Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

MECHANISMS FOR QUALITY-OF-SERVICE PROVISIONING IN NETWORKS WITH EXTENDED SERVICES

by
Zhen Qin

The emerging network traffic with various Quality-of-Service (QoS) requirements creates a demand for QoS service provisioning beyond the best effort service that Internet currently provides. QoS provisioning requires a framework that satisfies users' QoS and cost demand while maximizes benefits for network service providers. It is considered that QoS provisioning involves three issues: a) estimations of the network QoS performance, which can be achieved by performing network measurement; b) dissemination of the measured QoS states throughout the network with states exchanged among different network routers; and c) QoS routing. In this dissertation, these three issues are addressed.

In QoS networks, knowledge of the network status is crucial but current implemented network protocols cannot provide enough QoS measurement functions. For such purposes, a network measurement framework is proposed that runs active measurement tools to estimate multiple QoS classes. An important issue involved in the network measurement is the tasks conflict problem. This problem occurs when multiple active measurement tasks sending probing packets in the same network segment at the same time, and causes misleading report of QoS performance because the tasks' contention for network resources disturb each other's measurement. In this dissertation a novel scheduling algorithm is proposed to allow such contention among measurement tasks and to shorten the measurement period time.

In addition, flooding algorithm is dominantly used in link state dissemination. It faces the large overhead problem when used in QoS networks since a lot of QoS states need to be updated frequently. On the contrary, the alternative algorithm, single spanning tree dissemination, may not be able to achieve fast convergence or reliability. In the proposed

new scheme, Per-Hop pArtial-Spanning Tree Adjust (PASTA) for dissemination, link states can be distributed with low overhead and fast speed, and the computation complexity to build the tree is small compared to single spanning tree algorithm. The reliability of the dissemination is enhanced by the multi-spanning-tree approach and the back-trace method.

Furthermore, in most of the current QoS architectures, a traffic flow receive the unified service at every hop on the path, resulting that the end-to-end QoS provisioning lacks flexibility and granularity. To solve this problem, a nested DiffServ model is presented. In this model, service at each hop is quantified and divided into multiple classes, and the edge router on behalf of user is allowed to select different service at each hop. Under this framework, routing in terms of cost and QoS requirements can be regarded as a delay-least-low-cost (DCLC) problem, which is known to be NP-hard. An improved k-shortest path QoS routing algorithm is proposed to solve this problem.

MECHANISMS FOR QUALITY-OF-SERVICE PROVISIONING IN NETWORKS WITH EXTENDED SERVICES

by
Zhen Qin

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 and Computer Engineering

May 2010

APPROVAL PAGE

MECHANISMS FOR QUALITY-OF-SERVICE PROVISIONING IN NETWORKS WITH EXTENDED SERVICES

Zhen Qin

Dr. Roberto Rojas-Cessa, Dissertation Advisor — Date
Associate Professor, Department of Electrical and Computer Engineering, New Jersey Institute of Technology

Dr. Nirwan Ansari, Committee Member — Date
Professor, Department of Electrical and Computer Engineering, New Jersey Institute of Technology

Dr. Sotirios G. Ziavras, Committee Member — Date
Professor, Department of Electrical and Computer Engineering, NJIT

Dr. Yanchao Zhang, Committee Member — Date
Assistant Professor, Department of Electrical and Computer Engineering, NJIT

Dr. George Lapiotis, Committee Member — Date
Senior Research Scientist, Applied Research, Telcordia Technologies

BIOGRAPHICAL SKETCH

Author: Zhen Qin

Degree: Doctor of Philosophy

Date: May 2010

Undergraduate and Graduate Education:

- Master of Science in Electronic Engineering,
  Queen Mary, University of London, London, UK, 2003
- Bachelor of Science in Communications Engineering,
  Beijing University of Posts and Telecommunications, Beijing, P.R.China, 2002

Major: Electrical and Computer Engineering

Publications:

Z. Qin, R. Rojas-Cessa, and N. Ansari, "Task-Execution Scheduling Schemes for Network Measurement and Monitoring," *Elsevier Journal of Computer Communications*, Vol. 33, Issue 2, pp. 124-135, Feb. 2010.

R. Rojas-Cessa, and Z. Qin, "Proactive Routing for Congestion Avoidance in Network Recovery under Single-Link Failures," *in Proceedings of IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 822-825, Nov. 2009.

Z. Qin, R. Rojas-Cessa, and N. Ansari, "Descending-Order Clique-Based Task Scheduling for Active Measurements," *in Proceedings of IEEE Conference on High Performance Switching and Routing (HPSR)*, pp. 1-6, May 2007.

Z. Qin, R. Rojas-Cessa, and N. Ansari, "Distributed Link-State Measurement for Accurate QoS Routing," *in Proceedings of IEEE Conference on Military Communications (MILCOM)*, pp. 1-6, Oct. 2006.

Z. Qin, R. Rojas-Cessa, and N. Ansari, "OSPF-Based Adaptive and Flexible Security-Enhanced QoS Provisioning," *in Proceedings of IEEE Sarnoff Symposium*, Princeton, NJ, Mar. 2006.

To my wife and parents

ACKNOWLEDGMENT

I wish to express my sincere gratitude to my advisor, Dr. Roberto Rojas-Cessa, for giving me the opportunity to work with him and guiding me through this learning process. I am greatly indebted to my advisor for his inspiration and encouragement over my Ph.D. years. He is always open to new ideas and I really appreciate the support he gave me while working on my research. This research could not have been possible without his brilliant ideas. Not only was he readily available for me when I have questions, as he generously is for all his students, but also he carefully goes through the drafts of my work. Moreover, Dr. Rojas-Cessa also provided valuable facilities and brilliant guidelines for testing and evaluation of my dissertation. I owe much for his unending help to do the research and for his financial support during my graduate study.

I am also very grateful to Professor Dr. Nirwan Ansari, who offered his invaluable suggestion and dedicated help throughout these years. I feel fortunate and will never forget his support and contributions to my academic development.

I would also like to thank Dr. Sotirios Ziavras, Dr. Yanchao Zhang, and Dr. George Lapiotis for serving on my committee, reviewing this dissertation and making constructive comments.

My thanks go out to my lab mates Gang Cheng, Zhen Guo, Chuan-Bi Lin, Ziqian Dong, Lin Cai, Khondaker Salehin, Jie Yang and Jingjing Zhang who provided help and suggestions for my research. All my friends in and out of NJIT have been great source of support and fun. I thank them for making this journey productive and enjoyable.

I am grateful to the donors of the National Science Foundation which provide financial support for my study.

Finally, I expressed my special gratitude to my parents and my wife, for their dedicated and endless love. They supported me with their best in every aspect. Without them, I would not have achieved anything that I achieved today.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure Page

LIST OF FIGURES
(Continued)

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Challenges in QoS Network Provisioning

Nowadays, users demand wider spectrum of network services, many of them with Quality-of-Service (QoS) requirements. Popular applications, such as VoIP, streaming video, and online gaming have stringent requirement of delay, bandwidth, and jitter that current Internet's best effort mechanism cannot support [1, 2]. The implementation of multimedia and security service in the next generation networks is coupled with the limited QoS parameters the network (Internet) service providers (ISP) offer. QoS provisioning is a critical issue in designing current and next generation networks. However, providing satisfying and efficient QoS guarantees is challenging because of the following reasons:

- Granularity. The network is shared by various and large amount of end-users, most of which have their particular QoS requirements. This makes the provider consider specific performance guarantees for each traffic flow, so that the processing complexity of parameters of each flow is high and the required storage space is large, which causes processing delays in transmitting packets and challenges hardware processing and memory capability.

- Optimization of path and service selection. That is, to find a path that can provide a satisfying service to the end user, and minimize the user's cost or maximize the network provider's benefit, including maximizing network utilization. Optimization of such routing problem is known to be NP-complete.

- Service quantification. The quality of network service parameters are set up by different network applications. The common parameters include capacity, available bandwidth, delay, jitter, packet loss ratio; round trip time (RTT). Network security plays another critical role in the service provisioning; therefore it is also regarded as a QoS parameter in recent literature. To improve the flexibility of the service, it is necessary to separate this service into various classes so as to increase the options for users to select. However, some QoS parameters are difficult to be quantified. As an example, security may be quantified from user's perception perspective rather than by a quantifiable parameter. As another example, VoIP applications also estimate quality of service based on speaker's satisfaction feedback.

- Service pricing. QoS provisioning requires cooperation among multiple network service providers. Different providers can use mutual agreements on service charges and map service classes among partner ISPs. Service providers who are independent organizations, may be only concerned with its own profit, creating possible conflicts on fulfilling their agreements to provide the QoS service.

- QoS state distribution. The accuracy of QoS paths and service selections depends on the accuracy of the QoS states. The dynamic feature of QoS parameters [3–5] make it hard to maintain accurate link state information of the whole network. Otherwise, intolerable amount of network resources are wasted during the dissemination of link states, which causes disturbance to QoS service on data traffic and increases the complexity for the design of high performance network routing mechanisms.

- QoS measurement. The accuracy measurement of the QoS state is also important. While efforts in network measurement have produced a wide variety of hardware and software monitoring tools, demand for faster and more flexible measurement and monitoring technology is increasing. With the requirement of granularity of QoS, large amount of measurement processes need to be carried out. Thus, the measurement processes need to be scheduled carefully to avoid the contention for network resources; on the other hand, the measurement overhead must be limited within an acceptable level to save resources for user data traffic.

1.2 QoS Network Architecture

These issues have obtained attention from IETF and ITU-T for many years. In the IETF, the IP Performance Metric (IPPM) Working Group [6] is organized to define a set of QoS parameters (so called metrics in IPPM) and to provide quantitative characteristics of networks [7]. ITU-T proposed general Recommendation I.350 [8] or for IP in Recommendation Y.1540 [9], Y.1541 [10] and G.1010 [11], which recommends the use of a statistical probabilistic definition for the QoS parameters.

IETF also organizes several other working groups for QoS provisioning. Open Shortest Path First (OSPF) protocol [12] proposed by IETF is by now one of the most pervasive routing protocol implemented in the Internet. The OSPF protocol defines the link state or weight as inversely proportional to the link capacity. The IETF OSPF IGP Working Group extends the weight (cost) of a link as originally designed in OSPF by the association of QoS parameters with it [13], while the link state update policing and the routing based

on the QoS-mapped weight design remain unaddressed. Besides that, IETF organizes the Common Control and Measurement Plane (CCAMP) Working Group [14], which aims to provide a common control plane and a separate common measurement plane for physical path and core tunneling technologies of Internet and telecom service providers. CCAMP mainly focuses on MPLS networks and does not consider network-layer based QoS state measurement and monitoring.

On the existing QoS architecture, Integrated Services (Intserv) [15] and Diffserv [16] are well-known paradigms. The Intserv model is designed to provide network service to each individual data flow generated by users. Resources are reserved and allocated to each flow. A signaling protocol, namely, the Resource Reservation Setup Protocol (RSVP) [17] was proposed to assist Intserv model so that, the admission control and end-to end resource allocation is able to be performed in a dynamic way; the state of every flow can be updated in the Intserv routers. Since the Intserv requires per-flow management in the network and a router must keep information about the state of each flow, the QoS granularity is improved but with the cost of management and monitoring is high as the complexity, thus Intserv has limited scalability.

In a different approach, the Diffserv model only supports a number of services to end users, based on the Service Level Agreement (SLA) between the user and the service provider. The common service classes include best effort (BE), assured forwarding (AF) and expedited forwarding (EF) [18]. A data flow is marked by the terminal (host) or leaf router and classified, metered, shaped, and possibly re-marked at the ingress router where the flows are aggregated according to the service class and be forwarded to the core routers. At core routers, the aggregated flows are serviced according to the Per-Hop-Behavior (PHB) associated with each service classes. The core router is not required to keep the state of each traffic flow but only the service classes of the aggregated flows, thus Diffserv has better scalability but only supports coarse end-to-end QoS granularity. Flows

belonging to the same service class receive same end-to-end QoS service even if they have slightly different QoS requirements.

To balance the trade-off of granularity and scalability, the Intserv over Diffserv model was proposed [19], but whether Intserv over Diffserv model can fill this void remains an open issue, in which case it is necessary to map each individual flow's end-to-end QoS requirements from Intserv model to the Diffserv model. Recently a new approach of admission control referred to as endpoint admission control (EAC) has been proposed [20–22]. In EAC schemes, the end host sends out probing packets to collect and evaluate the end-to-end performance that a flow may experience. The connection request will be cancelled if the probing indicates that the current congestion level is high or the QoS provided by Diffserv network is not acceptable. Since it is end host instead of router to process the admission control on each individual flow, the scalability feature is saved in EAC. However, the probing path in EAC is pre-selected therefore EAC is separated from QoS routing but only admission control and service selection, thus the false routing may occur and the user's cost may increase.

1.3 Network Measurement Overview

As mentioned in section 1.1, QoS measurement has been researched and it remains as a challenging topic in network QoS provisioning. Currently there are two types of measurement methods that have been proposed for network measurements:

1. Active probing measurement: The router or end host deliberately sends probing packets (also named measurement packets) to the target router with precisely controlled departure time, and either the destination measures the arrival time or the source estimates the result from the feedback of the target router. This method has been prevalent since the late 1980's [23, 24]. It can be categorized into two classes: delay based and dispersion based. In the first category, path characteristics such as per-hop capacity, queuing delay, and link utilization are inferred based on the round-trip-time (RTT) or one-way delay of individual

packets. The dispersion based method can estimate capacity and available bandwidth, by observing the changes such as the space between the neighbors of probing packets pairs or trains. The probing traffic can be generated by marking the user data packets and using them as probing packets, so called in-band probing, or by injecting measurement packets only for probing purposes. This method is categorized as out-of-band probing [25]. Although inbound probing does not introduce additional traffic except for the marking bits, the marking and analysis processing at the measurement points may cause delay to the data traffic used for probing, hence currently most active measurement tools adopt out-of-band probing. This dissertation also focuses on out-of-band probing.

2. Passive measurement: In comparison to the active measurement, passive measurements do not inject probing packets into the network. They estimate the network performance according to the traffic information captured from the header of the incoming packets by network devices (for example, switches and routers) over various network paths. NetFlow [26], implemented in Cisco routers, is one example. It counts the number of bytes, packets, flag fields, time of the flow to measure bandwidth/link utilization between network backbone routers. IETF also organizes the workgroup for passive measurement framework [27]. Passive measurements can provide captured data with fine details, but large amount of per-flow information to be store requires large network storage resources such as memory of the router that are expensive. Furthermore, for high speed networks, the packets pass through a monitor point are counted in gigabyte per second units. Routers have to sample the traffic to alleviate the processing burden and large storage, which diminishes the accuracy of the measurement. The involved non-negligible installation and maintenance cost also impede the wide-spread deployment of passive measurement. Meanwhile, passive measurement lets ISPs know the performance of their domains, but lacks the ability to identify performance problems beyond a single ISP. Thus active measurements become a possible method to replace or complement passive measurements.

1.4 Scheduling of Network Measurement Tasks

Because different active measurement tools may be executed simultaneously at one measurement point, it is possible that different measurement tools contend with each other for the network (or node) resources and the transmission channels, such as processing time, bandwidth, and memory. Hence, the active measurement tasks that are performed at each measurement point need to be carefully scheduled to avoid both potential resource contention and measurement disturbance from each other measurement tasks. Otherwise, contentions might affect the network measurement results, which may be interpreted as a network problem, and mislead network administration and affect managing decisions. Even worse, measurement without scheduling may cause a traffic burst that may impair the users' data transmission quality.

Measurement tasks are also required to be executed as soon as possible so that the network performance state can be updated timely to different QoS management systems (e.g., QoS routers and server hosts). As the frequency of measurement tasks increases, the measurement traffic also increases, and this raises the possibility of measurement disturbance. Therefore, it is of interest to minimize the measurement time to perform the set of required measurements in each round (a set of measurement tasks that need to be performed within a specified deadline or period of time). The repetition of the measurement tasks, required for service monitoring, can be facilitated. That is, more active probing can be injected into the network each time interval.

In this dissertation, a novel scheduling scheme to resolve contention for resources of both periodic and on-demand measurement tasks from graph coloring perspective, called ascending order of the sum of clique number and degree of tasks. The scheme selects tasks according to the ascending order of the sum of clique number and conflict task degree in a conflict graph and allows concurrent execution of multiple measurement tasks for high resource utilization. The scheme decreases the average waiting time of all tasks in periodic measurement tasks scheduling. For on-demand measurement tasks, the proposed scheme

minimizes the waiting time of inserted on-demand tasks while keeping time space utilization high. In other words, the total time spent on finishing all the tasks is shortened.

1.5 Link State Dissemination

Current schemes for flooding-based link-state dissemination do not consider QoS parameters and cannot provide timely updates of link states. OSPF protocol, as an example, uses flood algorithm for link state update. For the sake of minimizing the administration complexity and overhead, the OSPF protocol recommends link state be updated periodically with intervals as large as 30 minutes. This is a low update frequency to obtain current link states for today's network capacities and speeds and for QoS-enabled networks. Spanning-tree based schemes have been proposed as an alternative to the flooding approach, but their convergence times are long and they may compromise service agreement compliance if a link of the spanning tree is affected. Therefore, the construction of the spanning tree cannot follow the latest link states but store a stale state. This makes the dissemination path suboptimal, causes slow responses with inaccurate state estimation, and makes the network intolerant to link failures.

1.6 QoS-Enabled Routing

The basic function for QoS routing is to find a path (as unicast routing) or tree (as multicast routing) to satisfy users' QoS requirement by selecting the proper link and service in a network. The Dijkstra algorithm to build a routing tree is the well-known shortest path calculation algorithm to find a routing path with only one constraint (for example, cost of path). However, in QoS routing, multiple constraints need to be considered. This routing process is named as multi-constraint routing. The constraint could be the minimum bandwidth and end-to-end delay the user requires, and the end-to-end cost the user budgets. Multi-constraint routing is an NP-hard problem [28]. There are there types of routing strategies that can address multi-constraint routing: source routing [29], distributed routing

[30] and hierarchical routing [31]. This dissertation mainly discusses the source routing approach.

CHAPTER 2

NETWORK QOS MEASUREMENT ARCHITECTURE AND SCHEDULING OF MEASUREMENT TASKS

2.1 Introduction

In this chapter, a distributed QoS state measurement architecture and a measurement task scheduling algorithm are proposed. The presented measurement framework is able to estimate multi-class QoS levels at each link of the network. The scheduling algorithm aims to minimize the scheduling time for each round of periodic measurement tasks, and to resolve resources the contention among various tasks. An analysis and simulation are presented in this chapter. They show the advantages of the proposed algorithm when compared to other existing schemes.

2.1.1 Active Measurement

Current QoS applications need QoS report to verify the service level agreement (SLA). Active measurement and passive measurement can help them to obtain such information. Because of the disadvantages of passive measurement as mentioned 1 and the controllable properties of active measurement, active measurement becomes a better solution for QoS measurements. [23, 32–42]. In active measurement, routers or end hosts, as the measurement points, deliberately send probing packets (also named measurement packets) to the target destinations with precisely controlled departure time, and either the destination measures the arrival time of such packet or the source estimates the resulting delay from the feedback of the target router or host [43–46]. The network information obtained by such measurements includes available bandwidth, capacity, and one-way delay, round trip time (RTT), jitter, and topology. The knowledge of such parameters facilitates various network administration tasks, network monitoring against network threats (e.g., denial-of-service attacks and hot spots), traffic engineering (e.g., QoS routing and link state update), and

billing (e.g., pricing based on traffic amount or QoS performance). Such active measurements have been implemented in some advanced existing networks [47–49]. Examples of tools for active measurement [43] include the simple ones such as Ping and Traceroute, and the sophisticated, such as Pipechar [50], Pathload [51], Cing [52], Clink [53], Nettimer [54], Pathrate [55], Pathchar [56], Yaz [57].

There are different tools to measure each QoS parameters and some tools may evaluate more than one parameter. Some of those tools are described below.

2.1.1.1 Bandwidth. In computer networking, digital bandwidth or just bandwidth often refers to a data rate measured in bits/s, for instance, network throughput. These measurement tools are used to evaluate two bandwidth metrics: capacity and available bandwidth. Capacity C_i of link i is defined as the maximum possible Internet Protocol (IP) layer transfer rate through that link. It should be noticed that the capacity of the link at IP layer delivers a lower rate than that its claimed transmission rate due to the overhead of data-link-layer encapsulation and framing. Given the claimed capacity and overhead of data link layer with C_{layer2} and H_{layer2}, the transmission time for an IP packet with size L is $T = (L + H_{layer2})/C_{layer2}$, so the capacity of the IP layer is $\frac{L}{T} = C_{layer2} \bullet \frac{1}{1+H_{layer2}/L}$, which is less than C_{layer2}. The available bandwidth of a link is the maximum throughput that the link can provide to an application, given the link's current cross traffic load. Because of its volatility, the available measurement should last a period to average the value. The existing active measurement techniques are mainly divided into four families:

- Variable Packet Size probing (VPS): This scheme estimates the capacity of each link along the path. The time-to-live (TTL) field in IP header is set to force probing packets to expire at the target link. The router at that link then feedback Internet Control Message Protocol (ICMP) time-exceeded error messages back to the source, which measures the RTT to that link. The tools Clink, Pchar, Pathchar, and Bing belong to this category. However, VPS probing may cause large capacity underestimation

errors if the path it passed includes a store-and-forward data link layer switch [58] because it introduces other delays hidden from the IP layer.

- Packet Pair/Train Dispersion Probing (PPTD): This scheme estimates end-to-end capacity (i.e., the minimal capacity of the links along the path, so called narrow link) of a path by sending multiple packet pairs which includes two-packets back-to-back with same size, or multiple back-to-back packets with same size as a train. For packet pair probing, if the previous dispersion between the first and last packet is ΔT_{i-1}, after passing the current link i with capacity C_i, the dispersion is $\Delta T_i = max(\Delta T_{i-1}, L/C_i)$. As the first dispersion $\Delta T_1 = L/C_1$, the final dispersion at the receiver is:

$$max_{i=1,2,...,n}(L/C_i) = L/min_{i=1,2,...,n}(C_i) \tag{2.1}$$

that is, inversely proportional to the path capacity $min(C_i)$, so the end-to-end capacity can be estimated. The drawback of this mechanism is the unrealistic assumption that there is no other traffic so called cross traffic in the network. To overcome this disadvantage, packet train is proposed. This approach follows the same algorithm as the one described above, but N packets are used for one train. In this case, the impact of cross traffic will be smaller as N increases. Figure 2.1 illustrates this method. Typical PPTD tools need the software installed on both sender and receiver side. Nettimer, Abing, Spruce, Pipechar, Bprobe, Cprobe, Sprobe, and Pathrate are some examples of PPTD tools.

- Self-Loading Periodic Streams (SLoPS): This scheme sends a number of equal-sized packets to the receiver, or saying sink, at rate R to estimate the end-to-end available bandwidth. The sink informs the source about the one-way delay of the probing stream. This delay keeps increasing means that the sending rate R is larger than the available bandwidth of the tight link (the link with input stream rate larger than output stream rate is called tight link) B_a so that queuing delay occurs. Otherwise, a non-

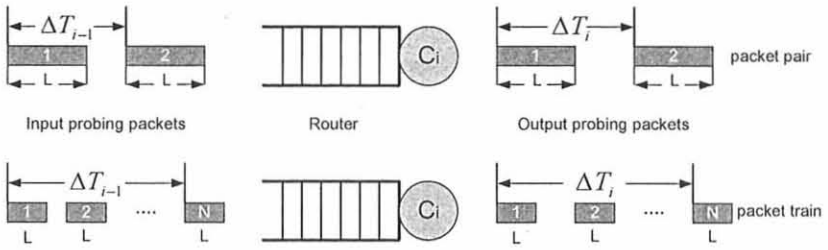

Figure 2.1 Packet pair and packet train dispersion.

increasing one-way delay accounts for $R < B_a$, and the source will keep sending probing packets with higher rate than R to approach the sending rate to B_a. In this case, there should be only one stream along the path, and the source must create a silent period between the successive streams to guarantee the average probing rate below 10% of B_a. Pathchirp, IGI, PTR, and Pathload are the members of this family.

- Trains of Packet Pairs (ToPP): This scheme was proposed to measure the available bandwidth of the path [59]. ToPP is based on the same principle as SLoPS where the sender sends several packet pairs to the receiver or with linear increasing rate. The initial dispersion inside the packet pair ΔT_s as well as the offered rate of the packet pairs is known:

$$R_s = L/\Delta T_s \tag{2.2}$$

If R_s is larger than the available bandwidth B_a, the receiver obtains a dispersion smaller than ΔT_s which infers a measured rate $R_m < R_s$, so that the sender increases the sending rate to one close to B_a. Besides that, ToPP is able to measure the capacity of the link with minimum available bandwidth (so called tightest link) in the path.

Some research groups compared the performance of different bandwidth estimation tools. Clink, Pchar, Pathchar, Abing, Spruce, Patchirp, IGI and Ipiechar are implemented end-to-end among different cities on France national research network [47]. The results show that most of the tested tools overestimate bandwidth because of the long routing path. Hence, per-link measurement may achieve better performance than end-to-end measure-

ment. Furthermore, the tools may be disturbed due to the network resource contention when cross traffic exists. Hence the active measurement tasks need to be carefully scheduled to balance the network resources.

2.1.1.2 One-Way Delay. Round-trip time (RTT) is often referred as an approximation delay, and it is easily measured by using Ping command. However, this measurement is the sum of both forward and reverse delays. Note that half RTT cannot infer the one-way delay as those two delays may not be the same in an asymmetry network, which is common in the real environment.

IETF standardized the one-way delay measurement framework [60]. In the proposal, the IP header of measurement packet is time-stamped when the packet is sent and received. The sink collects the one-way delay by computing the difference of sending and receiving timestamps. An important key issue is the synchronization of the clock between the source and the destination. GPS systems provide a solution to synchronize both sides within several 10s of μec but its high expense and strict requirements on antenna location (faint GPS signals require an antenna be mounted outside with a clear view of the sky) prevents it from wide deployment. A code division multiple access (CDMA) base station includes a GPS receiver that broadcasts the time to a CDMA handset, so CDMA receivers can also be used for synchronization. The disadvantage of this approach is that the transmission delay always fluctuates because the distance between the receiver and base station is unknown. Another approach Network-time protocol (NTP) [61] may achieve synchronization within several milliseconds, so that NTP can be used for delay measurement when the tolerance of the estimation error is larger than milliseconds. Internet2's project OWAMP [62], as an implementation of IETF OWAMP protocol, is based on NTP.

Some other mechanisms have attempted to evaluate delay without time synchronization. Choi et.al [63] proposed an algorithm in a TCP environment for this purpose. The forward (reverse) delay is the deduction of difference between the accumulated RTT mea-

sured by sender and the receiver side. However, it requires the initial forward delay in the first round, which needs a heuristic to predict it.

2.1.1.3 Packet Loss. IETF [64] also released a standard for one-way packet-loss measurement: at the source, the prepared packet with a timestamp is transmitted. The destination samples the coming packet within a time period ([64] proposed pseudo-random Poisson sampling, but this is not exclusive). If the packet fails to arrive within a reasonable time period or is corrupted, the destination considers it lost. Similarly to one-way delay measurement, clock synchronization is required so the transmitted packets can be counted by sender and receiver in exact same time period. On the other hand, to get an accurate projection of loss rate in a low-packet-loss network, a measurement point can either send the probing packet in a low rate but let the probing phase last long time, or transmit a high burst of packets in a transitory period. However, both solutions have disadvantages. The former produces coarse average loss ratio as the network performance is continually changed. Increasing the probing rate improves the measurement resolution, but the probe packets themselves can skew the results and disturb the existing traffic if the frequency is too high. [24] compared the probe-based packet-loss measurement with the passive method. The experiment reveals the active measurement suffers from high variance and do not correlate with those results from passive measurement. Thus the active measurement at low loss rates need be careful.

2.1.2 Passive Measurement

Most time the data of passive measurement is only accessible to the network operators, who use it to manage today's complex internet. Besides routing, the information is used for intrusion detection system to detect hot spots and denial-of-service attacks, for link state update, and billing.

2.1.2.1 NetFlow. NetFlow [26] is built into most Cisco switches and routers. It maintains a flow cache to contain flow records which maps the traffic relayed by the router. The current version 9 of NetFlow supports the IETF standard called IP Flow Information Export (IPFIX) to export the record by UDP to a collector application that analyzes and archives the data. The router retrieves the IP header of the traffic to identify the flow by its source and destination IP address, source and destination port, layer 4 protocol type, type of service (TOS) byte, and input logical interface. A new flow record is inserted into the flow cache if packet does not match current existing flow. NetFlow will let the flow end and be exported if TCP flags (FIN or RST) arrive, after a waiting threshold is passed (15sec, configurable), or 30 minutes (configurable) after the record is created, or else, if the flow cache is full. In addition to the flow IDs, the recorded entries also include the number of bytes and packets in the flow and the timestamps of the first and last packets. NetFlow looks up the record entry when a packet arrives, then update the corresponding entry. In high-speed networks with large number of flows, sampling policy is utilized by NetFlow to provide scalability. An early version of NetFlow provided deterministic sampling (named Sampled NetFlow), which selected every n_{th} packet for processing on a per-interface basis called "1 out of n". Then random sampling methodology (named random sampled NetFlow) is introduced in new version. This approach selects packets randomly with a fixed sampling probability, so it is more statistically accurate than deterministic sampling when traffic arrives in fixed patterns. However, it is true that sampling (often 1 in 100, or 1 in 1000) hides the full details of the traffic and causes estimation errors. Many researchers have provided improved sampling strategies to dynamically adjust sampling rates according to specific network scenarios [65].

2.1.2.2 DAG. Data Acquisition and Generation (DAG) [66] is a network monitoring interface card, originally developed by university of Waikoto, New Zealand, and later commercialized by Endace company. Regarded as accurate monitoring equipment, it is often

used to calibrate the active measurement software. It captures the whole traffic on the monitored link without packet loss. Based on its low CPU occupancy feature, DAG cards claim be able to handle data rates of up to OC192 and 10GigE with full line utilization. It uses GPS or CDMA timing signal from base station for clock synchronization. FPGA generates the timestamp for each packet, and filters and preprocesses the packets. The memory and processor are embedded in the card to process data. The captured data is then transferred to a PC through a PCI-Bus interface for post-processing and recording. The accuracy of the result if the packets arrival rate is far beyond the DAG processing capacity is unknown. DAG marks the timestamp as soon as the packet arrives, which is faster than flow-based measurement schemes. Therefore, the recorded time is closer to the ideal packet arrival time. Besides of the above two solutions, some other technology as sFlow [67] and RMON [68] also described passive measurement mechanisms.

2.1.3 Combination of Different Measurement Methods

2.1.3.1 End-to-End vs. Per-Link Measurement. From the above introduction, it can be seen that most of the active measurement tools support end-to-end though a few of them estimate the per-link performance. The performance of end-to-end QoS measurement may not be as accurate as the performance of per-link measurement. On the other hand, passive measurement tools mainly collect the QoS information of the local interface.

Traditional QoS routing needs the local state of each router, therefore per-link measurement may be more suitable. Per-link measurement can also be regarded as the case of end-to-end measurement where the path has no intermediate routers.

2.1.3.2 Combination of Active and Passive Measurement. Passive measurements can be less intrusive than the active ones. They provide relatively continuous measurements without any probing intrusion [24, 39]. This can assist ISPs to understand the network performance and analyze the network trends. On the other hand, active probing measurement

is more flexible and easier to be deployed. It can monitor the transitory phenomenon with various QoS measurement options. Based on this point, some proposals to combine active and passive measurements have been proposed in literature.

In [39] an available bandwidth monitoring tool is developed that it passively captures the existing application traffic whenever possible, and triggers active probes when no existing traffic is available. In this way, a constant series of measurement with lower overhead can be achieved.

2.2 Distributed QoS State Measurement

2.2.1 Service Vector

In order to increase the granularity, the QoS service of each router is categorized into various service classes. That is, assume there are n service classes, $S = (S_0, S_1, \ldots, S_{n-1})$ provided by each link in a network and that a data flow passes v routers. Then at router i and j, the user may select service Q_i and Q_j separately, where Q_i and Q_j can be the same or different classes and $Q_i \in S$, $Q_i \in S$. The service selected for each router is represented as a vector as $s = (s_0, s_1, \ldots, s_{v-1}, s_v)$, where s_k is the service selected at router k and $s_k \in S$. This vector is the so called *service vector*. Service vectors are used to find the suitable service classes in a single path so as to maximize:

$$G = max(U - C) \tag{2.3}$$

where U is the utility function and C is the cost. This combination of service vectors decouples the provisioning of end-to-end QoS at each router, thus resulting in an intermediate level of granularity and complexity between per-flow and per-group levels.

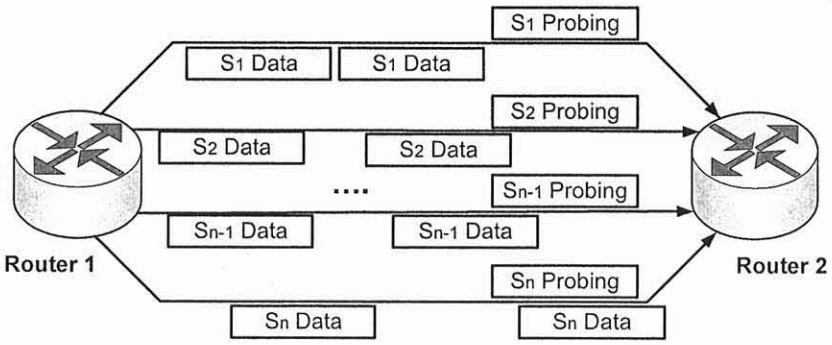

Figure 2.2 Probing packets to test various service classes between routers.

2.2.2 QoS Measurement Architecture

Following this consideration, it is necessary to measure the state of each service class. As Figure 2.2 shows, the framework proposes that one router sends the probing packets with different priority to measure the service class in each link.

As commented in 2.1.3, end-to-end measurement methods may not be suitable for the required QoS granularity. The system needs to measure the QoS state of each link between any neighboring routers, that is, the state of any router in the network (and in each interface). The most demanded QoS parameters are one-way delay, one-way available bandwidth, round-trip delay, and packet loss ratio. Currently, there are many measurement tools to evaluate them. Some of the available ones that can evaluate one-hop QoS state are shown here with the acceptable measurement quality.

Table 2.1 illustrates the summarized selection of the software considered. However, it is not restricted to these tools to measure the QoS state. In addition, other parameters for measurement, such as link capacity and jitter, can be considered.

2.2.3 Experiments Data and Analysis

The implemented measurement system is illustrated by Figure 2.3. Routers R_1 and R_2 are connected through a bidirectional cable with a capacity of 100Mbps. T_1 and T_2 are two terminals that run the selected measurement tools. They are connected to the router with 1

Table 2.1 Selected Measurement Tools for QoS Parameters.

QoS parameter	Measurement tool
One-way delay	OWAMP [62]
Round-trip delay	Ping
Available bandwidth	Pipechar [50] or Pathload [51]
Topology	Traceroute
Bandwidth capacity	Pathchar [56]

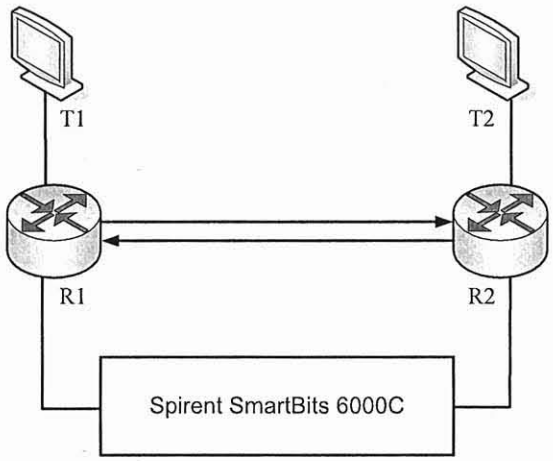

Figure 2.3 Experiment topology.

Gigabit Ethernet cable. A Spirent SmartBits 6000C system (traffic generator) [69] is used to generate the background traffic, also called cross traffic, sent between R_1 and R_2 with the frame size 60 bytes. The experiments include the following three parts:

2.2.3.1 Test of Measurement Tools. Through the SmartWindow application of the Spirent traffic generator, SmartBits generates bidirectional cross traffic between R_1 and R_2 with the traffic loads from 0 to 60% of the link capacity on each direction. Then, the measurement tools Ping and Pipechar are invoked from T_1. The probing traffic and the cross traffic have same priority. The measurement results are shown in Appendix A. It can be seen that the

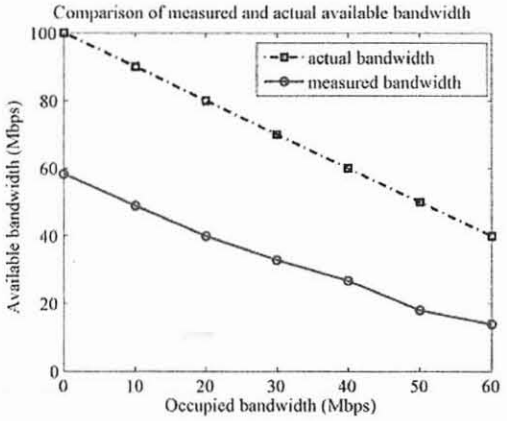

Figure 2.4 Comparison of the available bandwidth measured by Pipechar and the actual value.

average RTT measured from Ping and the average RTT from Pipechar are largely different. This may be caused by the different protocol that each tool uses. However, the measured RTT by SmartBits is close to the result obtained by Ping. Hence, Ping is selected for RTT measurement in future tests.

From the result shown in Appendix A, it can be seen that the available bandwidth measured by Pipechar is lower than the actual value (as measured by SmartBits), as Figure 2.4 shows. However, the difference between them approaches to a constant. This allows people to further find the reason and compensate the difference from the measured data.

During another experiment, SmartBits sends cross traffic with the lowest priority. The traffic is changed in same way as above experiment. Pathload is invoked from both terminal sides (for Pathload, T_2 invokes the software as the end side). In this case, the probing packets are sent with higher priority, so as to simulate that Pathload measures a high service class. The results presented by Pathload approaches to the actual SmartBits values. In the following experiment, the cross traffic shares same service class as Pathload. The obtained results show that Pathload values largely diverge from the SmartBits values once the cross traffic is sent as a burst, or when the gap between the frames has a large variance. Appendix B shows these results. Pathload measures the available bandwidth by increasing the probing-packet sending rate until it finds that the one-way delay of the

Table 2.2 Comparison of Network Consumption by Each Tool.

Measurement Tools	CPU / Memory Cost	Bandwidth Cost	Measurement Time(s)
Ping	very low	very low	< 2
Pipechar	Low	Low	> 20
Pathload	Low	Medium	7

packets is increased. Hence, the received difference is probably due to the variable gap between packets affects the one-way delay measurements in Pathload.

2.2.3.2 Analysis of probing packets. The length and amount of probing packets affects the accuracy of active measurements. It has been suggested that the optimal solution is to use around 60 probing packets with 700 bytes each [70]. In the experiment, while still using the above experiment topology, SmartBits generated bidirectional cross traffic with loads from 0% to 50% through the SmartWindow application, and at the same time SmartFlow application was used to send probing packets for one-way delay test. Compared with the measurement result obtained by SmartWindow, the probing packets with size of 64 bytes can achieve more accurate results than that obtained by using large size probing packets, independently of the number of packets are used.

2.2.3.3 Consumption of measurement tools. From the above experiments, the resource consumption by each tool is obtained, as Table 2.2 describes.

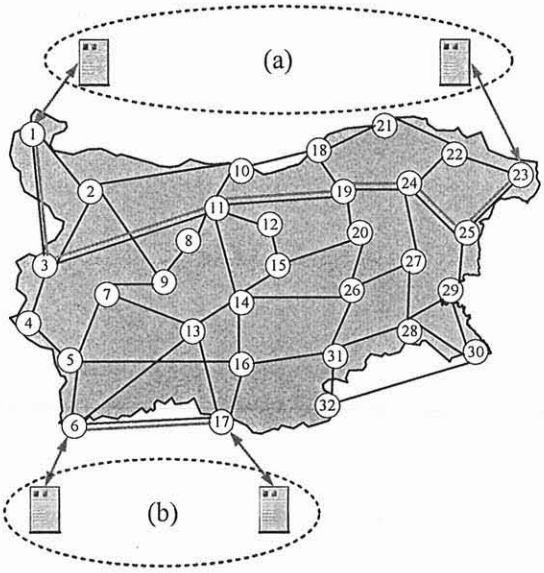

Figure 2.5 Network measurement implementation topology.

2.3 Task-Execution Scheduling Schemes for Network Measurement and Monitoring

Table 2.2 shows that the measurement tools potentially contend for network resources. Active measurements are launched from a specific measurement server (measurement point) connected to a router in the network to measure the end-to-end performance as shown in Figure 2.5.a. A few measurement tasks can also be implemented at routers, such as the Ping application, so that routers may measure the link state between any two neighbors, as illustrated in Figure 2.5.b. Without loss of generality, Figure 2.6 shows a measurement infrastructure designed by Internet2 E2E piPEs projects [71].

As discussed above, the measurement tasks need to be executed periodically. In each cycle, a measurement task $J^i_{x \to y}$ is denoted as one measurement process executed by the i_{th} measurement tool sending probing packets from measurement point x to point y. The same measurement task is processed periodically.

Independently of the measurement approach used, probing overhead is a general concern for active measurement mechanisms as it may affect the user traffic. For example, an active measurement experiment [70] showed that a 700-Byte packet size used in 60-packet

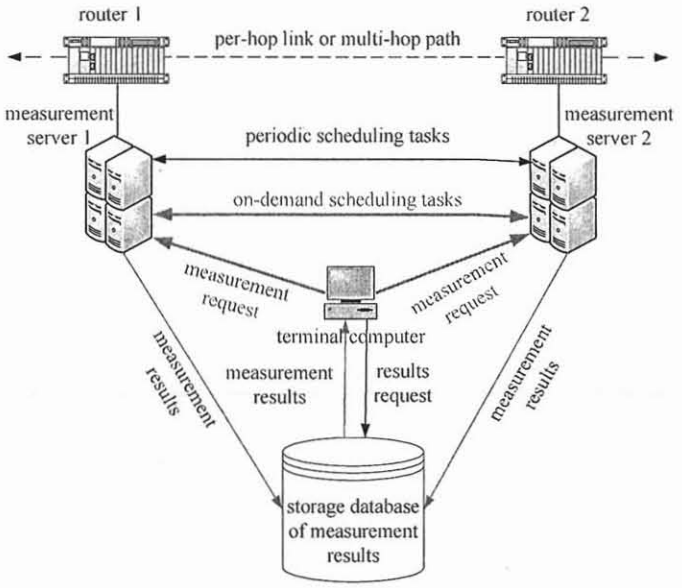

Figure 2.6 An example of network measurement infrastructure.

probing trains can achieve sufficiently accurate results of available bandwidth measurement per path on the Internet. In this case, one path overhead is about 42KB, and so measuring all end-to-end paths in a 200-nodes bidirectional mesh system requires about 1.7GB for just one snapshot if all network links are simultaneously tested. Therefore, the network resources need to be efficiently managed under active probing.

In addition, distributed measurement tasks may be executed simultaneously at one measurement point in a network. Hence, it is possible that different measurement tasks contend for network resources, including transmission channels and bandwidth. Measurement processes that are executed in different common points also contend for resources, such as processing time, bandwidth, and memory. The accuracy of some measurement processes may be affected by other measurement processes run concurrently. This contention for resources is called measurement conflict problem. To gain insight of the implications of contention for resources, Pipechar, Pathload, and Ping are executed in a host, all at the same time, to measure several parameters in the transmission from one host to another through a 100-Mbps fast Ethernet link [46]. It is observed that the measurement resources and

measurement processing time had a large discrepancy among those measurement tools, as shown in Table 2.2, and the obtained measurement results are instable because of the disturbance from other measurement processing. Sommers and Barford [72] also implemented a testbed through which the experiment results show that the measurements of packet loss and delay from active probes can be skewed significantly due to the contention of probing packets. Thus, the active measurement tasks that are performed at each measurement point need to be scheduled to avoid both potential resource contention and measurement disturbance from each other while achieving a satisfactory measurement in terms of time and accuracy.

To solve the above problem, a solution is designed to schedule the periodic and on-demand measurement tasks to achieve the following four goals:

1. Avoid conflicts among concurrent executed measurement tasks.
2. Network resources are not exhausted by measurement tasks.
3. Shorten the waiting time of each measurement task for the execution.
4. Shorten the total completion time of measurement tasks set, that is, improve the resource utilization.

To comply with the above requirements, this dissertation proposes an algorithm to schedule periodic tasks and to improve the measurement efficiency. It also proposes an algorithm to schedule on-demand measurement tasks that minimize the delay of both periodic tasks and the incoming on-demand tasks. Both algorithms are based on graph coloring theory, where each measurement task is treated as a vertex in a graph, and the contention/conflict by two tasks is represented as an edge connecting those two vertices.

2.3.1 Problem Analysis

According to the classification approach of scheduling introduced by Graham *et al.* [73], the task scheduling problem is defined in terms of a three-tuple classifications $[\alpha, \beta, \gamma]$,

where α defines the machine (processor) environment, β specifies the job's characteristics, and γ denotes the optimality criterion. Following this classification method, the measurement scheduling problem can be described as $[P, \{rec, r_i\}, \sum C_i]$. Here, P is the number of identical parallel processors to perform the required jobs. However, different from that approach [73], P is a variable instead. The value of P depends on the number of measurement tasks run simultaneously. Considering that n tasks need to be processed, the following relationship exists:

$$P \leq n \tag{2.4}$$

rec refers to the constraints on the resources used by the execution of measurement tasks. In order to minimize or to avoid the impact of probing packets on the performance of regular data traffic, a network resource constraint, such as the maximum bandwidth, is set at each measurement point. This is called measurement resource constraint (MRC) in this dissertation. Scheduling measurement tasks need to ensure that the total amount of resources consumed by the measurement tasks are within this constraint rec. Measurement task i is denoted as τ_i in the remainder of this chapter. The parameter r_i denotes the release time of a measurement task τ_i, upon which one instance of the task τ_i becomes available for processing or execution. $\sum C_i$ indicates that the optimal criterion chosen is to minimize the total completion time on P parallel processors, where C_i denotes the completion time of the measurement task τ_i. This optimal criterion reflects the fourth goal listed in this section. It is easy to see that the third goal is the sufficient and necessary condition of the fourth goal, as described by Lemma 2.1. Therefore, $\sum C_i$ can cover both the third and fourth goals.

Lemma 2.1. *Minimizing the total completion time of a set of measurement tasks is equivalent to minimizing the average waiting time of the measurement tasks in this set.*

Proof. For a measurement tasks set, the completion time of task τ_i is:

$$C_i = e_i + w_i \tag{2.5}$$

where e_i is the execution time of measurement task τ_i and w_i is the waiting time of task τ_i. Hence, the total completion time of the measurement tasks set is:

$$\begin{aligned} \sum C_i &= \sum e_i + \sum w_i \\ &= \sum e_i + m \times w_{avg} \end{aligned} \tag{2.6}$$

where m is the number of measurement tasks in the set and w_{avg} is the average waiting time of the tasks. Since the execution time of each measurement task is a constant, the sum of the execution time $\sum e_i$ is a constant too. According to Equation 2.6, minimizing $\sum C_i$ is equal to minimizing $m \times w_{avg}$, and thus is equal to minimizing w_{avg}. □

A scheduling algorithm can be further classified as preemptive or non-preemptive. In preemptive scheduling, the execution of a task can be interrupted prior to completion and resumed later. On the other hand, in non-preemptive scheduling, a task must be executed to completion once execution has started. In general, measurement task scheduling is regarded as non-preemptive scheduling as the measurement results are expected at completion and the measurement results may be time sensitive. Another issue with this problem that differentiates it from the others is the potential conflict that measurement tasks have with each other. This characteristic increases the complexity of the scheduling scheme because the tasks cannot be just sorted according to one parameter (e.g., deadline or execution time of the task), but also the conflict with scheduled tasks has to be considered.

2.3.2 Related Work

Round robin is one of the simplest scheduling schemes [48, 74, 75] where the tasks are executed by a fixed order in uni-processor systems and only one task is executed at a time. This scheme requires the longest processing time for measurement tasks as it does not admit concurrent execution.

Network Weather Service (NWS), a well-known network measurement infrastructure, adapts a token passing scheme [76] to ensure mutual exclusion between measurement

tasks. In this scheme, the measurement point that receives a token is entitled to execute a measurement task. Afterwards, the measurement point releases the token to a successor. However, this method does not allow concurrent execution of measurements.

Deadline driven scheduling (DSS), also known in the literature as the Earliest Deadline First (EDF) scheduling scheme [77], selects tasks based on their deadlines, and was originally defined for uni-processor execution.

It is shown that the problem of determining whether a given periodic task system is non-preemptively feasible on either a single processor or multiprocessors is NP-hard in a strong sense [78], [79]. To provide network measurement scheduling, a scheduling algorithm based on EDF that allows multiple concurrent executions, referred to as EDF-CE [80], was recently proposed. This approach initializes a queue that stacks all pending tasks to be processed in an EDF order, where the deadline is defined as the time before the task must be executed again. Whenever a task is ready to be released or a task finishes execution, the available tasks in the queue are scheduled. This method introduces the possibility of overlapping multiple tasks in some time slots, but it does not consider the utilization ratio; in other words, sorting the tasks in the pending queue with their deadlines ignores the fact that the concurrent execution of multiple tasks greatly depends on the existing conflicts between the tasks as much as on the tasks' deadlines.

2.3.3 Modeling of Network Measurement Scheduling Schemes

2.3.3.1 Definitions. Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ represent the measurement tasks set with up to n measurement tasks to be executed in the network. Here, τ_i is characterized by a three-tuple of parameters:

- $a(\tau_i)$: the time the measurement task is released, which is the task's arrival time.
- $e(\tau_i)$: the execution time required by a measurement task to complete the measurement.

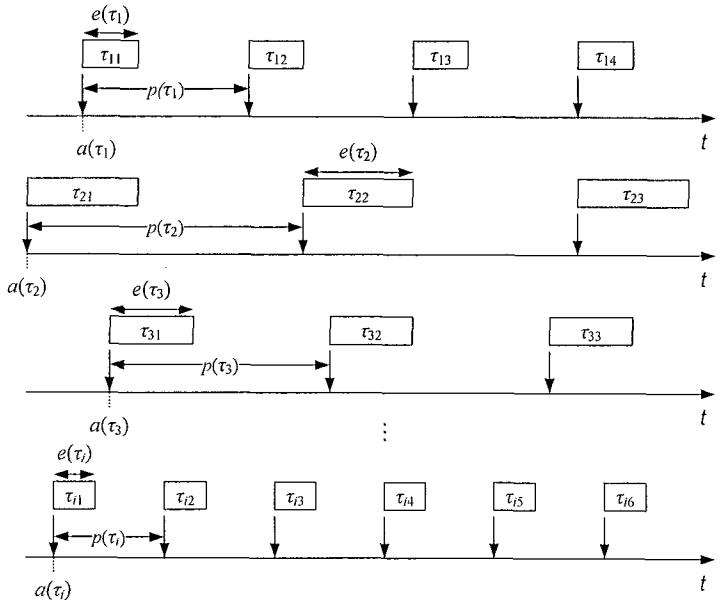

Figure 2.7 Illustration of network measurement tasks.

- $p\,(\tau_i)$: the period of the measurement task, or the time to execute task τ_i after the previous instance. This parameter describes how often a measurement task is executed.

A timetable of periodic measurements is constructed by sequences of tasks, each of which is executed again in $p\,(\tau_i)$ units of time, and each task requires execution of $e\,(\tau_i)$ time units. The j_{th} job (or repetition) of measurement task τ_i is denoted as τ_{ij}. Thus, the first job, τ_{i1}, of measurement task τ_i occurs at time $a\,(\tau_i)$; consecutive jobs generated by τ_i occur exactly $p\,(\tau_i)$ time units apart. Figure 2.7 illustrates an example delineating the terms defined above.

In a set of periodic tasks where the tasks (and the number of them) do not change and where each task can have any particular period, the combination of tasks' release times is finite. This is, after a long period of time, because of the task periodicity, the combination of release times repeats again. Therefore, for the measurement set τ, the term hyperperiod p_h is defined to be the period of time where all tasks in the set occur at different times and without replication of the combination of release times. That is, all periodic tasks in one hyperperiod are able to follow the same schedule as used in the previous hyperperiod.

The hyperperiod is defined as the least common multiple of the periods of all measurement tasks in τ.

$$p_h = lcm\left(p\left(\tau_1\right), p\left(\tau_2\right), \ldots, p\left(\tau_n\right)\right) \tag{2.7}$$

Without loss of generality, the execution time $e\left(\tau_i\right)$, initial available time $a\left(\tau_i\right)$, and the period $p\left(\tau_i\right)$ are defined as integer multiples of a time unit which is referred to as a time slot. The deadline of each job $d\left(\tau_{ij}\right)$ coincides with the period, that is, the job τ_{ij} should be completed before the next job $\tau_{i(j+1)}$ is available to be executed. According to this definition, Lemma 2.2 can be readily obtained:

Lemma 2.2. *Given a measurement tasks set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$, at any time instance, there is at most one job available to be executed for any measurement task $\tau_i \in \tau$.*

Proof. At any time instance, there must be a job available for execution at the beginning of that period. If there are some jobs generated from previous periods still pending for execution, those postponed jobs passed their own deadlines and they are considered as missed jobs. Hence, there is at most one job for each measurement task at any time-instance. □

2.3.3.2 Modeling of Measurement Scheduling. The proposed scheduling algorithms are based on graph theory. In the literature, there are some articles using graph coloring to solve time slots assignment problem [81–83], but most of them are designed for single processing, which are not fit for multi-task processing such as the network measurement scenario.

Consider a measurement tasks set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ to be executed in a network. Each measurement task can be represented as a node ($\in V$) in a graph and any two measurement tasks are connected by a link ($\in E$) if they are to be executed with mutual exclusion on the measurement point or channel. These tasks are said to be adjacent to each other. The graph $G(V, E)$ that describes these nodes and links is called a conflict graph. Figure

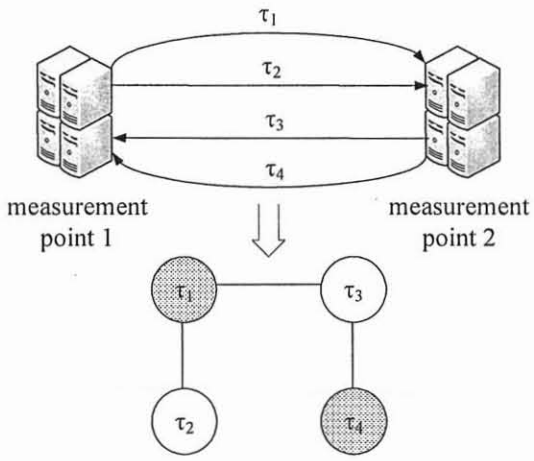

Figure 2.8 Illustration of the relationship between measurement tasks by a conflict graph.

2.8 illustrates an example of a conflict graph where two measurement tasks are to be executed between measurement points 1 and 2 in a full-duplex connection. Assume that task τ_1 contends with τ_2 for the available memory at measurement point 1, and at the same time, it contends for the transmission channel with τ_3. Task τ_3 also contends with τ_4 for available memory at measurement point 2. Therefore, these four tasks comprise a conflict graph with three links. In this example, measurement tasks τ_1 and τ_4 (represented by shaded nodes), or τ_2 and τ_3 (represented by unshaded nodes) can be concurrently executed.

In the considered network, there is a central controller to compute the schedule of all measurement tasks and to send out the schedule information to each measurement point. This central management mode is feasible and adopted in real network measurement frameworks. Scheduling is requested each time when a new job is available for execution and when a job execution has been completed. These time instances are named as scheduling points. There is a waiting queue to store the jobs available for execution. At each scheduling point, jobs stored in the waiting queue become eligible candidates for the scheduler. Based on the conflict relationship between the measurement tasks, these jobs that belong to different measurement tasks construct a conflict graph at the job level. The conflict relationship between jobs follows the same conflict relationship between measurement tasks. For

periodic tasks, the conflict relationship among them is known prior to performing scheduling because the submitted tasks and the amount of resources they consume are both known in advance. For on-demand tasks, the attributes of measurement tools are *a priori* so the tasks' conflicts are known once an on-demand task emerges.

As the measurement results obtained by earlier periodic measurement tasks are used to describe the current network performance, it is desired that the measurement tasks can be completed as soon as possible after a task is available for execution. Therefore, the scheduling problem is converted into a process to schedule the available jobs at each scheduling point so as to minimize the job waiting time for execution. At the same time, the scheduling of measurement jobs at one scheduling point is enunciated as the arrangement of the vertices of graph G at the job level such that none of the nodes connected with each other are scheduled for simultaneous execution. This process can be described as a *vertex coloring problem* as follows.

Scheduling of Measurement Tasks: *Given a conflict graph $G(V, E)$ with vertices $V = V(G)$, assign each vertex a color out of the range $[1, 2, \ldots, k]$ such that no two adjacent vertices have the same color.*

Here, each color maps to one time slot. The color set to be used by a vertex v_{ij} in the conflict graph is mapped to the time range $[t_c, d(\tau_{ij})]$ as described by Equation 2.8, where t_c is the current scheduling point and $d(\tau_{ij})$ is the deadline of the job mapped by vertex v_{ij}. That is, the scheduler only considers the time slots prior to a job's deadline.

$$[1, 2, \ldots, k] \rightarrow [t_c, d(\tau_{ij})] \tag{2.8}$$

Each measurement point is considered to have limited processing and storage (memory) capabilities, and each channel to have a limited bandwidth capacity. Therefore, the load of intrusive probing packets in active measurement needs to be restricted within a range, so as to minimize the disturbance of the measurement of the existing data traffic, as described by MRC values. A consumption matrix is proposed to describe such con-

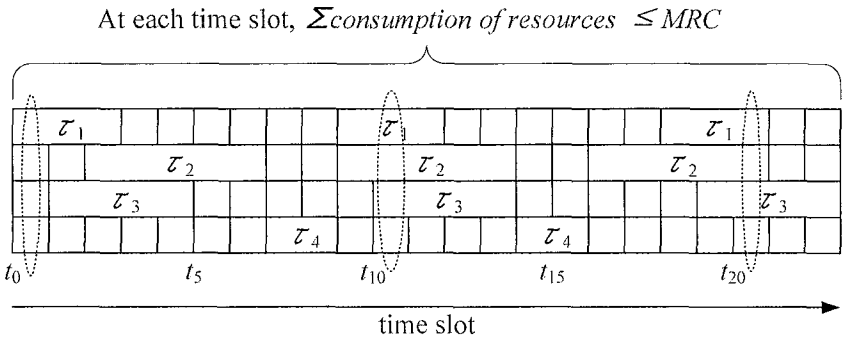

Figure 2.9 Consumption matrix.

straints. Denote the number of schedule slots and the number of the measurement jobs as the column and row of a matrix as shown in Figure 2.9. The resource utilization objective can be described as follows:

Resource Utilization of Measurement Tasks: *Jobs of measurement tasks set* $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ *with execution times* $e(\tau_1), e(\tau_2), \ldots, e(\tau_n)$, *can be represented as a* $p_h \times n$ *consumption matrix* A, *where a row indicates the task and its duration in time slots and the column indicates the time slot. The maximum number of rows is bounded by the amount of processing resources constrained by Equation 2.4. Each column as circled in Figure 2.9 represents the consumption of network resources at that particular time slot.*

The objective is to place the measurement tasks in the consumption matrix such that $\sum_{j=1}^{n} A_{ij} \leq MRC, \forall i \in [1, 2, \ldots, p_h]$, where p_h is the hyperperiod duration, i.e., the total consumption of resources by measurement tasks per time slot is within the measurement resource constraint.

2.3.4 Proposed Scheduling Schemes

This section introduces the proposed scheduling schemes for periodic and on-demand measurement tasks. The following definitions are used in the description of the proposed schemes.

- Clique: a maximal set of adjacent vertices of graph G.

- Clique number: the number of vertices in the largest clique of G, denoted as $\omega(G)$.
- Degree: degree of vertex v in graph G is the number of adjacent vertices of v in G, denoted as $d_G(v)$; the maximum degree of graph G is the largest number of $d_G(v)$, and it is denoted as $\Delta(G)$.

2.3.4.1 Periodic Measurement Tasks Scheduling Scheme. Following the model of the scheduling problem described above, the proposed algorithms consider the jobs stored in the waiting queue for scheduling at each scheduling point. If a job can be scheduled in the time range $[current\ scheduling\ point, deadline\ of\ job]$ without any conflict with already scheduled jobs at any given time slot, this job is removed from the waiting queue and the corresponding time slots for execution are marked in the consumption matrix; otherwise, the job is kept in the waiting queue and waits for consideration at the next scheduling point. Hence, the goal is to find a feasible scheme to schedule the maximum number of concurrent jobs at each scheduling point, so that the most time space in the consumption matrix can be utilized.

Consider available jobs in the waiting queue. Since their execution times are integer multiples of a time slot and the time slot can be mapped to a vertex, each task can be divided into a set of sub-vertices as follows:

In a conflict graph $G(V, E)$ at the job level, each vertex v_{ij} that maps job τ_{ij} has a set of sub-vertices $(\tau_{ij}^1, \tau_{ij}^2, \ldots, \tau_{ij}^\alpha)$, where α is the length of $e(\tau_{ij})$ in time slots.

As the sub-vertices of v_{ij} represent the different but consecutive time slots of a task, they are said to contend with each other (or to have a conflict with each other). These conflicts can be described by a complete sub-graph G_{ij} as in the example shown in Figure 2.10. Conflict graph G is further represented by its sub-vertices and it is denoted as $G^s\ (V^s, E^s)$. The clique number of a sub-graph G_{ij} is equal to the number of vertices in G_{ij}. Here, G^s is the graph constructed by sub-vertices.

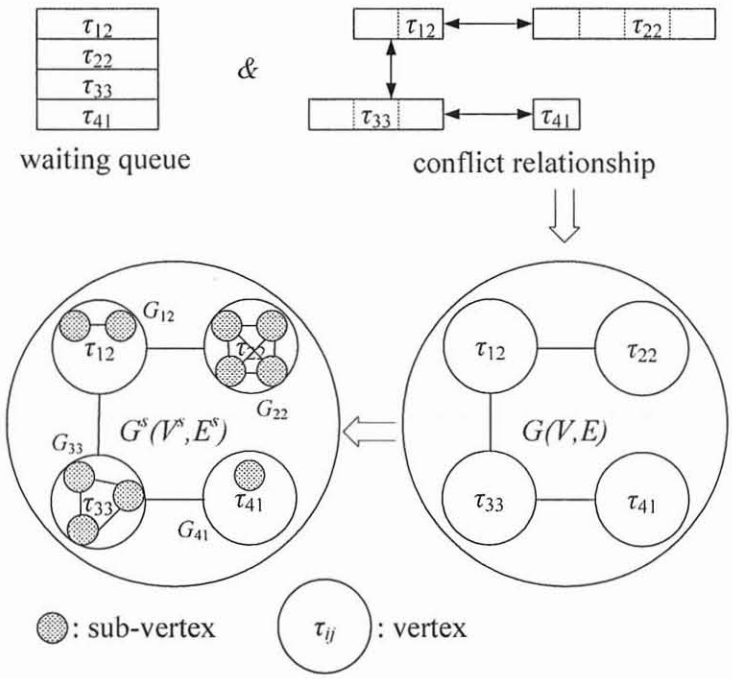

Figure 2.10 Example of sub-graph.

As each color represents one time slot, each sub-vertex in graph G^s is a candidate for a color assignment, so that any two adjacent sub-vertices must not possess the same color. Each sub-vertex is restricted to allowed colors that satisfy the relationship denoted by Equation 2.8. This is called the list coloring problem. To solve this problem, the sub-vertices are sorted in the ascending order of their degree in graph G^s:

$$\forall v_{ij}^l \in G_{ij}\ d_G^s(v_{ij}^l) = d_G(v) + \omega(G_{ij}) = d_G(v) + e(\tau_{ij}) \tag{2.9}$$

The rationale to schedule jobs in this fashion is the expectation that a sub-vertex with a small degree has a few conflicts; therefore, a large number of tasks might be scheduled at the same time. In a network with a measurement scheduling environment, this can be described by two aspects. For a sub-vertex v_{ij}^l and its adjacent sub-vertex v_x:

- v_{ij}^l and v_x map to the same job: Then, the low degree implies the job has a short execution time. This part is represented as the execution time of the vertex $e(\tau_{ij})$,

or by the clique number of the sub-graph $\omega(G_{ij})$. Scheduling a job with a short execution time will leave more available time slots for other jobs in the waiting queue.

- v_{ij}^l and v_x map to different job: Then, the low degree of the sub-vertex indicates the job might have few conflicts with other available jobs in the waiting queue. Scheduling a job with few conflicts allows additional jobs to be executed concurrently, thus increasing the resource utilization.

The scheduling procedure is described below:

- Step 1. At current scheduling point t_c, check if there is a new job available for execution. If so, the new job is placed in the waiting queue.

- Step 2. Map the candidate jobs in the waiting queue to a conflict graph G and convert G into sub-graph G^s.

- Step 3. Sort the sub-vertices in the ascending order of their degree, as described by Equation 2.9.

- Step 4. Schedule the first job as indicated by the sorted sequence. Any sub-vertex v_{ij}^l selected to be scheduled will be colored with other sub-vertices belonging to the same job τ_{ij} with consecutive colors. The used colors are the intersection set as $\overline{colors_{in-conflict}} \cap [t_c, d(\tau_{ij})]$ where $[t_c, d(\tau_{ij})]$ is the time interval from t_c to $d(\tau_{ij})$, $colors_{in-conflict}$ is the set of available colors possessed by the on-going conflict jobs, and $\overline{colors_{in-conflict}}$ is the complementary set of $colors_{in-conflict}$, i.e., the available colors that can be used by v_{ij}^l.

- Step 5. Check if the colored job and other on-going jobs violate the resource constraint MRC. If there is no violation, remove the colored job from the waiting queue, remove the corresponding sub-vertices from the sorted sequence, and add the completion time of the job to the scheduling point list.

- Step 6. Color the next sub-vertex in the sorted sequence. Repeat Steps 4 to 5.
- Step 7. Go to the next scheduling point. Repeat Steps 1 to 6.

The algorithm of periodic measurement-tasks scheduling is described by the pseudo code in Figure 2.11.

2.3.4.2 On-Demand Measurement Tasks Scheduling Scheme. During the execution of the periodic measurement, a network administrator may request sporadic on-demand measurement tasks to test specific network performance parameters at a particular time. Furthermore, on-demand tasks might conflict with some periodic or on-demand tasks. Each on-demand task has also defined execution and deadline times, and it is considered with either a priority higher than or equal to that of the scheduled periodic tasks. The proposed scheduling scheme for on-demand measurement tasks is able to handle both of these two cases adaptively. The goal of scheduling on-demand tasks with higher priority is to execute the on-demand tasks as soon as possible while minimizing the latency of the periodic tasks caused by the insertion of on-demand tasks. On the other hand, scheduling on-demand measurement tasks with the same priority as periodic tasks aims to shorten the average waiting time for all measurement tasks including on-demand and periodic tasks.

The proposed method schedules all the tasks with higher priority first, and then schedules the remaining on-demand and periodic tasks according to the ascending order of the degree of sub-vertices, as explained below:

- Step 1. When a new on-demand task arrives at t_c, check the priority type of the on-demand task. If its priority is high, store this on-demand task to the waiting queue of high priority tasks Q_{high}. If the priority is equal to that of the periodic tasks, the on-demand task is stored to $Q_{regular}$.
- Step 2. Schedule all the candidate jobs in the waiting queue of high priority tasks Q_{high}. In the pre-computed schedule, all the jobs of periodic tasks that finish their

Periodic Tasks Scheduling Algorithm

Input: measurement task set τ 3-tuple parameters of tasks $a(\tau_i), e(\tau_i), p(\tau_i)$

tasks conflict matrix F measurement resource constraint MRC

Output: start time of jobs T

Initialize: hyperperiod p_h, scheduling point list S, waiting queue $Q = \{\}$

while S is not empty

if new available job $\tau_{new} \in Q$

remove the old instance of τ_{new} due to expiration

end

$Q = Q + \tau_{new}$

set up conflict graph with sub-vertices $f : (F, Q) \rightarrow G^s(V^s, E^s)$

sort Q by ascending order of degree of sub-vertices $d_{G^S}(v)$

for each job τ_{ij} in sorted Q

find the available colors range R

if consecutive sequence $L = e(\tau_{ij})$ exists in R & consumption of τ_{ij} and on-going jobs satisfies MCR

assign the first long enough consecutive colors L to τ_{ij}

$Q = Q - \tau_{ij}$, record start time of τ_{ij} in T

update scheduling point list:

$S = S + completion\ time\ of\ \tau_{ij}$

end

end

Go to next scheduling point

end

Figure 2.11 Pseudo code of scheduling algorithm for periodic measurement tasks.

execution before t_c and the jobs that are still being executed at time t_c are discarded/cancelled. The jobs that start processing after t_c are considered as rescheduled. Follow Steps 2 to 6 of the previous scheduling procedure for periodic tasks. Note that the scheduling points are updated so the completion time of the scheduled jobs in Q_{high} are added into the scheduling points list. After this step, all the possible jobs in Q_{high} must be either scheduled or expired because there are no available time slots to be scheduled before the job's deadline.

- Step 3. Add those jobs that start processing after t_c in the pre-computed schedule to the waiting queue of regular priority tasks $Q_{regular}$. Schedule all candidate jobs in $Q_{regular}$ following the previous scheduling procedure for periodic tasks.

Figure 2.12 shows an example to illustrate this scheduling procedure. In this example, the on-demand task τ_{od} conflicts with periodic tasks τ_1 and τ_3, as shown in Figure 2.12.a. If the priority of τ_{od} is higher than that of other periodic tasks, then when it arrives at t_c, all periodic jobs that start the execution after t_c are stored in $Q_{regular}$ while τ_{od} is stored in Q_{high}. Thus, τ_{od} is the first to obtain a schedule. As shown in Figure 2.12.b, τ_{od} is first scheduled and only the schedule of job τ_{32} is changed. If τ_{od} has same priority as other periodic task, then τ_{od} and all periodic jobs that start the execution after t_c are stored in $Q_{regular}$ and sorted in the ascending order of sub-vertices' degree. As shown in Figure 2.12.c, τ_{od} is scheduled with longer waiting time than in Figure 2.12.b, but rescheduling for other periodic jobs is unnecessary.

The algorithm of on-demand measurement tasks scheduling is described by the pseudo code shown in Figure 2.13.

2.3.4.3 Computational Complexity Analysis. According to Lemma 2.2, there are at most n jobs in the waiting queue if there are n tasks in the measurement tasks set. Using a simple sorting algorithm such as binary tree sort, the computational complexity of sorting n jobs is $n \lg(n)$. In one hyperperiod, assume there are m scheduling points which

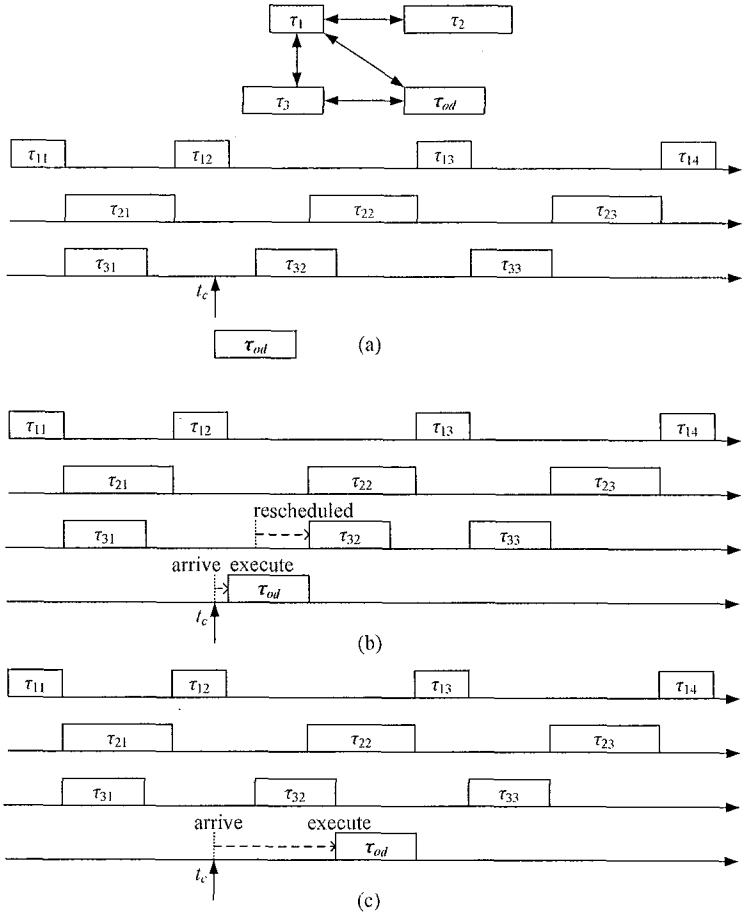

Figure 2.12 Example of scheduling on-demand measurement task: (a) pre-computed schedule; (b) on-demand task has higher priority; (c) on-demand task has same priority as periodic tasks.

On-demand Tasks Scheduling Algorithm

Input: measurement task set τ

3-tuple parameters of tasks $a(\tau_i)$, $e(\tau_i)$, $p(\tau_i)$

tasks conflict matrix F

on-demand task τ_{od}

measurement resource constraint MRC

pre-computed schedule T_o

Output: start time of jobs T

Initialize hyperperiod p_h

Initialize scheduling point list S

Initialize waiting queue $Q_{high} = \{\}, Q_{regular} = \{\}$

if priority of τ_{od} is high

$Q_{high} = Q_{high} + \tau_{od}$

elseif priority of τ_{od} is regular

$Q_{regular} = Q_{regular} + \tau_{od}$

end

set up conflict graph with sub-vertices:

$f : (F, Q_{high}) \rightarrow G^s_{high}(V^s_{high}, E^s_{high})$

sort Q by ascending order of degree of sub-vertices $d_{G^S_{high}}(v)$

schedule each job in Q_{high} and update scheduling point list

set up conflict graph with sub-vertices:

$f : (F, Q_{regular}) \rightarrow G^s_{regular}(V^s_{regular}, E^s_{regular})$

sort Q by ascending order of degree of sub-vertices $d_{G^S_{regular}}(v)$

schedule each job in $Q_{regular}$ and update scheduling point list

Figure 2.13 Pseudo code of scheduling algorithm for on-demand measurement tasks.

indicate the time jobs arrive, then the computational complexity of the proposed algorithm is $mn\lg(n)$. Denote C as the number of unique completion times of all jobs and K as the total number of jobs to be executed in a hyperperiod. Then the following relationship exists:

$$K = \sum_{i=1}^{n} \frac{p_h}{p(\tau_i)}$$

$$m \leq \sum_{i=1}^{n} \frac{p_h}{p(\tau_i)} + C \leq 2 \sum_{i=1}^{n} \frac{p_h}{p(\tau_i)}$$

Therefore, the computational complexity of the proposed algorithm is $n\lg(n)\sum_{i=1}^{n}\frac{p_h}{p(\tau_i)}$, and thus the complexity can be decreased by limiting the upper-bound of p_h. Some previously proposed methods aimed to achieve this goal [84], but this is out of scope of this dissertation.

2.3.5 Simulation Results

The proposed algorithms are compared them with other scheduling algorithms for the performance study.

2.3.5.1 Schemes for Comparison. The algorithms should be able to process multiple measurement tasks at the same time for a fare comparison to the proposed algorithms for their execution on an infrastructure with sufficient resources. All of these algorithms have the same computational complexity as the proposed ones. These algorithms are described next:

- *Round-Robin* The original round robin scheme is improved here to empower it with the concurrent execution capability. The improved scheme selects tasks for execution by following a pre-defined order. The scheme performs scheduling at each scheduling point. At a scheduling point, all the available jobs waiting to be scheduled are

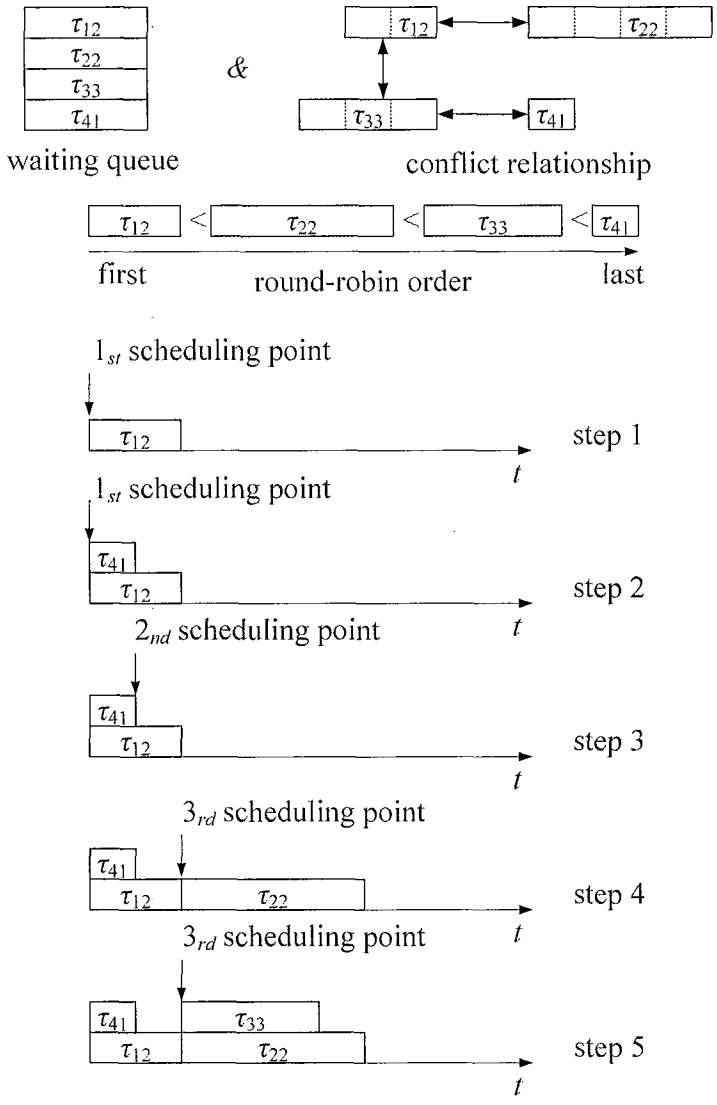

Figure 2.14 Illustration of the improved round robin scheduling algorithm.

selected in a pre-defined round-robin order. If there is no conflict with current on-going task, the job is scheduled; otherwise, the job is kept in the queue to be considered/scheduled at the next scheduling point. This algorithm is described in Figure 2.14.

- *Descending Order of Sub-Vertices' Degree (DOSD)* This scheme, also introduced here for comparison purposes, follows a similar procedure as described in Section 2.3.4.1 for the ascending order version, except that this scheme sorts the jobs in the waiting queue in the descending order of the degree of the sub-vertices mapped to the jobs, in Step 3.

2.3.5.2 Evaluation Method. The algorithms are compared in terms of the average normalized waiting time of all jobs in one hyperperiod that is defined as below:

$$Avg.\ normalized\ waiting\ time = avg(\sum \frac{w(\tau_{ij})}{p(\tau_{ij})})$$

where $w(\tau_{ij})$ is the waiting time of the job τ_{ij}. $w(\tau_{ij})$ is formally defined as the difference between the time that the job starts execution and the beginning time the job is available to be executed. In the worst case, some measurement jobs may be missed due to time expiration (i.e., the waiting time exceeds the task period). The waiting time of the missed job is defined to be equal to its period time.

As the network performance is monitored by periodic measurement requests, the measurement jobs are expected to be scheduled at desired sampling times that the interval time between any two consecutive samplings is a constant. However, because of the conflict of the network measurement tasks, the measurement jobs are scheduled at the time deviated from the desired sampling times. The average normalized waiting time is used to reflect how severe such deviation impacts the acceptance of the measurement sampling results. For example, if a measurement task with period equal to 20 minutes waits for 1.5 minute to start execution, the measurement result is still acceptable to be used as periodic samples.

However, if a measurement task with period 2 minutes waits for 1.5 minute for execution, the measurement sample obtained is far from the expected measurement sampling time.

Another evaluation parameter is the execution success ratio of jobs to be executed, which is defined as:

$$Execution\ success\ ratio = \frac{number\ of\ executed\ jobs\ in\ one\ hyperperiod}{number\ of\ total\ jobs\ in\ one\ hyperperiod}$$

2.3.5.3 Simulation Results of Periodic Tasks Scheduling. In this simulation, the period of the periodic measurement tasks is uniformly distributed in the range of [11,100] time units, and the execution times of the periodic measurement tasks are uniformly distributed in the range of [2,10] time units. The initial time of task $a\ (\tau_i)$ is randomly selected in the range of [1,5] time units. The conflict probability value is increased from 0 to 1.0 with increments of 0.05. A conflict probability of 0 between two tasks means that there is no conflict between them, therefore there is no edge connecting these two vertices in the conflict graph. A conflict probability of 1.0 means that there is a conflict between any two tasks, which corresponds to a fully connected conflict graph. There might be a high conflict probability in a real network where the ongoing measurement tasks demand network resources for exclusive use. As an example, the simultaneous measurement of bandwidth, delay, jitter and other parameters at a gateway in a small network could be a network performance bottleneck as all measurement tools contend for the memory, processing time, and uplink/downlink bandwidth of that gateway. To observe the maximum performance of the scheduling schemes, the measurement resource is assumed to be large enough so there is no MRC constraint on measurement tasks. The performance of the algorithms is compared in 10 and 20 periodic tasks scenario. The simulation is run 1000 times (i.e., for each time a random tasks set and the conflict relationship are generated) for each scenario.

Figure 2.15 shows the average normalized waiting times of 10 periodic tasks for these schemes. The figure shows that the proposed scheme has the lowest average normalized waiting times, and EDF-CE has the highest.

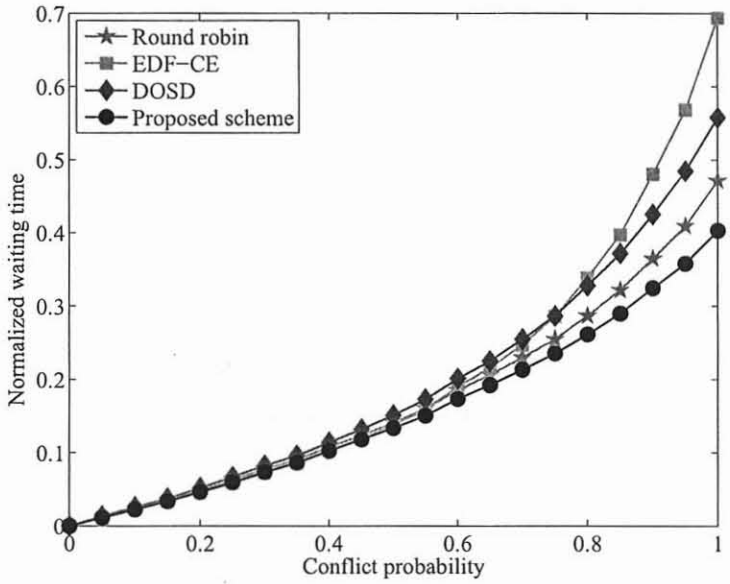

Figure 2.15 Normalized waiting time for 10 periodic measurement tasks.

Figure 2.16 shows the success ratio of 10 periodic tasks of the compared schemes. The figure shows that as the conflict probability increases, the success ratio of the schemes decreases. Here, the success ratio of the proposed scheme is the highest among other schemes as this scheme misses scheduling the fewest number of tasks as compared to the other schemes, while DOSD, which sorts the task in the opposite order, has the lowest success ratio. The combination that increases the success ratio seems to be the selection of a small task and with a small number of conflicts.

Figure 2.17 shows the normalized waiting times of these schemes with 20 tasks. The outcome for 20 tasks is similar to the case with 10 tasks, where the proposed scheme achieves the lowest waiting time. The advantage of using the proposed scheme is more pronounced for scenarios with a larger number of tasks.

Figure 2.18 shows that the proposed scheduling scheme and the EDF-CE scheme provide similar execution success ratio, which is the highest success ratio as compared to round robin and DOSD schemes. It can be seen that when the conflict probability is lower than 0.5, the performance of all algorithms is similar, but as the conflict probability

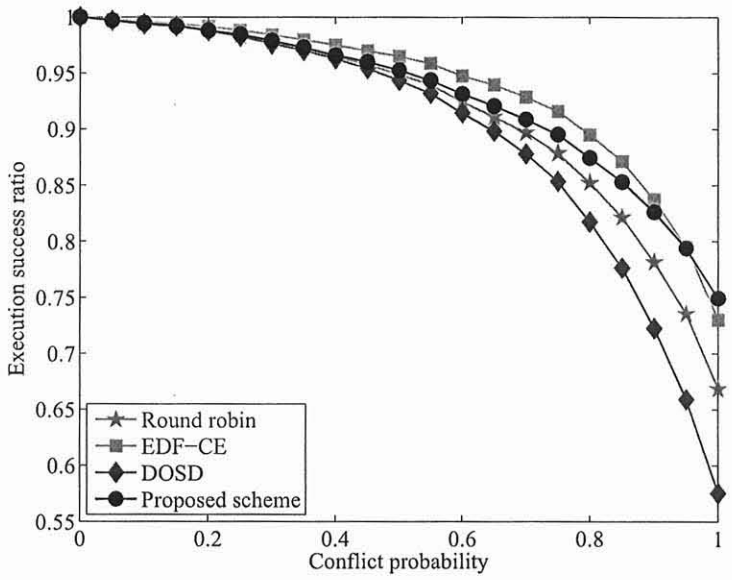

Figure 2.16 Execution success ratio for 10 periodic measurement tasks.

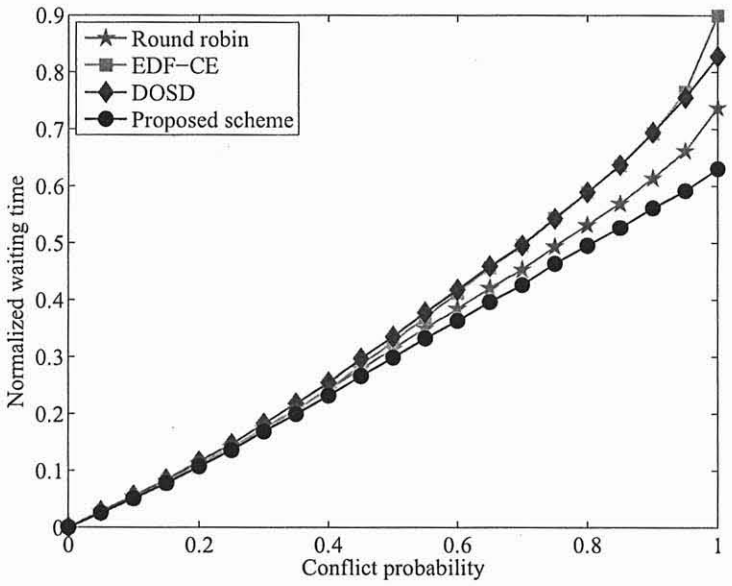

Figure 2.17 Normalized waiting time for 20 periodic measurement tasks.

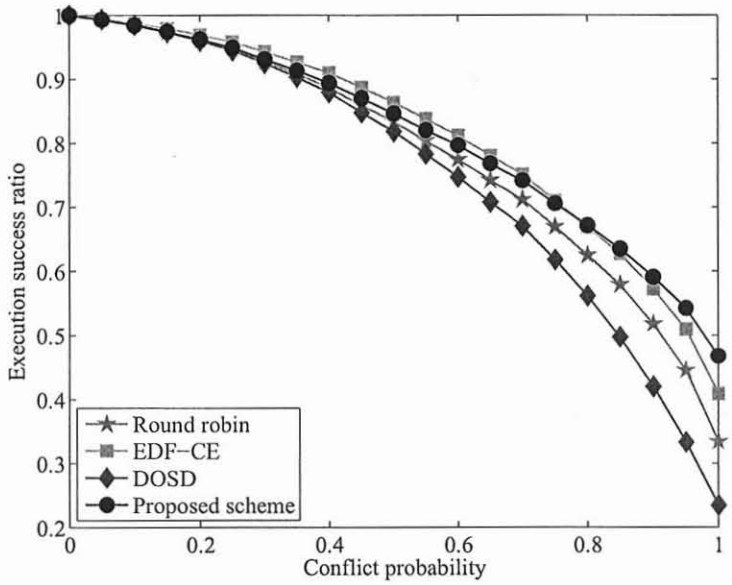

Figure 2.18 Execution success ratio for 20 periodic measurement tasks.

increases, the performance differences of the schemes become more pronounced. As an interesting observation, when the conflict probability is 1, where no more than one job can be executed at a time by any of the schemes, the waiting time and success ratio of the schemes show differences. The low success probability of DOSD is expected as it selects jobs with long execution time first, and the remaining time will then be left to a large number of tasks that may be delayed close to or beyond the end of their periods; therefore, a large number of jobs are missed. In the proposed scheduling algorithm, the degree of a sub-vertex is decided by the length of the execution time of the job, so that scheduling by the ascending order of the degree means that the job with the shortest execution time is scheduled first. This selection can potentially save a larger number of time slots for the subsequent jobs in the waiting queue. Therefore, the performance of this algorithm is also the highest with the conflict probability of 1.0.

Another scenario is simulated that the execution times of periodic measurement tasks are non-uniformly distributed. The periodic measurement task set is composed of 10 measurement tasks. The execution time of 5 measurement tasks are uniformly distributed in

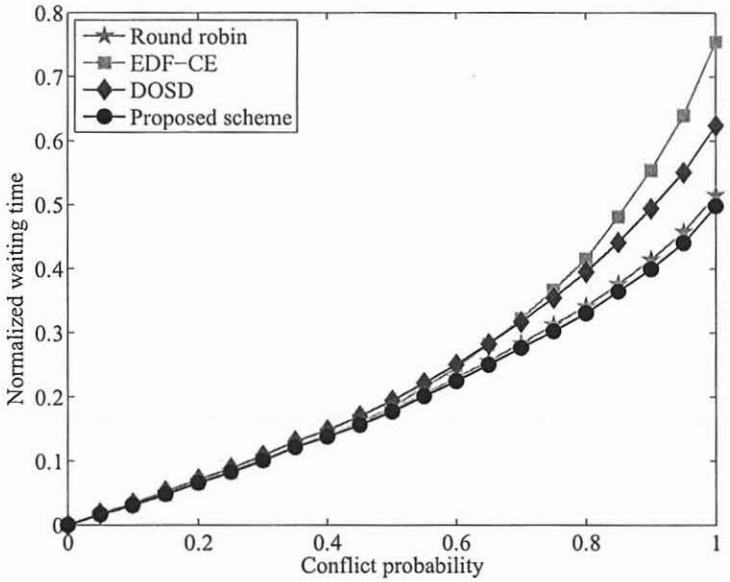

Figure 2.19 Normalized waiting time for 10 periodic measurement tasks with non-uniformly distributed execution times.

the range of [2,10] time units while the execution time of the rest of 5 tasks are randomly selected in the range of [8,10] time units. The period of the tasks is uniformly distributed in the range of [11,100] time units. The initial available time of a task is randomly selected in the range of [1,5] time units. The simulation is run 1000 times.

Figure 2.19 shows the average normalized waiting times under non-uniform distribution in the execution time of the 10 tasks. The large number of tasks with long execution times is not beneficial to the proposed scheme, but the proposed scheme still achieves the lowest normalized waiting time among all compared schemes. The round-robin scheme achieves similar normalized waiting times (although slightly higher) to those of the proposed scheme. The other schemes are favored by this distribution of execution times, but their normalized waiting times are larger than those of the proposed scheme. This indicates that the measurement samples generated by scheduling schemes in comparison are more biased from the regular measurement sampling points, so that the jitter of the time intervals between any two inter-sampling points is large.

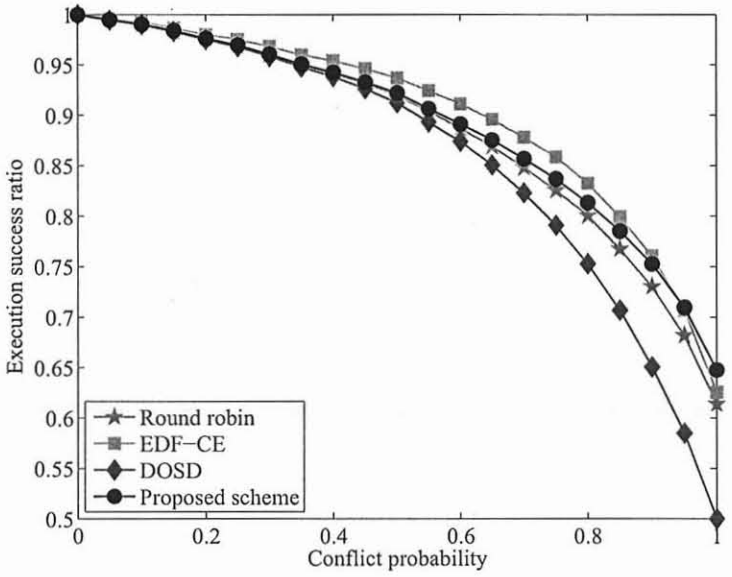

Figure 2.20 Execution success ratio for 10 periodic measurement tasks with non-uniformly distributed execution times.

Figure 2.20 shows the execution success ratios of these schemes for tasks with non-uniformly distributed execution times. The results show that the execution success ratios of all these schemes are lower than the values obtained under execution times with a uniform distribution. The consideration of a larger number of tasks with long execution times makes the scheduling schemes less efficient, and more tasks miss their executions. Nevertheless, the results show that the proposed scheme achieves the highest execution success ratio.

2.3.5.4 Simulation Results of On-Demand Tasks Scheduling. The scenario that the scheduling of periodic tasks combined with on-demand tasks is also simulated. The performance is evaluated according to the average waiting time instead of average normalized waiting time of the jobs since there is no period for the on-demand tasks. In this scenario, there are 10 periodic tasks, and on-demand tasks are created at arbitrary time slots. The periodic tasks are combined with on-demand tasks that are created at arbitrary time slots, where the arrival of an on-demand measurement task is created with a probability of 0.05

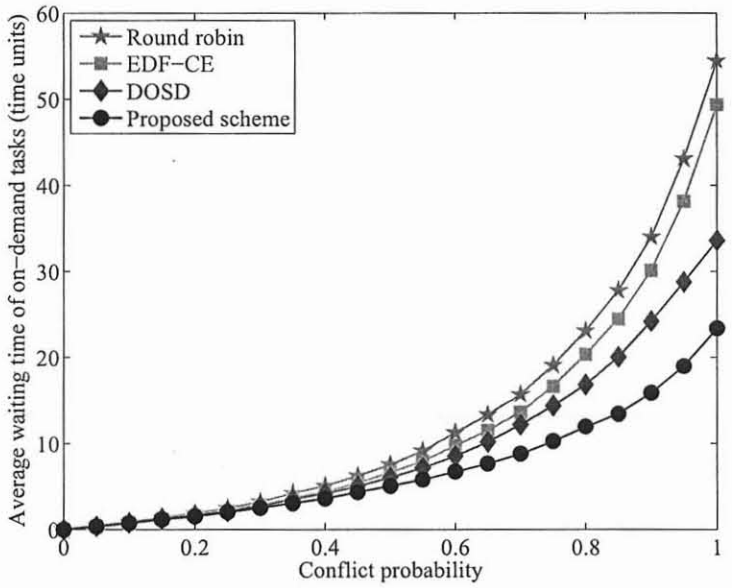

Figure 2.21 Average waiting time for on-demand measurement tasks of on-demand tasks in a combination with periodic tasks.

for each time slot. For scheduling (and execution), the priority of on-demand tasks is set to be equal to that of periodic tasks.

The execution and period times are uniformly distributed in the ranges of [2,10] and [11,100] time slots, respectively. As in the previous section, the conflict probability among all measurement tasks (including both periodic and on-demand tasks) increases from 0 to 1.0 with steps of 0.05. The simulation runs for 500 times.

Figure 2.21 shows the average waiting times measured only on the on-demand tasks. The results indicate that the proposed algorithm can achieve the lowest waiting time for on-demand tasks among the considered algorithms as all task are considered with the same priority levels. However, different from the cases with periodic tasks only, the round-robin scheme shows the lowest performance (the longest average waiting time) as some tasks cannot be re-organized with the addition of on-demand tasks because periodic tasks would still follow the pre-determined round-robin order. However, the other schemes follow similar trends as those observed for periodic tasks only.

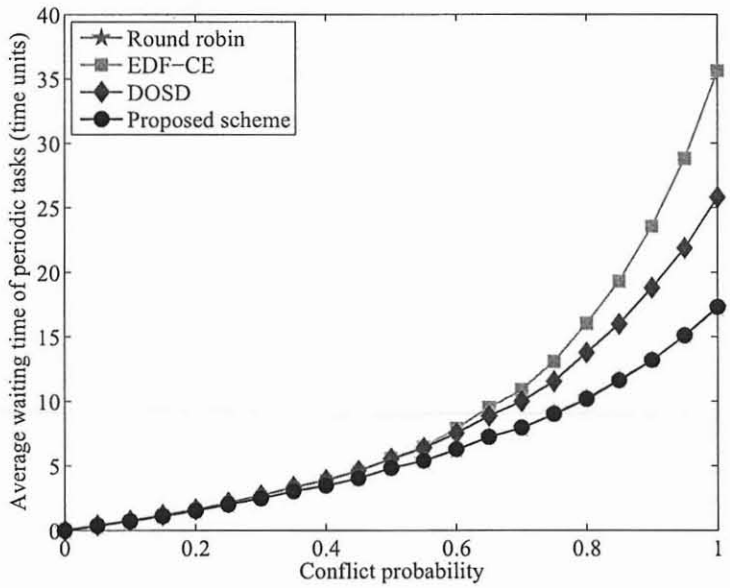

Figure 2.22 Average waiting time of periodic tasks when they are combined with on-demand tasks.

Figure 2.22 shows the average waiting times of the periodic tasks only, under this scenario. The results show that the periodic tasks undergo similar average waiting times as in the case of periodic tasks only, and the round-robin scheme and the proposed scheme achieve the lowest average waiting times. The performance of round-robin is high in this scenario as the pre-determined order followed by this scheme isolates the periodic task from the arrivals of on-demand tasks. The proposed scheme, however, accommodates the on-demand tasks and still achieves an efficient outcome, or the lowest average waiting times.

Figure 2.23 shows the normalized waiting times of the periodic measurement tasks. This graph also corroborates the previous observations, where the periodic tasks have similar results to the case of only periodic tasks, with the proposed scheme achieving the highest performance and the EDF-CE scheme achieving the lowest performance.

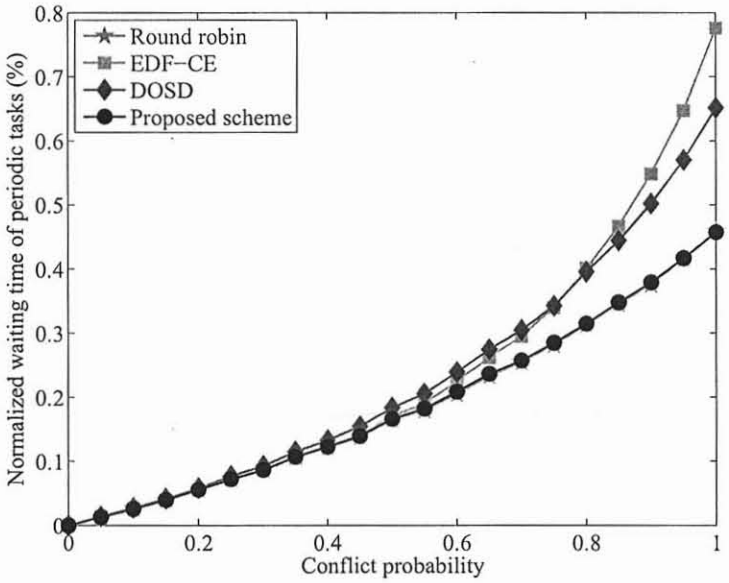

Figure 2.23 Normalized waiting time of periodic tasks when they are combined with on-demand tasks.

2.4 Summary

QoS routing is a basic and important function to fulfill QoS provisioning in current networks. However the lack of accurate link state information can jeopardize the QoS routing outcomes. This chapter presents a framework for distributed measurement of link state parameters that can be implemented in the next generation routers. The framework makes an emphasis on active probing, where routers launch various measurement processes to evaluate the QoS state for each service class between the router itself and the neighboring routers. In order to avoid interference of active probing on the measurement mechanism and the existing traffic while guaranteeing the measurement accuracy, scheduling of measurement processes are analyzed. Based on graph coloring theory, it is proposed to describe the measurement tasks relation by using a conflict graph, and to convert this scheduling problem into a graph coloring problem. Two algorithms are proposed to schedule tasks according to the ascending order of the degree of sub-vertices in the conflict graph, one for periodic measurement tasks, and another for on-demand measurement tasks. Each sub-vertex represents one basic time unit for the execution time of the task. The results showed that the

proposed scheduling schemes provide the shortest average waiting time for cases where periodic tasks are considered in the network as well as when on-demand task are added in a network with existing periodic tasks. The proposed schemes also achieve the highest utilization of network resources as shown by achieving the highest execution success ratios in the presented results. In addition, the schemes are able to schedule the on-demand tasks with either higher or equal priority with respect to that of the periodic measurement tasks.

CHAPTER 3

EFFICIENT AND RELIABLE DISSEMINATION OF LINK STATE INFORMATION

This chapter presents a novel scheme called Per-Hop pArtial-Spanning Tree Adjust (PASTA) approach to enhance the efficiency and reliability of link state dissemination. In the proposed approach, link states are distributed through the spanning tree, which is updated at each hop by considering the latest local QoS states. This approach can also guarantee that every node is notified as long as it remains connected to rest of the network. Combined with multiple spanning trees method, the reliability and fastness of the dissemination is further improved. The complexity analysis and implementation feasibility are also discussed.

3.1 Introduction

Currently implemented dissemination mechanisms are based on flooding link-states advertisement (LSA) packets, such as OSPF [12]. In these mechanisms, the routers send their own LSA packets and forward those created by other routers to all the neighbors until every router in the network area knows the updated link states. The time at what this occurs in the network is called convergence time. Flooding mechanisms are robust enough to guarantee that link states get disseminated even in the case of link and node failures as long as every node in the network remains connected.

On the other hand, link states are defined by various QoS parameters in QoS-enabled network. Considering the growth of both QoS parameters and the scale of the Internet, dissemination of the large amounts of QoS link state information generates a significant dissemination overhead by flooding methods. The dissemination overhead has the potential to occupy a significant amount of transmission bandwidth that can diminish the network utilization by user's data traffic. Furthermore, the link states may have been stale when

a new LSA arrives to the destination node because of congestion in the transmission. To minimize the overhead, various implementation schemes attempt to reduce the state update frequency. As for example, the OSPF protocol uses a state refreshing time of 30 minutes. However, approaches like this also reduce the timeliness of link state update, which may cause false routing. Therefore, dissemination efficiency and accurate state awareness is hard to achieve with flooding-based algorithms.

To overcome the large overhead of dissemination for flooding schemes, some router vendors have proposed to reduce the flooding of already disseminated and unchanged information [85]. Other policies for link-state update have been proposed. In threshold-based policy, a state update is triggered only when the percentage change in the link state value exceeds the predefined threshold [86]. In class-based policy, the range of link state values is partitioned into classes and an update is triggered whenever the link state changes sufficiently to cross a class boundary [86, 87]. However, these approaches reduce the dissemination overhead at the expense of increasing the convergence time and link state accuracy degradation [88]. As another alternative, link-state dissemination based on spanning-tree has been considered. A localized flooding approach where the dissemination network is based on a spanning-tree scheme instead was proposed [89]. To decrease the communication overhead to maintain the tree, a protocol called Topology Broadcast based on Reverse Path Forwarding (TBRPF) was proposed [90]. This protocol uses Reverse Path Forwarding (RPF) to broadcast the link states through the spanning tree in the reverse-path direction. The construction of the tree is based on the path with the minimal number of hops from every node to the source of the update.

Although the dissemination overhead is reduced, the above spanning-tree-based approaches have not considered the continuous update of the spanning tree itself in a QoS environment, thus the tree keeps using the same links even the QoS states of those links deteriorate, which causes longer convergence time or even dissemination failure. The example in Figure 3.1 illustrates the drawback of such spanning-tree-based methods. In this

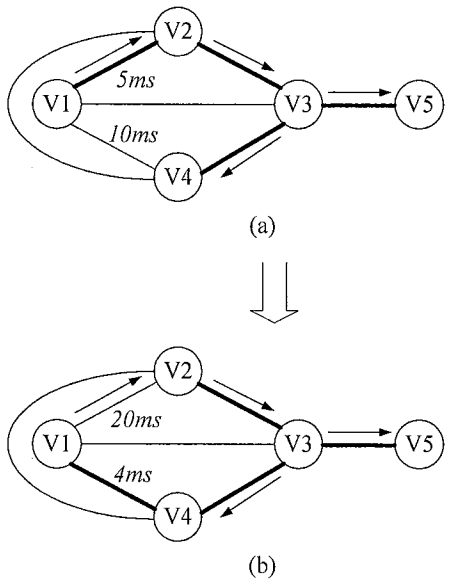

Figure 3.1 An example of the outdated spanning tree.

spanning tree, the information travels from node V_1 to node V_2 ($V_1 \rightarrow V_2$), as the bold-line link in Figure 3.1(a). The delay state of the link ($V_1 \rightarrow V_2$) changes from 5 ms to 20 ms, and that of link ($V_1 \rightarrow V_4$) changes from 10 ms to 4 ms. If node V_1 realizes the state changes of the outgoing links ($V_1 \rightarrow V_2$ and $V_1 \rightarrow V_4$), the optimal spanning tree would use ($V_1 \rightarrow V_4$), as Figure 3.1(b) shows. However, the existing spanning-tree-based schemes let nodes use existing spanning tree as described in Figure 3.1(a). In this case the node is not sensitive to the local state change, or saying, the spanning tree is not adaptive with the link changes. The spanning tree is recomputed after the last convergence is completed, which means every node receives the updated link state advertisement message. Therefore, the convergence time in this example increases to 15 ms (20 ms-5 ms) instead of decreasing to 6 ms (10 ms-4 ms).

Different from spanning-tree schemes, flooding schemes is designed to broadcast LSA packets throughout network. Hence, the links with best performance (shortest delay, largest available bandwidth etc.) are always utilized. Without considering the potential

congestion caused by flooding overhead, the convergence time of a flooding approach is the shortest for state dissemination algorithms.

Reliability of the link state dissemination is another crucial requirement to support QoS networks. Spanning-tree-based approaches have not been considered for providing reliability, that a dissemination tree may be susceptible to failure if a single link fails. For example, if a single link or node fails, then the tree might become partitioned into two or more subtrees, and the link state information generated from one subtree may be unreachable to the rest of the network even if the network is still interconnected by links that are not included in the dissemination tree. In the graph shown at the top of Figure 3.1, if the link ($V_2 \rightarrow V_3$) fails, then the spanning tree becomes partitioned into sub-trees V_1, V_2 and V_4, V_3, V_5. Therefore, the LSA packets cannot be forwarded from one subtree to another.

A novel approach, called Per-hop pArtial-Spanning-Tree Adjustment (PASTA) scheme, is proposed to minimize both the link-state dissemination overhead and the convergence time. Furthermore, to provide reliability to the dissemination tree, the back-trace scheme and the multiple spanning tree (MST) scheme are proposed. In the proposed schemes, the weight of a link is defined by its QoS state. These schemes are able to find and update a feasible link-state dissemination tree with smaller delay and more transmission bandwidth compared to the original tree. Because network nodes are aware of the QoS state of their outgoing links in the proposed schemes, the spanning tree can be updated partially at each hop according to the latest local states combined with the stored and current states received from other nodes.

In the reminder of this chapter, Section 3.2 describes preliminary definitions and terms. Section 3.3 introduces the PASTA, MST, and back-trace schemes. Section 3.4 estimates the dissemination overhead and computation complexity of these mechanisms. Section 3.6 presents the summary.

3.2 Preliminary Definition

Throughout the remainder of this chapter, the following terms are used indistinctly: *node*, *router*, and *vertex*, and *edge* and *link*.

- Cospanning tree: the cospanning tree T^* of a spanning tree T of a graph G is a subgraph of G. It has all the vertices of G and exactly those edges of G that are not in T [91].

- Component: a maximal connected subgraph of a graph. That is, between any two nodes in the same component, there is always a path connecting them. However, there is no path connecting two nodes which are in different components [91]. An isolated node by itself is treated as a single component.

- Edge cut: A set of edges that if the set are removed from a connected graph, will disconnect the graph into one or more components.

- Minimum edge cut: An edge cut such that there is no other edge cut containing fewer edges. If any edge is placed back in graph G, G will be reconnected.

3.3 QoS-Based Link State Dissemination Schemes

3.3.1 Formulation of Link-State Dissemination Problem

Given a network topology $G(V, E)$, for each directed edge $e \in E$, the weight/state of e at time t_j is denoted as $w(e, t_j)$ or $w(e)$. At each node, the QoS states of all the links are kept in matrix $W_i(G)$ for node i. The matrix is also denoted as $W_i(G, t_j)$, where t_j is the time of the last link state update. In addition, $w^p(e)$ and $w^c(e)$ denote the previous state and current state of the link, respectively; $W^p(G)$ and $W^c(G)$ denote the previous state and current state of the graph G, respectively. The spanning tree is denoted as $T_o(q)$, where o is the root node of the initial spanning tree which is built right after state matrix at each node is initialized. q is the root node of the current updated spanning tree. $T_o(q, W_q(G))$ means the tree is built from the state matrix $W_o(G)$ at node q.

According to above definitions, given that node i obtains the latest link states of the network $W_i(G, t_a)$ at time t_a, the spanning tree $T_i(i)$ may be built at this node based on $W_i(G)$. At this moment, $W_i(G, t_a)$ is the current state of graph G, so it is equal to $W_i^c(G)$:

$$W_i^c(G) = W_i(G, t_a)$$

$$\begin{aligned} T_i(i) &= T_i(i, W_i^c(G)) \\ &= T_i(i, W_i(G, t_a)) \end{aligned} \tag{3.1}$$

If local link state $w_i(e_j)$ changes at time t_b:

$$w_i^p(e_j) = w_i(e_j, t_a)$$

$$w_i^c(e_j) = w_i(e_j, t_b)$$

$$\begin{aligned} W_i^c(G) &= W_i(G - e_j, t_a) \cup w_i^c(e_j) \\ &= W_i(G - e_j, t_a) \cup w_i(e_j, t_b) \\ &= W_i(G, t_b) \end{aligned}$$

$$W_i^p(G) = W_i(G, t_a) \tag{3.2}$$

According to the conventional spanning-tree-based dissemination algorithm, as in TBRPF, node i disseminates this update of e_j along the tree $T_i(i)$. However, since the tree is built according to Equation 3.1, the tree might be outdated:

$$\begin{aligned} T_i(i) &= T_i(i, W_i(G, t_a)) \\ &= T_i(i, W_i^p(G)) \end{aligned} \tag{3.3}$$

There are three types of constraints to characterize performance matrix of a network that are also used to build the dissemination tree: concave, additive, and multiplicative [92]. For any path $p(l_1, l_2, \ldots, l_n)$ which means the path p is composed of n elements: $l_1, l_2, \ldots, l_n$, the constraint on $w(p)$ for path p is explained as:

$$\begin{aligned}
\text{concave as} \quad & w(p) = \min\left(w(l_1), w(l_2), \ldots, w(l_n)\right) \\
\text{additive as} \quad & w(p) = w(l_1) + w(l_2) + \ldots + w(l_n) \\
\text{multiplicative as} \quad & w(p) = w(l_1) \cdot w(l_2) \cdot \ldots \cdot w(l_n) \\
\text{so} \quad & \lg w(p) = \lg w(l_1) + \lg w(l_2) + \ldots + \lg w(l_n)
\end{aligned}$$

A concave constraint is determined by the bottleneck of the path, for instance, the minimum available bandwidth of the path. An additive constraint, where delay and jitter are examples, is determined as the sum of the state of each link in a path. Here delay is considered as the most important QoS parameter as the convergence time of link-state dissemination is in function of link delays. A multiplicative constraint, such as packet-loss ratio, is determined as the product of the states of the links on the path, and it can be converted to be an additive constraint by a logarithmic conversion. Without losing generality, additive constraints are considered in the following case analysis.

Assume that node i is the root node of the original spanning tree, and node k is one of the nodes of the tree. In this dissertation the out-of-current-tree links are called external links. The possible cases of the link state, in which $w_k(e_j)$ can change, are:

1. $e_j \notin T_i(i)$ and $w_k^c(e_j) > w_k^p(e_j)$

   This case means that the current state of external link(s) degrades. Therefore, there is no need to update the current spanning tree.

2. $e_j \notin T_i(i)$ and $w_k^c(e_j) < w_k^p(e_j)$

   In this case, the external links can provide better service than before. Therefore, update of the tree is necessary.

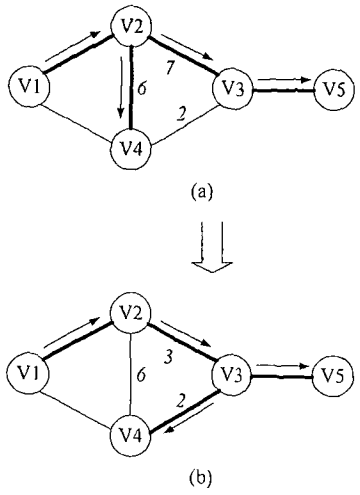

Figure 3.2 An example to illustrate case 4.

3. $e_j \in T_i(i)$ and $w_k^c(e_j) > w_k^p(e_j)$

   Then the performance of the spanning tree $T_i(i)$ worsens because the state of edge e_j degrades. Thus, update of the spanning tree is necessary.

4. $e_j \in T_i(i)$ and $w_k^c(e_j) < w_k^p(e_j)$

   In this case the performance of the spanning tree improves. However, it is possible the state change of the link may result in the optimal dissemination path changes. As the example shown in Figure 3.2, when the weight of edge $V_2 \rightarrow V_3$ decreases from 7 to 3, the spanning tree $T_{V_1}(V_1)$ as depicted in Figure 3.2 (b) is prior to the original spanning tree depicted in Figure 3.2 (a). Hence, it is required to update the spanning tree.

From the above analysis, it can be seen that all the cases, except for case 1, need the adjustment of the spanning tree.

3.3.2 PASTA Link-State Dissemination Algorithm

PASTA algorithm is proposed in this section. Based on a spanning tree, PASTA is designed to reduce the amount of overhead that the flooding algorithms has and to reduce the large convergence time of existing spanning tree algorithms.

It is considered that at the initialization stage of link-state dissemination, the nodes do not have all link-state $W(G)$ from every node, but only those of their local links and their neighbors. The spanning tree cannot be built without complete link states of the network. Therefore, at the initial step the link states are broadcasted by flooding. After this initial step, flooding is no longer performed and the mechanism to update the tree follows the proposed PASTA algorithm.

The PASTA adjustment procedure is described by the flow chart as illustrated in Figure 3.3. This procedure follows four stages.

3.3.2.1 Initialization Stage After the initial flooding phase at time t_0, the current link states are disseminated throughout the network, so every node builds a state matrix:

$$W^c_{V_k}(G) = W_{V_k}(G, t_0)$$

$$T_{V_k}(V_k) = T_{V_k}(V_k, W_{V_k}(G, t_0)) \tag{3.4}$$

3.3.2.2 Rebuilding the Tree at the Original Root Node As every node is aware of the states of its own outgoing links, the dissemination can be triggered from any node in the network. Assume that node $V_k \in V$ is the root node of the tree, and it triggers the next link state dissemination process. After time interval $\lambda > \alpha$, if one of node V_k's output links e_i

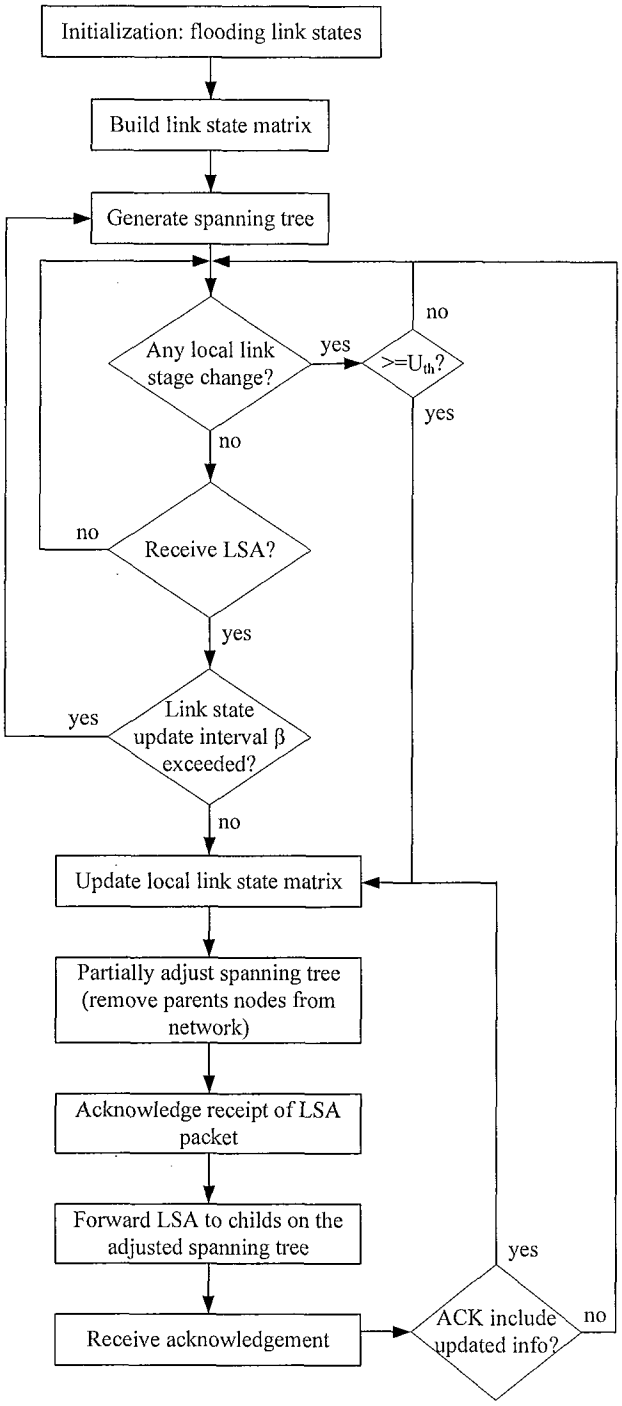

Figure 3.3 Flow chart of the PASTA algorithm.

changes the state from $w_k(e_i, t_0)$ to $w_k(e_i, t_0 + \lambda)$, so:

$$w^c_{V_k}(e_i) = w_{V_k}(e_i, t_0 + \lambda)$$

$$\begin{aligned} W^c_{V_k}(G) &= W_{V_k}(G - e_i, t_0) \cup w^c_{V_k}(e_i) \\ &= W_{V_k}(G - e_i, t_0) \cup w_{V_k}(e_i, t_0 + \lambda) \\ &= W_{V_k}(G, t_0 + \lambda) \end{aligned} \tag{3.5}$$

The parameter α is the minimum threshold time to perform the next link state update, as defined in the OSPF protocol. That is, the minimum interval between two consecutive updates cannot be smaller than α to prevent unnecessary continuous updates from a node.

Normalized link-state change, $\Delta(e_i)$, is:

$$\Delta(e_i) = \frac{\left| w^c_{V_k}(e_i) - w^p_{V_k}(e_i) \right|}{w^p_{V_k}(e_i)} \tag{3.6}$$

According to the policy of the threshold-based scheme to trigger a link-state update process [86], the spanning tree update must use a threshold U_{th} to avoid frequent updates owning to the small fluctuations in the link state. Following this rule, the update policy is explained as:

- $\Delta(e_i) \geq U_{th}$, state dissemination triggered by node V_k
- $\Delta(e_i) < U_{th}$, no dissemination triggered by node V_k

Given an state change $\Delta(e_i) \geq U_{th}$, this node would have generate a LSA packet to disseminate the new link state $w_{V_k}(e_i, t_0 + \lambda)$. Before the LSA dissemination, node V_k updates the recorded link states as in Equation 3.5. Based on the updated matrix $W_{V_k}(G, t_0 + \lambda)$, the root node V_k computes the dissemination spanning tree $T_{V_k}(V_k)$:

$$\begin{aligned} T_{V_k}(V_k) &= T_{V_k}(V_k, W^c_{V_k}(G)) \\ &= T_{V_k}(V_k, W_{V_k}(G, t_0 + \lambda)) \end{aligned} \tag{3.7}$$

Therefore, the tree might be rebuilt (with the same root node) depending on the latest link states. After that, the tree structure $T_{V_k}(V_k)$ and the updated link state $w_{V_k}(e_i, t_0 + \lambda))$ are marked into the LSA packets, which are forwarded to V_k's child node V_{k+1}.

3.3.2.3 Tree-Adjustment at A Child Node During the LSA transmission phase, the state of the outgoing link e_j connected to V_{k+1} may change at time t_1 as $w_{V_{k+1}}(e_j, t_1)$. Therefore, β is introduced as a threshold time to update the link state and then consider the following two cases:

In the first case, the LSA packet arrives to node V_{k+1} within time $t_1 + \beta$. Then, the spanning tree $T_{V_k}(V_k)$ may be adjusted based on the states marked in LSA sent by the parent and the current local link state. The tree is denoted as $T_{V_k}(V_{k+1})$ if there is a tree update. The tree is updated according to the following Equation 3.8:

$$\begin{aligned} T_{V_k}(V_{k+1}) &= T_{V_k}(V_{k+1}, W^c_{V_{k+1}}(G)) \\ &= T_{V_k}(V_{k+1}, (W_{V_{k+1}}(G - \{e_i, e_j\}, t_0) \cup w^c_{V_k}(e_i) \cup w^c_{V_{k+1}}(e_j))) \\ &= T_{V_k}(V_{k+1}, (W_{V_{k+1}}(G - \{e_i, e_j\}, t_0) \cup w_{V_k}(e_i, t_0 + \lambda) \cup w_{V_{k+1}}(e_j, t_1))) \end{aligned} \tag{3.8}$$

However, because some nodes of the spanning tree have been updated during dissemination, the notified nodes and the edges connected to them can be excluded from the update process. Therefore, the above Equation 3.8 can be simplified:

$$G_{\overline{V_k}} = G - G(V_k, E_{V_k})$$

$$\begin{aligned} T_{V_k}(V_{k+1}) &= T_{V_k}(V_{k+1}, W^c_{V_{k+1}}(G_{\overline{V_k}})) \\ &= T_{V_k}(V_{k+1}, W_{V_{k+1}}(G_{\overline{V_k}} - e_j, t_0) \cup w^c_{V_{k+1}}(e_j)) \\ &= T_{V_k}(V_{k+1}, W_{V_{k+1}}(G_{\overline{V_k}} - e_j, t_0) \cup w_{V_{k+1}}(e_j, t_1)) \end{aligned} \tag{3.9}$$

where $G_{\overline{V_k}}$ is the graph where node V_k and the edges connected to node V_k are removed.

The spanning tree $T_{V_k}(V_{k+1})$ replaces the old tree $T_{V_k}(V_k)$ in the received LSA packet. The LSA packet is updated with $w_{V_{k+1}}(e_j, t_1)$ in addition to the updated information of $w_{V_k}(e_i, t_0 + \lambda)$ already included. The new LSA packet is then forwarded to the next node of $T_{V_k}(V_{k+1})$. In this way, one LSA packet is able to disseminate several state changes along the node.

The second case is when the LSA packet arrives to node V_{k+1} after time $t_1 + \beta$. The same procedure as in the previous case is followed, except that the forwarded LSA packet does not include the local state $w_{V_{k+1}}(e_j, t_1)$, because at time $t_1 + \beta$, a new link state dissemination, which is independent from the current dissemination $T_{V_k}(V_k)$, is generated according to a spanning tree $T_{V_{k+1}}(V_{k+1})$, which uses V_{k+1} as the original root node.

3.3.2.4 Acknowledgement to the Parent Node In the OSPF protocol, each newly received LSA is acknowledged. This is usually implemented by sending link state acknowledgment (ACK) packets. Considering that, the child node here is proposed to send an ACK packet back to the parent node once the child node successfully receives an LSA packet. Furthermore, the current updated link state $w_{V_{k+1}}(e_j, t_1)$ is marked so included in the ACK packet and sent back to the parent V_k from child node V_{k+1}. If the ACK packet has been sent already, for instance, from the parent node to the grandparent, a separate LSA packet is generated and is sent back.

After the LSA packet is transmitted to the next child node, Steps 2 to 4 are performed iteratively until all the nodes in the network are notified.

PASTA algorithm is illustrated with the following example. As shown in Figure 3.4, assume that node V_1 is the current disseminating node. V_1 computes the adjusted tree $T_{V_0}(V_1)$ based on the information of the network without the connection to parent node V_0. Hence, node V_1 becomes the temporary root node of the tree $T_{V_0}(V_1)$. Then, the LSA packet is forwarded from V_1 to the child node through the partially adjusted tree $T_{V_0}(V_1)$. Assume

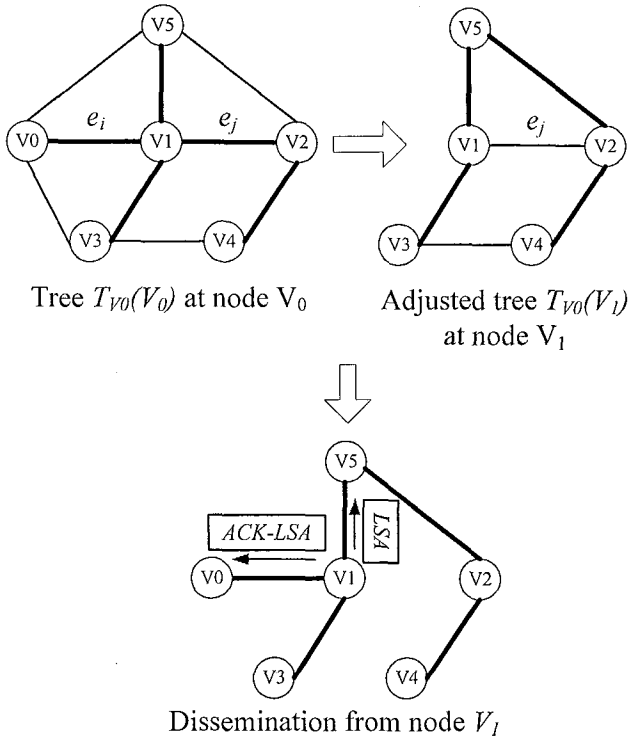

Figure 3.4 Spanning tree adjustment and acknowledgement from node V_1.

that the change of edge e_j is above U_{th}. Then the current state $w^c(e_j)$ is also marked in this LSA packet so that the next hop can obtain the latest link-state from both ancestors V_0 and V_1. At the same time, the ACK packets that carry the state of $w^c(e_j)$ are sent back to V_0.

3.3.3 Back-Trace Algorithm

If the current node cannot reach its child (i.e., isolated node) because either the edge in between fails or the link-state is poor and cannot forward the link state dissemination, a recovery algorithm, called back-trace, is proposed. The back-trace algorithm has the objective to build a tree split by a link failure or by a link that cannot deliver LSA packet.

Following the PASTA algorithm described in Section 3.3.2.3 and based on the graph $G_{\overline{V_{k+1}}}$, if current node V_{k+2} cannot find a path to forward the LSA to its child node V_{k+3}, it generates a fail report and send it to the parent node V_{k+1}. The fail report lists the event trap that which node fails to receive the new LSA. The parent node V_{k+1} attempts to find a

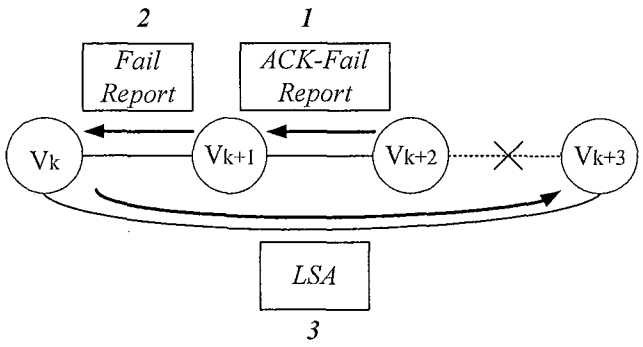

Figure 3.5 An example of back-trace algorithm.

path to V_{k+3} and to disseminate all the updated link states (including those from child V_{k+2} sent through the ACK packet) to this grandchild. If node V_{k+1} cannot find such a path, it continuously sends the fail report back to its parent, if any. This procedure is executed iteratively until a node can successfully forward the LSA packet to the isolated node or until the root node is reached (this means that all the nodes fail to reach V_{k+3}). Figure 3.5 depicts a simple example of back-trace algorithm.

Because the current node is required to acknowledge the parent node about the status of the received LSA, as in the PASTA algorithm through ACK packet, the fail report and the state of the isolated child node are marked in the ACK packet and sent back to the parent node. If no ACK packet is sent from current node, a separate ACK packet may be generated by the current node to deliver the fail report and the state of the isolated node.

3.3.4 Multiple Spanning Tree (MST) Algorithm

To evaluate the reliability of the scheme, the following criteria is defined according to [93]:

***Criterion I**: Given a network topology $G(V, E)$, for any link e that is not the minimum edge cut of G, if the link state information is still reachable to all nodes V after e is removed from G, this link state dissemination scheme is regarded as reliable.*

This definition can be used to verify that a single spanning tree in [89, 90] is not reliable if there is only one edge between two nodes. Furthermore, if the edge is removed,

the graph becomes multiple components. The PASTA algorithm assisted by the back-trace algorithm provides high reliability by avoiding the selection of local failed links. However, it is necessary to further enhance the reliability from the protection perspective.

***Lemma I**: Given a network $G(V, E)$, a subnet $G'(V, E')$ is regarded as a reliable topology (RT) if for any edge $e \in E'$, $\{e\} \neq E'_{cut}$ exists as long as $\{e\} \neq E_{cut}$ exists, where E'_{cut} and E_{cut} are the minimum edge cut of $G'(V, E')$ and $G(V, E)$.*

A reliable topology here is a subnet over which the link state dissemination is reliable as defined by Criterion I. Also by this criterion, it can be concluded that a link state information dissemination approach is reliable if the number of members in the minimum edge cut in the dissemination graph is not smaller than two. If the network does not have a minimum edge cut with only one member, all of its subnets that contain same nodes set V do not have the one-member minimum edge cut either. Each link in the RT has alternative edges. Thus, finding a reliable dissemination depends on having a reliable subnet topology or RT. However, this problem needs to be solved through a spanning cycle process [91], which is NP-hard and unable to achieve fast convergence. Hence, the work presented in [93] is extended, by proposing to update link states through multiple spanning trees in terms of Lemma II, so called multiple spanning tree (MST) algorithm.

***Lemma II**: Consider a connected network $G(V, E)$, and assume $T(V, E'_0)$ is a spanning tree of G, and $\overline{T}(V, \overline{E'_0})$ is the cospanning tree of G. Then $\overline{T}(V, \overline{E'_0})$ may include h $(h \geq 1)$ components $G_1(V_1, E_1)$,$G_2(V_2, E_2)$,...,$G_h(V_h, E_h)$ with spanning trees $T_1(V_1, E'_1)$, $T_2(V_2, E'_2)$, ..., $T_h(V_h, E'_h)$, respectively. Thus, the topology $G'(V, E')$ formed by $T(V, E'_0)$ and $T_i(V_i, E'_i)$, with $i = 1, \ldots, h$, is an RT of $G(V, E)$.*

The tree building procedure is described by an example shown in Figure 3.6. After the spanning tree $T(V, E'_0)$ is selected, the remainder links and all nodes constructs two components $G_1(V_1, E_1)$, $G_2(V_2, E_2)$. The spanning tree of each component can be derived as $T_1(V_1, E'_1)$, $T_2(V_2, E'_2)$. Thus $T(V, E'_0)$, and $T_1(V_1, E'_1)$, $T_2(V_2, E'_2)$ are the RT of $G(V, E)$. The link state information dissemination on this topology is reliable.

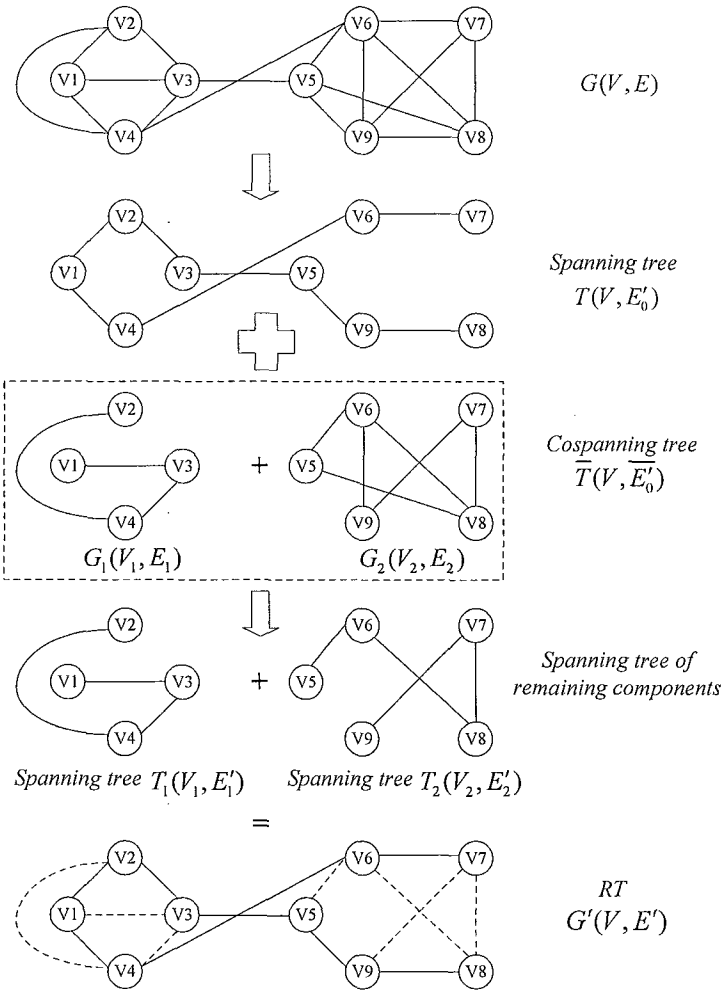

Figure 3.6 Example of RT build-up procedure.

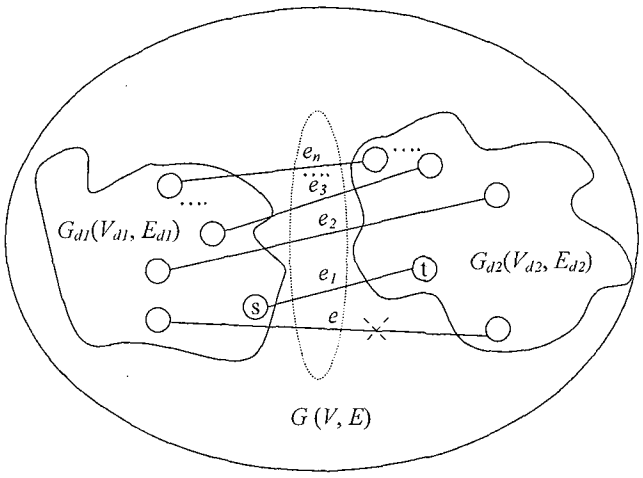

Figure 3.7 Illustration of a minimum edge cut of $G(V, E)$:$\{e_1, e_2, \ldots, e_n\}$.

The above tree combination can be proved as a RT by contradiction. First, assume that there is a link $e \in E'$ that satisfies $\{e\} = E'_{cut}$ and $\{e\} \neq E_{cut}$, where E'_{cut} and E_{cut} are the minimum edge cut set of $G'(V, E')$ and $G(V, E)$. Then assume $E_{cut} = \{e, e_1, e_2, \ldots, e_n\}$ as Figure 3.7 shows. After edge e is removed, $G'(V, E')$ is divided into two parts $G'_{d_1}(V_{d_1}, E'_{d_1})$ and $G'_{d_2}(V_{d_2}, E'_{d_2})$. It can be derived that $e_1, e_2, \ldots, e_n \notin E'$, otherwise $e_1, \ldots, e_n$ still connect G'_{d_1} and G'_{d_2} after e is removed from E', then $\{e\}$ is not a minimum edge cut set of $G'(V, E')$, and the assumption $\{e\} = E'_{cut}$ is contradicted.

Thus, denote s ($s \in V_{d_1}$) and t ($t \in V_{d_2}$) as two nodes connected by e_1. Since $e_1 \notin E'$ and spanning tree $E'_0 \in E'$, e_1 exists in cospanning tree $\overline{E'_0} = E' - E'_0$. On the other hand, there is $\{e\} = E'_{cut}$ and so does $e \in E'_0$. Then, after removing spanning tree E'_0, e_1 belongs to one of the components $G_i(V_i, E_i)$ and still connects the divided parts $G'_{d_1}(V_{d_1}, E'_{d_1})$ and $G'_{d_2}(V_{d_2}, E'_{d_2})$. Therefore, the assumption is contradictory and the claim is true for Lemma II. Based on this Lemma, $G'(V, E')$ combined with multiple spanning trees is also RT.

3.3.5 Spanning Tree Selection Algorithm at Each Node

By using the PASTA mechanism, a partial spanning tree can be rebuilt at each node. Currently, the most widely used approaches for building a routing tree are Dijkstra's algorithm

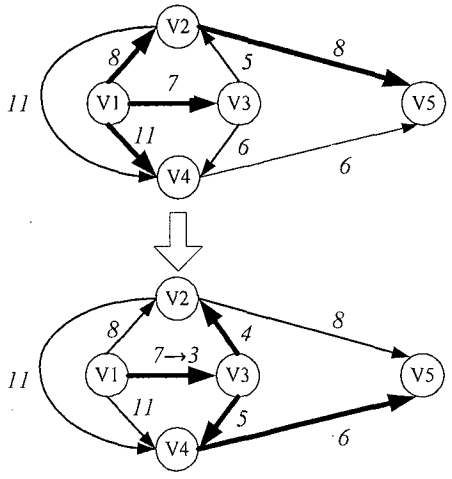

Figure 3.8 The example of the LDF algorithm.

and Prim's algorithm. Both of them construct a new tree without recurring to the information of the last (and obsolete) tree, so the spanning tree structure is not stable and the computation load is relatively high.

In [94], it is proposed to construct the spanning tree according to the difference of the decrement of the distance from the root node to the other nodes. This approach reduces the access times to each node in the network so the computation load diminishes and some area of the old spanning tree can be kept in the new tree. It is called largest-decrement-first (LDF) algorithm in this dissertation. A simple example describes the LDF scheme. The initial spanning tree is shown as the top graph in Figure 3.8. Then, the weight of the edge $V_1 \rightarrow V_3$ changes from 7 to 3 because a change of the link state. According to the Dijkstra's algorithm, node V_2 is first accessed since its temporary shortest distance to the root V_1 is smaller than that of V_4. Then V_5 is accessed through V_2 with distance 17. However, after later selecting V_4, the path to V_5 needs to be reselected as (V_1, V_4, V_5) because of the shortest distance of 14. In contrast, if the LDF scheme is used, V_4 is first selected as the distance has the largest reduction (i.e., 3) and it becomes the closest node. Then V_5 is selected because it has the second largest deduction (i.e., 2). Finally, V_2 is chosen (distance reduces by 1).

Derived from this method, the spanning tree selection algorithm is described by the pseudo code in Figure 3.9. Assume that the only constraints on the link state dissemination are the available bandwidth of the link and the convergence delay bound. Other parameters can be used.

3.4 Complexity Analysis

This section analyzes the complexity of the proposed schemes and it is compared with that of the existing schemes.

3.4.1 Overhead Complexity

In the flooding algorithm, the overhead of dissemination is proportional to number of links and the number of service classes. Given $G(V, E)$ with S service classes, the dissemination overhead complexity of flooding mechanism is estimated as $O(S|V||E|)$, where $|V|$ and $|E|$ are the number of nodes and edges in G. If m link state updates occur within a time period P, the total overhead complexity is $O(mS|V||E|)$.

As for the single spanning tree algorithm, which uses a single link between any two nodes, a total of $|V| - 1$ links are used for dissemination. Therefore, the overhead is $O(S(|V| - 1))$. For m link state updates, the overhead is $O(mS(|V| - 1))$.

In the PASTA algorithm, with r multiple spanning trees, the overhead is $rS(|V| - 1)$. It can be seen that PASTA lightly sacrifices overhead, but in average, a network would only need a couple of spanning trees to provide the required reliability, so the overhead complexity is not increased significantly. Moreover, for m link state updates within time period P, the overhead complexity is $xrS(|V| - 1)$. Here, the parameter $x \in [1, \ldots, m]$ and $x = 1$ mean that only one LSA packet is used to disseminate all link states and the LSA arrives before the local state exceeds the time threshold β. Thus, the overhead complexity of the PASTA-MST algorithm is equal to or smaller than that of a single-spanning tree algorithm. The following example may give some light to the above point. Considering

Spanning tree selection algorithm

Remove link whose bandwidth is: $B_{avail} < B_{required}$

if $e_j \notin T_i(i)$ and $w_k^c(e_j) > w_k^p(e_j)$ no adjustment on spanning tree T

else adjust T based on LDF

Initialize

$\forall v \in V$ Parent $P(v,T) = \phi$, Child $C(v,T) = \phi$, Length from root to v $L(v,T) = 0$,

Node status $U(v,T) =' unchecked'$, $inqueue(Q, \{root(T), 0, 0\})$

define $spring(v,T)$: set of 'checked'child nodes of v

$\forall v \in V, w_v^c(e_j) = w_v^p(e_j) + \pi$ //π is the change of state

$NewDist = L(SourceNode(e_j), T) + w_v^c(e_j)$; $\Delta = NewDist - L(EndNode(e_j, T)$

if $\pi < 0$

if $NewDist < L(EndNode(e_j, T)$ $inqueue(Q, e_j, NewDist, \Delta)$ **end**

end

if $\pi > 0$

if $e_j \in T$ $N = spring(e_j, T) \cup N$ **end**

for $e \in inLink(N)$ $inqueue(DB, e_j, NewDist, \Delta)$ **end**

end

$\Delta' = retriveMin(Q)$; //according to Δ in Q

Update: $C(v,T)$, $P(v,T)$, $L(v,T)$

for $e \in outLink(N)$

if $L(EndNode(e), T) - w(e) > L(sourceNode(e), T)$

$||U(EndNode(e), T) =' unchecked'$

$\Delta = NewDist - L(EndNode(e), T)$

$inqueue(Q, e, NewDist, \Delta)$

end

end

Figure 3.9 Pseudo code of spanning tree selection algorithm.

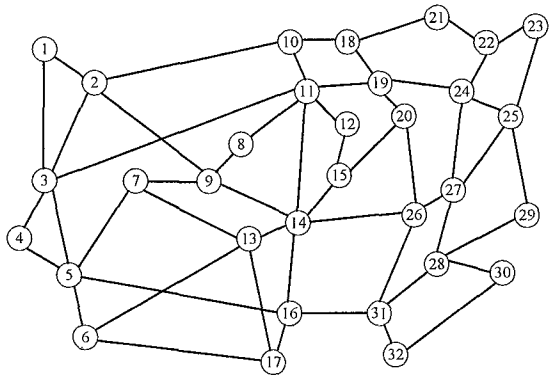

Figure 3.10 The topology of 32-node network.

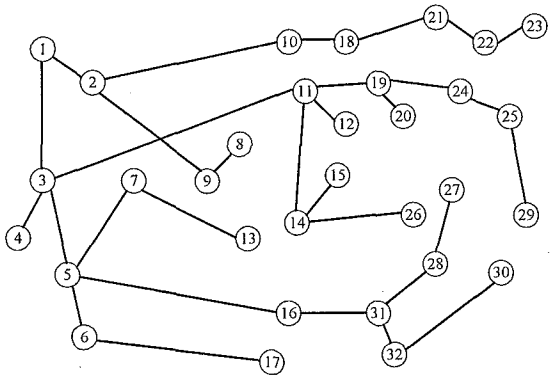

Figure 3.11 The single tree for link state dissemination in a 32-node network.

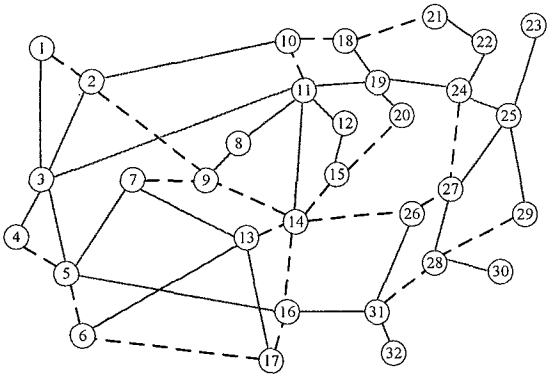

Figure 3.12 The trees generated by the PASTA algorithm in a 32-node network.

the 32-node network as depicted in Figure 3.10, there are 54 bidirectional links in this network. Given one single update from each node during a short period, if using the blind flooding method, all nodes transmit LSA packets through all 54 links, so there are at least $54 * 32 = 1728$ update packets transmitted through the network. On the other hand, with single spanning tree algorithm, a tree will be generated to transmit the LSA packets. One possible tree is shown in Figure 3.11. The total LSA packets generated from each node to its neighbor nodes on the tree are 31, which means the dissemination overhead for this update is 31. If the PASTA-MST algorithm is used, the primary tree and the partial back up tree are indicated by the solid line and the black dashed line, respectively, in Figure 3.12. The number of LSA packets generated by the nodes on the primary tree for one single update is at most 31, and the LSA packets on the partial second tree are at most 20, so the dissemination overhead is a number of packets smaller than or equal to 51.

3.4.2 Time Complexity

The time complexity of the PASTA algorithm is determined by how the elements are stored in queue. If the queue is implemented as a linear data structure, the time complexity of the flooding algorithm is $O(|V|)$. For existing single-spanning tree algorithm, as the complete spanning tree which covers all the nodes in whole network each time it is rebuilt, the time complexity of each node is $O(|V|^2)$ by using Dijkstra's algorithm or the Prim's algorithm. Therefore, the time complexity, including all the nodes in whole network is $O(|V|^3)$. Compared to flooding and single-spanning tree algorithm, multiple spanning trees are adjusted at every hop in the PASTA-MST algorithm. The edges already used in the dissemination are discarded in the computation. Assume that either the Prim's or Dijkstra algorithm is used at each node to compute the spanning tree update. Then the time complexity of PASTA-MST for the whole network is:

$$\begin{aligned} r\sum_{n=1}^{|V|} O(n^2) &= O(r|V|(|V|+1)(2|V|+1)/6) \\ &= O(r|V|^3) \end{aligned}$$

Compared to a linear process, a binary heap structure is a more efficient and practical option. With a binary heap structure, the time complexity of the LDF algorithm per node is $O(|E|lg(|V|))$, so by using the LDF algorithm, the time complexity of PASTA-MST algorithm for the whole network is:

$$\begin{aligned} r\sum_{n=1}^{|V|} O(|E|\,lg(|n|)) &= O(r\,|E|\sum_{n=1}^{|V|} lg(|n|) \\ &< O(r\,|E|\,|V|\,lg(|V|)) \end{aligned}$$

Therefore, the time complexity of PASTA-MST algorithm has the same order as the existing single-spanning-tree-based algorithm at each node.

On the other hand, the accuracy of the PASTA-MST algorithm is greatly improved as the node utilizes the current local states to construct the new spanning tree, so an optimal dissemination path can be selected. Moreover, since more than one spanning tree is used to disseminate the link states, the destination nodes obtain the LSA packets from different routes. The convergence time is counted as the time difference when a node first issues an LSA until the time the last node receives the LSA. Hence, the efficiency of the dissemination is also improved by using multiple spanning trees in the PASTA-MST algorithm.

3.5 Simulation Analysis

To estimate the performance, a simulation is built to compare the proposed schemes with the blind flooding algorithm and the conventional spanning-tree-based algorithm that uses one spanning tree to disseminate the link states. The network is randomly generated with an average degree of nodes equal to or smaller than three. The delay of a link is randomly generated between 1 to 100 time units. The link delay is used as a parameter to assign link weight values. The simulation is run for 10,000 times while a unique network topology is generated each round.

Firstly, the scenario where the network is stable during each dissemination cycle is simulated. In this case, there is at most one link change the value of its delay. Figure 3.13

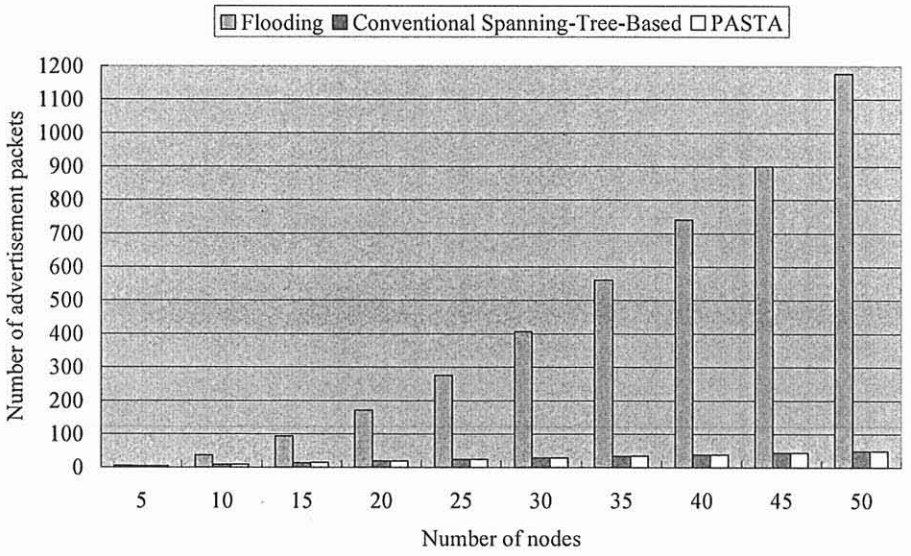

Figure 3.13 Comparison of average overhead for different dissemination methods with one link changing events.

shows the comparison of the average overhead among these three schemes for a network with nodes 5, 10, 15 to 50 nodes, with a increase step of 5 nodes. It is easily seen from the figure that the flooding method causes the most severe dissemination overhead. The PASTA and conventional spanning-tree-based algorithms have almost the same amount overhead for each network. This is because both of them use a tree structure to disseminate the link states.

Figure 3.14 depicts the simulation results of the average convergence time by these algorithms, for the cases when the dissemination process is processed successfully. From this figure, it can be seen that all three methods have similar convergence times, but the flooding algorithm takes the shortest time to finish dissemination while the conventional spanning-tree-based algorithm takes slightly longest convergence time among all three algorithms. It is because almost all the links in this scenario keep the link state, so in most cases PASTA shares the same dissemination route as conventional spanning-tree-based algorithm uses.

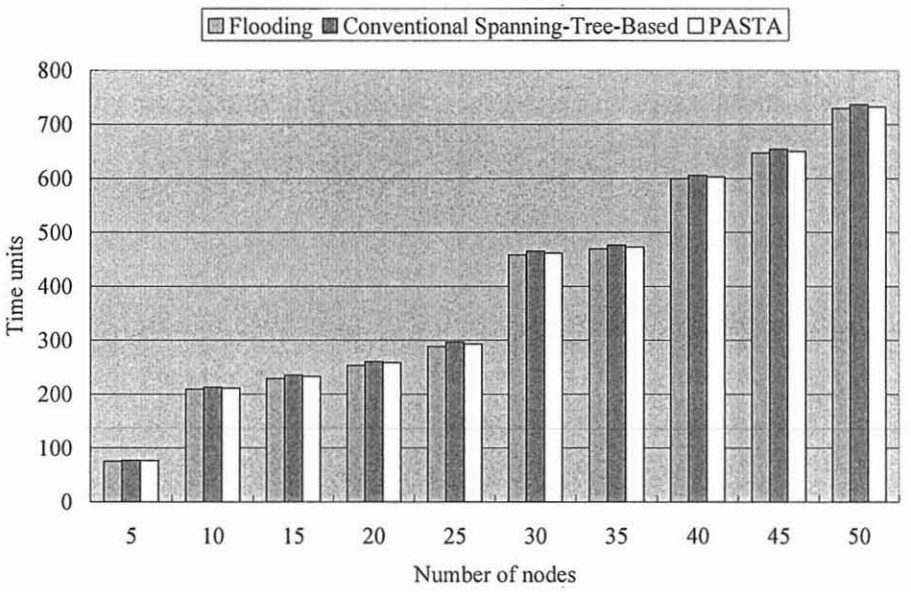

Figure 3.14 Comparison of average convergence time for different dissemination methods with one link changing events.

However, note that in some simulation cases, the dissemination process didn't converge because the link used for dissemination failed and it was no longer available. The parameter *number of dissemination failure events* is used to evaluate this attribute for each algorithm. As Table 3.1 depicts, the flooding and the proposed PASTA algorithms share the lowest dissemination failure rate. Both of them suffer from sporadic dissemination failure in case that the generated network includes the isolated node or when crucial link (e.g., a minimum edge cut contains only one member) in the network fails. On the other hand, the conventional spanning-tree-based algorithm has the highest failure rate, since this scheme is not able to adjust the dissemination path when the link on the spanning tree fails.

Another scenario that the link state of the network dynamically changes, which is more realistic, is also simulated. Here the number of link-state changing events is proportional to the number of nodes in the network (6 times of the number of nodes). In Figure 3.15, it can be observed that the flooding algorithm and PASTA algorithm spend similar time to finish dissemination, both of which are smaller than the convergence time of the conventional spanning-tree-based algorithm. The reason is that the flooding algorithm

Table 3.1 Number of Dissemination Failure Events for Different Dissemination Methods.

Nodes	PASTA	Flooding	Conventional Spanning-Tree-Based
5	2769	2769	4012
10	196	196	1187
15	16	15	634
20	7	7	465
25	3	3	392
30	3	3	322
35	2	2	271
40	2	2	239
45	1	1	212
50	0	0	189

always broadcast the message through all possible links so the links with lowest delay are always utilized to transmit LSAs. On the other hand, the conventional spanning-tree-based algorithm is not as timely as the PASTA algorithm to adjust the dissemination routes. Therefore, PASTA is more efficient to find the shortest-delay link during dissemination than the conventional spanning-tree-based algorithm.

Figure 3.16 represents the results of average amount of overhead that these three schemes cost under this scenario. Similar to the first scenario, the flooding algorithm has a large dissemination overhead. On the contrary, PASTA and conventional spanning-tree-based algorithms have a smaller overhead compared to the flooding algorithm.

Following up the specific example given in Section 3.4, the 32-node network as depicted in Figure 3.10 is considered here. The delay of a link is uniformly distributed between 1 to 100 time units. The simulation is run for 10,000 rounds. Assume in each dissemination round, there are k changes of delay on links where $k \geq 1$. Moreover, the

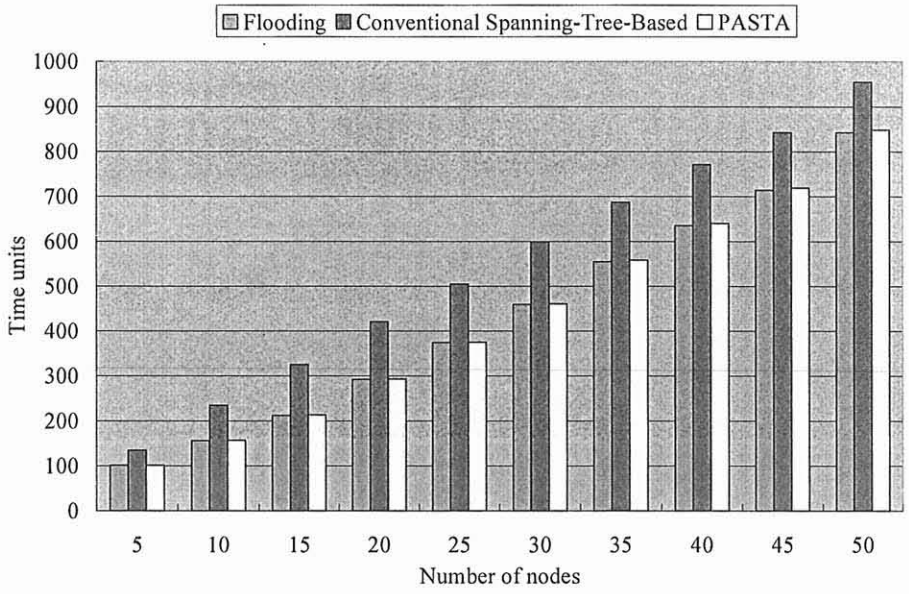

Figure 3.15 Comparison of average convergence time for different dissemination methods with multiple link changing events.

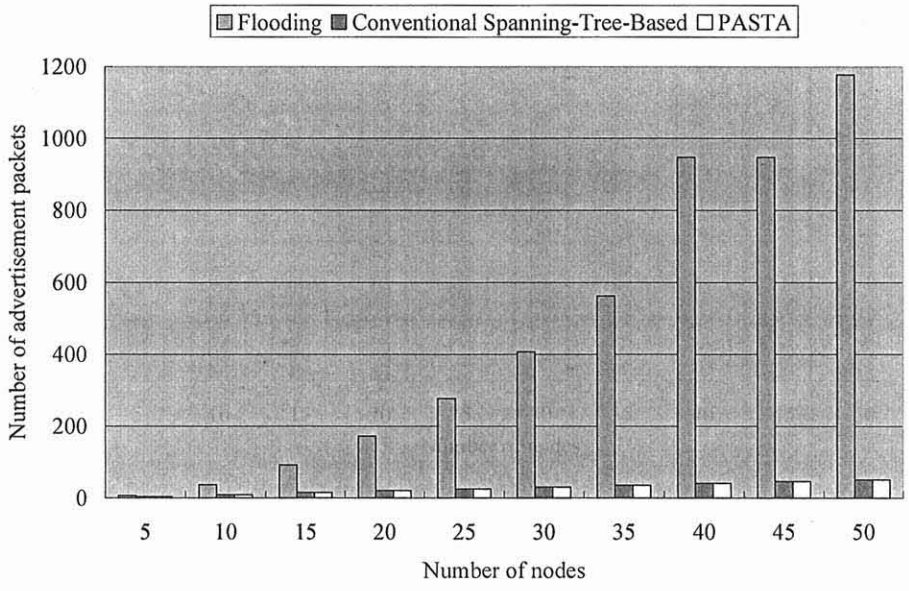

Figure 3.16 Comparison of average overhead for different dissemination methods with multiple link changing events.

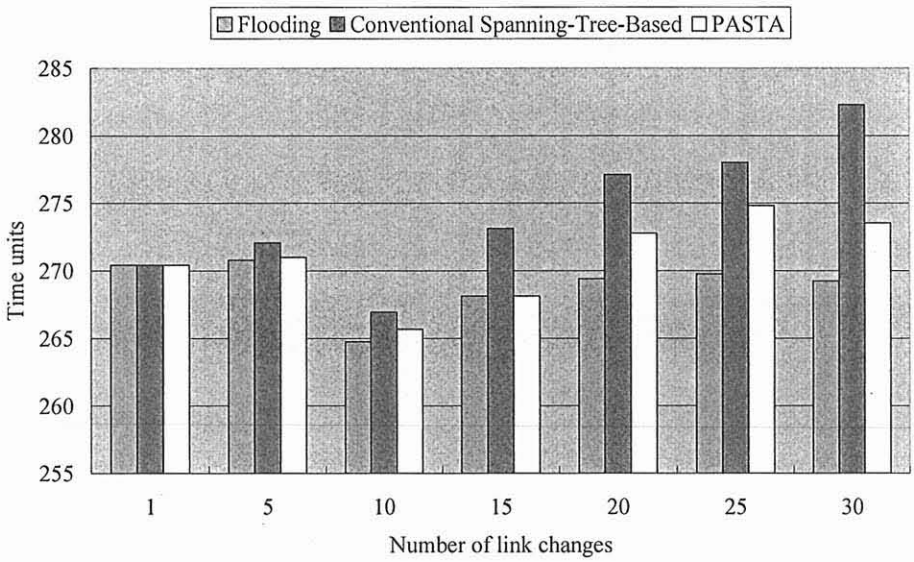

Figure 3.17 Comparison of average convergence time for 32-node network with simultaneously link changing.

root node of the spanning tree is randomly selected in each round. Firstly, the scenario that all link changes happen on the same time slot is simulated. Figure 3.17 illustrates the average convergence time for flooding, conventional spanning-tree-based algorithm, and the PASTA algorithms when the link state changing does not cause dissemination failure. The results show that the average convergence time of link state distribution by PASTA and flooding algorithm are almost the same, and both are shorter than that of the conventional spanning-tree-based algorithm. It can be seen that the time difference between conventional spanning-tree-based algorithm and PASTA/flooding increases when the number of link changes increase. It is because the larger the link changes, the larger the possibility of delay increment on the conventional spanning tree, but the PASTA algorithm can adjust the tree to fit for the current situation so the convergence time is not affected much.

The dissemination overhead between these three algorithms in above scenario is also compared. As depicted in Figure 3.18, the dissemination overhead of PASTA and conventional spanning-tree-based algorithms is significantly smaller than that of flooding algorithm.

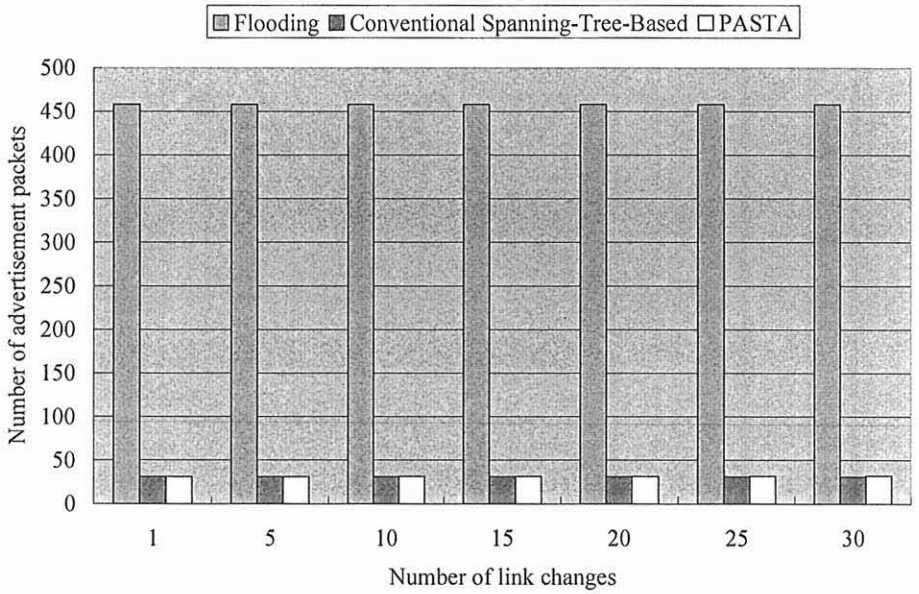

Figure 3.18 Comparison of average overhead for 32-node network with simultaneously link changing.

Then the failure probability of each link is set as 0.01. The simulation is run for 10,000 rounds. The dissemination failure rate is used to compare the robustness of the algorithm rate, which is defined as the number of dissemination failure events divided by the total number of dissemination rounds. Table 3.2 shows the comparison results, from which it can be seen that conventional spanning-tree-based algorithm is more frangible than the other two.

The second scenario simulated is that the links states can change on various time slots. Figure 3.19 compares the average convergence time for flooding, conventional spanning-tree-based algorithm and PASTA when the link changing does not cause dissemination failure. Similar to the results of the above scenario, here the average convergence time of the distribution by PASTA and flooding are both shorter than that supported by conventional spanning-tree-based algorithm. The time differences between conventional spanning-tree-based algorithm and PASTA/flooding are also larger when the number of link changes increase.

Table 3.2 Dissemination Failure Rate for 32-node Network with Simultaneously Link Changing.

Number of link changes	PASTA	Flooding	Conventional Spanning-Tree-Based
1	0	0	0.58%
5	0	0	2.87%
10	0	0	5.73%
15	0	0	8.60%
20	0	0	11.51%
25	0	0	14.36%
30	0	0	17.24%

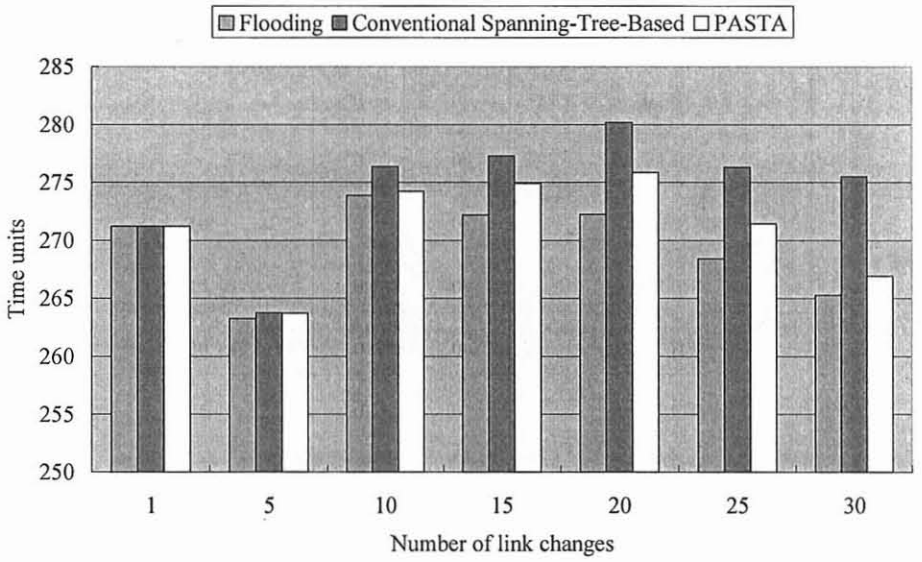

Figure 3.19 Comparison of average convergence time for 32-node network with link changing on various time slots.

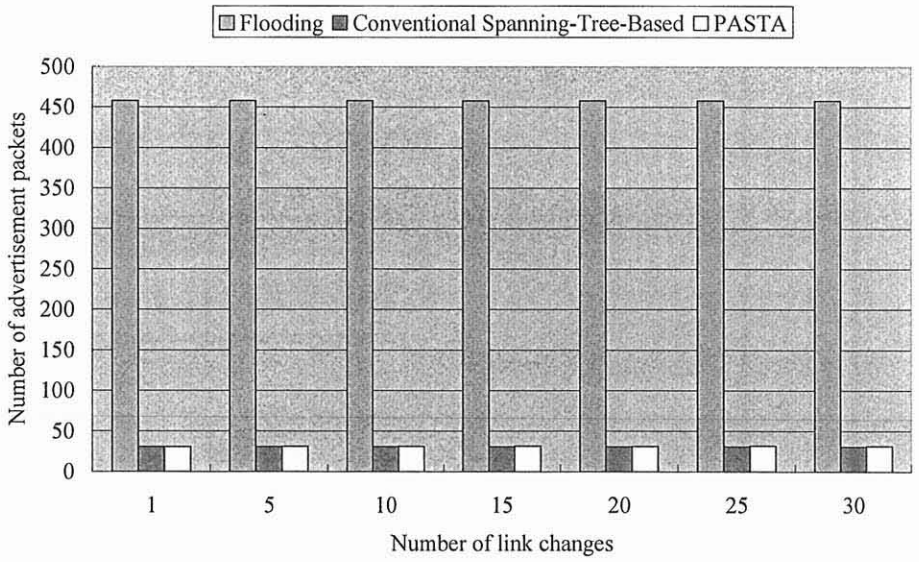

Figure 3.20 Comparison of average overhead for 32-node network with link changing on various time slots.

Figure 3.20 illustrates the comparison results of dissemination overhead between the algorithms in this scenario. It is easily seen that PASTA and conventional spanning-tree-based algorithms have a smaller overhead than the flooding algorithm.

Via setting the failure probability of each link as 0.01 and running the simulation for 10,000 times, the failure rate of the algorithms is compared in such scenario. As described in Table 3.3, conventional spanning-tree-based algorithm has a larger dissemination failure rate than PASTA and flooding algorithm, which indicates its robustness is much less than the latter two algorithms.

The above simulation results further approves that PASTA algorithm is more efficient to disseminate the links states with lower overhead than flooding algorithms, and more robust than conventional spanning-tree-based algorithm. The convergence time of PASTA is almost at same level as flooding algorithm, which is the fastest among all the algorithms compared here.

Table 3.3 Dissemination Failure Rate for 32-node Network with Link Changing on Various Time Slots.

Number of link changes	PASTA	Flooding	Conventional Spanning-Tree-Based
1	0	0	0.58%
5	0	0	2.87%
10	0	0	5.72%
15	0	0	8.60%
20	0	0	11.50%
25	0	0	14.33%
30	0	0	17.22%

3.6 Summary

In this chapter, a novel per-hop based partial spanning-tree adjustment (PASTA) algorithm is proposed to overcome the inefficiency problem of the current single-spanning-tree approaches. Combined with multiple spanning tree distribution and back-trace algorithm, this scheme is able to provide protection in case of the unavailability of the link in each spanning tree. The spanning tree is searched by the QoS performance of the links, so the link with fast transmission and high reliability capability is always selected. Through the analysis of the overhead cost and time complexity, it is proved that PASTA has the same order of complexity as other tree-based link state distribution algorithms, but its convergence time and reliability are greatly enhanced, and its overhead is kept low when compared to flooding based algorithms. A multiple spanning tree scheme for reliability and the back-trace mechanism for reconstructing a disconnected tree were proposed. All these mechanisms are used together for efficiently disseminate link-state information in a QoS network.

CHAPTER 4

OSPF-BASED ADAPTIVE AND FLEXIBLE QOS ROUTING

In this chapter, a QoS-enabled routing scheme is proposed to avoid false routing or low network utilization when using service vectors presented in Chapter 2. The routing scheme is based on OSPF protocol so as to guarantee feasible deployability of service vectors in existing networks. Furthermore, a network architecture is introduced to integrate security into the set of QoS parameters.

4.1 Introduction

As analyzed in Chapter 1, Intserv and Diffserv are two paradigms that differ in the level of accuracy on service provisioning and QoS granularity for a scalable implementation. Therefore neither one can satisfy a large number of service requirements and provide high service granularity at the same time. One way to solve this problem is by using a nested Diffserv model, where each group of flows can have a subset of requirements. This model can be combined with the explicitly endpoint admission control (EEAC) scheme [22] that represents the nested-Diffserv service levels as service vectors (SVs). The EEAC scheme can be performed in two phases: the probing (or exploring) phase, to determine link state, and the data transmission phase, which is performed after the probing (and call acceptance) processes. In the probing phase, the end host sends probing packets to the destination host to collect the SV information, which includes the service states of the routers along the end-to-end path. After receiving feedback information from the end server and retrieving the state from the probing packets, the end host compares all possible service class combinations for this specific path, and computes the utilization and cost to find the most suitable service classes to be used at each router per flow basis. The selected service vector is marked in the data packets during the data transmission phase. Each router checks

the vector, and provides the cost of the corresponding QoS service in it. This EEAC-SV model improves the QoS granularity to $O(p^q)$, where p is the number of routers and q is the number of service classes in the network (or end-to-end path). The flexibility feature increases the probability of minimizing the cost for the user and network utilization for the service provider. However, this scheme, as other EAC models, assumes that the path is pre-selected so that the probing path and data transmission path are always fixed. This assumption simplifies the analysis, but it may not be accurate in the case of considering a real networks. The reason is that in routing mechanisms, the above QoS provisioning scheme may not be able to provide the flexibility achievable by EEAC if SVs are not considered in the path calculation.

Here, OSPF is considered as the widely used QoS routing model. In Section 4.2, two examples of link-state routing are presented to show that the combination of OSPF and EEAC may cause false routing, which results in low utilization of the network resources and in high cost. To solve this problem, it is proposed here to select SVs during the path selection phase based on OSPF. The performance of different service classes of a link can be detected by the neighboring routers and disseminated by PASTA in a timely fashion. Furthermore, the concept of security-enabled QoS concept (SQoS) and a solution to achieve the optimal path selection are also presented introduced in this chapter.

4.2 Drawbacks of EEAC with Path Selection

Without loss of generality, OSPF can be used to select a path for the EEAC scheme. The weight of link is defined in OSPF to be inversely proportional to the capacity of the link [12]. This configuration cannot reflect the accurate QoS state of the network. [88, 95] extended OSPF by redefining the weight of link to reflect QoS performance of link, such as delay or available bandwidth. However, none of the above OSPF protocols can make the EEAC and other EAC algorithms survive from false routing.

Figure 4.1 shows an example of a simple network with routers $N = (N_1, N_2, N_3, N_4, N_5, N_6)$. Assume the delay of the path is determined by the QoS requirement. The service class $S = (S_1, S_2, S_3, S_4)$ is thus categorized by the delay of the link as shown below. The cost C of the service is represented as:

$$S = \begin{cases} S_1 & (delay < 5ms) \\ S_2 & (5ms \leq delay < 10ms) \\ S_3 & (10ms \leq delay < 20ms) \\ S_4 & (delay \geq 20ms) \end{cases}$$

$$C = \begin{cases} 6 & (S = S_1) \\ 3 & (S = S_2) \\ 2 & (S = S_3) \\ 1 & (S = S_4) \end{cases}$$

Here, $W_i, i \in (1, 10)$, as shown in the graph, is the weight of each link recorded by OSPF. Assume that a flow from N_1 to N_6 has a delay request of less than 25 ms. OSPF will select shortest path as $P_1 = (N_1, N_2, N_6)$ according to the addition of weights on the path. The EEAC-SV scheme then executes the probing processes along P_1, but no SV may satisfy the delay request for less than 25ms, so that the request is denied. However, the rest of the paths, such as $P_2 = (N_1, N_5, N_6)$ and $P_3 = (N_1, N_3, N_5, N_6)$, can satisfy the user's request with the proper service class selection. Furthermore, it is easy to see that the SV (S_2, S_2) with P_2 and with the lowest cost ($C = 3 + 3 = 6$) is the optimal solution. Therefore, the EEAC-SV scheme suffers of false routing in this case.

As another example, the link weight is set as the function of delay, as shown in the brackets in Figure 4.1, or:

$$W_i = f(delay) = \lceil delay \rceil .$$

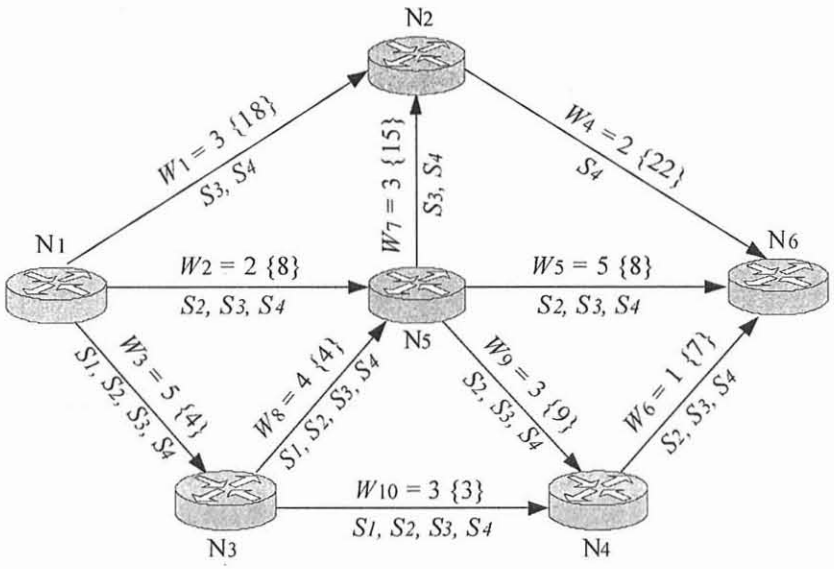

Figure 4.1 Network topology of the first example

Then $P_4 = (N_1, N_3, N_4, N_6)$ is the shortest path found by OSPF, which makes EEAC select (S_1, S_2, S_2) or (S_2, S_1, S_2) as the solution (where the cost is 12). However, the optimal answer in this case is $P_2 = (N_1, N_5, N_6)$ with service (S_2, S_2) in tandem (where cost is 6). The non optimal solution increases the cost and diminishes the network utilization.

4.3 OSPF-based Adaptive and Flexible QoS Provisioning

To overcome the above problem, a new architecture based on OSPF is proposed. In this description, the weight of link i (W_i) is proposed to be represented as a vector that contains all the available service classes the link can provide, documented as:

$$Wi = (S_1, S_2, ...S_k)$$
$$s.t. \quad (S_1, S_2, ...S_k) \in S \tag{4.1}$$

Generally, the service class in a QoS model is defined in function of various QoS parameters. Denote the service class as $S_i = (Q_i^1, Q_i^2, \ldots, Q_i^k)$, where $Q_i^j, j \in (1, \ldots, k)$ is the j_{th} QoS component in the i_{th} service class. For instance, a service class may be defined as

(delay, jitter, packet loss, available bandwidth). In this way, different service classes share the common network resources, so they can be compared by the QoS parameters and sorted in a linear order. In the proposed model, only the highest available service class needs to be marked as the weight of the link. However, the services of the lower classes can also be provided. Here, the amount of data used to represent the state class is reduced, that is, the overhead of link state update is decreased. Same as in the original OSPF protocol, the link states are exchanged by link state advertisement (LSA) packets among the routers in the network, so that every router has the same QoS link state database. When the user's request comes, the edge router, i.e. source node, selects the shortest path that satisfies Equation 2.3 in Chapter 2. All the service classes lower than or equal to the weight of the link are candidates for selection. The edge router here is different from that in generic QoS routing in the sense that the router not only selects the path the data go through but also the service classes of the link as requested. If the user's requirement can be satisfied, the SV as the selected service classes are marked into the data packets during data transmission. The traversed routers read the service class from each packet and provide the corresponding service. If no SV can be found to fulfill users' demand, the request is denied.

As for the link-state update, the router estimates the state of the connected link and launches the state update mechanism when any of the states change. However, the estimation of the performance of each service cannot be accurate because if the state is updated too frequently these updates generate large traffic overhead. Besides, to prevent the LSA packets and the following data packets from increasing their overhead, the values of the QoS parameters in each service class are divided into several service levels, so that $\lceil \log_2 M \rceil$ bits can represent M service classes.

OSPF sets a link to disseminate its state information every 30 minutes [12]. This update interval may be too large for the architecture discussed here, because the dynamic changes of the QoS parameters may cause the state already known to other routers to be-

come outdated. For such purposes, the update is triggered when the state of a link crosses a boundary of service level, which is called as a class-based triggering mechanism [86].

4.4 Combination of Security and QoS

It is well known that the security level of a given communication depends on the individual user. Therefore, it is difficult to evaluate security uniformly and classify its service level [96]. In order to shape the problem, the security protection capability of the router is estimated and it is linearly mapped to the level of security satisfaction of users. The most widely considered security capabilities of the router can be listed as encryption, DoS detection, authentication, and virus filtering [97]. Besides these, more security components can be extended in this framework. In this chapter the following criterion is created to evaluate them, but other standards are equally applicable:

- Encryption K_e: measured by the bit-length of the encryption key (e.g., 64 bits, 128 bits) and strength of cryptographic algorithm (e.g., RSA, DES).
- Virus scanning K_v: measured by the number of viruses and worms that the anti-virus software can detect and the false-alarm ratio.
- DoS detection K_d: measured by the false positive ratio of intrusion detection system (IDS) under uniform DoS attack testing.
- Authentication K_a: measured according to the robustness of the authentication mechanism. It might contain a weak or strong password, biometric, and smart cards with on-board display and input interfaces.

Here, link security state is expressed as a vector K_a, K_v, K_d, K_a. From the user's viewpoint, the security of link i is the addition of the above four components:

$$K_i = a_1 K_e^i + a_2 K_v^i + a_3 K_d^i + a_4 K_a^i \tag{4.2}$$

Assume there are n links in the path under study, and that the security level of the path is the link with minimum security:

$$K_p = \min(K_1, K_2, \ldots, K_n) \tag{4.3}$$

a_i, $i \in (1, 2, 3, 4)$ is the sensitivity weight of individual security component, as a user is concerned about the different security components that are specific to a given situation. As security protection capability mentioned here is considered to change at lower rates than the other QoS parameters, the security values are updated with a low frequency, such as once every 24 hours. During each update interval, this value is considered to remain constant in each service class. This decreases the computation work of routers and produces no increases in the complexity of the link state update.

Equation 2.3 in Chapter 2 is utilized for service vector selection. The cost of the security service S_i in each router is related to the processor occupancy time and strength level $C_p(S_i)$, the occupied memory $C_m(S_i)$, the bandwidth $C_b(S_i)$, and the disk space $C_d(S_i)$ for a database to store virus or DoS attack patterns. The cost function of security service is the sum of all of them:

$$C_{security}(S_i) = C_p(S_i) + C_m(S_i) + C_b(S_i) + C_d(S_i). \tag{4.4}$$

4.5 Path Selection Algorithm Analysis

Generally data flows can specify their QoS requirements in terms of four parameters: the available bandwidth B_{req}, the maximum jitter request J_{req}, the maximum delay request D_{req}, and the minimum security requirement K_{req}. The path and SV selection problem thus can be described as the problem to maximize Equation 2.3 in Chapter 2 as long as it is eligible for the selected path p_j,

$$\begin{aligned} B_p(P_j) \geq B_{req}, \quad D_p(P_j) \leq D_{req} \\ J_p(P_j) \leq J_{req}, \quad K_p(P_j) \geq K_{req} \end{aligned} \tag{4.5}$$

From the user perspective, the utility function U reflects the degree of users' satisfaction to the QoS service. Users' QoS requirement can be elastic or inelastic. With elastic demand the user can tolerate some degree of service deterioration if QoS provisioning is lower than the user expected constraint; while inelastic ones means otherwise. Here only

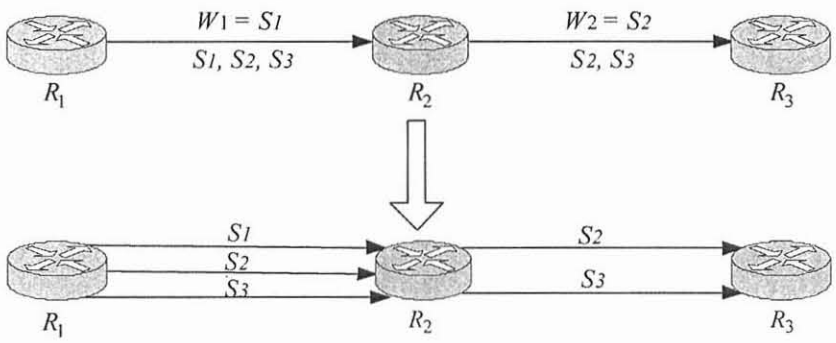

Figure 4.2 The illustration of virtual links with service class.

inelastic QoS requirements are considered so U is either 1 or 0. Maximizing Equation 2.3 in Chapter 2 is equivalent to minimize cost function C (i.e. the multi-constrained least cost routing with multi-service is selectable), which is an NP-complete problem [28]. Firstly consider the several service classes in each link. To convert their multiple-to-one relationship to one-to-one mapping, each service class is regarded as a virtual link, as Figure 4.2 shows.

To simplify the above problem, the users' constraints are categorized into two different classes. The algorithm is analyzed with each type of constraints. The combination of those algorithms gives the solutions but that is beyond the scope of this chapter. One class is called concave or bottleneck constrained, such as the cases for available bandwidth, and security. This can be solved by using an extension of the Dijkstra algorithm as in Figure 4.3. Assume a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of links. s and d are the source node and destination node, respectively. For the m_{th} service class and $|E|$ links in the graph G, the time complexity of the pre-process part is $O(m\,|E|)$; the time complexity of main-selection part is $O(n^2)$. Thereby the total complexity of this algorithm is $O(n^2)$, which has the same order of complexity as Dijkstra algorithm.

The other class is called as additive constrained, such as delay and jitter are. Assume the set of feasible paths from node s to d is F. The cost function is C. Then the problem is

QoS-Routing (G, s, d, K_{req})

Pre-Process Part

for each $e \in E$

$temp = \phi$

Set m service classes

for each $k \in m$

if $K_{e^k} \geq K_{req}$ **then**

$temp := temp \cup \{k\}$

Recode virtual links with lowest service class

$E := E - \left\{e^k\right\}, \forall k \in m$ except $k = \min\{temp\}$

Main Selection

$Dijkstra(E, s, d)$

Figure 4.3 The path selection algorithm for concave constraint.

defined as:

$$C = \min\{F\}$$

$$s.t. \quad \forall p_j \in F,\ D_{p_j} \leq D_{req}\ (or\ J_{p_j} \leq J_{req})$$

With respect to the lowest cost, the problem is known as Delay-Constrained-Least-Cost (DCLC). Many mechanisms are proposed to solve it in polynomial time, among which k-shortest paths (KSP) is a good solution. The proposed KSP scheme here is based on Jimenez and Marzal's Recursive Enumeration Algorithm (REA) [98]. The idea is to list k shortest paths from s to d with increasing costs of weight in a directed graph. The algorithm first invokes Dijkstra's shortest path algorithm to build the shortest path tree. Each path from s to current node v is the concatenation of the path from s to $pre(v)$ and the link $(pre(v), v)$, while $pre(v)$ is the adjacent predecessor node. The k_{th} shortest path $\pi^k(v)$ is thus selected from the candidate set $C^k(v)$ according to the following generalized Bellman's equation:

$$C^k(v) = \begin{cases} \pi^1(u) \cdot v \forall u \in pre(v)\ \ if\ k = 1\ \&\ v \neq s, \\ \qquad\qquad or\ k = 2\ \&\ v = s; \\ (C^{k-1}(v) - \pi^g(u).v) \cup \pi^{g+1}(u) \cdot v\ \ otherwise; \end{cases}$$

$$\pi^k(v) = \begin{cases} s & if\ k = 1\ \&\ v = s; \\ argmin_{\pi \in C^k(v)} L(\pi) & otherwise; \end{cases}$$

where $\pi^{k-1}(u) = \pi^g(u) \cdot v$ and $L(\pi)$ is the weight of the path π.

The above recursive computation to obtain the k shortest paths solution is finished in $O(m + Kn\log(m/n))$ time. To avoid the worst case when K is large, the following algorithm is proposed here for the DCLC problem. Once the delay constraint is above the threshold, the k_{th} longest path is tracked based on the longest path tree instead of the shortest one.

Build Shortest and Longest Path Tree

$path_{shortest} = Dijkstra(V, E);$

$path_{longest} = Dijkstra(V, E);$

$threshold = h * (Delay(path_{shortest})$

$+Delay(path_{longest}));$

K Paths Selection

if $Delay_{constraint} < Delay(path_{shortest})$ **then**

Request is denied

else if $Delay_{constraint} \leq threshold$

invoke k shortest path algorithm

else if $Delay_{constraint} \geq threshold$

invoke k longest path algorithm

$cost = \min(cost(\pi^i(d)), \forall i \in k)$

Figure 4.4 The path selection algorithm for additive constraint.

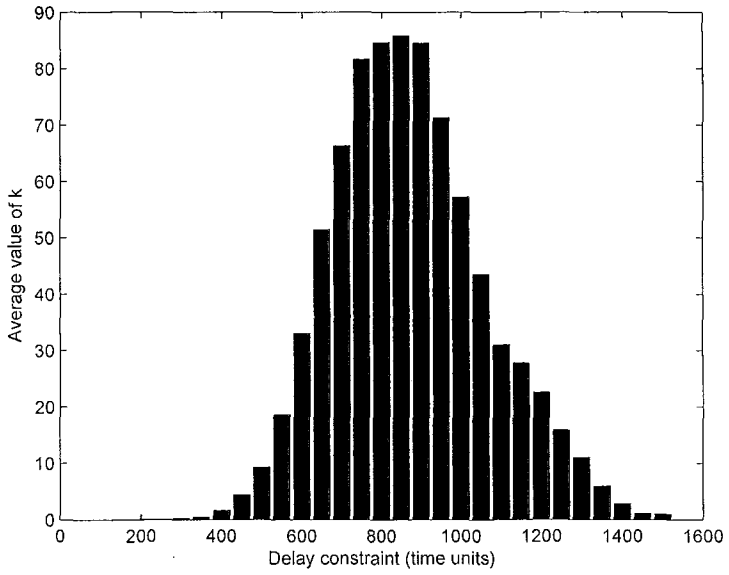

Figure 4.5 Average number of iterations of the proposed algorithm in 32-node network.

The algorithm depicted in Figure 4.4 is simulated in a 32-node bidirectional network in [99] by running 10,000 requests. Each link is replaced by three virtual links to represent the service classes. Without loss of generality, the delay of the virtual link is uniformly distributed from 1 to 500. The source and destination node is 1 and 30. Here, h is set to 0.5. The delay constraint set is from 150 to 1500 with 50 between intervals. Figure 4.5 shows the average number of iterations k, where k is found without setting the upper bound of k and considering that there are optimal feasible paths. The figure shows that the number of iterations is not large and has not unlimited increase when the delay constraint increases. In reality, as the service class is distributed uniformly among links, the variety of construction paths decreases. This means that the number of k is actually smaller than that shown in the figure.

4.6 Summary

In this chapter an extended OSPF framework is presented that SVs and path selection are integrated into one phase to provide the flexibility of QoS and high utilization of the net-

work. An efficient routing algorithm as well as the combination of security and QoS is introduced. To overcome the complexity of deployability of new mechanisms, SVs are embedded into OSPF routing to guarantee feasible deployability into existing networks.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

In this dissertation, the issue of QoS provisioning in next generation network has been addressed. The main contributions are:

1. Observing the importance of network measurement in QoS provisioning. The issue of active and passive measurement has been investigated. The framework of active measurement for QoS classes is proposed in the dissertation. A scheduling scheme has been proposed to resolve measurement tasks conflict problem.

2. For the purpose of fast and reliably disseminating link state information throughput the networks with low overhead, an efficient link state dissemination scheme has been proposed. This scheme enables each hop on the distributing tree to adjust the path according to latest link information, combined with back-trace scheme and multi-spanning-tree approach.

3. In QoS routing, multiple constraints need to be considered and high granularity of routing paths is required to improve network utilization. A KSP-based QoS routing algorithm is provided in this dissertation.

5.2 Future Work

The author will continue the ongoing work on QoS provisioning in next generation networks. Based on the proposed network framework, the author will focus on designing and implementing the network measurement tools and network provisioning strategy. Moreover, the author would like to implement the past research to wireless networks.

APPENDIX A

TEST RESULTS OF PING AND PIPECHAR

The table shown below records the test results of Ping and Pipechar in Chapter 2.

Table A.1 Test Result of Ping and Pipechar.

Load ($R_1 \rightarrow R_2$)	Load ($R_2 \rightarrow R_1$)	Ping	Pipechar		
		Average RTT(ms)	Available bandwidth(Mbps)	Minimal one-way delay(ms)	Average RTT(ms)
60%	60%	fail	fail	fail	fail
0	60%	0.509	13.884	6.70	14.12
50%	50%	fail	fail	fail	fail
0	50%	0.505	18.036	5.11	12.20
40%	40%	28.455	fail	fail	fail
0	40%	0.511	26.766	3.47	8.37
30%	30%	0.828	fail	fail	fail
0	30%	0.316	32.892	2.94	7.17
20%	20%	0.406	fail	fail	fail
0	20%	0.290	39.933	2.53	6.25
10%	10%	0.291	39.691	2.54	6.31
0	10%	0.263	48.913	2.20	5.10
0	0	0.252	58.347	1.97	4.60

APPENDIX B

TEST RESULTS OF PATHLOAD

The table shown below records the test results of Pathload in Chapter 2.

Table B.1 Test Result of Pathload.

Load	Load	Pathload	
($R_1 \rightarrow R_2$)	($R_2 \rightarrow R_1$)	Available bandwidth(Mbps)	Measurement Time(sec)
60%	60%	fail	N/A
0	60%	96.00-97.50	6.80
50%	50%	72.40-96.60	7.45
0	50%	95.70-97.20	6.83
40%	40%	95.30-96.80	6.81
0	40%	95.80-97.30	6.78
30%	30%	95.40-96.90	6.83
0	30%	95.70-97.20	6.81
20%	20%	95.90-97.40	6.80
0	20%	95.80-97.30	6.89
10%	10%	95.90-97.40	6.80
0	10%	95.80-97.30	6.81
0	0	95.80-97.30	6.81

REFERENCES

[1] Y. Xu and R. Guerin, "Individual QoS versus aggregate QoS: a loss performance study," *IEEE/ACM Trans. Networking*, vol. 13, pp. 370–383, Apr. 2005.

[2] X. Xiao and L. Ni, "Internet QoS: a big picture," *IEEE Network Mag.*, vol. 13, pp. 8–18, Mar. 1999.

[3] 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," in *ACM SIGCOMM Computer Communication Review*, Apr. 2001, pp. 25–37.

[4] S. Floyd and V. Paxson, "Difficulties in simulating the internet," *IEEE/ACM Trans. Networking*, vol. 9, pp. 392–403, Aug. 2001.

[5] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the self-similar nature of ethernet traffic (extended version)," *IEEE/ACM Trans. Networking*, vol. 2, pp. 1–15, Feb. 1994.

[6] IETF working group on IP Performance Metrics (ippm). [Online]. Available: http://www.ietf.org/html.charters/ippm-charter.html

[7] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis, "Framework for IP performance metrics," RFC 2330, IETF, May 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2330.txt

[8] *General Aspects of Quality of Service and Network Performance in Digital Networks, including ISDNs*, ITU-T Recommendation I.350, Mar. 1993.

[9] *Internet Protocol Data Communication Service - IP Packet Transfer and Availability Performance Parameters*, ITU-T Recommendation Y.1540, Mar. 2000.

[10] *Network performance objectives for IP-based services*, ITU-T Recommendation Y.1541, Feb. 2006.

[11] *End-User Multimedia QoS Categories*, ITU-T Recommendation G.1010, Nov. 2001.

[12] J. Moy, "OSPF version 2," RFC 2328, IETF, Apr. 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2328.txt

[13] IETF working group on Open Shortest Path First IGP (ospf). [Online]. Available: http://www.ietf.org/html.charters/ospf-charter.html

[14] IETF working group on Common Control and Measurement Plane (ccamp). [Online]. Available: http://www.ietf.org/html.charters/ccamp-charter.html

[15] IETF working group on Integrated Services (intserv). [Online]. Available: http://www.ietf.org/html.charters/OLD/intserv-charter.html

[16] IETF working group on Differentiated Services (diffserv). [Online]. Available: http://www.ietf.org/html.charters/OLD/diffserv-charter.html

[17] IETF working group on Resource Reservation Setup Protocol (rsvp). [Online]. Available: http://www.ietf.org/html.charters/OLD/rsvp-charter.html

[18] J. Babiarz, K. Chan, and F. Baker, "Configuration guidelines for DiffServ service classes," RFC 4594, IETF, Aug. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4594.txt

[19] Y. Bernet *et al.*, "A framework for integrated services operation over Diffserv networks," RFC 2998, IETF, Nov. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2998.txt

[20] L. Breslau, E. Knightly, S. Shenker, I. Stoica, and H. Zhang, "Endpoint admission control: Architectural issues and performance," in *Proc. ACM SIGCOMM Conf.*, no. 4, Aug. 2000, pp. 69–81.

[21] F. Kelly, P. Key, and S. Zachary, "Distributed admission control," *IEEE J. Select. Areas Commun.*, vol. 18, pp. 2617–2628, Dec. 2000.

[22] J. Yang, J. Ye, S. Papavassiliou, and N. Ansari, "A flexible and distributed architecture for adaptive end-to-end QoS provisioning in next-generation networks," *IEEE J. Select. Areas Commun.*, vol. 23, pp. 321–333, Feb. 2005.

[23] V. Sharma and M. Suma, "Estimating traffic parameters in Internet via active measurements for QoS and congestion control," in *Proc. IEEE GLOBECOM*, Nov. 2001, pp. 2527–2531.

[24] P. Barford and J. Sommers, "Comparing probe- and router-based packet-loss measurement," *IEEE Internet Computing*, vol. 8, no. 5, pp. 50–56, Sep. 2004.

[25] R. Kapoor, L.-J. Chen, L. Lao, M. Gerla, and M. Y. Sanadidi, "CapProbe: a simple and accurate capacity estimation technique," in *ACM SIGCOMM Computer Communication Review*, Oct. 2004, pp. 67–78.

[26] "Cisco IOS NetFlow," White Paper, Cisco, Oct. 2007. [Online]. Available: http://www.cisco.com/en/US/products/ps6601/prod_white_papers_list.html

[27] N. Brownlee, C. Mills, and G. Ruth, "Traffic flow measurement: Architecture," RFC 2722, IETF, Oct. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2722.txt

[28] Z. Wang and J. Crowcroft, "Quality of service routing for supporting multimedia applications," *IEEE J. Select. Areas Commun.*, vol. 14, no. 7, pp. 1228–1234, Sep. 1996.

[29] R. Guerin and A. Orda, "QoS based routing in networks with inaccurate information: theory and algorithms," *IEEE/ACM Trans. Networking*, vol. 7, no. 3, pp. 350–364, Jun. 1999.

[30] D. Ghosh, V. Sarangan, and R. Acharya, "Quality-of-service routing in IP networks," *IEEE Trans. Muitimedia*, vol. 3, pp. 200–208, Jun. 2001.

[31] W. Tsai, C. Ramamoorthy, W. Tsai, and O. Nishiguchi, "An adaptive hierarchical routing protocol," *IEEE/ACM Trans. Networking*, vol. 38, pp. 1059–1075, Aug. 1989.

[32] L. Ciavattone, A. Morton, and G. Ramachandran, "Standardized active measurements on a tier 1 IP backbone," *IEEE Commun. Mag.*, vol. 41, pp. 90–97, Jun. 2003.

[33] T. Tsugawa *et al.*, "Inline bandwidth measurements: Implementation difficulties and their solutions," in *IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services (E2EMON)*, May 2007.

[34] G. D. Santos *et al.*, "UAMA: a unified architecture for active measurements in IP networks; End-to-end objetive quality indicators," in *IEEE/IFIP Integrated Network Management Symposium*, May 2007, pp. 246–253.

[35] M. Zhanikeev *et al.*, "Active performance measurement for IP over all-optical networks," in *IEEE/IFIP Int. Conf. in Central Asia on Internet*, Sep. 2006.

[36] M. Zhanikeev and Y. Tanaka, "A testbed for agent-based multi-purpose extensible active measurement," in *Int. Conf. on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM)*, Mar. 2006.

[37] M. Mushtaq and T. Ahmed, "Adaptive packet video streaming over P2P networks using active measurements," in *Proc. IEEE Computers and Communications Symposium*, Jun. 2006, pp. 423–428.

[38] R. Mishra and V. Sharma, "QoS routing in MPLS networks using active measurements," in *IEEE Conf. on Convergent Technologies for Asia-Pacific Region (TENCON)*, Oct. 2003, pp. 323–327.

[39] M. Zangrilli and B. Lowekamp, "Comparing passive network monitoring of grid application traffic with active probes," in *Proc. Int. Workshop on Grid Computing (GRID)*, Nov. 2003, pp. 84–91.

[40] M. Aida, K. Ishibashi, and T. Kanazawa, "CoMPACT-Monitor: change-of-measure based passive/active monitoring weighted active sampling scheme to infer QoS," in *Proc. Applications and the Internet (SAINT) Workshops*, Feb. 2002, pp. 537–542.

[41] P. Calyam, D. Krymskiy, M. Sridharan, and P. Schopis, "Active and passive measurements on campus, regional and national network backbone paths," in *Proc. IEEE Conf. on Computer Communications and Networks (ICCCN)*, Oct. 2005, pp. 537–542.

[42] K. Mase and Y. Toyama, "End-to-end measurement based admission control for VoIP networks," in *Proc. IEEE Int. Conf. Commun.*, Apr. 2002, pp. 1194–1198.

[43] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy, "Bandwidth estimation: Metrics, measurement techniques, and tools," *IEEE Network Mag.*, vol. 17, no. 6, pp. 27–35, Nov. 2003.

[44] M. Luckie and A. McGregor, "IPMP: IP measurement protocol," in *Proc. Passive and Active Measurement (PAM) Workshop*, Apr. 2002, pp. 168–176.

[45] S. Shalunov and B. Teittelbaum, "One-way active measurement protocol (OWAMP)," RFC 3763, IETF, Apr. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3763.txt

[46] Z. Qin, R. Rojas-Cessa, and N. Ansari, "Distributed link-state measurement for QoS routing," in *Proc. Military Communications Conf.*, Oct. 2006.

[47] Y. Labit, P. Owezarski, and N. Larrieu, "Evaluation of active measurement tools for bandwidth estimation in real environment," in *IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services (E2EMON)*, May 2005, pp. 71–85.

[48] Active measurement project (AMP). National laboratory for Applied Network Research (NLANR). [Online]. Available: http://www.nlanr.net/

[49] F. Strohmeier, H. Dorken, and B. Hechenleitner, "Aquila distributed QoS measurement," in *In Proc. of COMOCON8 Conference*, 2001, pp. 177–185.

[50] Pipechar. [Online]. Available: http://www-didc.lbl.gov/NCS/

[51] Pathload. [Online]. Available: http://www.cc.gatech.edu/fac/Constantinos.Dovrolis/bw-est/pathload.htm%l

[52] K. Anagnostakis, M. Greenwald, and R. Ryger, "Cing: measuring network-internal delays using only existing infrastructure," in *Proc. IEEE INFOCOM*, Mar. 2006, pp. 2112–2121.

[53] A. Downey. Clink: a tool for estimating internet link characteristics. [Online]. Available: http://allendowney.com/research/clink/

[54] K. Lai and M. Baker, "Nettimer: A tool for measuring bottleneck link bandwidth," in *Proc. of the USENIX Symposium on Internet Technologies and Systems*, Mar. 2001.

[55] C. Dovrolis, P. Ramanathan, and D. Moore, "What do packet dispersion techniques measure?" in *Proc. IEEE INFOCOM*, Apr. 2001, pp. 905–914.

[56] Pathchar. [Online]. Available: http://www.caida.org/tools/utilities/others/pathchar/

[57] J. Sommers, P. Barford, and W. Willinger, "Laboratory-based calibration of available bandwidth estimation tools," *Microprocess. Microsyst*, vol. 31, pp. 222–235, Jun. 2007.

[58] R. S. Prasad, C. Dovrolis, and B. A. Mah, "The effect of layer-2 store-and-forward devices on per-hop capacity estimation," in *Proc. IEEE INFOCOM*, Apr. 2003, pp. 2090–2100.

[59] B. Melander, M. Bjorkman, and P. Gunningberg, "A new end-to-end probing and analysis method for estimating bandwidth bottlenecks," in *Proc. IEEE GLOBECOM*, Nov. 2000, pp. 415–420.

[60] G. Almes, S. Kalidindi, and M. Zekauskas, "A one-way delay metric for IPPM," RFC 2679, IETF, Sep. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2679.txt

[61] IETF working group on Network Time Protocol (ntp). [Online]. Available: http://www.ietf.org/html.charters/ntp-charter.html

[62] OWAMP. Internet2. [Online]. Available: http://e2epi.internet2.edu/owamp/

[63] J. Choi and C. Yoo, "One-way delay estimation and its application," *Elsevier Computer Commun.*, vol. 28, no. 7, May 2005.

[64] G. Almes, S. Kalidindi, and M. Zekauskas, "A one-way packet loss metric for IPPM," RFC 2680, IETF, Sep. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2680.txt

[65] C. Estan, K. K. adn D. Moore, and G. Varghese, "Building a better NetFlow," in *Proc. ACM SIGCOMM Conf.*, no. 4, Aug. 2004.

[66] J. Cleary, S. Donnelly, I. Graham, A. McGregor, and M. Pearson, "Design principles for accurate passive measurement," in *Proc. Passive and Active Measurement (PAM) Workshop*, Apr. 2000.

[67] Traffic monitoring using sFlow. sFlow. [Online]. Available: http://www.sflow.org/sFlowOverview.pdf

[68] IETF working group on Remote Network Monitoring (rmonmib). [Online]. Available: http://www.ietf.org/html.charters/OLD/rmonmib-charter.html

[69] "Smartbits: White papers," Spirent Inc. [Online]. Available: http://www.spirent.com/Solutions-Directory/Smartbits.aspx

[70] N. Nu and P. Steenkiste, "Evaluation and characterization of available bandwidth probing techniques," *IEEE J. Select. Areas Commun.*, vol. 21, no. 6, pp. 879–894, Aug. 2003.

[71] E2E piPEs. Internet2. [Online]. Available: http://e2epi.internet2.edu/e2epipes/

[72] J. Sommers and P. Barford, "An active measurement system for shared environments," in *Proc. ACM SIGCOMM Conf. on Internet Measurement (IMC)*, Oct. 2007.

[73] R. Graham, E. Lawler, J. Lenstra, and A. Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey," in *Annals of Discrete Mathematics*, 1979, pp. 5:287–326.

[74] T. McGregor, H. Braun, and J. Brown, "The NLANR network analysis infrastructure," *IEEE Commun. Mag.*, vol. 38, pp. 122–128, May 2000.

[75] S. Kalidindi and M. Zekauskas, "Surveyor: An infrastructure for internet performance measurements," in *Internet Global Summit(INET)*, Jun. 1999.

[76] R. Wolski, N. Spring, and C. Peterson, "Implementing a performance forecasting system for metacomputing: The network weather service," in *Supercomputing*, Aug. 1997.

[77] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[78] K. Jeffay, D. Stanat, and C. Martel, "On non-preemptive scheduling of periodic and sporadic tasks," in *Proc. IEEE Real-Time Systems Symposium*, Dec. 1991, pp. 129–139.

[79] Y. Cai and M. Kong, "Nonpreemptive scheduling of periodic tasks in uni- and multiprocessor systems," *Algorithmica*, vol. 15, no. 6, pp. 572–599, Jun. 1996.

[80] P. Calyam *et al.*, "Enhanced EDF scheduling algorithms for orchestrating network-wide active measurements," in *Proc. IEEE Real-Time Systems Symposium*, Dec. 2005.

[81] I. Gopal, M. Bonuccelli, and C. Wong, "Scheduling in multibeam satellites with interfering zones," *IEEE Trans. Commun.*, vol. 31, no. 8, pp. 941–951, Aug. 1983.

[82] W. Chen, P. Sheu, and J. Yu, "Time slot assignment in TDM multicast switching systems," in *Proc. IEEE INFOCOM*, Apr. 1991, pp. 1296–1305.

[83] A. Bagchi and S. Hakimi, "Data transfers in broadcast networks," *IEEE Trans. Computers*, vol. 41, no. 7, pp. 842–847, Jul. 1992.

[84] J. Goossens and C. Macq, "Limitation of the hyperperiod in real-time periodic task set generation," in *Proc. RTS Embedded System (RTS'01)*, 2001, pp. 133–147.

[85] "OSPF flooding reduction," Cisco IOS Software, May 2000. [Online]. Available: http://www.cisco.com/en/US/docs/ios/12_1t/12_1t2/feature/guide/dt_ospff.pdf

[86] G. Apostolopoulos *et al.*, "Quality-of-service based routing: A performance perspective," in *Proc. ACM SIGCOMM Conf.*, Oct. 1998, pp. 17–28.

[87] G. Cheng and N. Ansari, "ROSE: A novel link state information update scheme for QoS routing," in *Proc. IEEE High Performance Switching and Routing*, 2005, pp. 24–28.

[88] G. Apostolopoulos *et al.*, "QoS routing mechanisms and OSPF extensions," RFC 2676, IETF, Aug. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2676.txt

[89] J. Moy, "Flooding over a subset topology," Internet draft, IETF, Feb. 2001. [Online]. Available: http://tools.ietf.org/id/draft-ietf-ospf-subset-flood-00.txt

[90] B. Bellur and R. Ogier, "A reliable, efficient topology broadcast protocol for dynamic networks," in *Proc. IEEE INFOCOM*, vol. 1, Mar. 1999, pp. 178–186.

[91] K. Thulasiraman and M. Swamy, Eds., *Graphs: Theory and Algorithms*. New York: Wiley-Interscience, 1992.

[92] K. Long, S. Cheng, and J. Ma, "Internet QoS: architectures, strategies and mechanisms," *High Technology Letters 7 (1)*, pp. 13–21, 2001.

[93] N. Ansari, G. Cheng, and R. Krishnan, "Efficient and reliable link state information dissemination," *IEEE Commun. Lett.*, vol. 8, pp. 317–319, May 2004.

[94] P. Narvaez, K. Siu, and H. Tzeng, "New dynamic SPT algorithm based on a ball-and-string model," *IEEE/ACM Trans. Networking*, vol. 9, pp. 706–718, Dec. 2001.

[95] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world," *IEEE J. Select. Areas Commun.*, vol. 20, no. 4, pp. 756–767, May 2002.

[96] A. D. Kermytis, V. Misra, and D. Rubenstein, "SOS: Secure overlay services," Columbia University, Tech. Rep. EE200415-1, Feb. 2002.

[97] A. Chakrabarti and G. Manimaran, "Internet infrastructure security: A taxonom," *IEEE Network Mag.*, vol. 16, pp. 13–21, Nov. 2002.

[98] V. M. J. Pelayo and A. M. Varo, "Computing the k shortest paths: A new algorithm and an experimental comparison," in *Proc. 3rd Int. Workshop on Algorithm Engineering (WAE'99), Lecture Notes in Computer Science 1668*, Jul. 1999, pp. 15–29.

[99] S. Chen and K. Nahrsted, "On finding multi-constrained paths," in *Proc. IEEE Int. Conf. Commun.*, Jun. 1996, pp. 874–899.