0$ and $c(u,v) > 0$). If $k_{u,v} > \lfloor \beta_k c(u,v) \rfloor$,

$$\frac{k_{u,v}}{c(u,v)} + \max\{0, \frac{1-\varepsilon k_{u,v}}{(1+\varepsilon)c(u,v)}\} \geq \frac{\lfloor \beta_k c(u,v) \rfloor + 1}{c(u,v)} > \frac{\beta_k c(u,v)}{c(u,v)} = \beta_k. \tag{6.27}$$

Since $\alpha_k \leq \lambda \leq \beta_k$, if $\alpha_k c(u,v) \notin \mathbb{Z}^+$,

$$k_{u,v}^- = \lfloor \alpha_k c(u,v) \rfloor \leq k_{u,v} \leq \lfloor \beta_k c(u,v) \rfloor = k_{u,v}^+, \tag{6.28}$$

Therefore,

$$\begin{aligned} k_{u,v}^+ - k_{u,v}^- \quad \in \quad & \{0,1\} \Rightarrow \lambda \in [\alpha_k, \beta_k] \cap \\ & \left(\left[\alpha_{k^-}^{e(u,v)}, \beta_{k^-}^{e(u,v)} \right] \cup \left[\alpha_{k^+}^{e(u,v)}, \beta_{k^+}^{e(u,v)} \right] \cup \left[\alpha_k, \frac{k_{u,v}^-}{c(u,v)} \right] \right). \end{aligned} \tag{6.29}$$

So, if a feasible ε-approximation function $f_\varepsilon(x) = \lceil \lambda x \rceil$, $\alpha_k \leq \lambda \leq \beta_k$, exists, $\bigcap_{e_{u,v} \in E} \pi_{u,v} \neq \Phi$, where

$$\pi_{u,v} = [\alpha_k, \beta_k] \cap \left(\left[\alpha_{k^-}^{e(u,v)}, \beta_{k^-}^{e(u,v)} \right] \cup \left[\alpha_{k^+}^{e(u,v)}, \beta_{k^+}^{e(u,v)} \right] \cup \left[\alpha, \frac{k_{u,v}^-}{c(u,v)} \right] \right). \tag{6.30}$$

Note that $\pi_{u,v} = [\alpha_k, \beta_k]$ if $c(u,v) = \max_{e_{w,x} \in E} \{c(w,x)\}$. We can simplify $\pi_{u,v}$ as

$$\pi_{u,v} = \left[\alpha_{k^-}^{e(u,v)}, \beta_{k^-}^{e(u,v)} \right] \cup \left[\alpha_{k^+}^{e(u,v)}, \beta_{k^+}^{e(u,v)} \right] \cup \left[\alpha_k, \frac{k_{u,v}^-}{c(u,v)} \right]. \tag{6.31}$$

If $\bigcap_{e_{u,v} \in E} \pi_{u,v} \neq \Phi$, by Theorem 9, a feasible ε-approximation function $f_\varepsilon(x) = \lceil \lambda x \rceil$, $\alpha_k \leq \lambda \leq \beta_k$, exists. ■

Next, we will try to find the smallest feasible λ based on Theorem 10. Since $[\alpha_k, \frac{k^-_{u,v}}{c(u,v)}]$ is a point only when $\alpha_k c(u,v) \in \mathbb{Z}^+$, the point can be eliminated by checking if α_k is the solution (optimal λ). If α_k is feasible, the smallest feasible λ in $[\alpha_k, \beta_k]$ must be α_k. Otherwise, we set

$$\pi_{u,v} = \left[\alpha_{k^-}^{e(u,v)}, \beta_{k^-}^{e(u,v)}\right] \cup \left[\alpha_{k^+}^{e(u,v)}, \beta_{k^+}^{e(u,v)}\right]. \tag{6.32}$$

Let

$$\pi = \bigcap_{k^-_{u,v}=k^+_{u,v}} \pi_{u,v} = \bigcap_{k^-_{u,v}=k^+_{u,v}} \left[\alpha_{k^-}^{e(u,v)}, \beta_{k^-}^{e(u,v)}\right] = \left[\max_{k^-_{u,v}=k^+_{u,v}} \left\{\alpha_{k^-}^{e(u,v)}\right\}, \min_{k^-_{u,v}=k^+_{u,v}} \left\{\beta_{k^-}^{e(u,v)}\right\}\right]. \tag{6.33}$$

The remaining problem is thereby to compute

$$\pi \cap \left(\bigcap_{k^-_{u,v}\neq k^+_{u,v}} \pi_{u,v}\right). \tag{6.34}$$

It should be noted that if $\pi_{u,v}$ ($e_{u,v} \in E$) are independent with each other, the computational complexity of computing $\pi \cap \left(\bigcap_{k^-_{u,v}\neq k^+_{u,v}} \pi_{u,v}\right)$ could be rather high. However, as shown in Fig. 6.2, if

$$\beta_{k^+}^{e(u,v)} - \beta_{k^-}^{e(u,v)} > \frac{1}{c_{\max}}, \tag{6.35}$$

i.e., $\beta_{k^+}^{e(u,v)} - \beta_{k^-}^{e(u,v)}$ is larger than the period of links that have costs of $c_{\max}$, at most one of two regions, $\left[\alpha_{k^-}^{e(u,v)}, \beta_{k^-}^{e(u,v)}\right]$ and $\left[\alpha_{k^+}^{e(u,v)}, \beta_{k^+}^{e(u,v)}\right]$, has a non-empty intersection with the kth period of the links with cost $c_{\max}$ because either $\beta_{k^+}^{e(u,v)} > \beta_k$ or $\beta_{k^-}^{e(u,v)} < \alpha_k$. Hence, the next theorem is introduced for the purpose of further reducing the complexity of computing $\pi \cap \left(\bigcap_{k^-_{u,v}\neq k^+_{u,v}} \pi_{u,v}\right)$.

Theorem 11 *Given a link $e_{u,v} \in E$, if $k^+_{u,v} - k^-_{u,v} = 1$, and $c(u,v) < \frac{c_{\max}}{1+\varepsilon}$, either*

$$\left[\alpha_{k^-}^{e(u,v)}, \beta_{k^-}^{e(u,v)}\right] \cap [\alpha_k, \beta_k] = \Phi, \tag{6.36}$$

or

$$\left[\alpha_{k+}^{e(u,v)}, \beta_{k+}^{e(u,v)}\right] \cap [\alpha_k, \beta_k] = \Phi. \tag{6.37}$$

Proof: When $k_{u,v}^{+} - k_{u,v}^{-} = 1$,

$$\begin{aligned} & \frac{k_{u,v}^{+}}{c(u,v)} + \max\{0, \frac{1 - k_{u,v}^{+}\varepsilon}{(1+\varepsilon)c(u,v)}\} - \frac{k_{u,v}^{-}}{c(u,v)} - \max\{0, \frac{1 - k_{u,v}^{-}\varepsilon}{(1+\varepsilon)c(u,v)}\} \\ = & \frac{1}{c(u,v)} + \max\{0, \frac{1 - k_{u,v}^{+}\varepsilon}{(1+\varepsilon)c(u,v)}\} - \max\{0, \frac{1 - k_{u,v}^{-}\varepsilon}{(1+\varepsilon)c(u,v)}\} \\ \geq & \frac{1}{c(u,v)} - \frac{\varepsilon}{(1+\varepsilon)c(u,v)} = \frac{1}{(1+\varepsilon)c(u,v)} > \frac{1}{c_{\max}}, \end{aligned} \tag{6.38}$$

while

$$\beta_k - \alpha_k = \frac{1 - \max\{0, \frac{1-\varepsilon k}{1+\varepsilon}\}}{c_{\max}} \leq \frac{1}{c_{\max}}. \tag{6.39}$$

Hence, either

$$\left[\alpha_{k-}^{e(u,v)}, \beta_{k-}^{e(u,v)}\right] \cap [\alpha_k, \beta_k] = \Phi, \tag{6.40}$$

or

$$\left[\alpha_{k+}^{e(u,v)}, \beta_{k+}^{e(u,v)}\right] \cap [\alpha_k, \beta_k] = \Phi. \tag{6.41}$$

■

By Theorem 11, given a network in which $\forall\ e(u,v)$, either $c(u,v) = c_{\max}$ or $c(u,v) < \frac{c_{\max}}{1+\varepsilon}$, we have

$$\left[\alpha_{n(u,v)}^{e(u,v)}, \beta_{n(u,v)}^{e(u,v)}\right] \cap [\alpha_k, \beta_k] \neq \Phi \tag{6.42}$$

only when $n(u,v) \in \{k^+_{u,v}, k^-_{u,v}\}$, where $n(u,v)$ is the integer such that Eq. 6.42 is satisfied. Hence,

$$\bigcap_{e_{u,v}\in E} \pi_{u,v} = \bigcap_{e_{u,v}\in E} \left[\alpha^{e(u,v)}_{n(u,v)}, \beta^{e(u,v)}_{k(u,v)}\right] = \left[\max_{e_{u,v}\in E}\{\alpha^{e(u,v)}_{n(u,v)}\}, \min_{e_{u,v}\in E}\{\beta^{e(u,v)}_{n(u,v)}\}\right]. \tag{6.43}$$

In other words,

$$\begin{aligned} 1+\varepsilon &< \frac{c_{\max}}{\max\limits_{c(u,v)\neq c_{\max}}\{c(u,v)\}} \\ &\Longrightarrow \bigcap_{e_{u,v}\in E} \pi_{u,v} = \bigcap_{e_{u,v}\in E} \left[\alpha^{e(u,v)}_{n(u,v)}, \beta^{e(u,v)}_{k(u,v)}\right] \\ &= \left[\max_{e_{u,v}\in E}\{\alpha^{e(u,v)}_{n(u,v)}\}, \min_{e_{u,v}\in E}\{\beta^{e(u,v)}_{n(u,v)}\}\right]. \end{aligned} \tag{6.44}$$

Accordingly, the computational complexity of computing a feasible linear scaling factor in $[\alpha_k, \beta_k]$ becomes $O(m)$, which results from computing $\max\limits_{e_{u,v}\in E}\{\alpha^{e(u,v)}_{n(u,v)}\}$ and $\min\limits_{e_{u,v}\in E}\{\beta^{e(u,v)}_{n(u,v)}\}$.

6.4 Proposed Algorithms for Searching the Optimal Linear Scaling Factor

In this section, we will propose two efficient algorithms for searching the optimal linear scaling factor, which can be incorporated into ε-approximation algorithms to reduce their computational complexities. We first introduce the next theorem, based on which an upper bound on the optimal linear scaling factor can be derived.

Theorem 12 *Given an instance of DCLC, the linear scaling ε-approximation solution computed with a linear scaling factor λ has a cost less than*

$$c^* + \frac{h}{\lambda}, \tag{6.45}$$

where c^* and h are the cost and the number of hops of the optimal feasible path of DCLC, respectively.

Proof: Assume the optimal path of DCLC is p, and the path computed by using a linear scaling factor λ is $\widehat{p}$. Therefore,

$$\begin{aligned} \sum_{e(u,v)\in\widehat{p}} c(u,v)\lambda &\leq \sum_{e(u,v)\in\widehat{p}} \lceil c(u,v)\lambda \rceil \leq \\ \sum_{e(u,v)\in p} \lceil c(u,v)\lambda \rceil &< \sum_{e(u,v)\in p} [c(u,v)\lambda + 1] \\ &= \sum_{e(u,v)\in p} c(u,v)\lambda + h. \end{aligned} \tag{6.46}$$

Hence, the feasible path computed with linear scaling factor λ has a cost less than $c^* + \frac{h}{\lambda}$.

■

Based on Theorem 12, the next lemma provides an upper bound on the optimal feasible linear scaling factor of a given instance of DCLC.

Lemma 13 *Given an instance of DCLC, the linear scaling ε-approximation solution computed by using a linear scaling factor $\lambda = \frac{n-1}{L\varepsilon}$ is an ε-approximation solution, where L is a lower bound on c^*.*

Proof: By Theorem 12, the path computed with linear scaling factor $\lambda = \frac{n-1}{L\varepsilon}$ has a cost less than

$$c^* + \frac{h}{\lambda} = c^* + \frac{h}{n-1} L\varepsilon \leq c^* + L\varepsilon \leq c^*(1+\varepsilon). \tag{6.47}$$

Hence, the path computed with a linear scaling factor $\lambda = \frac{n-1}{L\varepsilon}$ is an ε-approximation solution. ■

How to compute L is beyond the scope of this chapter. Readers can refer to [58] and [59] for related information. We adopt $\mu = \frac{n-1}{L\varepsilon}$ as an upper bound on the optimal linear scaling factor by Lemma 13.

The first algorithm, called Optimal Linear Scaling ε-approximation Algorithm (OLSA), is recommended for cases where $\forall e(u,v) \in E$, either $c(u,v) = c_{\max}$ or $c(u,v) < \frac{c_{\max}}{1+\varepsilon}$, so that Theorem 11 can be applied to reduce the computational complexity

for searching the optimal linear scaling factor. As mentioned before, we use μ as a loose upper bound for the optimal linear scaling factor. The basic idea behind OLSA is that the optimal linear scaling factor λ can be found by Theorems 10 and 11 by gradually increasing the integer k until λ reaches μ, implying that k is limited by

$$k_{\max} = \lfloor \mu c_{\max} \rfloor , \tag{6.48}$$

in OLSA. Without loss of generality, we assume $c_{\max}= 1$ for the rest of the chapter, which can be achieved simply by dividing all the link costs by $c_{\max}$.

Thus, denote $\varsigma_k = [k + \max\{0, \frac{1-\varepsilon k}{1+\varepsilon}\}, k + 1]$, OLSA consists of the following steps:

Step 1. Initialize $k = 0$.

Step 2. AApply Theorems 10 and 11 to find a feasible linear scaling factor in ς_k. If $\bigcap \pi_{u,v} \neq \Phi$ at the kth iteration, the optimal feasible λ must be the lower bound of $\bigcap \pi_{u,v}$. Hence, set λ to the lower bound of $\bigcap \pi_{u,v}$, and the optimal feasible linear scaling factor is found. End.

Step 3. $k = k + 1$. If $k = k_{\max}$, go to step 4. Otherwise, go to step 2.

Step 4. By Lemma 13, let $\lambda = \mu$.

Since the computational complexity of computing a feasible linear scaling factor in ς_k is, by Theorems 10 and 11, $O(m)$, and there are totally $\lceil \lambda c_{\max} \rceil$ iterations in OLSA, the computational complexity of OLSA is

$$O(\lceil \lambda c_{\max} \rceil \, m) = O(\lceil \lambda \rceil \, m). \tag{6.49}$$

It can be observed that the worst case computational complexity of OLSA is very low. On the other hand, since OLSA always locates the optimal λ, the computational complexities of ε-approximation algorithms deploying OLSA can be reduced, especially in some special cases. For example, for a given network $G(N, E)$, and $\forall e(u, v) \in E$, $c(u, v) \in \{k/3, k =$

$1, 2, ...\}$, $\lambda = 3$ is a feasible linear scaling factor, which is independent of ε. As a result, the computational complexities of ε-approximation algorithms are greatly reduced when ε is small. Moreover, in this case, the computational complexities become polynomial, no longer pseudo polynomial anymore.

Remark 1 *As mentioned earlier, most ε-approximation approaches adopt Bellman-Ford-like algorithms, e.g., RSP [59] and DAD [62]. Therefore, their worst-case computational complexities are linearly proportional to the upper bound of the path costs (the computational complexity of DAD is linearly proportional to the delay bound). For example, given a delay bound d and an upper bound U on the cost of the optimal feasible path, the computational complexities of RSP [59] is $O(mU)$ (it should be noted that link costs must be integers to deploy RSP). Given a linear scaling factor λ and an upper bound U on path costs, in order to achieve 100% success ratio in finding the optimal feasible path with a cost less than U, the upper bound is adjusted to $O(\lambda U)$ after all link costs are linearly scaled by λ. Accordingly, the computational complexities of ε-approximation*

algorithm is proportional to $O(m \lceil \lambda U \rceil)$. Note that U must be larger than c_{max}. Otherwise, we can prune the links whose costs are larger than U. Therefore,

$$O(m \lceil \lambda U \rceil) > O(\lceil \lambda c_{max} \rceil m) = O(\lceil \lambda \rceil m), \tag{6.50}$$

which implies that adopting OLSA will not increase the computational complexities of ε-approximation algorithms. Note that the upper bound of λ is $\mu = \frac{n-1}{L\varepsilon}$, and in [58], an efficient method has been introduced to find U and L such that $\frac{U}{L} \leq \alpha$, where α is a constant. Therefore, the worst-case computational complexity of OLSA is

$$O\left(m \left\lceil U \frac{n-1}{L\varepsilon} \right\rceil\right) = O(\frac{mn}{\varepsilon}), \tag{6.51}$$

which is, to our best knowledge, no larger than the computational complexity of any DCLC ε-approximation solution, implying that OLSA can be adopted by ε-approximation

algorithms to minimize their linear scaling factors without increasing their computational complexities. On the other hand, since the computational complexities of ε-approximation algorithms are linearly proportional to the linear scaling factor, by adopting OLSA, their computational computational complexities can be reduced. Finally, we need to point out that since OLSA is proposed under the assumption that $\lceil \cdot \rceil$ is used for scaling, it may not be directly applicable to algorithms, e.g., DSA [62] and S-OPQR [59], in which $\lfloor \cdot \rfloor$ is deployed.

Note that OLSA is not applicable to the cases where $\exists e(u,v) \in E$ such that $c(u,v) \neq c_{\max}$ and $c(u,v) \geq \frac{c_{\max}}{1+\varepsilon}$. Thus, we propose our second algorithm, Transformed Optimal Linear Scaling ε-approximation Approach (T-OLSA), by first setting the costs of the links with costs no less than $\frac{c_{\max}}{1+\varepsilon}$ to $c_{\max}$, and then applying OLSA.

Theorem 14 *Given an instance of DCLC, construct $G'(V,E)$ by setting the costs of the links in $G(V,E)$ whose costs are not less than $\frac{c_{\max}}{1+\varepsilon}$ to $c_{\max}$, and then construct $\widetilde{G}(V,E)$ from $G'(V,E)$ by linearly scaling the link costs by λ, where λ is the optimal linear scaling factor computed by OLSA in $G'(V,E)$. Assume the least cost feasible path in $\widetilde{G}(V,E)$ is $\widetilde{p}$. $\widetilde{p}$ has a cost in $G(V,E)$ no larger than $(1+\varepsilon)^2 c^*$, where c^* is the cost of the optimal feasible path in $G(V,E)$.*

Proof: Assume p is the optimal feasible path of $G(V,E)$, p' the optimal feasible path in $G'(V,E)$, and $c'(p')$ the cost of p' in $G'(V,E)$. Since $\widetilde{G}(V,E)$ is constructed from $G'(V,E)$ by linearly scaling the link costs by λ, and $\widetilde{p}$ is the optimal feasible path of $\widetilde{G}(V,E)$ (or $\widetilde{p}$ is the solution of an ε-approximation algorithm computed in $\widetilde{G}(V,E)$),

$$c'(\widetilde{p}) \leq (1+\varepsilon)c'(p'). \tag{6.52}$$

Define $c'(u,v)$ as the cost of link $e_{u,v}$ in $G'(V,E)$. Since p' is the optimal feasible path in $G'(V,E)$, and $G'(V,E)$ is constructed by setting the costs of the links in $G(V,E)$ whose

costs are no less than $\frac{c_{\max}}{1+\varepsilon}$ to $c_{\max}$, implying that for any link $e(u,v)$, $c'(u,v) \leq (1+\varepsilon)c(u,v)$, and thus,

$$c'(p') \leq c'(p) = \sum\nolimits_{e_{u,v} \in p} c'(u,v) \leq \sum\nolimits_{e_{u,v} \in p} c(u,v)(1+\varepsilon) = (1+\varepsilon)c^*,$$

where $c'(p)$ is the cost of p in $G'(V,E)$. Thus

$$c'(\widetilde{p}) \leq (1+\varepsilon)c'(p') \leq (1+\varepsilon)^2 c^*. \tag{6.53}$$

Therefore,

$$c(\widetilde{p}) \leq c'(\widetilde{p}) \leq (1+\varepsilon)^2 c^*, \tag{6.54}$$

where $c(\widetilde{p})$ is the cost of $\widetilde{p}$ in $G(V,E)$, i.e., $\widetilde{p}$ has a cost in $G(V,E)$ no larger than $(1+\varepsilon)^2 c^*$.

■

Thus, T-OLSA consists of the following steps:

Step 1. Set the costs of the links whose cost are no less than $\frac{c_{\max}}{1+\varepsilon'}$ as $c_{\max}$, where $\varepsilon' = \sqrt{1+\varepsilon} - 1$.

Step 2. Initialize $k = 0$.

Step 3. Apply Theorems 10 and 11 to find a feasible linear scaling factor in ς_k. If $\bigcap \pi_{u,v} \neq \Phi$ at the kth iteration, the optimal feasible λ must be the lower bound of $\bigcap \pi_{u,v}$. Hence, set λ to the lower bound of $\bigcap \pi_{u,v}$, and the smallest feasible linear scaling factor is found. End.

Step 4. $k = k + 1$. If $k = k_{\max}$, go to step 5. Otherwise, go to step 2.

Step 5. By Lemma 13, let $\lambda = \mu$ and set $\varepsilon' = \varepsilon$.

Lemma 15 *Given an instance of DCLC and ε, by letting $\varepsilon' = \sqrt{1+\varepsilon} - 1$, a path with cost no larger than $(1+\varepsilon)c^*$ can be found by an ε-approximation algorithm (with parameter ε') if T-OLSA is adopted for finding the linear scaling factor.*

Proof: Assume the path computed by the ε-approximation algorithm with parameter $\varepsilon' = \sqrt{1+\varepsilon} - 1$ is p. Therefore, by Theorem 14,

$$c(p) \leq (1+\varepsilon')^2 c^* = (1+\varepsilon)c^*, \tag{6.55}$$

where c^* is the cost of the optimal feasible path. ■

Observe that T-OLSA does not totally rely on OLSA. Since $\varepsilon' < \varepsilon$, by Lemma 1, the computational complexity may unnecessarily increase if λ is larger than μ. In this case, we can directly solve the original problem (link costs are not modified) by setting

$$\lambda = \mu. \tag{6.56}$$

Therefore, the worst-case computational complexity of T-OLSA is the same as that of OLSA.

6.5 Summary

In this chapter, having observed that the computational complexity of many ε-approximation algorithms is linearly proportional to the linear scaling factor, we have investigated the issue of finding the optimal linear scaling factor in order to reduce their computational complexities. Two algorithms, Optimal Linear Scaling Algorithm (OLSA) and Transformed Optimal Linear Scaling Algorithm (T-OLSA), have been proposed. We have analytically shown that incorporating the two algorithms into DCLC ε-approximation solutions not only does not increase but in fact reduce their computational complexities because the optimal linear scaling factor can always be found. We have also shown that in some special cases, with our proposed algorithms, the worst-case computational complexities of ε-approximation solutions are no longer pseudo-polynomial, i.e., they become strictly polynomial and independent of ε. Besides the DCLC ε-approximation solutions, our proposed algorithms can be applied to all linear ε-approximation solutions

in which $\lceil \cdot \rceil$ is used for scaling. Similarly, algorithms can also be developed for cases in which $\lfloor \cdot \rfloor$ is deployed for scaling.

CHAPTER 7

MINIMIZING THE IMPACT OF STALE LINK STATE INFORMATION ON QOS ROUTING

7.1 Introduction

Designing a QoS routing algorithm based on a specific update policy has been rarely considered. In this chapter, we show that general QoS routing algorithms without considering the staleness of the link state information may introduce unignorable percentage of false routing. Hence, under the assumption that trigger-based link state update policies are deployed for updating link state information in the networks, we introduce and investigate the problem of finding the most probable feasible path without stochastic link state knowledge, and propose a novel QoS routing algorithm, which is capable of minimizing the impact of stale link state information. As a result, the probability of false routing is minimized. Our proposed routing algorithm, Heuristic Most Probable Path (HMPR) algorithm, is based on Iterative Multiple Additive Constrained Path (IMACP) selection and DynAmic Extended Bellman-ford (DAEB) algorithm. IMACP is a high performance and progressive solution capable of finding a path in meeting multiple additive constraints with a fairly low computational complexity. DAEB is a dynamic programming solution for the special case of Least Cost Multiple Additively Constrained Path (LCMACP) selection where the link costs are non-negative integers. We show by theoretical analysis and extensive simulations that HMPR not only minimizes the undesirable effect of the staleness of link state information but also achieves high success ratio in finding the feasible path with very low average computational complexity.

7.2 Proposed Routing Algorithm

We first lay out the assumptions made in this chapter. Practically, protocol overhead will be intolerably high if link state is updated whenever a minor change occurs. Hence, for the sake of reducing the protocol overhead, the staleness of link state information is inevitably introduced. In this chapter, we focus on minimizing the impact of the staleness of link state information (introduced by the link state update policy) on QoS routing. Hence, we ignore the effect of the dissemination delay on the link state update and simply assume that the dissemination delay is zero, i.e., all nodes can instantly receive the link state information upon an update. In this section, for simplicity, we only consider the class based link state update policies. We will show that our proposed routing algorithm can be easily extended to cases where other trigger-based update policies are adopted. We will not focus only on a single QoS metric (e.g., bandwidth) as in [27] because the routing algorithm proposed in this chapter should be able to compute feasible paths subject to multiple constraints. Assume that $c_1^j(t)$, $c_2^j(t)$, ..., $c_M^j(t)$ are M QoS metrics associated with link j at moment t, and we partition the ith metric of link j into k_i^j classes

$$((0, B_{i,j}^1), (B_{i,j}^1, B_{i,j}^2), ...(B_{i,j}^{k_i^j-1}, B_{i,j}^{k_i^j})). \tag{7.1}$$

Many studies have been done to characterize the Internet traffic, revealing interesting facts such as Long-Range Dependence or multi-fractal behaviors [33]-[35]. Therefore, we cannot assume that the link state is memoryless. Hence, each QoS metric of a link ($c_1^j(t)$, $c_2^j(t)$, ..., and $c_M^j(t)$) is viewed as a random process with memory. Without loss of generality, we assume that for any i and j, $k_i^j = k_i$ and $B_{i,j}^\beta = B_i^\beta$, i.e., $\forall i \in \{1, 2, ..., M\}$, the ith link metrics of all links are partitioned into the same set of classes, where k_i represents the number of classes for the ith metric and B_i^β the lower bound of the $(\beta + 1)$th class of the ith metric. As mentioned above, an update is triggered only when the available bandwidth crosses a class boundary. Denote $C_1^{j,u}(t)$, $C_2^{j,u}(t)$, ..., $C_M^{j,u}(t)$ as the latest updated link state of link j of node u at moment t, i.e., from the perspective of node u, the M QoS metrics

of link j at moment t are $C_1^{j,u}(t)$, $C_2^{j,u}(t)$, ..., $C_M^{j,u}(t)$. Since we ignore the staleness of link state information introduced by the dissemination delay, for any two nodes u and v,

$$C_i^{j,u}(t) = C_i^{j,v}(t) = C_i^j(t). \tag{7.2}$$

Moreover, $\exists \alpha_i^j(t)$ ($0 \leq \alpha_i^j(t) \leq k_i$) such that

$$B_i^{\alpha_i^j(t)-1} \leq C_i^j(t) \leq B_i^{\alpha_i^j(t)}. \tag{7.3}$$

Since we assume that the partitions (classes) for the metrics of all links are the same, as long as $C_i^j(t)$ is available to a node, $B_i^{\alpha_i^j(t)-1}$ and $B_i^{\alpha_i^j(t)}$ are also known. In fact, if the assumption does not hold for some networks, we can encode $B_i^{\alpha_i^j(t)-1}$ and $B_i^{\alpha_i^j(t)}$ in the update packets by which link state information is distributed. At any moment, when a connection request arrives at a node, we assume that the node tries to compute a path meeting the QoS requirement of the connection according to its available link state information (source routing). If, from the perspective of the node, there is enough network resource (bandwidth) to accommodate this connection, it starts a setup process for the connection. Otherwise, it rejects the connection request immediately. Ideally, the connection is accepted if there is actually enough network resource, and rejected otherwise. However, due to inaccurate link state information ($C_i^j(t) \neq c_i^j(t)$) and the adopted routing algorithm, there are two possible undesirable cases:

- False positive: There is actually not enough network resource to accommodate a connection, but is indicate otherwise by its link state information. Since a setup process will be initialized by the node, network resource is wasted.

- False negative: A connection can actually be accepted by the network, but is rejected by the node because of inaccurate link state information or failure of the adopted routing algorithm in finding a feasible path.

Collectively, both cases are referred to as the false routing in this paper. Designing a link state update policy to improve the accuracy of updated link state information is beyond the scope of this paper. Hence, our objective here is to design a highly efficient and effective QoS routing algorithms, which is capable of minimizing the occurrences of above two cases, while achieving high success ratio in finding a real feasible path. We formulate our problem as following:

Definition 15 *Given a network $G(N, E)$, where N is the set of nodes, and E the set of links. Assume c_1^j, c_2^j, ..., c_M^j and C_1^j, C_2^j, ..., C_M^j are the real M QoS metrics associated with link j, $j = 1, 2, ..., m$, and the ones from the perspective of the nodes, where m is the number of links. Find a (real) feasible path p from node s to a destination subject to $\forall i = 1, 2, ..., M$, $\sum_{j \in p} c_i^j < \rho_i$ if c_i^j is an additive metric, or $\min_{j \in p} c_i^j > \rho_i$ if c_i^j is a concave metric, where $\rho_1, \rho_2, ..., \rho_M$ are given QoS constraints.*

Generally, QoS constraints can be classified into three categories: concave, additive, and multiplicative. Since multiplicative constraints can be converted to additive constraints by using the logarithm function, we only consider concave and additive constraints in this paper.

Ideally, we hope that our routing algorithm can achieve 100% success ratio in finding a real feasible path and no false negative and positive. However, since from the perspective of a node, it cannot know the real link metrics from the updated ones, i.e., it is impossible to know c_i^j from C_i^j, no false negative and positive is impossible.

On the other hand, we can reduce the probability of false negative and positive by taking the advantage of the relationship between the current link status and the updated state information. For instance, given a network shown as Fig. 7.1, in which we denote $B^{\alpha^j - 1}$, C^j, and $B^{\alpha_i^j}$ of link j in the form of $(B^{\alpha^j - 1}, C^j, B^{\alpha^j})$ (since there is only one constraint, we ignore the sub index i in this example). Assume the metric is concave and a connection request from node a to node c with constraint of $\rho = 0.45$ arrives. Both paths (a, b, c) and (a, d, c) seem to be feasible paths according to updated link state information because

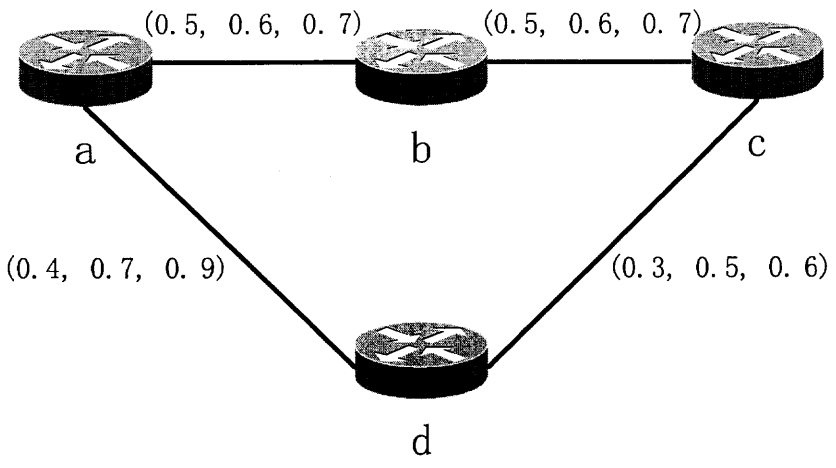

Figure 7.1 A 4-node network.

$\min_{j \in p} C^j > \rho$, $p \in \{(a, b, c), (a, d, c)\}$, where $(a_1, a_2, ..., a_h)$ represents an $(h-1)$-hop path from node α_1 to node α_h sequentially traversing nodes $\alpha_1, \alpha_2, ..., \alpha_h$. However, since the lower bounds of current classes of links $e(a, d)$ and $e(d, c)$ are 0.4 and 0.3, respectively, it is possible that $\min_{j \in (a,d,c)} c^j < \rho$, i.e., the path (a, d, c) can be an infeasible path. On the other hand, since the current link states of links $e(a, b)$ and $e(b, c)$ are both $(0.5, 0.6, 0.7)$, implying that

$$\min_{j \in (a,b,c)} c^j > \min_{j \in (a,b,c)} B^{\alpha^j - 1} = 0.5 > \rho. \tag{7.4}$$

Hence, we can guarantee that the path (a, b, c) is a real feasible path. Denote α_i^j as the integer such that

$$B_i^{\alpha_i^j - 1} \leq C_i^j \leq B_i^{\alpha_i^j} \Rightarrow B_i^{\alpha_i^j - 1} \leq c_i^j \leq B_i^{\alpha_i^j}. \tag{7.5}$$

We call tightening the ith metric of link j as setting the ith metric of link j as $B_i^{\alpha_i^j - 1}$ when ith metric is concave, and setting the ith metric of link j as $B_i^{\alpha_i^j}$ when ith metric is additive.

Theorem 16 *Assume $G_t(N, E, i)$ is the network constructed from $G(N, E)$ by tighening the ith metrics of all links. If p is a feasible path of $G_t(N, E, i)$, p must also be a feasible path of $G(N, E)$.*

Proof: Consider the case that the ith metric is concave. By Eq. 7.5,

$$\min_{j \in p} B_i^{\alpha_i^j - 1} \geq \alpha_i \Rightarrow \min_{j \in p} c_i^j \geq \rho_i. \tag{7.6}$$

Therefore, p statisfies the ith constraint when it is concave. Consider the other case that the $i$the metric is additive. By Eq. 7.5,

$$\sum_{j \in p} B_i^{\alpha_i^j} \leq \alpha_i \Rightarrow \sum_{j \in p} c_i^j \leq \rho_i. \tag{7.7}$$

i.e., the sum of ith metrics of p is less than the constraint. Therefore, p is a feasible path of $G(N, E)$. ■

Denote $G_t(N, E)$ as the network constructed from $G(N, E)$ by tightening all metrics of links. We can eliminate false positive by above theorem because the feasible path computed in $G_t(N, E)$ must also be a feasible path of $G(N, E)$, i.e., without knowing the exact link state information, no false positive may be achieved by Theorem 16. However, a feasible path of $G(N, E)$ may not be a feasible path of $G_t(N, E)$. Hence, it is possible that we fail to find a feasible path in $G_t(N, E)$ although a real one exists in $G(N, E)$. Therefore, the false negative may be increased if we only compute a path in $G_t(N, E)$.

Definition 16 *For a concave constraint ρ_i, $i = 1, 2, ..., M$, link j is referred to as an ensured feasible link if $B_i^{\alpha_i^j - 1} \geq \rho_i$, an unsured feasible link if $C_i^j \geq \rho_i$ and $B_i^{\alpha_i^j - 1} < \rho_i$, an unsured infeasible link if $C_i^j < \rho_i$ but $B_i^{\alpha_i^j} \geq \rho_i$, and an ensured infeasible link if $B_i^{\alpha_i^j} < \rho_i$.*

If no feasible path can be found in $G_t(N, E)$, we hope find a path with largest probability to be a real feasible path. However, in this paper, we only assume that a node

knows the updated link state information, i.e., no stochastic knowledge of link metrics is available to the nodes. Hence, we cannot adopt a probablistic approach as [50]. Instead, we propose an alternative way to minimize the probability of false routing. Assume p is a path consisting of links 1, 2, ..., h, which satisfies $\min_{j=1,2,...,h}\{C^j\} > \rho$ and $\min_{j=1,2,...,h}\{B^{\alpha^j-1}\} < \rho$, where ρ is the concave constraint. Assume $\forall j \in \{1, 2, ..., h\}$, $p(c^j > \rho | C^j > \rho, B^{\alpha^j-1} < \rho) = \delta$. Since $\min_{j=1,2,...,h}\{C^j\} < \rho$ and $\min_{j=1,2,...,h}\{B^{\alpha^j-1}\} < \rho$, there mush exist $j \in \{1, 2, ..., h\}$ such that $C^j > \rho$ and $B^{\alpha^j-1} < \rho$. Moreover, assume the link metrics on any two link are independent with each other, and there are totally $\widetilde{h}$ links, i_1, i_2, ..., $i_{\widetilde{h}}$, in p satisfying that $\forall j \in \{i_1, i_2, ..., i_{\widetilde{h}}\}$, $C^j > \rho$ and $B^{\alpha^j-1} < \rho$ ($\widetilde{h}$ unsured feasible links). The probability for p to be a real feasible path becomes $\delta^{\widetilde{h}}$, i.e., the probability for p to be real feasible path decrease exponentially with the number of unsured feasible links. Hence, for a concave metrics, to find a path with the largest probability to be a real feasible path is to minimize the number of its unsured feasible links. Theoretically, if we fail to find a feasible path consisting only of ensure feasible links and unsured feasible link, it is still possible to find a real feasible path that includes some unsured infeasible links. However, since probability for the path to be a real feasible path is relative small and false positive routing wastes network resources, we do not prefer such path in this paper.

Next, we show how to compute the path with largest probability to be a feasible path for the case of an additive metric. For an additive constraint, the probability for p to be a real feasible path is

$$p(\sum_{j=1}^{h} c^j < \rho | \sum_{j=1}^{h} C^j = \beta < \rho) = \frac{p(\sum_{j=1}^{h} c^j < \rho, \sum_{j=1}^{h} C^j = \beta < \rho)}{p(\sum_{j=1}^{h} C^j = \beta < \rho)}. \quad (7.8)$$

As mentioned before, for a class based link state update policy, an update is triggered when the current QoS metric crosses a class boundary. In addition, C^j is a fixed value (not a random variable). Hence, assume the probability density function (*pdf*) of c^j is $f^j(x)$, the

pdf of c^j under the condition of C^j is

$$\frac{f^j(x)}{\int_{B^{\alpha_j-1}}^{B^{\alpha_j}} f^j(y)dy}, B^{\alpha_j-1} \leq x \leq B^{\alpha_j}, \tag{7.9}$$

where B^{α_j-1} and B^{α_j} are the lower bound and upper bound of the current class of link j, respectively. Therefore,

$$p(\sum_{j=1}^{h} c^j < \rho | \sum_{j=1}^{h} C^j = \beta < \rho) = p(\sum_{j=1}^{h} x_j < \rho), \tag{7.10}$$

where x_j is the random variable whose probability density function is Eq. 7.9. Therefore, similar to [50], let $\xi = \sum_{j=1}^{h} x_j$, we can approximately assume ξ as a random variable with normal distribution by Central Limit Theorem (CRL). Assume the mean and variance of ξ are u and σ^2, respectively. Hence,

$$\sigma^2 = \sum_{j=1}^{h} \sigma_j^2, \tag{7.11}$$

and

$$u = \sum_{j=1}^{h} u_j \approx \sum_{j=1}^{h} C^j = \beta, \tag{7.12}$$

where u_j and σ_j are the mean and variance of x_j, respectively. The approximate probability for path p to be a real feasible path is

$$p(\sum_{j=1}^{h} x_j < \rho) \approx \int_{-\infty}^{\rho-\beta} \frac{1}{2\pi\sigma} e^{-\frac{y^2}{2\sigma^2}} dy. \tag{7.13}$$

Many works [50] have been reported in the literature to address the problem of finding the Most Probable Delay Constraint Path (MP-DCP). However, since no statistic link state information is available to the nodes, we cannot directly adopt the approach as [50]. Instead, we propose a heuristic solution. By Eq. 7.13, It can be observed that the probability for p to be real feasible path is only related to $\rho - \beta$ and σ. Note that we cannot compute

σ in this paper. Since ρ is the given constraint, we can simply maximize the probability by minimizing β.

With above analysis, we divide our proposed routing algorithm into two parts:

- In the first part, we try to find a real feasible path in $G_t(N, E)$. By Theorems 16, the path computed in the first part must be a feasible path.
- In the second part, we compute the heuristic most probable feasible path.

In the first part, since the links not satisfying the concave constraints can be pruned, our problem is converted to find a path satisfying additive constraint(s). If there is only one additive constraint, it can be easily solved by the shortest path search algorithms such as Bellman-Ford and Dijkstra algorithm. Note that finding a path subject to multiple additive constraints is NP-complete. Hence, we propose a high performance progressive routing algorithm, Iterative Multiple Additively Constrained path (IMACP) selection, which can progressively achieve 100% success ratio in finding a feasible path with fairly low average computational complexity. IMACP uses a subroutine, Iterative All Hops K-shortest Path selection (IAHKP), a solution to All Hops K-shorest Path selection (AHKP) problem. The progressive property of IMACP is very useful in practice: the number of iterations of IMACP is adaptively minimized such that its computational complexity is minimized while we guarantee to find a feasible path with 100% success ratio.

In the second part of our proposed routing algorithm, for the purpose of minimizing the impact of inaccurate link state information on the routing performance, we propose a novel solution based on a dynamic programming subroutine, DAEB to compute the heuristic most probable feasible path.

7.2.1 Iterative All Hops K-shortest Path Selection

As mentioned above, our proposed algorithm to MACP, IMACP, is based on a solution ot AHKP. For completeness, we first present the definition of AHKP.

Definition 17 *All Hops k-shortest Paths (AHKP) Selection Problem: Given a network* $G(N, E)$ *in which each link* $e(u, v)$ *is associated with an additive cost* $w(u, v)$. *For a given source node* $s \in N$ *and maximal hop count* H, $H < n$, *find, for each hop count value* h, $1 \leq h \leq H$, *and a destination node* $u \in N$, *the* k *shortest* h*-hop constrained paths from* s *to* u.

It can be observed that AHSP is a special case (s to i, d_i as the degree of node i, and $i_1, i_2, \ldots, i_{d_i}$ as its neighboring nodes. Assume there exists a virtual link $\widehat{e}(s, i)$ between the source node s and any other node i, whose cost is infinity. Similar to AHSP, we can compute the least cost h-hop paths from s to i, $p_1^h(s, i), p_2^h(s, i), \ldots, p_k^h(s, i)$, as follows:

1. $\forall i \in N$, $p_1^1(s, i) = e(s, i)$ and $p_g^1(s, i) = \widehat{e}(s, i)$, $g = 2, 3, ..., k$, (if, in reality, no link between the source s and node i exists, $p_1^1(s, i) = \widehat{e}(s, i)$).

2. $p_1^h(s, i), p_2^h(s, i), \ldots, p_k^h(s, i)$ are computed by selecting the k least cost h-hop paths from the paths $p_g^{h-1}(s, i_d) + e(i_d, i)$, $d = 1, 2, ..., d_i$, $g = 1, 2, ..., k$. If, in reality, the total number of h-hop paths from s to i is less than k, we assume that there exist virtual h-hop paths whose costs are infinity.

We can apply the above two steps to the previous example to verify its correctness. Denote $D_{i,g}^h$ as the cost of $p_g^h(s, i)$.

Proposition 17 $p_1^h(s, i), p_2^h(s, i), \ldots, p_k^h(s, i)$ *are the* h*-hop* k *shortest paths among all the* h*-hop paths from source* s *to node* i.

Proof: When $h = 1$, from the definition of the initial values of $D_{i,g}^1$ ($i \neq s$), $p_g^1(s, i)$, $g = 1, 2, ..., k$, are the one hop k-shortest paths from s to i. We assume that the proposition is correct for $h = m$. We shall prove by deduction that it is true for $h = m + 1$. Assume when $h = m + 1$, if $\exists i \neq s$ such that $p_g^{m+1}(s, i)$, $1 \leq g \leq k$, is not one of the k-shortest paths in all $(m + 1)$-hop paths from s to i ($D_{i,g}^{m+1}$ is larger than the cost of any $(m + 1)$-hop

k-shortest path from s to i). Further assume that path $\widehat{p}^{m+1}(s,i)$ is not one of $p_g^{m+1}(s,i)$, $g = 1, 2, ..., k$, and has a length smaller than that of $p_g^{m+1}(s,i)$. The predecessor node of node i in $\widehat{p}^{m+1}(s,i)$ is j, the path from s to j in $\widehat{p}^{m+1}(s,i)$ is $\widehat{p}^{m}(s,j)$ (note that $\widehat{p}^{m}(s,j)$ may not be one of the k-shortest m-hop paths from s to j, and by the earlier assumption, $p_g^m(s,j)$, $g = 1, 2, ..., k$, are the k-shortest m-hop paths from s to j), the cost of $\widehat{p}^{m+1}(s,i)$ is c, and the cost of $\widehat{p}^{m}(s,j)$ is c'. Thus,

$$c < D_{i,g}^{m+1} \tag{7.14}$$

If $\widehat{p}^{m}(s,j)$ is one of $p_g^m(s,j)$, $g = 1, 2, ..., k$, or $\exists g \in \{1, 2, ..., k\}$ such that $c' = D_{j,g}^m$, $\widehat{p}^{m+1}(s,i)$ is resulted by concatenating $\widehat{p}^{m}(s,i)$ with link $e(i,j)$, i.e., $\widehat{p}^{m+1}(s,i)$ is one of $p_g^m(s,n_d^i) + e(n_d^i,i)$, $d = 1, 2, ..., d_i, g = 1, 2, ..., k$. Since $\widehat{p}^{m+1}(s,i)$ is not one of $p_g^{m+1}(s,i)$, $g = 1, 2, ..., k$, and $p_g^{m+1}(s,i)$, $g = 1, 2, ..., k$, are the k-shortest paths of $p_g^m(s,n_d^i) + e(n_d^i,i)$, $d = 1, 2, ..., d_i$, $g = 1, 2, ..., k$, $\forall g \in \{1, 2, ..., k\}$,

$$c = D_j^m + c(i,j) \geq D_{i,g}^{m+1} \tag{7.15}$$

which contradicts Eq. 7.14, where D_j^m and $c(i,j)$are the costs of $\widehat{p}^{m}(s,j)$ and $e(i,j)$, respectively. Hence, $\widehat{p}^{m}(s,j)$ is not one of $p_g^m(s,j)$, i.e., $\forall g \in \{1, 2, ..., k\}$,

$$c' \geq D_{j,g}^m.. \tag{7.16}$$

So, $\forall g, u \in \{1, 2, ..., k\}$, the cost of $\widehat{p}^{m+1}(s,i)$ is

$$c = c' + c(i,j) \geq D_{j,g}^m + c(i,j) \geq D_{i,g}^{m+1}. \tag{7.17}$$

which contradicts Eq. 7.14. So, when $h = m + 1$, $\forall i \in \{1, 2, ..., N\}$, $p_g^{m+1}(s,i)$, $g = 1, 2, ..., k$, are the k-shortest paths in all $(m+1)$-hop paths from s to i.

Thus, for any node $i \in \{1, 2, ..., N\}$, $p_g^h(s,i)$, $g = 1, 2, ..., k$, must be the k-shortest paths in all h hop paths from s to i. ■

For the purpose of avoiding loops, we adopt a simple method: associating paths with indicators. For example, we associate a path traversing nodes s, 1, 5, and 7 with an integer array of size n, in which 1st, 5th, and 7th array elements are set to 1, and the rest to 0. Hence, we can easily find out if a node is in the path by only checking the corresponding array element's value. By this method, we can prevent loops without increasing the worst-case computational complexity.

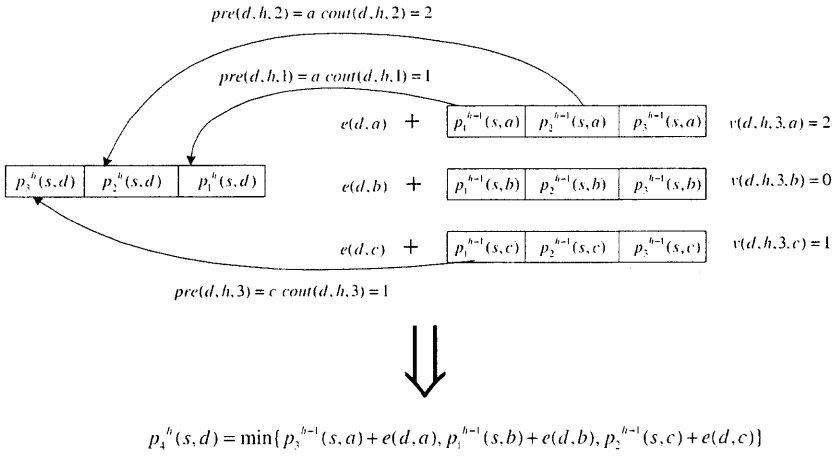

Figure 7.2 An illustration of iteratively computing the all hops k-shortest path.

Namely, IAHKP is capable of iteratively computing the all hops k-shortest path, i.e., it computes the all hops shortest paths from the source to all other nodes in the first iteration, the all hops second shortest paths in the second iteration, and so on. We first intuitively introduce how IAHKP works, i.e., how IAHKP computes the all hops kth shortest paths when the all hops $(k-1)$ shortest path have been computed. As shown in Fig. **??**, assume the three paths on any node are in the increasing order of their costs, and $p_1^h(s,d) = p_1^{h-1}(s,a) + e(d,a)$, $p_2^h(s,d) = p_2^{h-1}(s,a) + e(d,a)$, and $p_3^h(s,d) = p_1^{h-1}(s,c) + e(d,c)$. Hence, since $p_4^h(s,d)$ is selected as the 4th shortest path of $\{p_g^{h-1}(s,u) + e(d,u), u = a, b, c$ and $g = 1, 2, 3, 4\}$, and on any node, the paths are in the increasing order of their costs, $p_4^h(s,d) = \min\{p_3^{h-1}(s,a) + e(d,a), p_1^{h-1}(s,b) + e(d,b), p_2^{h-1}(s,c) + e(d,c)\}$, i.e., $p_4^h(s,d)$

is the shortest path of $p_3^{h-1}(s,a) + e(d,a)$, $p_1^{h-1}(s,b) + e(d,b)$, and $p_2^{h-1}(s,c) + e(d,c)$. Denote $pre(i,h,g)$ as the predecessor node of node i on $p_g^h(s,i)$ and $count(i,h,g)$ as the number such that

$$p_g^h(s,i) = p_{count(i,h,g)}^{h-1}(s, pre(i,h,g)) + e(i, pre(i,h,g)), \tag{7.18}$$

i.e., $p_g^h(s,i)$ is the path constructed by concatenating the $count(i,h,g)$th shortest $(h-1)$-hop path from s to $pre(i,h,g)$ and the link $e(i,pre(i,h,g))$. For example, since $p_3^h(s,d) = p_1^{h-1}(s,c) + e(d,c)$, $pre(d,h,3) = c$ and $count(d,h,3) = 1$. Denote $v(i,h,k,\alpha)$ as the maximum number among all the $count(i,h,g)$, $g = 1,2,...,k$, that satisfy $pre(i,h,g) = \alpha$ (for the neighboring node α of i satisfying that for all $g = 1,2,...,k$, $pre(i,h,g) \neq \alpha$, $v(i,h,k,\alpha) = 0$). For instance, in the previous example, $v(d,h,3,a) = 2$, $v(d,h,3,b) = 0$, and $v(d,h,3,c) = 1$. Hence, it can be observed that

$$\begin{aligned} p_4^h(s,d) \;=\; & \min\{p_{v(d,h,3,a)+1}^{h-1}(s,a) + e(d,a), p_{v(d,h,3,b)+1}^{h-1}(s,b) \\ & + e(d,b), p_{v(d,h,3,c)+1}^{h-1}(s,c) + e(d,c)\}. \end{aligned} \tag{7.19}$$

We illustrate the procedure of computing $p_4^h(s,d)$ in Fig. 7.2. Therefore, if all hops k-shortest paths have been computed, IAHKP computes the all hops $(k+1)$th shortest paths as follows:

1. $\forall i \in N, p_{k+1}^1(s,i) = \widehat{e}(s,i)$.
2. $p_{k+1}^h(s,i) = \min\{p_{v(i,h,k,\alpha)+1}^{h-1}(s,\alpha) + e(i,\alpha), \alpha = i_1, i_2, \ldots, i_{d_i}\}$.

Note that since IAHKP iteratively computes the all hops k-shortest paths, for any node and hop count h, the paths computed sequentially are in increasing order of their costs.

The relaxation procedure and pseudo codes of IAHKP are shown in Figs. 7.4 and 7.3, respectively, where $w(p)$ denotes the weight of path p and $\pi(i)$ the predecessor node of node i.

```
IAHKP algorithm (G(N,E), s, d)
1   Initialize   v(i, 2, 1, j)=0   for any pair of   i, j∈N
2   ∀i,∈N, initialize   p_g^1(s,i), g=1,2,...,k
3   for u = 2 to k
4           for h = 2 to H
5                   for all i ∈N
6                           for all e(i,j) ∈E
7                                   relax(i,j,u,h)   // relaxation procedure
8                           end for
9                   v(i,h,u,π(i)) = v(i,h,u,π(i)) + 1   //update v(i,h,u,π(i))
10                  end for
11          end for
12  end for
```

Figure 7.3 The pseudo code of IAHKP.

```
Relax(i, j, k, h)
1   if  w(p_k^{h+1}(s,i)) > w(p^h_{v(i,h,k,j)+1}(s,i)) + e(i, j), then
2           p_k^{h+1}(s,i) = p^h_{v(i,h,k,j)+1}(s, j) + e(i, j)
3           π(i) = j
4   end if
5   return
```

Figure 7.4 The relaxation procedure of IAHKP.

Computational Complexity: As mentioned above, $p_g^h(s,i)$ is computed by selecting the shortest path among the set of paths $\{p^{h-1}_{v(d,h,g,a)+1}(s,\alpha) + e(i,a),\ \alpha = i_1, i_2, \ldots, i_{d_i}\}$, whose computational complexity is $O(d_i)$. Hence, the computational complexity of computing the gth shortest h-hop paths for all nodes is $\sum_{i=1}^{N} O(d_i) = O(m)$. Since there are H hops and k shortest paths, the computational complexity of IAHKP is

$$O(kHm). \tag{7.20}$$

Memory Complexity: We can divide memory complexity into two parts: the memory used to record the paths and the memory used in the process of computing. For any given node i, hop count h, and $1 \leq g \leq k$, define

- $n_0 = i$ and $g_0 = g$.
- $n_j = pre(n_{j-1}, h-j, g_{j-1})$ and $g_j = count(n_{j-1}, h-j, g_{j-1})$, $j \leq h$.

Hence, it can be observed that $p_g^h(s,i) = (s, n_{h-1}, n_{h-2}, ..., n_1, i)$, i.e., $p_g^h(s,i)$ sequentially traverses nodes $s, n_{h-1}, n_{h-2}, ..., n_1$, and i. Therefore, all paths can be backward reconstructed as long as for any node i, hop count h, and $1 \leq g \leq k$, $pre(i,h,g)$ and $count(i,h,g)$ are available. Hence, for a single node, the memory cost to record all k shortest h-hop paths is $O(k)$. Since there are H hops and n nodes, the first part of the memory cost is $O(kHn)$. As mentioned before, we use indicator arrays (of size n) to avoid loops, which contributes to the memory cost of the second part. The total memory cost used by indicator arrays is $O(Hn^2)$ for all nodes and the all hops gth shortest paths. Since there are k shortest path, the second part of the memory cost resulting from the indicators is $O(kHn^2)$. Combining memory costs mentioned above together, the memory complexity of our proposed algorithm is $O(kHn^2)$.

7.2.2 Multiple Additively Constrained Path Selection

Without causing confusion, we simply assume all M constraints are additive in this section. MACP is defined as following.

Definition 18 *Multiple Additively Constrained Path Selection (MACP): Assume a network is modeled as a directed graph $G(N, E)$, where N is the set of all nodes and E is the set of all links. Each link connected from node u to v, denoted by $e_{u,v} = (u, v) \in E$, is associated with M additive parameters: $w_i(u, v) \geq 0$, $i = 1, 2, ..., M$. Given a set of constraints $\rho_i > 0$, $i = 1, 2, ..., M$, and a pair of nodes s and d, the objective of MACP is to find a path p from s to t subject to $w_i(p) = \sum_{e_{u,v} \in p} w_i(u, v) < \rho_i$, $i = 1, 2, ..., M$.*

Our solution, IMACP, is based on IAHKP. However, IAHKP is only adapted to the case where there is one additive cost associated with each link. Hence, we need to pick a cost function which maps multiple additive metrics of a link (path) into a single cost. Generally, there are two kinds of cost functions: link-based and path-based cost function. The link-based cost functions map multiple metrics into a cost, and the cost of a path is defined as the sum of its link costs. The path cost is not sum of its link costs for the case of path-based cost functions; it is the function of the path metrics. Specifically, given a path p and a function $f(\cdot)$, the cost of p is

$$\sum_{e(u,v) \in p} f(w_1(u, v), w_2(u, v), ..., w_M(u, v)) \tag{7.21}$$

if $f(\cdot)$ is defined as a link-based cost function. And the cost of p is

$$f(w_1(p), w_2(p), ..., w_M(p)) \tag{7.22}$$

if $f(\cdot)$ is defined as a path-based cost function. Assume p is constructed by concatenating two sub-paths p_1 and p_2, whose costs are respectively x and y, with computational complexity of $O(1)$, we can simply compute the cost of p as $x + y$ if the cost function is link-based. However, if the cost function is path-based, the computational complexity

to compute the cost of p, $f(w_1(p), w_2(p), ..., w_M(p))$, is $O(M)$. Hence, we adopt the link-based cost function for IMACP for the sake of computational complexity. With extensive simulations, we show that even with link-based cost function, IMACP still achieves 100% success ratio in finding a feasible path with fairly low average computational complexity because of IAHKP, i.e., because of the progressive properties of IMACP, IMACP can guarantee to find a feasible path with low computational complexity. We divide IMACP into two parts: forward IAHKP and backward IAHKP. We search for a feasible path from the source to the destination in the forward IAHKP with the cost function of

$$f(x_1, x_2, ..., x_M) = \frac{x_1}{\rho_1} + \frac{x_2}{\rho_2} + ... + \frac{x_M}{\rho_M}. \tag{7.23}$$

If we fail to find a feasible path in the first iteration of forward IAHKP, we reverse the search (from the destination to the source) with backward IAHKP. In IMACP, the kth iteration of forward IAHKP start after the $(k-1)$th iteration of backward IAHKP and the kth iteration of backward IAHKP follows the kth iteration of forward IAHKP until a feasible path is found or we can ensure that no feasible path exists. The cost function of backward IAHKP is different from that of forward IAHKP. Assume the least cost path, $p_{\min}$, found in forward IAHKP is not feasible, we modify Eq. 7.23 as the cost function of backward IAHKP such that

- If a feasible path, $p_{feasible}$, exists, with a new cost function $\widetilde{f}(\cdot)$, $p_{\min}$ cannot be the least cost path of the backward IAHKP.
- If $p_{\min}$ is still the least cost path in the backward IAHKP, we ensure that no feasible path exists, and IMACP is terminated.

We first introduce next theorem with which the cost function for backward IAHKP is selected.

Theorem 18 *Given a linear cost function* $f(x_1, x_2, ..., x_M) = \sum_{i=1}^{M} \psi_i x_i$, $\psi_i \geq 0$ *for all* $i = 1, 2, ..., M$, *no feasible path exists if the least cost path has a cost larger than* $f(\rho_1, \rho_2, ..., \rho_M)$.

Proof: By constradiction. Assume path p satisfies the constraints and $p_{\min}$ the least cost path. Since $p_{\min}$ is the least cost path and has a cost larger than $f(\rho_1, \rho_2, ..., \rho_M)$,

$$f(p) \geq f(p_{\min}) > f(\rho_1, \rho_2, ..., \rho_M). \tag{7.24}$$

Hence,

$$\sum_{i=1}^{M} \psi_i w_i(p) > \sum_{i=1}^{M} \psi_i \rho_i. \tag{7.25}$$

On the other hand, since p is a feasible path, $\forall i = 1, 2, ..., M$,

$$w_i(p) < \rho_i. \tag{7.26}$$

Note that $\psi_i \geq 0$ for all $i = 1, 2, ..., M$. Therefore, by Eq. 7.26,

$$\sum_{i=1}^{M} \psi_i w_i(p) < \sum_{i=1}^{M} \psi_i \rho_i, \tag{7.27}$$

which contradicts Eq. 7.25, and thus theorem is proved. ■

Lemma 19 *Given a linear cost function* $f(x_1, x_2, ..., x_M) = \sum_{i=1}^{M} \psi_i x_i$, $\psi_i \geq 0$ *for all* $i = 1, 2, ..., M$, *a feasible path must have a cost less than* $f(\rho_1, \rho_2, ..., \rho_M)$.

Proof: Assume p is a feasible path. Hence,

$$w_i(p) < \rho_i. \tag{7.28}$$

Since $\psi_i \geq 0$ *for all* $i = 1, 2, ..., M$,

$$f(p) = \sum_{i=1}^{M} \psi_i w_i(p) < \sum_{i=1}^{M} \psi_i \rho_i = f(\rho_1, \rho_2, ..., \rho_M). \tag{7.29}$$

Hence, the cost of p has a cost less than $f(\rho_1, \rho_2, ..., \rho_M)$. ■

By above theorem, if $f(p_{\min})$ is larger than $f(\rho_1, \rho_2, ..., \rho_M)$, Theorem 18 is invoked and no feasible path exists. Hence, we terminate the search. In other words, we run the backward IAHKP only when

$$f(p_{\min}) < f(\rho_1, \rho_2, ..., \rho_M). \tag{7.30}$$

If we select a function such that

$$\widetilde{f}(p_{\min}) = \widetilde{f}(\rho_1, \rho_2, ..., \rho_M), \tag{7.31}$$

it satisfies the requirements for the cost functions of the backward IAHKP because

- If there exists a feasible path $p_{feasible}$, implying that $\widetilde{f}(p_{feasible}) < \widetilde{f}(\rho_1, \rho_2, ..., \rho_M) = \widetilde{f}(p_{\min})$. Hence, $p_{\min}$ cannot be the least cost path again in the backward IAHKP.
- If $p_{\min}$ still is the least cost path of the backward IAHKP, by theorem 18, no feasible path exists because the least cost path has a cost equal to $\widetilde{f}(\rho_1, \rho_2, ..., \rho_M)$.

Since $p_{\min}$ is not a feasible path, $\exists \xi \in \{1, 2, ..., M\}$ such that $w_\xi(p_{\min}) > \rho_\xi$. Hence, the cost function we adopt for the backward IAHKP is

$$\widetilde{f}(x_1, x_2, ..., x_M) = \left(\sum_{i=1, i \neq \xi}^{M} \frac{x_i}{\rho_i} \right) + \left(\frac{f(\rho_1, \rho_2, ..., \rho_M) - f(p_{\min})}{w_\xi(p_{\min}) - \rho_\xi} \right) x_\xi. \tag{7.32}$$

It can be verified that $\widetilde{f}(p_{\min}) = \widetilde{f}(\rho_1, \rho_2, ..., \rho_M)$.

Theorem 20 *Denote $p^h_{k,forward}$ as the h-hops kth shortest paths from the source to the destination in the forward IAHKP. If for all $g < k$, $0 < h < n$, $p^h_{g,forward}$ is not a feasible path, and $f(p^h_{k,forward}) \geq f(\rho_1, \rho_2, ..., \rho_M)$, no feasible path exists.*

Proof: Given a path p, consider the case that $p \in \{p^h_{g,forward}, g = 1, 2, ..., k-1\}$. Since for all $g < k$, $0 < h < n$, $p^h_{g,forward}$ is not a feasible path, p is also not a feasible

path. Consider the otherse case that $p \notin \{p^h_{g,forward}, g = 1, 2, ..., k-1\}$.

$$f(p) \geq \min_{h=1,2,...,n} \{f(p^h_{k,forward})\} \geq f(\rho_1, \rho_2, ..., \rho_M). \tag{7.33}$$

By Lemma 19, p is not a feasible path, and thus the theorem is proved. ■

Similarly, we have following Theorem.

Theorem 21 *Denote $p^h_{k,backward}$ as the h-hops kth shortest paths from the source to the destination in the backward IAHKP. If for all $g < k$, $0 < h < n$, $p^h_{k,backward}$ is not a feasible path, and $f(p^h_{k,backward}) \geq f(\rho_1, \rho_2, ..., \rho_M)$, no feasible path exists.*

Proof: proof is similar to that of Theorem 20. ■

With above two theorems, we can ensure that no feasible path exists and terminate the search when they are invoked.

7.2.3 Finding the Most Probable Feasible Path Without Stochastic Link State Knowledge

As mentioned above, for a given concave constraint, we try to minimize the number of unsured feasible links on the computed path such that the probability for it to be real feasible path is maximized. Hence, we define a cost which represents the number of unsured feasible links in path.

Definition 19 *Given a link, which is either ensured feasible link or unsured feasible link for all concave metrics. The cost of the link is defined as the number of metrics by which the link is unsured feasible link, and the cost of a path is the sum of its link costs.*

For example, as shown in Fig. 7.5, two concave metrics, (1, 2, 3) and (1.2, 2.1, 3.5), (1, 2, 3) and (0.5, 0.8, 1.2), and (0.6, 0.8, 1) and (0.5, 0.8, 1.2), are respectively associated with links $e(a, b)$, $e(b, c)$, and $e(c, d)$. Assume the concave constraints are both 0.7. By above definition, the costs of $e(a, b)$, $e(b, c)$, and $e(c, d)$ are 0, 1, and 2,

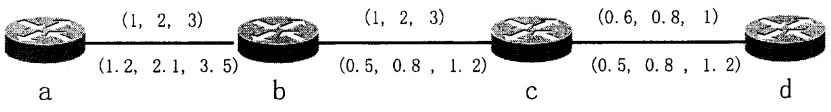

Figure 7.5 An illustration of the definition of cost.

```
Algorithm DAEB(G(V,E), s, f(·))
1   for all  v ≠ s
2            w(v, 0) = 0
3            p(v, 0) = {s}
4   end for
5   for i= 1, 2, ..., U
6            for  v ∈ V
7                    for e(u ,v)∈E
8                            p_temp(v, i)=p(u, i-c(u, v)) + {v}   // a temporary path
9                            w(v, i) = w(v, i-1)
10                           if   w(v, i) > f(p_temp(v, i))
11                                   p(v, i) = p_temp(v, i)
12                                   w(v, i) = f(p_temp(v, i))
13                           end if
14                   end for
15           end for
16  end for
```

Figure 7.6 The pseudo code of DAEB.

respectively, and the cost of path consisting of the three links is sum of link costs, 3. As a result, we convert the problem of finding the most probable concave constrained path into finding the least cost path with above definition. Combining with the additive constraints, our problem becomes finding the least cost path subject to multiple additive constraints. In [7], a efficient algorithm for the Multiple Constrained Optimal Path (MCOP) selection has been proposed. Note that the cost in our problem represents the number of unsured feasible links in a path, which are no-negative integers. Meanwhile, if there is only one additive metric, our problem becomes the Delay Constrained Least Cost path selection (DCLC), which can be polynomially solved when link costs are integers. Hence, instead of adopting the solution of [7], we propose a new efficient algorithm in this paper, which use a dynamic programming algorithm, DAEB, as a subroutine, whose pseudo code is shown in Fig. 7.6. Similar to [62], we ignore the fact that some link costs are zero for the initial description. Fig. 7.6 illustrates the pesudo code of DAEB, in which s is the source node, $f(\cdot)$ a weight function that maps the additive QoS metrics of a path into a single weight, $p(v,i)$ the least weight path from s to v with cost of i, $w(v,i)$ the weight of $p(v,i)$, and U the upper bound on the cost of a path. Since the probability for a path to be a feasible path decreases exponentially with its cost, we only select very small U and $U << n$ such that false positive is minimized. Assume the first r metrics are the all additive metrics, and they are indexed from 1 to r. In order to maximize the probability of finding a real feasible path, we adopt the weight function as following:

$$f(p) = \max\{\frac{w_1(p)}{\rho_1}, \frac{w_2(p)}{\rho_2}, ..., \frac{w_r(p)}{\rho_r}\}, \tag{7.34}$$

where $w_i(p) = \sum_{j \in p} C_i^j$, which has been heuristically shown to be a high performance cost function in [7]. As stated in [62], we can handle zero-delay links by running an invocation of Dijkstra algorithm to update weight of paths due to zero cost. Hence, the overall computational complexity of the second part of our proposed algorithm is $O(mrU + m + n\log n)$, in which $O(mrU)$ results from running the DAEB, and $O(m + n\log n)$ from the Dijkstra

algorithm. Notice that the weight of the computed paths $p(v, k)$ non-increases with its cost k. Therefore, there exits conflict between selecting the least cost path and the least weight path. We set up a strategy to pick a path from $p(d, k)$, $k = 1, 2, ..., U$. A path p is eligible for being selected only when $\forall i = 1, 2, ..., r$,

$$\sum_{j \in p} C_i^j < \rho_i. \tag{7.35}$$

If there exists a zero cost path satisfying that $\forall k = 1, 2, ..., r$,

$$\sum_{j \in p} B_i^{\alpha_i^j} < \rho_i. \tag{7.36}$$

By Theorem 1, it must be a real feasible path and can be computed by the first step of our proposed algorithm. Next, we only consider that the case that $\exists i$ such that

$$\sum_{j \in p} B_i^{\alpha_k^j} > \rho_i \Rightarrow \max\{\frac{\sum_{j \in p} B_1^{\alpha_1^j}}{\rho_1}, \frac{\sum_{j \in p} B_2^{\alpha_2^j}}{\rho_2}, ..., \frac{\sum_{j \in p} B_r^{\alpha_r^j}}{\rho_r}\} > 1. \tag{7.37}$$

For the purpose of minimizing the probability of false routing, path p can be selected only when

$$\max\{\frac{\sum_{j \in p} B_1^{\alpha_1^j}}{\rho_1}, \frac{\sum_{j \in p} B_2^{\alpha_2^j}}{\rho_2}, ..., \frac{\sum_{j \in p} B_r^{\alpha_r^j}}{\rho_r}\} < 1 + \varepsilon, \tag{7.38}$$

where ε is a selected constant. If there are multiple paths satisfying both Eqs. 7.35 and 7.38, we select the path having the least cost. If no path satisfies both Eqs. 7.35 and 7.38, we select the least cost path satisfying Eq. 7.35. Otherwise, we just assume that no feasible path exists. Note that ε should be selected according the adopted link state update policy. The more accurate the link state information (the larger number of classes), the smaller ε. For instance, given a update policy with which $\forall i$ and j,

$$\frac{B_i^{\alpha_i^j}}{C_i^j} < 1 + \widetilde{\varepsilon}. \tag{7.39}$$

Intuitively, the more accurate link state information, the smaller $\widetilde{\varepsilon}$. Therefore, for the path satisfying Eq. 7.35,

$$\begin{aligned} & \max\{\frac{\sum_{j\in p} B_1^{\alpha_1^j}}{\rho_1}, \frac{\sum_{j\in p} B_2^{\alpha_2^j}}{\rho_2}, ..., \frac{\sum_{j\in p} B_r^{\alpha_r^j+1}}{\rho_r}\} \\ < & \max\{\frac{\sum_{j\in p} B_1^{\alpha_1^j}}{\sum_{j\in p} C_1^j}, \frac{\sum_{j\in p} B_2^{\alpha_2^j}}{\sum_{j\in p} C_2^j}, ..., \frac{\sum_{j\in p} B_r^{\alpha_r^j}}{\sum_{j\in p} C_r^j}\} < 1+\widetilde{\varepsilon}. \end{aligned} \tag{7.40}$$

Hence, Eq. 7.38 has an effect on the path selection only when

$$\varepsilon < \widetilde{\varepsilon}. \tag{7.41}$$

Since the path computed by our proposed routing algorithm is the heuristic most probable path, we call our algorithm as Heuristic Most Probable Routing algorithm (HMPR).

7.2.4 Remarks

Above we have proposed a routing algorithm that can be applied to the cases where the classb-based link state update policy are adopted for updating link state information. Note that we can also apply the algorithm for the cases of other trigger-based link state update policies. For example, assume threshold update policy is adopted for updating link state information. Given a link j whose current link state metric is C^j (assume one metric). By the definition of threshold update policy, an update will be triggered at the moment of t if

$$\frac{|c^j(t) - C^j|}{C^j} > \tau \Rightarrow c^j(t) = (1+\tau)C^j \text{ or } c^j(t) = (1-\tau)C^j. \tag{7.42}$$

Hence, we can simply set $B^{\alpha^j - 1} = (1-\tau)C^j$ and $B^{\alpha^j} = (1+\tau)C^j$, and our proposed routing algorithm can be applied. Specifically, for any given trigger based update policy, assume the current link state metrics of link j are C_1^j, C_2^j, ..., C_M^j, and an update will be triggered when

$$c_i^j(t) = B_i^{\alpha_i^j - 1} < C_i^j, i = 1, 2, ..., M, \tag{7.43}$$

or

$$c_i^j(t) = B_i^{\alpha_i^j} > C_i^j, i = 1, 2, ..., M. \tag{7.44}$$

Hence, it can be observed that our proposed routing algorithm can be applied.

Note that our proposed routing algorithm is a generic one for trigger-based update policies. More efficient ones may be proposed for a specific update policy. For instance, assume equal class based link state update policy is adopted, and all link metrics are partitioned into the same number of classes ($(0, B)$, $(B, 2B)$,...$((k-1)B, kB)$). Therefore, for any link j,

$$B_i^{\alpha_i^j} \in \{B, 2B, ..., kB\}, \tag{7.45}$$

$$B_i^{\alpha_i^j - 1} \in \{0, B, ..., (k-1)B\}, \tag{7.46}$$

and

$$C_i^j \in \{\frac{2\vartheta - 1}{2}B, \vartheta = 1, 2, ...\}. \tag{7.47}$$

It can be observed that we can convert all link metrics and the lower and upper bounds of classes into integers by dividing them with $\frac{B}{2}$. It has been proved that MCP can be polynomial solved if all link metrics are integers [69]. Therefore, more efficient solutions may be proposed in this case.

For the purpose of minimizing the probability of false positive, we exclude the paths with unsured infeasible links in this paper. However, this approach may increase probability of false negative. Therefore, we may extend our proposed routing algorithm such that the paths with unsured infeasible links are considered as candidate feasible paths. Moreover, instead of setting the cost of a link as the number of metrics by which the link is unsured feasible link, it may be necessary to have different treatment for different metrics. For

example, given a link j and two constraints, ρ_1 and ρ_2, the cost of link j may be set as ϕ_1 if it is a unsured feasible link for ρ_1, and ϕ_2 if it is a unsured feasible link for ρ_2. This approach may reduce the probability of false routing for the case that the updated metrics have different level of accuracy. For instance, assume the numbers of classes for two metrics, say A and B, are k_1 and k_2 ($k_1 > k_2$), respectively, and as a result, every node has more accurate link state information on A than B. Assume the costs of paths p_1 and p_2 by Definition 19 are both 1. But the unsured feasible link on path p_1 is from the perspective of A, and the one on path p_2 is from the perspective of B. Since the nodes have more accurate state information on A than B, we prefer p_1 because it has larger probability to be a real feasible path. Hence, instead of adopting the Definition 19, we present next more general cost definition.

Definition 20 *Given a link j and M concave constraints, ρ_1, ρ_2, ..., ρ_M. If $\exists \rho_i$, $i = 1, 2, ..., M$, such that the link is ensure infeasible link, its cost, $cost(j)$, is defined as infinity. Otherwise,*

$$cost(j) = \sum_{i=1}^{M} \left[\frac{\lambda_i}{2} \left(\frac{\rho_i - B_i^{\alpha_i^j - 1}}{\left| \rho_i - B_i^{\alpha_i^j - 1} \right|} + 1 \right) + \frac{\pi_i}{2} \left(\frac{\rho_i - C_i^j}{\left| \rho_i - C_i^j \right|} + 1 \right) \right], \tag{7.48}$$

where λ_i and π_i are the weights, from the perspective of ith constraint, set for the unsured feasible and unsured infeasible link, respectively.

We may even balance traffic load in the network by assigning different links with different cost functions, which is, however, beyond the scope of this paper.

7.3 Simulations

We divide our simulations into two parts, in which the performances of IMACP and HMPR are evaluated, respectively.

7.3.1 Simulation 1

In this section, our main objective is to evaluate the average computational complexity of IMACP when it achieves 100% success ratio in finding the path meeting the multiple additive constraints. Moreover, by giving a small upper bound on the number of iteration of IMACP, we show that it still achieves very satisfactory performance.

The adopted network topologies are the 32 nodes network [54] and a 50 nodes generated by using the Doar's model [63]. The number of constraints are two, which are set equal to each other in the simulation and increased from 0.5 to 5.9 with step of 0.2. Each data is gotten by running 10,000,000 requests. The source and destination nodes are uniformly random picked in the network. Each additive QoS metrics of a link is a uniformly distributed random variable from 0 to 1.

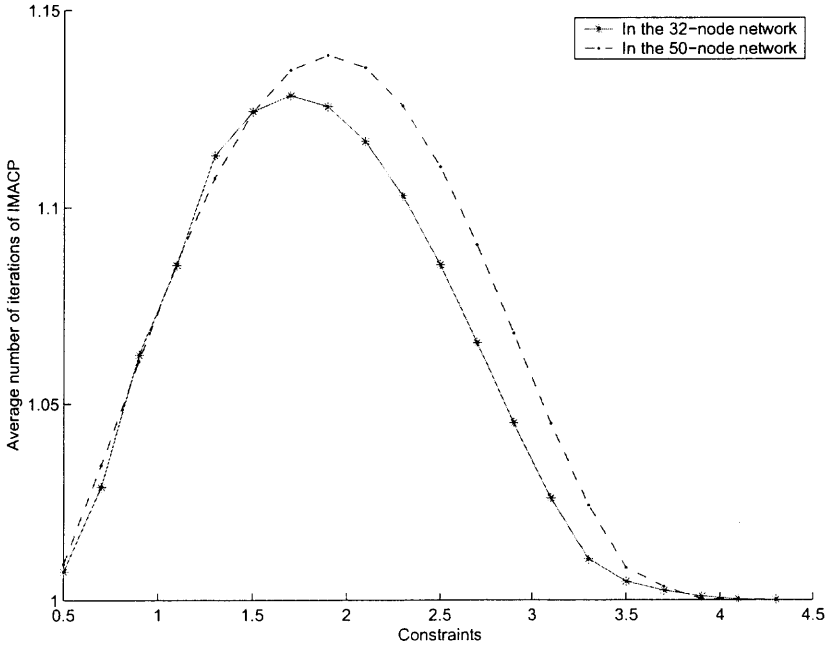

Figure 7.7 The average number of iterations of IMACP in the 32-node network and the 50-node network.

We illustrate the average computational complexity of IMACP in Fig. 7.7. It can be observed that the average computational complexity of IMACP is fairly low: with average

computational complexity of twice that of standard Bellman-Ford algorithm, IMACP achieves 100% success ratio in finding a path meeting the constraints.

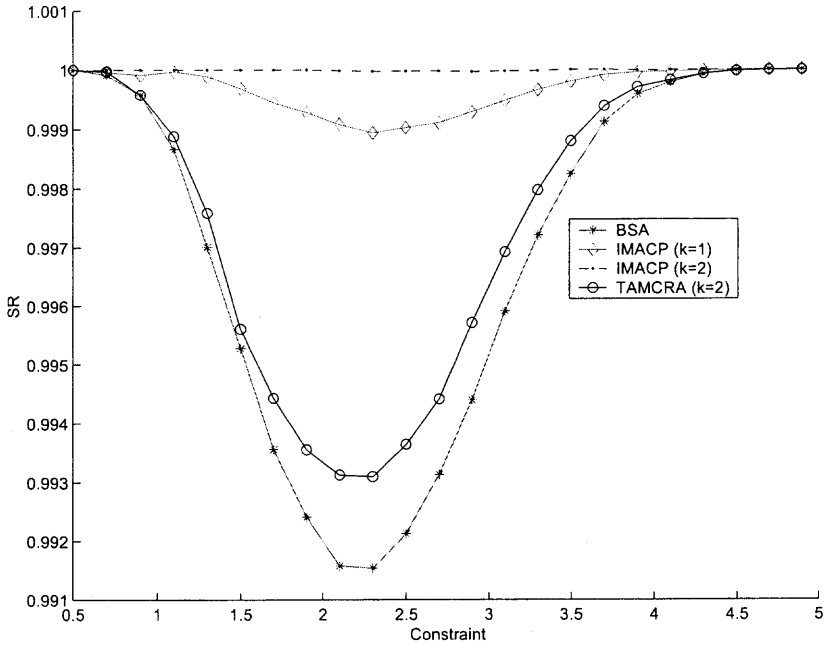

Figure 7.8 The success ratios of the algorithms in the 32-node network.

In real networks, it is necessary to set an upper bound on the worst-case computational complexity on IMACP in order to promptly response to the connection requests. Here, giving a small upper bound on the number of iterations of IMACP ($k = 1$ and 2), we evaluate its performance with the performance index, Success Ratio (SR), which is defined as below:

$$SR = \frac{\text{The number of optimal paths of the algorithm}}{\text{The number of optimal paths of the optimal algorithm}}. \tag{7.49}$$

The algorithm that can always locate the optimal feasible path as long as a feasible path exists is referred to as the optimal algorithm. Here, it is achieved simply by flooding which is very exhaustive. In addition, we also show SRs of the state-of-the-art algorithms, TAMCRA [53] and BSA [55], for the comparison purpose. By Figs. 7.8 and 7.9, IMACP

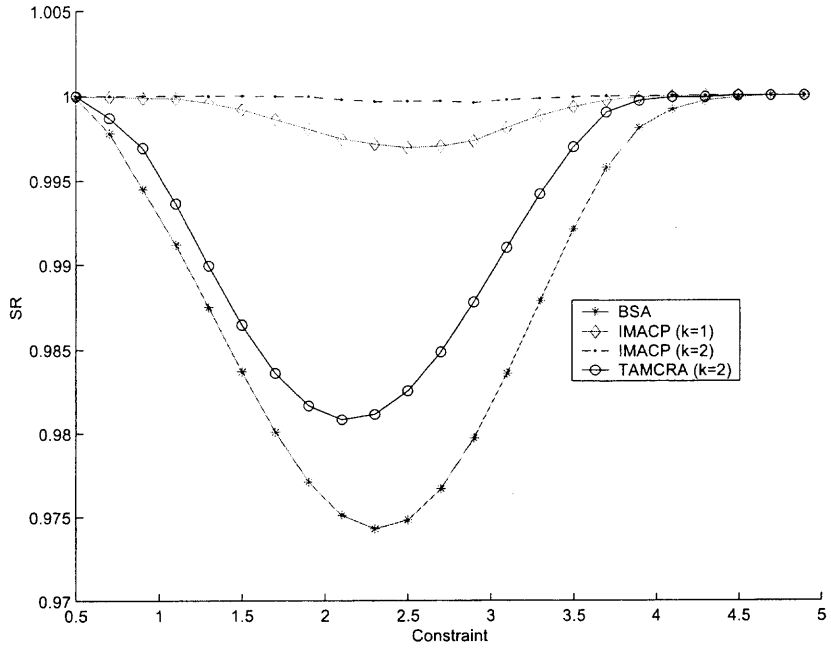

Figure 7.9 The success ratios of the algorithms in the 50-node network.

outperforms TAMCAR ($k = 2$) and BSA ($B = 1000$) in terms of the success ratio. Actually, due to the progressive property of IMACP, providing the average computational complexity of IMACP, we can provide a lower bound on its success ratio for the case that a upper bound on the number of iterations is given. Denote $SR(k)$ as the success ratio of IMACP when the number of iterations is upper bounded by k ($SR(0) = 0$), and ave as the average iteration of IMACP when it can guarantee 100% success ratio. Denote

$$s(k) = SR(k) - SR(k-1). \tag{7.50}$$

Hence, we have

$$ave = \sum_{i=1}^{\infty} is(i) > \sum_{i=1}^{k} s(i) + (k+1) \sum_{i=k+1}^{\infty} s(i). \tag{7.51}$$

Note that

$$\sum_{i=1}^{\infty} s(i) = 1. \tag{7.52}$$

Hence,

$$ave > SR(k) + (k+1)(1 - SR(k)) \Rightarrow SR(k) > 1 - \frac{ave - 1}{k}. \tag{7.53}$$

We can verify this lower bound with the simulation results we provided in Figs. 7.7, 7.8, and 7.9.

7.4 Simulation 2

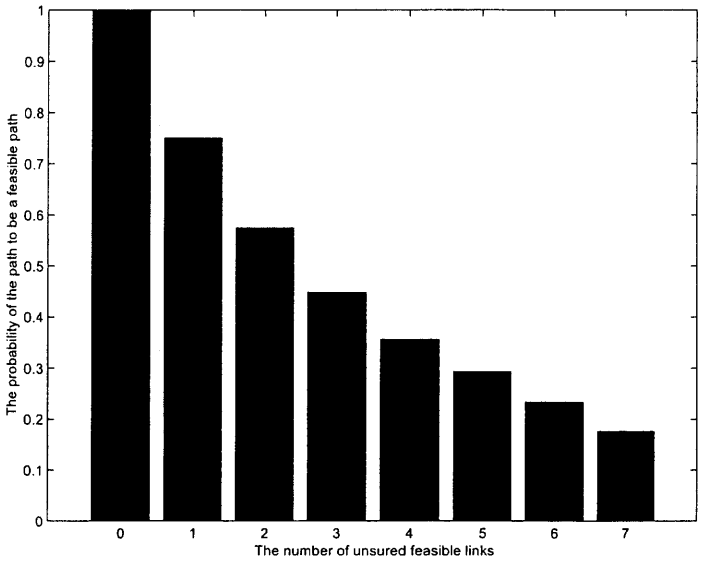

Figure 7.10 The probability for a path to satisfy a concave constraint decrease exponentially with the number of unsured feasible links on it.

In the previous section, we have shown that IMACP can achieve 100% success ratio in finding the path meeting the additive constraints with fairly low average computational complexity. Since the worst case computational complexity of DAEB is $O(mrU + m +$

$n \log n$), the average computational complexity of our proposed routing algorithm, HMPR, is also fairly low. In all the simulation of this section, we assume the link state update policy is the equal-class based update policy, and all link metrics are partitioned into the same number of classes ($k_1 = k_2 = ... = k_M = nc$). Before evaluating the performance of HMPR, we show that the generic QoS routing algorithm without considering the staleness of link state information may introduce significant portion of false routing. We adopt only two constraints in this simulation: bandwidth and delay. Similar to most source routing algorithms, in which the links do not have enough bandwidth are pruned, we simply prune all the unsured infeasible and ensured infeasible links, and treat the other links as the ones satisfying the concave constraints. Specifically, assume the first QoS metric is bandwidth, link j is pruned from the network if $C_1^j < \rho_1$. Then, we run the shortest path search algorithm, Dijkstra algorithm, to find the least delay path. For convenience, we call this algorithm as Simple Dijkstra Algorithm (SDA) for the rest of section. The network topology is the 32-node network. We assume the traffic load of the network is 0.8 and the available bandwidth is uniformly distributed from 0 to 0.4 (average available bandwidth is 0.2). Note that by the definition of equal-class based link state update policy, for any $i = 1, 2$ and j,

$$C_i^j = \left(\left\lceil \frac{c_i^j}{nc} \right\rceil + 0.5 \right) nc, \tag{7.54}$$

implying that from the perspective of a network node, the bandwidth and delay of link j are $\left(\left\lceil \frac{c_1^j}{nc} \right\rceil + 0.5 \right) nc$ and $\left(\left\lceil \frac{c_2^j}{nc} \right\rceil + 0.5 \right) nc$, respectively. In the simulation, the delay constraint is increased from 0.5 to 5.5 by the step of 0.2, and $nc = 10$.

We first demonstrate the relationship between the probability for a path to satisfy a concave constraint and the number of its unsured feasible links in Fig. 7.10. The probability roughly exponentially decreases with the number of unsured feasible links, which is consistent with our analysis. Observe that when the number of unsured feasible links is larger than 3, the probability for a path to satisfy the bandwidth constraint is less

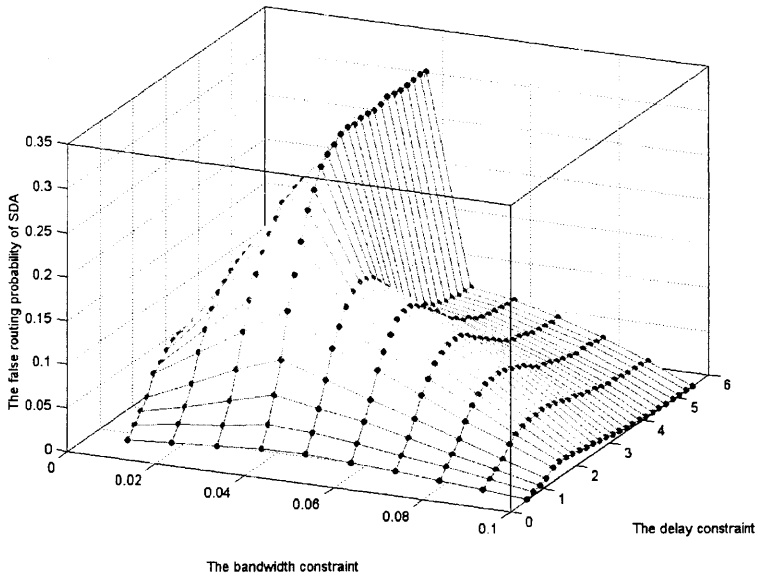

Figure 7.11 The false routing probabilities of SDA.

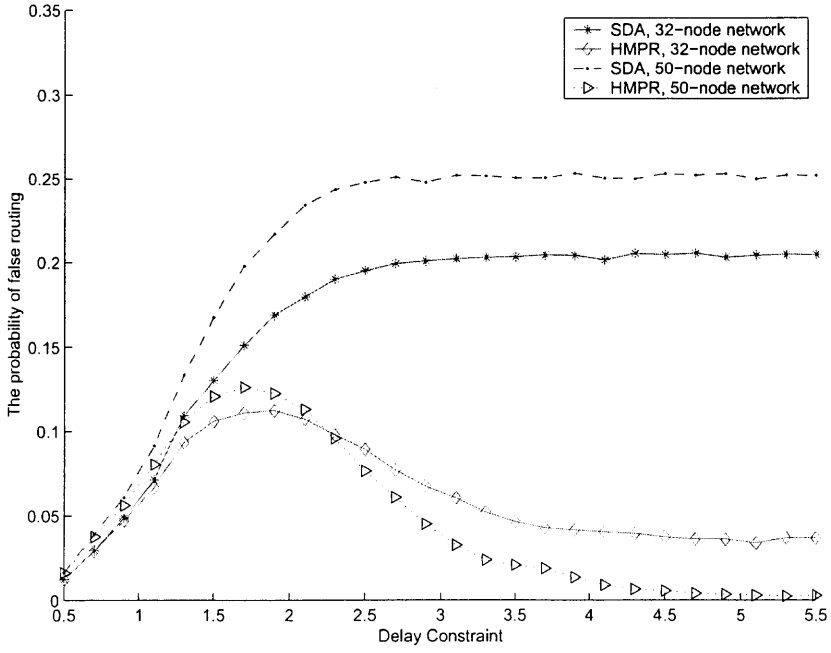

Figure 7.12 The false routing probabilities of SDA and HMPR.

than 0.5. Hence, we set $U = 3$ in our simulation. For the purpose of minizing the false routing probability, we adopt ε such that $\forall i$ and j

$$\varepsilon = \min\{\frac{B_i^{\alpha_i^j}}{C_i^j}\} - 1 = \frac{2}{(1 + \frac{nc-1}{nc})} - 1 = \frac{1}{2nc-1}. \tag{7.55}$$

In Fig. 7.11, we illustrate the variation of fasle routing probability of SDA with dealy and bandwidth constraints. It can be observed that the impact of the state link state information on the performance of routing algorithm is very serious, i.e., without consider the staleness of link state information, SDA introduces significant percentage of false routing, which can be upto 35% in the 32-node network. It should be noted that when the bandwidth constraint ρ_1 is larger than 0.05, the fals routing probability becomes very small in our simulation. The bandwidth are partioned into 10 classes in the simulation ($nc = 10$), and by the defintion of equal-class based update policy, the updated values are $\frac{2\vartheta-1}{2}B$, $\vartheta = 1, 2, ...$ Hence, for any link j, either

$$C_1^j = 0.05 \Rightarrow C_1^j < \rho_1 \text{ and } B_1^{\alpha^j} = 0.1 > \rho_1, \tag{7.56}$$

, implying that link j is an unsured infeasible link, or

$$C_1^j \geq 0.15 \Rightarrow B_1^{\alpha^j - 1} > \rho_1, \tag{7.57}$$

implying that link j is an ensured feasible link, i.e., when the bandwidth constraint is between 0.05 and 0.1, there is no unsured feasible link. As the result, the probability of false routing become small. For the purpose of evaluating the effect of our propsed routing algorithm on minimizing the false routing probability, we thus only focus on the case that bandwidth constraint is less than 0.05, and assume it is uniformly distributed from 0 to 0.05. The simulation results is illustrated in Fig. 7.12, in which the performance of SDA and HMPR is compared. By Fig. 7.12, we can find that HMPR has much lower false routing probability than SDA, i..e, the impact of the stale link state information is minimized by HMPR. The probability of false routing of SDA increases with the network size.

Meanwhile, HMPR is more scalable than SDA in terms of the false routing probability, i.e., the false routing probability of HMPR has only a small variation with the network size. Note that the false routing probability can be reduced by increasing the accuracy of link state information. In our simulation, it can be achieved by increasing the number of classes (nc). However, this approach increases the protocol overhand and the burden on the network resource, and thus is not preferable. On ther other hand, since we can achieve much lower false routing probability with HMPR, given a upper bound on the false routing probability, we can save network resource from distributing the link state information by deploying HMPR. Note that in our simulations, HMPR has relative higher probability of false routing in the 32-node network than in the 50-node network when the delay constraint is larger than 2.5. This result comes from the fact that the 32-node network is a sparse network and it is divided into two networks when all links expcet insured feasible links are removed (in this case, the step 1 of HMPR does not work), while the 50-node network is more dense than the 30-node network.

7.5 Conclusions

In this section, we introduce and investigate the issume of minimizing the impact of stale link state information on the performance of routing algorithm without any link state stochastic knowledge, and have proposed a Heuristic Most Probable Routing (HMPR) algorithm under the assumption that trigger-based link state update policies are adopted. With extensive simulations, we show that HMPR can effectively minize the effect of the staleness of link state information. As a result, the false routing probability is greatly reduced. In addition, we have proposed a high performance routing algorithm, IMACP, to locate the path meeting multiple additive constraints. IMACP is based on a progressive solution to AHKP. We show that without setting an upper bound on the number of iterations of IMACP, its average computational complexity is still fairly low. The most distinguished property of IMACP is its progressive property, which is very useful in practical: it can

adaptively minimize its computational complexity without sacrificing its performance. Furthermore, a dynamic programming algorithm, DAEB, has been presented to solve the special case of LCMACP that the link costs are no-negative integers.

CHAPTER 8

FINDING A LEAST HOP(S) PATH SUBJECT TO MULTIPLE ADDITIVE CONSTRAINTS

8.1 Introduction

In this chapter, for the purpose of saving network resources, we introduce and investigate a new problem referred to as the least hop(s) multiple additively constrained path (LHMACP) selection, which is NP-complete. We propose the Fast All Hop(s) k-shortest Path (FAHKP) algorithm, which is an efficient solution to AHKP. Through extensive analysis and simulations, we show that our proposed heuristic algorithm, based on FAHKP, is highly effective in finding a least hop path subject to multiple additive constraints with very low computational complexity; it achieves near 100% success ratio in finding a feasible path while minimizing its average hop count.

8.2 Problem Formulation

As discussed before, we only consider additive constraints. Without loss of generality, the problem is formulated as follows:

Definition 21 *Least Hop(s)Multiple Additively Constrained Path Selection (LHMACP): Assume a network is modeled as a directed graph $G(N, E)$, where N is the set of all nodes and E is the set of all links. Each link connected from node u to v, denoted by $e_{u,v} = (u, v) \in E$, is associated with M additive parameters: $w_i(u, v) \geq 0$, $i = 1, 2, ..., M$. Given a set of constraints $\rho_i > 0$, $i = 1, 2, ..., M$, and a pair of nodes s and d, the objective of LHMACP is to find the least hop(s) path p from s to t subject to $w_i(p) = \sum_{e_{u,v} \in p} w_i(u, v) < \rho_i$, $i = 1, 2, ..., M$.*

Definition 22 *Any path p from s to t that meets the requirement,* $W_i(p) = \sum_{e_{u,v} \in p} w_i(u,v) < c_i$, *for all* $i = 1, 2, ..., M$, *is a feasible path.*

We denote $p_1 + p_2$ as the concatenation of two paths p_1 and p_2, and $c(p)$ as the cost of path p. Note that, given two paths, p_1 and p_2, and their costs, if the cost of a path p is defined as $c(p) = f(W_1(p), W_2(p), ..., W_M(p))$, where $f()$ is a cost function, the computational complexity of computing the cost of $p_1 + p_2$,

$$c(p_1 + p_2) = f(W_1(p_1) + W_1(p_2), W_2(p_1) + W_2(p_2), ..., W_M(p_1) + W_M(p_2)), \quad (8.1)$$

is $O(M)$, while it is $O(1)$ ($c(p_1 + p_2) = c(p_1) + c(p_2)$) if the cost of a path is defined as the sum of its link costs. Hence, we adopt the latter definition of the cost of a path for the sake of the computational complexity. The least cost path is also referred to as the shortest path in this chapter.

8.3 Proposed Algorithm

Similar to HMPR, we propsed our algorithm of LHMACP based on a novel solution, Bidirectional k-shortest Extended Bellman-Ford (BFAHKP) algorithm, to AHKP. As mentioned in the previous chapter, the least cost h-hop paths from s to i, $p_1^h(s,i)$, $p_2^h(s,i)$, ... , $p_k^h(s,i)$, can be computed as follows:

1. $\forall i \in N$, $p_1^1(s,i) = e(s,i)$ and $p_g^1(s,i) = \widehat{e}(s,i)$, $g = 2, 3, ..., k$, (if, in reality, no link between the source s and node i exists, $p_1^1(s,i) = \widehat{e}(s,i)$).

2. $p_1^h(s,i)$, $p_2^h(s,i)$, ... , $p_k^h(s,i)$ are computed by selecting the k least weight h-hop paths from the paths $p_g^{h-1}(s,i_d) + e(i_d,i), d = 1, 2, ..., d_i$, $g = 1, 2, ..., k$.

In order to reduce the computational complexity, we introduce an algorithm Fast Algorithm (FA), to select $p_1^h(s,i)$, $p_2^h(s,i)$, ... , $p_k^h(s,i)$ from the paths $p_g^{h-1}(s,n_d^i) + e(n_d^i,i)$, $1 \leq d \leq d_i, 1 \leq g \leq k$. We first illustrate how FA works by a simple example.

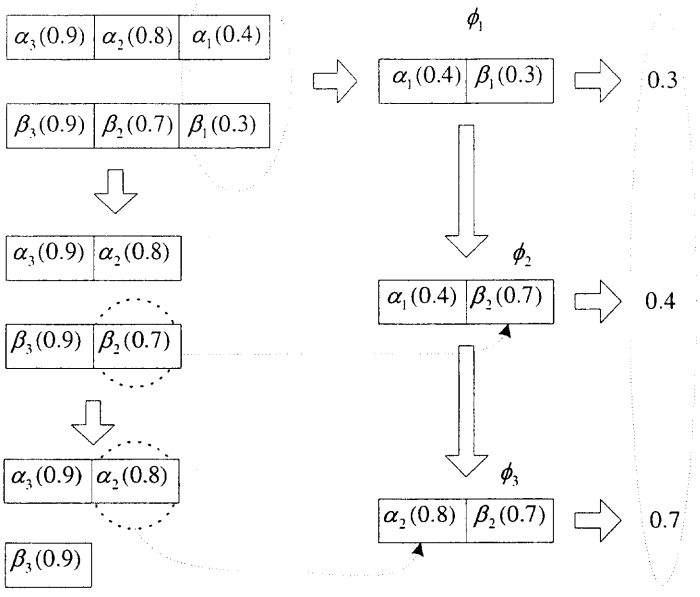

Figure 8.1 The demonstration of the FA algorithm.

As shown in Fig. 8.1, there are two sorted path sets ($k = 3$), $\{\alpha_1, \alpha_2, \alpha_3\}$ and $\{\beta_1, \beta_2, \beta_3\}$, whose costs are $\{0.4, 0.8, 0.9\}$and $\{0.3, 0.7, 0.9\}$, respectively. Note that

$$\min\{\min_{j=1,2,3}\{c(\alpha_j)\}, \min_{j=1,2,3}\{c(\beta_j)\}\} = \min\{c(\alpha_1), c(\beta_1)\}. \tag{8.2}$$

Let

$$\boldsymbol{\phi_1} = \{\alpha_1, \beta_1\}; \tag{8.3}$$

the least cost path in the two sets is the least cost path in $\boldsymbol{\phi_1}$. In this example, it is β_1. Furthermore, since the two path sets are sorted by their costs and β_1 is the least cost path, the second least cost path in the two sets must be the least cost path between α_1 and β_2, i.e., let $\boldsymbol{\phi_2} = (\boldsymbol{\phi_1} \cap \overline{\{\beta_1\}}) \cup \{\beta_2\}$, and the second least cost path in the two sets is the least cost path in $\boldsymbol{\phi_2}$. Similarly, the jth least cost path in the two sets is the least cost path in $\boldsymbol{\phi_j}$, which can be proved by deduction, where $\boldsymbol{\phi_j} = (\boldsymbol{\phi_{j-1}} \cap \overline{\{\pi_{j-1}\}}) \cup \{\nu_j\}$, π_{j-1} is the least cost path in $\boldsymbol{\phi_{j-1}}$, and ν_j is the next path to π_{j-1} in the corresponding set. Moreover,

$1 \leq j < v \leq k$, $c(\pi_j) \leq c(\pi_v)$, i.e., the paths (π_j) are sequentially computed in increasing order of their costs. As shown in Fig. 8.1, the output paths sequentially computed by FA are β_1, α_1, and β_2 with the corresponding costs of 0.3, 0.4, and 0.7, respectively, which are in increasing order.

Denote P_i^h as the set $\{p_g^h(s,i), 1 \leq g \leq k\}$. The following FA procedure is used to compute the k least cost $(h+1)$-hop paths from the source node s to node i:

Step 1 Set $\mathbf{P}_i^{h+1} = \Phi$ and $\mathbf{P} = \{p_1^h(s,n_d^i) + e(n_d^i, i), d = 1, 2, ..., d_i\}$, where Φ is the empty set. Sort the paths in $\mathbf{P}$ by their costs by the Heapsort Algorithm [40]. Since the number of paths in $\mathbf{P}$ is d_i, the computational complexity of this step is $O(d_i \log d_i + d_i)$ (the computational complexity of using the Heapsort algorithm is $O(d_i \log d_i)$).

Step 2 Denote u as the number of paths in $\mathbf{P}_i^{h+1}$. Initialize $u = 1$.

Step 3 Assume the least cost path of $\mathbf{P}$ is $p_u = p_v^h(s,n_d^i) + e(n_d^i, i)$, for some $d \in \{1, 2, ..., d_i\}$ and $v \in \{1, 2, ..., k\}$. Remove p_u from $\mathbf{P}$. Check if node $i$is included in $p_v^h(s,n_d^i)$ with its associated indicator array. If so, go to Step 5. Otherwise, place it at the tail of $\mathbf{P}_i^{h+1}$, i.e., $p_u^{h+1}(s,i) = p_u$. Copy the indicator array associated with $p_v^h(s,n_d^i)$ to that of $p_u^{h+1}(s,i)$ and set the value of its ith element by 1.

Step 4 If u is equal to k, the k least cost $(h+1)$-hop paths from the source node s to node i are obtained; Stop. Otherwise, $u = u + 1$.

Step 5 Insert $p_{v+1}^h(s,n_d^i) + e(n_d^i, i)$ into P by the binary search [40]. Since there are currently $d_i - 1$ elements in $\mathbf{P}$, the computational complexity of this step is $O(\log(d_i - 1) + 1)$. Then, go to Step 3.

$\mathbf{P}$ is sorted in Step 1 in order to reduce the computational complexity of Step 5. Otherwise, the computational complexity of Step 5 is $O(d_i)$, which is resulted from finding the least cost path in $\mathbf{P}$. Using the FA procedure, our proposed algorithm, named Fast All

```
Algorithm FAHKP(G, s, t)
1       for all i from 1 to N
2               set p_1^1(i) = c(s,i)
3       end for
4       for h = 1 to H
5               for all i from 1 to N
6                       run FA algorithm with input of i and h (FA(i,h))
7               end for
8       end for
```

Figure 8.2 The pseudo-code of the FAHKP algorithm.

Hops k-shortest Path (FAHKP) algorithm, can be illustrated by the pseudo code shown in Fig. 8.2.

Computational Complexity: Note that Step 5 is executed $k-1$ times in FA. Hence, the computational complexity of using FA to compute the k shortest $(h+1)$-hop $(1 \leq h \leq H-1)$ paths from the source node s to node i is $O(d_i \log d_i + (k-1)\log(d_i-1) + k - 1 + d_i)$ $= O(d_i \log d_i + (k-1)\log(d_i-1))$. Accordingly, the computational complexity of using FA to compute the k shortest $(h+1)$-hop paths from the source node s to all other nodes is the sum of those of computing the k shortest $(h+1)$-hop paths from the source node s to every single node, which is $O(\sum_{i=1}^{n}(d_i \log d_i + (k-1)\log(d_i-1))$. Moreover, it can be observed from the pseudo code of our proposed algorithm that there are H loops to compute all hops k shortest paths from the source node s to all other nodes, i.e., the computational complexity is

$$\begin{aligned} & O(H(\sum_{i=1}^{n}(d_i \log d_i + (k-1)\log(d_i-1)))) \\ \leq\ & O(n(\sum_{i=1}^{n}(d_i \log d_i + (k-1)\log(d_i-1)))). \end{aligned} \tag{8.4}$$

If there exists a bound D on the maximum node degree, i.e., $\forall i \in N$, $d_i \leq D$, the worst-case computational complexity of FAHKP is bounded by

$$\begin{aligned} & O(n(\sum_{i=1}^{n}(d_i \log d_i + (k-1)\log(d_i - 1)))) \\ \leq\ & O(nm \log D + nk \log D) \leq O(nm \log n + nk \log n). \end{aligned} \tag{8.5}$$

Note that this computational complexity bound is very loose, but it is rather low already and almost does not increase with k when $k < m$; it could be much less in reality.

Memory Complexity: The memory complexity of our proposed algorithm can be divided into two parts: the memory used to record the computed paths (for the purpose of reconstructing them after computing) and the one consumed during the computing procedure (FA). Denote $pre(i, h, g)$ as the predecessor node of i on $p_g^h(s, i)$, and $count(i, h, g)$ as the number satisfying that

$$p_g^h(s, i) = p_{count(i,h,g)}^{h-1}(s, pre(i, h, g)) + e(i, pre(i, h, g)), \tag{8.6}$$

i.e., $count(i, h, g)$ is the number such that $p_g^h(s, i)$ is constructed by concatenating the $count(i, h, g)$th shortest $(h-1)$-hop path from s to $pre(i, h, g)$ and the link $e(i, pre(i, h, g))$. Hence, for any given node i, hop count h, and $1 \leq g \leq k$, define

- $n_0 = i$ and $g_0 = g$.
- $n_j = pre(n_{j-1}, h - j + 1, g_{j-1})$ and $g_j = count(n_{j-1}, h - j + 1, g_{j-1})$, $j \leq h$.

It can be observed that $p_g^h(s, i) = (s, n_{h-1}, n_{h-2}, ..., n_1, i)$, i.e., all paths can be backward reconstructed as long as for any node i, hop count h, and $1 \leq g \leq k$, $pre(i, h, g)$ and $count(i, h, g)$ are available, where $(s, n_{h-1}, n_{h-2}, ..., n_1, i)$ represents a path sequentially consisting of nodes $s, n_{h-1}, n_{h-2}, ..., n_1, i$. Hence, for a single node, the memory cost to record all k shortest h-hop paths is $O(k)$. Since there are H hops and n nodes, the first part of the memory cost is $O(kHn) \leq O(kn^2)$. As mentioned before,

we use indicator arrays (of size n) to avoid loops, which contribute to the memory cost of the second part. The total memory cost used for indicator arrays is $O(kn^2)$ for all n nodes and the k shortest h-hop paths. Observe that the indicators for the h-hop paths are only used when we compute the $(h+1)$-hop paths. We can erase the indicators associated with the h-hop paths when all the $(h+1)$-hop paths are computed. Hence, the part of the memory cost resulting from the indicators is $O(kn^2)$. Combining memory costs mentioned above together, the memory complexity of our proposed algorithm as $O(kn^2)$. It can be observed that the introduction of the indicators to avoid loops does not increase the worst-case memory complexity of our proposed algorithm.

Similar to BEB, our proposed algorithm for LHMACP consists of two parts: forward FAHKP and backward FAHKP. The cost functions used in two parts are different. We adopt the same cost functions here as BEB, i.e.,

- We search for a feasible path from the source to the destination in the forward FAHKP with the cost function of

$$f(x_1,\ x_2,\ ...,\ x_M) = \frac{x_1}{c_1} + \frac{x_2}{c_2} + ... + \frac{x_M}{c_M}. \tag{8.7}$$

- Assume no feasible path is found with forward FAHKP and the least cost path is $p_{\min}$, the cost function $\widetilde{f}(\cdot)$ adopted in the backward FAHKP is

$$\begin{aligned} &\widetilde{f}(x_1,\ x_2,\ ...,\ x_M) \\ &= \left(\sum_{i=1, i\neq\xi}^{M} \frac{x_i}{\rho_i}\right) + \left(\frac{f(\rho_1, \rho_2, ..., \rho_M) - f(p_{\min})}{w_\xi(p_{\min}) - \rho_\xi}\right) x_\xi. \end{aligned} \tag{8.8}$$

Fig. 8.3 shows the pseudo-code of our proposed QoS routing algorithm, referred to as BFAHKP (Bi-directional FAHKP). The key properties that distinguish BFAHKP from previously proposed algorithms are:

- Intuitively, the more links (hops) on a path, the more network resources are consumed. Hence, minimizing the length or hops of a feasible path is preferred.

```
Algorithm BFAHKP(G, s, t, c)
1       if FAHKP(G, s, t, c, LeastCost) = SUCCESS
2               return SUCCESS
3       else
4               if LeastCost > f(c)
5                       return no feasible path exists
6               else
7                       compute the new cost function f̄(•)
8                       if FAHKP(G, t, s, c, LeastCost) = SUCCESS
9                               return SUCCESS
10                      else
11                              if LeastCost > f̄(c)
12                                      return no feasible path exists
13                              end if
14                      end if
15              end if
16      end if
17      return FAIL /*fail to find a feasible path */
```

Figure 8.3 The pseudo-code of the BFAHKP algorithm.

Based on BFAHKP, our algorithm can essentially minimize the hops of the feasible path.

- Assume the link weights are randomly distributed, and define $P_r\{W_1(p) \leq c_1, W_2(p) \leq c_2, ..., W_M(p) \leq c_M | c(p) = \alpha, H(p) = n\}$ as the probability that a path p is a feasible path with $c(p) = \alpha$, and its hop count, $H(p) = n$. The probability of the shortest path to be a feasible path may not be the largest in all possible paths. Note that, instead of computing only the shortest path, BFAHKP finds all hops k shortest paths from a source to a destination that increases the probability of finding a feasible path.

In order to reduce the runtime, we stop the search whenever a feasible path is found.

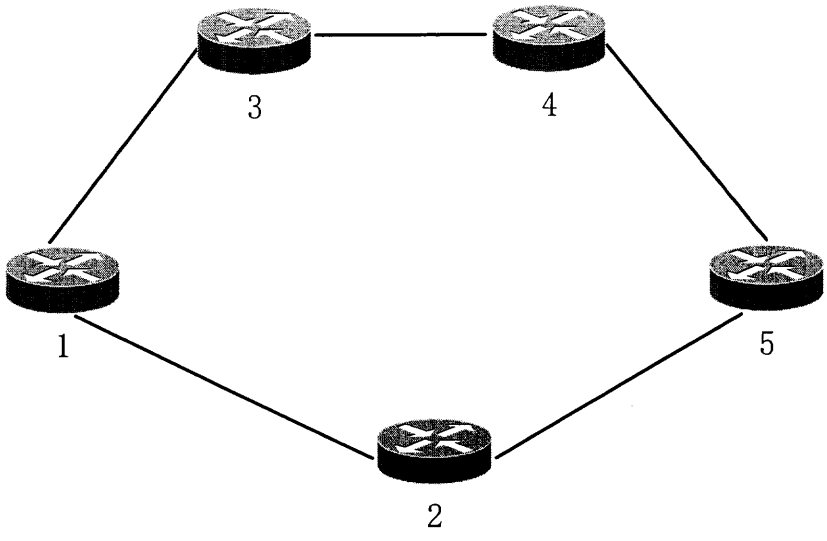

Figure 8.4 A network consisting of 5 nodes.

8.4 Simulations

We evaluate the performance of our proposed routing algorithm by comparing it with the Binary Search Algorithm (BSA) [55], H_MCOP [7],and TAMCRA [53]. Note that H_MCOP was originally designed to solve the multiple constrained optimal path selection problem. It can also be used to solve the LHMACP problem by setting the cost of each link to 1. On the other hand, it should be noted that BFAHKP and IMACP are both based on the solutions to AHKP and use same cost functions. They have the same success ratio in finding the path subject to multiple additive constraints when the same number of shortest paths (k) are chosen for them. Hence, in the chapter, we only evaluate the effectiveness of BFAHKP in minimizing the number of hops of feasible paths (people can refer to the previous chapter for the ability of BFAHKP in finding the feasible paths). We do not simply adopt the average hop of computed feasible paths as one performance index because it may introduce unfairness in comparison. For example, given a network shown as Fig. 8.4 and a set of constraints, we conduct two searches (from node 1 to node 5) with two routing algorithms, α and β (the link QoS metrics are different in the two searches). In the first

search, both algorithms locate a 2-hop feasible path (1-2-5), while in the second search, the algorithm α fails to find a feasible path, but algorithm β does (path 1-3-4-5). Obviously, algorithm β performs better than α in the simulation. However, if the average hop of the computed feasible path is adopted as the only performance index (the average hop of the feasible path computed by algorithm α in the two searches is 2, while it is 2.5 for algorithm β), it turns out that algorithm α outperforms β. Note that the optimal algorithm is achieved by hop-by-hop flooding, and it can always locate the least hop feasible path as long as a feasible path exists. Therefore, its average hop is the lower bound of all feasible paths. Furthermore, given any feasible path p, there must exist a corresponding optimal one (the least hop feasible path, which could be p) that has the same source and destination as p. Hence, instead of using the average hop of computed feasible paths as a performance index, we adopt the Average Hop Ratio (AHR), where AHR is defined as the ratio between the average number of hops of the computed feasible paths and that of the corresponding optimal ones, i.e.,

$$\text{AHR} = \frac{\text{Average hop of the computed paths of the algorithm}}{\text{Average hop of corresponding optimal paths}}. \tag{8.9}$$

We adopt the same 100-node network topologie as the one in Chapter 5. in our simulations, we set $k = 1, 2$ because BFAHKP already achieves near optimal performance when for $k = 2$. In all simulations, the link weights are independent and uniformly distributed from 0 to 1, and all data are obtained by running 100,000 requests. We adopt two constraints in the simulations, and set them equal to each other. The constraints are increased from 0.5 to 6 with a step size of 0.2.

Fig. 8.5 demonstrates the AHRs of different algorithms in the 100-node network. It can be observed that our proposed algorithm, BFAHKP, achieves near optimal average hop, i.e., our algorithm can minimize the hops of the feasible path found. Note that although H_MCOP achieves relatively low AHR (compared to TAMCRA and BSA), its success ratio in finding a feasible path is not satisfactory.

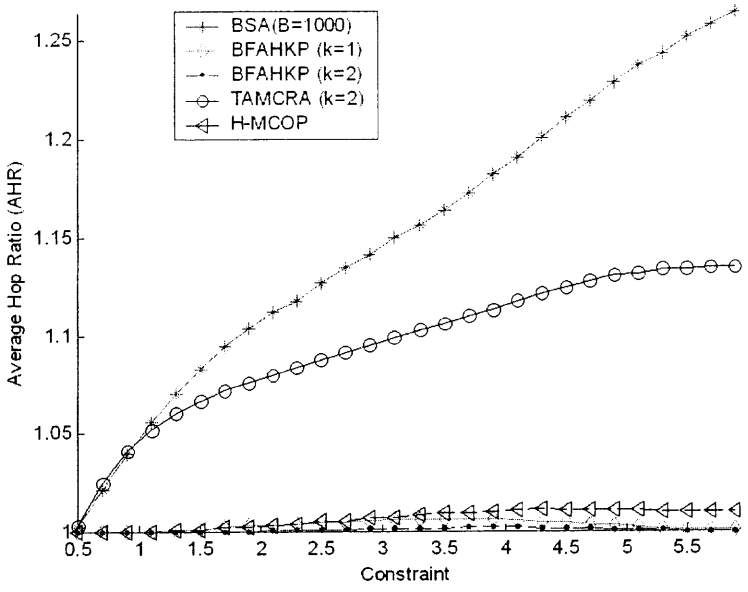

Figure 8.5 AHRs of algorithms in the 100-node network.

8.5 Conclusions

We have proposed an efficient algorithm (BFAHKP), which can achieve a very high success ratio in finding a feasible path for the least hop(s) multiple additively constrained routing. Extensive simulations show that BFAHKP is a high performance routing algorithm in terms of both the success ratio in finding a feasible path and the average hop of solutions. With a slight modification, our algorithm can also be employed for solving many other problems, such as the DCLC problem. Moreover, the success ratio of our proposed algorithm may be further improved by, similar to [10], using a non-linear cost function, i.e., the cost of a path is the function of its weights, which, however, will increase the computational complexity.

CHAPTER 9

CONCLUSIONS AND FUTURE WORKS

9.1 Conclusions

In this dissertation, we have addressed the issue of QoS provisioning in high speed networks, especially the QoS routing perspective. The main contributions are:

1. Observing that most proposed link state update schemes are not efficient enough because they update link state information without considering the QoS requirements of connections, we have investigated the issue of when to update link state information based on rate-distortion analysis, and proposed an efficient link state update scheme. Through extensive simulations, we have shown that our proposed scheme outperforms the state-of-art ones in terms of both the average update rate and false routing probability.

2. For the purpose of reliably distributing link state information throughout the networks, we have proposed an efficent reliable link state information dissemination scheme. The basic idea behind the scheme is that instead of distributing link state information on a tree, we disseminate link state information on a Reliable Topology (RT), which guarantees that in the case of a single link failure, as long as the network is still connnected, all nodes are also connected in RT. At the same time, since the RT adopted in our dissertation is a Tree-based Reliable Topology (TRT), in which the total number of links is only twice that of nodes, the protocol overhead of our proposed scheme is fairly low.

3. Many routing algorithms reported in the literature tackle the QoS routing problems by converting it into a shorest path searching problem with a cost function which maps multiple QoS metrics associated with a link into a single cost. However, the

performance of these algorithms are not satisfactory, i.e., they suffer either high computational complexities or low success ratio in finding a feasible path. Therefore, we have introduced and investigated the issue of finding All Hops Shortest Path (AHSP), and presented a tight lower bound on the optimal comparison-based solution of AHSP, which is the same as that of the standard Bellman-Ford algorithm. Since an optimal comparison-based solution of AHSP yields more paths than the standard Bellman-Ford (BF) algorithm, while their worst-case computational complexity remains the same, the performance of routing algorithms based on BF can be improved.

4. Based on an optimal comparison based solution to AHSP, Extended Bellman-Ford algorithm (EB), we have proposed two efficient routing algorithms, BEB and DEB, respectively, for MACP and DCLC. By extensive simulations, we have shown that they outperform their contenders.

5. Many works reported in literature tackle delay constrained least cost path selection by using ε-approximation schemes and scaling techniques, i.e., by mapping link costs into integers or, at least, discrete numbers, a solution that satisfies the delay constraint and has a cost within a factor of the optimal one can be computed with pseudo polynomial computational complexity. In this paper, having observed that the computational complexities of the ε-approximation algorithms using the linear scaling technique are linearly proportional to the linear scaling factors, we have investigated the issue of finding the optimal (the smallest) linear scaling factor to reduce the computational complexities, and have proposed two algorithms, Optimal Linear Scaling Algorithm (OLSA) and Transformed Optimal Linear Scaling Algorithm (T-OLSA). We have analytically shown that both algorithms are always able to locate the optimal linear scaling factor with the computational complexities negligible to those of the ε-approximation solutions. As a result, the computational

complexities of ε-approximation solutions can be effectively reduced. We have also shown that in some special cases, with our proposed algorithms, the worst-case computational complexities of ε-approximation solutions are not pseudo-polynomial any more, i.e., they are strictly polynomial and independent of ε.

6. Although the algorithms such as the ε-approximation approaches can achieve 100% or near 100% success ratio, their worst-case computational complexities are too high to be practical (assume ε is very small in ε-approximation algorithms so that their success ratios are close to 1). The algorithms such as [55] have the advantage of having low computational complexities. However, they cannot guarantee in finding a feasible path when it exists. Moreover, their success ratios in finding a feasible path may decrease sharply with the network size. In order to increase the success ratio, many proposed algorithms deploy k-shortest paths selection solutions (the number of shortest paths are generally fixed in these algorithms), instead of the shortest path selection solutions. However, the computational complexities of the algorithms increase unnecessarily in the case in which a feasible path can be found with only one shortest path searching algorithm. Since the computational complexities of the routing algorithms based on k-shortest paths selection solutions increase with the number of shortest path, k, we can minimize their computational complexities if we can adaptively minimize k, i.e., we can always locate the (optimal) feasible path with the minimized k. Hence, we proposed routing algorithms that can iteratively compute the All Hops k-shortest paths (AHKP), i.e., the algorithms are capable of computing the all hops shortest path in the first iteration, then the all hops second shortest path in the second iteration, and so on. Based on the algorithms, we proposed a solution to achieve the objective of minimizing the number of shortest paths such that the computational complexities of routing algorithms can be minimized.

7. Intuitively, the more links (hops) on a path, the more network resources are consumed. Hence, minimizing the length or hops of a feasible path is preferred. Accordingly, we introduced and investigated the issue of computing the least hops path subject to multiple additive constraints.

9.2 Future Works

I will continue my ongoing work on QoS routing in high speed networks. Based on the route-based link state update model, I will focus on designing, implementing, and evaluating a fundamentally new route-based update policy and corresponding routing strategy. In addition, I would like to extend my past research to wireless networks, e.g., Ad Hoc and Sensor networks.

9.2.1 Congestion Control

I have devoted my endeavor to fair bandwidth allocation for AF traffic. Collaborating with my colleagues, I also have worked on traffic scheduling for Expedite Forwarding (EF) traffic and queue management for Best Effort (BE) traffic. Having obtained a good understanding of the congestion control in the context of Differential Service (DiffServ), I would like to investigate more QoS issues in both wired and wireless domain. For example, since there is no definition on how the bandwidth should be distributed between the AF out-of-profile (OUT) and BE traffic during times of congestion, I may deal with this problem by a new core-stateless queue management similar to CSPFQ. Furthermore, it is also possible for me to propose new traffic scheduling algorithms based on our proposed traffic model.

9.2.2 Overlay Network

Overlay networks have emerged as a powerful and highly flexible method for delivering content without modifications of the underlying network. Today's overlay solutions are developed independently under the implicit assumption that they will run isolated.

However, as more and more overlay networks are put into use, this assumption no longer holds, and the performance of these networks may be greatly undermined. Hence, I would like to address the problem of how to share the resources among competing overlays in an economical and computationally practical fashion. In addition, I also like to address the fault recovery and security issues in overlay networks.

9.2.3 Reliability and Security

As the Internet evolves to an infrastructure carrying critical applications, Internet faces increasing challenges in providing dependable packet delivery. Today's Internet is a loose interconnection of networks and accessed by users with diverse interests. Due to its sheer scale, faults, including the failures of hardware and software, human operational errors, and malicious attacks, have become very common. I expect to research on, but not limited to i) robust routing in the presence of failures, ii) increasing the life time and reliability of sensor networks, and iii) maintaining connectivity in ad-hoc network. In a long run, I aim to provide the resilient routing in both wired and wireless networks.

9.2.4 Information Theory for Network Issues

Classic information theory has been widely deployed in the developments of communications. However, there is no unified, basic theory yet that extends information theory to networks, and information theory is also rarely used or considered in network design. Motivated by my research in information theory based link state information update schemes, I am particularly interested in applying information theory to optimize network designs and protocols. As wireless networks have experienced a phenomenal growth in the last decade, it is imperative to develop a network information theory to understand the fundamental throughput and delay performance limits of wireless networks, which is also my another interest.

REFERENCES

[1] S.V. Raghavan and S.K. Tripathi, Network Multimedia System: Concepts, Architecture, and Design, Prientice Hall, Inc., 1998.

[2] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control - the single node case," *Proceedings of the INFOCOM'92*, pp. 915 –924, May, 1992.

[3] L. Zhang, "Virtual clock: A new traffic control algorithm for packet switching networks," *Proceedings of ACM SIGCOMM'90*, pp. 19-29, September 1990.

[4] J.C.R. Bennett and H. Zhang, "WF2Q: Worst-case fair weighted fair queueing," *Proceedings of IEEE INFOCOM'96*, pp. 120-128, March 1996.

[5] S. Golestani, "A self-clocked fair queueing scheme for broadband applications," *Proceedings of IEEE INFOCOM'94*, pp. 636-646, June 1994.

[6] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Service," *IETF RFC* 2475, 1998.

[7] T. Korkmaz and M. Krunz, "Routing multimedia traffic with QoS guarantees," *IEEE Transactions on Multimedia*, 2003, 5, (3), pp. 429-443.

[8] X. Yuan, "Heuristic algorithm for multiconstrained quality-of-service routing, " *IEEE/ACM Transactions on Networking*, 2002, 10, (2), pp. 244-256.

[9] S. Deering, D.L. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei; "The PIM architecture for wide-area multicast routing," *IEEE/ACM Transactions on Networking*, vol. 4, pp. 153-162, 1996.

[10] S. Yan, M. Faloutsos, and A. Banerjea, "QoS-aware multicast routing for the Internet: the design and evaluation of QoSMIC," *IEEE/ACM Transactions on Networking*, vol. 10, pp. 54-166, 2002.

[11] B. Wang and J.C. Hou, "Multicast routing and its QoS extension: problems, algorithms, and protocols," IEEE Network, vol. 14, pp. 22-36, 2000.

[12] S. Chen, K. Nahsted, and Y. Shavitt, "A QoS-aware multicast routing protocol," *IEEE Journal on Selected Areas in Communications,* vol. 18, pp. 2580-2592, 2000.

[13] Z. Wang and J. Crowcroft, "Quality of Service routing for supporting multimedia applications," *IEEE Journal on Selected Areas on Communications*, 1996, 14, (7), vol. 14, pp. 1228-1234, 1996.

[14] Y. Joo, V. Ribeiro, A. Feldmann, A. C. Gilbert, and W. Willinger, "TCP/IP traffic dynamics and network performance: a lesson in workload modeling, flow control, and trace-driven simulations," *Proceedings of ACM SIGCOMM'01*, vol. 21, pp. 25-37, 2001.

[15] S. Floyd and V. Paxson, "Difficulties in simulating the Internet," *IEEE/ACM Transactions on Networking*, vol. 9, pp. 392-403, 2001.

[16] W.E. Leland, W. Willinger, M.S. Taqqu and D. V. Willson, "On the self-similar nature of Ethernet traffic," *IEEE/ACM Transactions on Networking*, vol. 2, pp. 1-15, 1994.

[17] J. Moy, "OSPF version 2," RFC2328, *IETF*, 1998.

[18] S. Chen and K. Nahsted, "An overview of quality of service routing for next-generation high-speed network: problems and solutions," *IEEE Network*, 1998, 12, (6), pp. 64-79.

[19] R. Guerin and A. Orda, "QoS based routing in networks with inaccurate information: theory and algorithms," *Proceedings of the INFOCOM'97*, pp. 75-83, 1997.

[20] A. Kolarov and J. Hui, "Least Cost Routing in Multiple-Service Networks," *Proceeding of the INFOCOM'94*, pp. 1482-1489,1994

[21] B. Zhang, M. Krunz, H. T. Mouftah, and C. Chen, "Stateless QoS Routing in IP Networks," *Proceedings of IEEE GLOBECOM'01*, pp. 1600-1604, 2001.

[22] C. Hon, "Routing Virtual Circuit with Timing Requirement in Virtual Path Based ATM Networks," *Proccedings of IEEE INFORCOM'06*, pp. 320-328, 1996.

[23] R.K. Boel and J.H. van Schuppen, "Distributed routing for load balancing," *Proceedings of IEEE*, vol. 77, pp. 210-221, 1989.

[24] G. Manimaran, H. S. Rahul, and C. S. R. Murthy, "A New Distributed Route Selection Approach forChannel Establishment in Real-Time Networks," *IEEE/ACM Transactions on Networking*, vol.7, pp. 318-335, 1999.

[25] D. Ghosh, V. Sarangan, and R. Acharya,"Quality-of-Service Routing in IP Networks," *IEEE Transactions on Multimedia*, pp. 200-208, vol. 3, 2001.

[26] ATM Forum, *Private Network Interface (PNNI) v1.0 Specifications*, 1996.

[27] G. Apostolopoulos, R. Guerin, S. Kamat, and S. Tripathi, "Quality-of-service based routing: A performance perspective," *Proceedings of ACM SIGCOMM 1998*, vol. 28, pp. 17–28, 1998.

[28] A. Shaikh, J. Rexford, and K. G. Shin, "Evaluating the impact of stale link state on quality-of-service routing," *IEEE/ACM Transactions on Networking*, 2001, 9, (2), pp. 162-176.

[29] Q. Ma and P. Steenkiste, "Quality-of-service routing for traffic with performance guarantees," *Proceedings of IFIP Int. Workshop Quality of Service 1997*, pp. 115–126, 1997.

[30] L. Breslau, D. Estrin, and L. Zhang, "A simulation study of adaptive source routing in integrated services networks," Computer Science Department, University of Southern California, Tech. Rep. 93-551, 1993.

[31] M. Peyravian and R. Onvural, "Algorithm for efficient generation of link-state updates in ATM networks," *Computer Networks and ISDN System*, vol. 29, pp. 237–247, 1997.

[32] X. Li, L. K. Shan, W. Jun, and N. Klara, "QoS Extension to BGP," *Proceedings of IEEE ICNP'02*, pp. 100- 109, 2002.S.

[33] W.E. Leland, W. Willinger, M.S. Taqqu and D. V. Willson, "On the self-similar nature of Ethernet traffic," *IEEE/ACM Transactions on Networking*, vol. 2, pp. 1-15, 1994.

[34] B.A. Mah, "An empirical model of HTTP network traffic," Proceedings of the IEEE INFOCOM'97, vol. 2, pp. 592-600, 1997.

[35] M. E. Crovella and A. Bestavros, "Self-similarity in World Wide Web traffic: evidence and possible causes," Proceedings of ACM SIGMETRICS'96, pp. 160-169, 1996.

[36] A. Zinin and M. Shand, "Flooding optimizations in link-state routing protocols," *IETF*, draft-ietf-ospf-isis-flood-opt-01.txt, 2001.

[37] J. Moy, "Flooding over a subset topology," *IETF*, draft-ietf-ospf-subset-flood-00.txt, 2001.

[38] B. G. Ogier, "A reliable, efficient topology broadcast protocol for dynamic networks," Proceedings of 1999 *IEEE Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 1, pp. 178-186, 1999.

[39] K. Thulasiraman and M. N. S. Swamy, *Graphs: Thoery and Algorithms*, Wiley-Interscience, U.S.A, 1992.

[40] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, Cambridge, MA: MIT Press, 1990.

[41] H. Huang and J. Copeland, "Hamiltonian cycle protection: a novel approach to mesh WDM optical network protection," P*roceedings of 2001 IEEE Workshop on High Performance Switching and Routing*, pp. 31-35, May, 2001.

[42] R. Guerin and A. Orda, "Computing shortest paths for any number of hops," *IEEE/ACM Transactions on Networking*, 2002, 10, (5), pp. 613-620.

[43] D.R. Karger, D. Koller, and S. J. Phillips, "Finding the hidden path: Time bounds for all-pairs shortest paths," *SIAM J. Computing*, 1993, vol. 22, pp. 1199-1217.

[44] R. Guerin and A. Orda, "QoS based routing in networks with inaccurate information: theory and algorithms," *Proceedings of the INFOCOM'97*, pp. 75-83, 1997.

[45] J. Wang, W. Wang, J. Chen, and S. Chen, "A randomized QoS routing algorithm on networks with inaccurate link-state information," *Proceedings of WCC-ICCT 2000*, vol. 2, pp. 1617-1622, 2000.

[46] D. H. Lorenz and A. Orda, "QoS routing in networks with uncertain parameters," *IEEE/ACM Transactions on Networking*, 1998, 6, (6), pp. 768-778.

[47] S. Chen and K. Nahrstedt, "Distributed QoS routing with imprecise state information," *Proceedings of 7th International Conference on Computer Communications and Networks*, pp. 614-621, 1998.

[48] A. Orda and A. Sprintson, "Precomputation schemes for QoS routing," *IEEE/ACM Transactions on Networking*, 2003, 11, (4), pp. 578-591.

[49] G. Liu and K. G. Ramakrishnan, "A*Prune: an algorithm for finding K shortest paths subject to multiple constraints," Proceedings of IEEE INFOCOM 2001, vol. 2, pp. 743-749, 2001.

[50] T. Korkmaz and M. Krunz, "Bandwidth-delay constrained path selection under inaccurate state information," *IEEE/ACM Transactions on Networking*, 2003, 11, (3), pp. 384-398.

[51] A. Juttner, B. Szyiatovszki, I. Mecs, and Rajko, "LaGrange relaxation based method for the QoS routing problem," *Proceedings of IEEE INFOCOM 2001*, vol. 2, pp. 859-868, 2001.

[52] D. Eppstein, "Finding the k shortest path," *Proceedings of 35th Annual Symposium on Foundations of Computer Science*, pp. 154-165, 1994.

[53] H. De Neve and P. Van Mieghem, "A multiple quality of service routing algorithm for PNNI," *Proceedings of 1998 IEEE ATM workshop*, pp. 324-328, 1998.

[54] S. Chen and K. Nahrsted, "On finding multi-constrained path," *Proceedings of IEEE ICC'98*, vol. 2, pp. 874-899, 1998.

[55] T. Korkmaz., M. Krunz, and S. Tragoudas, "An efficient algorithm for finding a path subject to two additive constraints," *Proceedings of the ACM SIGMETRICS'2000*, pp. 318-327, 2000.

[56] C. Pomavalzi, G. Chakraborty, and N. Shiratori, "QoS based routing algorithm in integrated services packet networks," *Proceedings of IEEE 1997 Conference on Network Protocols*, pp. 167-174, 1997.

[57] R. Widyono, "The design and evaluation of routing algorithms for real-time channels," *Technical Report TR-94-024*, University of California at Berekey, 1994.

[58] D. H. Lorenz and D. Raz, "A simple efficient approximation scheme for the restricted shortest path problem," *Operations Research Letters*, 2001, vol. 28, pp. 213-219.

[59] D. H. Lorenz and A. Orda, "Efficient QoS partition and routing of unicast and multicast," *Proceedings of 8th International Workshop on Quality of Service*, pp. 75-83, 2001.

[60] R. Hassin, "Approximation schemes for the restricted shortest path problem," *Mathematics of Operations Research*, 1992, 2, (2), pp. 36-42.

[61] D. Raz, and Y. Shavitt, "Optimal partition of QoS requirements with discrete cost functions," *IEEE Journal on Selected Areas in Communications*, 2000, vol. 12, (18), pp. 2593-2602.

[62] A. Goel, K.G. Ramakrishnan, D. Kataria, and D. Logothetis, "Efficient Computation of Delay-sensitive Routes from One Source to All Destinations," *Proceedings IEEE Infocom 2001*, pp. 854-858, 2001.

[63] M.B. Doar, "A better model for generating test networks," *Proceedings of IEEE GLOBECOM'96*, pp. 86-93, 1996.

[64] G. Cheng and N. Ansari, "An Information Theory Based Framework for Optimal Link State Update," *IEEE Communications Letters*, vol. 8, pp. 692-694, 2004.

[65] N. Ansari, G. Cheng, and R.N. Krishnan, "Efficient and reliable link state information dissemination," *IEEE Communications Letters*, vol. 8, pp. 317 - 319, 2004.

[66] Y. Jia, I. Nikolaidis, and P. Gburzynski, "Multiple path routing in networks with inaccurate link state information," *Proceedings of the IEEE ICC'01*, vol. 8, pp. 2583–2587, 2001.

[67] G. Cheng and N. Ansari, "Finding all hops k-shortest paths," *Proceedings of IEEE PACRIM'03*, vol. 1, pp. 474 - 477, 2003.

[68] G. Cheng and N. Ansari, "Achieving 100% success ratio in finding the delay constrained least cost path," to appear in *IEEE GLOBECOM'04*, 2004.

[69] S. Chen, "Routing Support for Providing Guaranteed End-to-End Quality-of-Service," Ph.D. dissertation, Department of Computer Science, University of Illinois at Urbana-Champaign, May 1999.