TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

Ever since the ancient time, people have known how to communicate in a preliminary way. To facilitate and improve ways of communication, our ancestors built up systems of communication, and the first postal service, for instance, is one of the great improvements in communications before Warring States (403-221 B.C.) Through the message delivery system, distant messages could be exchanged in a certain way so that it has replaced the face-to-face communication. However, with the development of human civilizations, the message transmission system in postal service has also evolved from the preliminary dove to circuits and finally to the present circuit-switching network, packet-switching network and the upcoming ALL-IP network[1]. All-IP network is a network that uses Internet Protocol (IP) to transport of all user data and signaling. In order to carry time sensitive communication services, the transmission delay time in an ALL-IP network has to be controlled. To provide controlled delay time, we need proper mechanisms to measure delay time and select paths for request traffics. Routing is a critical task of a packet-switching network to decide the path to deliver a packet. However, most routing approaches are not for time sensitive services.

Only a few are designed for time sensitive services. These time sensitive routing algorithms are designed at the time when the link bandwidth is the scarcest resource. As the link bandwidth grows rapidly in recent years due to the advance of optical communication technologies, link bandwidth is no longer the only scarce resource. The processing speed of nodes, e.g. routers, becomes another critical source of delay time. Thus, we hypothesize that node delay is a significant part in time sensitive routing for high-speed packet-switching network. In this paper, we show the importance of node delay time and developed an algorithm to provide controlled delay time in high-speed packet-switching networks.

1.1 Switching Network Technologies

Current communication networks can be categorized into two types: circuit-switching networks to support time sensitive services, such as voice, and packet-switching networks to support other types of services, mainly non-time sensitive data services.

1.1.1 Circuit-Switching Networks

Since the first invention of telephone by Alexander Graham Bell in about 100 years ago, the development of tele-communication networks has become the foundation of the build-up of circuit-switching network. At first, telephone connections require operators who manually switched phone lines into correct positions so that the connection could be built up. After years of development, instead of manual services, such complicated and tiresome jobs are done automatically by private branch exchanges (PBXs), as in Figure 1.1. This system is very reliable (try to think how rarely the system fails to connect the receiver when you dialed the numbers correctly). Furthermore, circuit-switching network operates with very few overheads in transmitting datas, so it provides good quality, such as low delay and jitter, in real-time communications. However, it is also extremely inefficient and expensive because the connection is made at the beginning of a session; no matter it is a conversation, a fax transmission, a modem session or whatever based on the voice transfer. The connection is maintained until being terminated, that means a certain portion of the network is reserved exclusively for that connection whether something (voice) is taking place at the moment or not. If one party puts down the phone or remain silent, or if a fax machine is sending or receiving data for a period of time, the line(the circuit) between the two terminal devices is still unavailable for other activities even though it is not being used at the moment. In a conversation, while one is speaking and another is usually listening. There might be up to 50% of a typical voice conversation being actually silence. A tremendous network capacity is thus wasted.

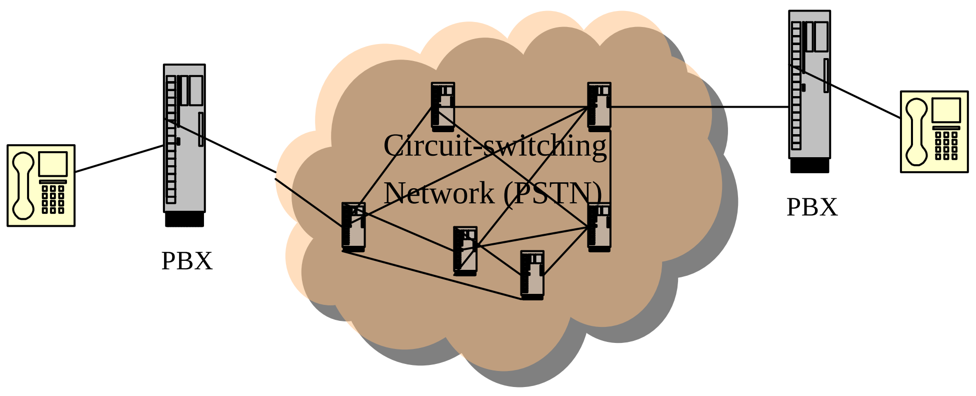


Figure 1.1: A circuit-switching network.

Although the system is inefficient and expensive, people still enjoy and get used to the convenience of circuit-switching system. With the rapid improvements in communication technologies, people of today can talk over the phone easily and inexpensively even they are apart by oceans. It provides high quality services to real-time voice communications that other networks hardly have. The word "Circuit-Switching Network" equaled to "Communications" in so many years. It was not until late 1960's that this domination started loosening, when packet-switching networks were introduced to the world.

1.1.2 Packet-Switching Networks

In 1960's, a new transmission approach was invented—packet-switching. It breaks the rules made by circuit-switching networks. Instead of occupying a line for the entire connection time, packet-switching networks break the digital stream of 1s and 0s into pieces of the some specific length. These pieces, or we may call them "packets", are then put in the so-called "header" envelopes with some information. In other words, when the packet is originated, a serial number and error correction messages are created and attached to the packet to indicate the sequence number of the packets. Similar to the role of switching connected and disconnected circuits in circuit-switching network, routers or computers in packet-switching networks take the functions. They read the address of a packet and forward it to another router closer to the destination. At the destination, a few hundreds of milliseconds or some seconds later, the packets are received, reassembled in correct order, and converted back into the original message. We show an example of how packet-switching works in Figure 1.2.

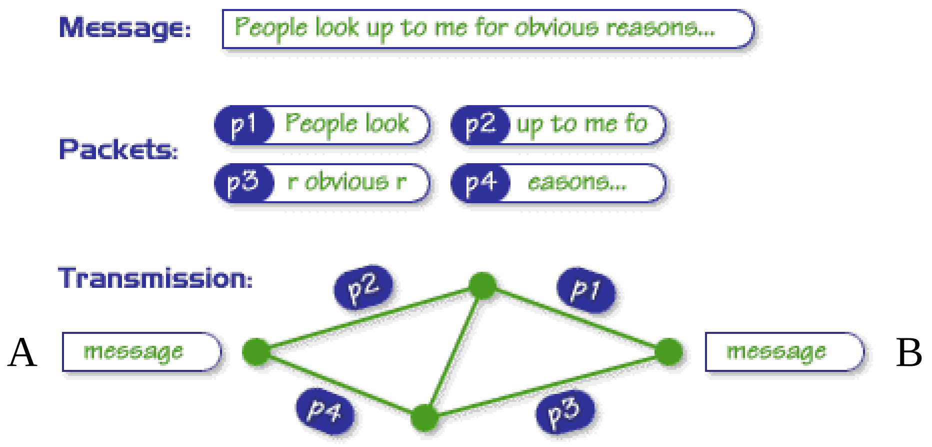


Figure 1.2: Sending a message over a network as a series of packets.

The main difference between circuit-switching network and packet-switching network is the improvement of link utilization. Although the session in packet-switching (4 links) may use more links than circuit-switching network (2 links), the links between A and B are not occupied when the session proceeds. It implies that the links could be released earlier for other sessions in packet-switching network than in circuit-switching network.

In some aspects, we may view a packet-switching network as a network of queues. Each node contains queues where incoming packets are queued before they are sent out on an outgoing link. If the rate of certain packets arrive at a switch point exceeds the rate a packets can be transmitted, the queues grow. When it happens, the queuing mechanism causes delay, and if the queues overflow, packets thus lost. It is so called congestion. Loss of data generally causes retransmissions that may either add to the congestion or result in less-effective utilization of the network.

In 1990's, Internet, which evolutes from ARPANET, bursts. ARPANET is a packet-switching network using Internet Protocol(IP)[23] as it's major communication protocol. With the popularity of Internet, IP becomes the most popular communication protocol to transmit data communication services. Researchers in packet-switching area focus on the bandwidth utilization, which is a method that may maximize bandwidth utilization but sacrifice packet delay time. The inherent packet delay time caused by the signal propagation, router processing, and packet transmitted over slow transmission links, is naturally ignored in the packet-switching network researches.

A brief comparison between circuit-switching and packet-switching networks is shown in Table 1.1.

Table 1.1: Circuit-switching Networks v.s. Packet-switching Networks.

Resource	**Circuit-switching**	**Packet-switching**
Dedicated path	Yes	No
Available bandwidth	Fixed	Dynamic
Reliability	Yes	No
In order delivery	Yes	No
Store-and-forward transmission	No	Yes
Call setup	Required	Not required
When congestion occurs	At setup time	On every packet
Charge	Per time unit	Per packet or per time unit

1.1.3 Network Convergence and ALL-IP Network

In so many years, these two types of switching techniques have built up high capacity networks in the real world. The cost of co-managing two types of networks is huge but the applications are usually not integrated well. For example, if someone wants to contact the customer service of a company, he/she has to look up the company web page for the customer service numbers. He/she cannot contact the customer service just via a mouse click, as surfing on the web. Even if the user applies ISDN, xDSL or other broadband services using the same line, he/she still cannot achieve it easily, because the services still are not integrated well.

It was not until recently that service providers have been required to deploy different networks for different applications, such as voice, video, private data, and etc. Each of these networks evolved independently as separate networks, so it was technically impossible to run all of these applications across a common infrastructure. Due to the existent technological limitations, each of these incredibly expensive parallel infrastructures was built to support different application requirements. Internet was just another example of a separate network that was designed to provide connections among various researchers, military and educational institutions. In general, users prefer an application independent network, which can serve for all purposes, not just for certain purposes. People really need to build an application independent network, a network with no specific application purpose. This network is built for the general purpose capable to carry all the traditional services on their own networks and to move them onto this common infrastructure.

The deployment of global IP infrastructure, as well as the recent technological advances, makes the integration possible. For service providers and their customers, they can get a better return on their assets, lower operational cost, and bring new services to a worldwide market. Therefore, the network convergence emerges. In response to the network usage explosion, huge bandwidth growth, multiple services and mutual connectivity of network, IP is chosen to be the common network layer protocol of next generation networks. An All-IP network uses a packet-switching network to carry all the traffics that were delivered by both packet-switching and circuit-switching networks. The delay requirement of real-time applications, such as VoIP, Video Conference, and even on-line games, etc, must be fulfilled on the ALL-IP networks. These time sensitive services can be easily served by a circuit-switching network, which connects end hosts via real circuits. Once the connections are set properly discarding the previous call set up time, the users only suffer little delay in communications. On the other hand, the delay time requirement of time sensitive services is a big challenge to the ALL-IP network. The inherit packet delay time becomes an important issue in managing an ALL-IP network.

Traditional researches in packet-switching network management field only take into account the packet transmission time over transmission links but the nodes. It is because the line prices were expensive and the bandwidth was poor in the past years. Link delay was the dominating factor in the traditional "slow" packet-switching networks. As the link bandwidth grows rapidly in recent years due to the advance of optical communication technologies, link bandwidth is no longer the only scarce resource. The processing speed of a node, e.g. a router, becomes another critical source of delay time. A network management mechanism must take node delay into account to achieve a better performance.

Table 1.2: Comparison of PSTN/IN, Internet and ALL-IP Network.

	Circuit-Switching (PSTN/IN)	**Packet-Switching (Internet)**	**ALL-IP Network**
Multimedia services	No	Yes	Yes
QoS-enabled	Yes (voice only)	No	Yes
Network intelligence	Yes	No	Yes
Intelligent CPE	No	Yes	Yes
Underlying transport network	Circuit-switching Network	Packet-switching Network	Packet-switching Network
Integrated control and management	No	Yes	Yes
Service reliability	High	Low	High
Service creation	Complex	Ad-hoc	Systematic
Ease of use of service	Medium	High	High
Modularity	Low	Medium	High
Architecture openness	Low	High	High

1.2 Delay Time Analysis

In this section, we will to illustrate the categories of delays, and explain why they happen and what we can do to improve them.

1.2.1 One Trip Delay Time

One trip delay time is the accumulation of delay time along a transmitting path. It is usually an important metric to measure the quality of a transmission. An example of the one trip delay is shown in Figure 1.3.

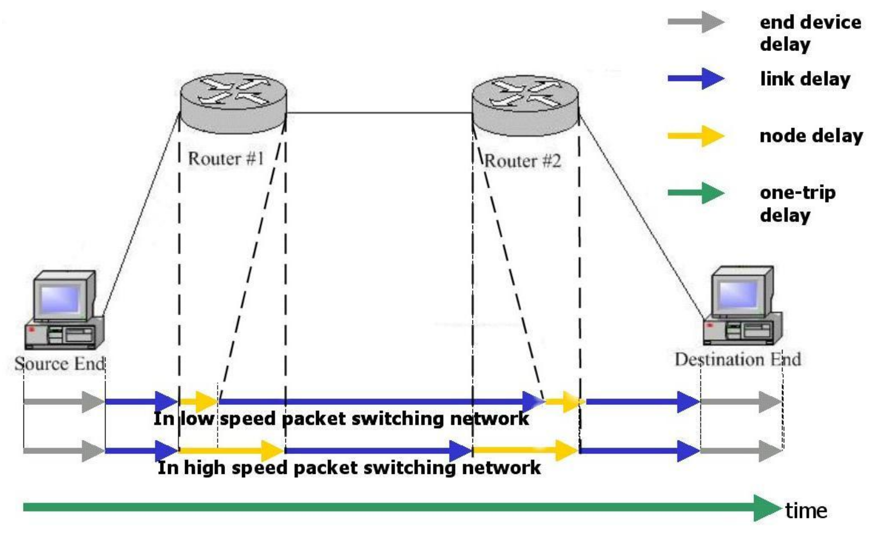


Figure 1.3: One trip delay time.

1.2.2 Categories of Delay Time Components

The components of the delay time in a path are categorized into the following three categories: link delay, node delay and end-host delay.

1.2.2.1 Link Delay

The first place that catches the developer's sight is the delays on links. This contains three types of delays, the queuing delay, the propagation delay and the transmission delay. Link delay was a major concern when people searches a path for time sensitive services.

1.2.2.1.1 Propagation Delay

Propagation delay is the time of an electrical or optical signal transmitted along a specific link. The propagation delay of a link is the length of the link divided by the propagation speed, which depends on the physical characteristics of the media and the signal. For the same link, the propagation delay is a constant if the length and the media keep unchanged.

For example, the propagation delay time of an xDSL line is the same as that of a voice line, because the transmission media is still a copper wire with the same length.

1.2.2.1.2 Transmission Delay

Transmission delay is the time of a data unit being transmitted alone a specific link with the propagation delay time ignored. For example, we may need 82ms to transmit a 128kb data, and 41ms to transmit a 64kb data along the T1 link. The transmission delay time is a measure of the link bandwidth. Users can reduce the transmission time by either reducing the data size or increasing link bandwidth.

1.2.2.1.3 Queueing Delay for a Link

Queueing delay for a link is defined as the time of a packet waiting in a queue, before it is being transmitted. It could be improved by using different queueing policy, such as changing the queue policy from FIFO to RED[2,3].

1.2.2.2 Node Delay

Node delay is the delay time occurs within a node. It contains two types of delays: the processing delay and the queuing delay for a processor. With link bandwidth gets closer to node processing capacity, node delay increases its share in one trip delay time. We may recall the impact caused by the Code-Red worm in 2001. Like the first Internet worm in 1988, the code-red worm invaded numerous of computers, and most of the invaded hosts were not aware of it. However, unlike other worms, this worm did not cause a very heavy traffic loading to the hosts, but generated many small-sized packets such that the mediate routers and switches cannot process them all instantaneously. These short packets did not occupy too much link bandwidth. Instead, they caused heavy loads on routers and switches and hence increased packet delay time tremendously. The routers and switches spent much time finding routes for numerous short worm packets but transferring real data traffics. The link loading was not heavy because the traffic was small.

This situation might be repeated when the tasks carried by routers are more and more complex. Then, node delay becomes a dominant part when choosing a path for a delay sensitive service in a high-speed network.

1.2.2.2.1 Processing Delay

Router’s processing power determines processing speed and processing delay time. For simplicity, we call it “node processing capacity”. The more data a processor handles, the longer processing delay will be. Furthermore, applying faster processor could reduce the processing delay.

1.2.2.2.2 Queueing Delay for a Node

Similar to the queueing delay for a link, it is defined as the time of a packet waiting in a queue before being processed. It could be improved by using different scheduling policy, such as changing policy from FIFO to Weighted Round-Robin (WRR.)

1.2.2.3 End-Host Delay

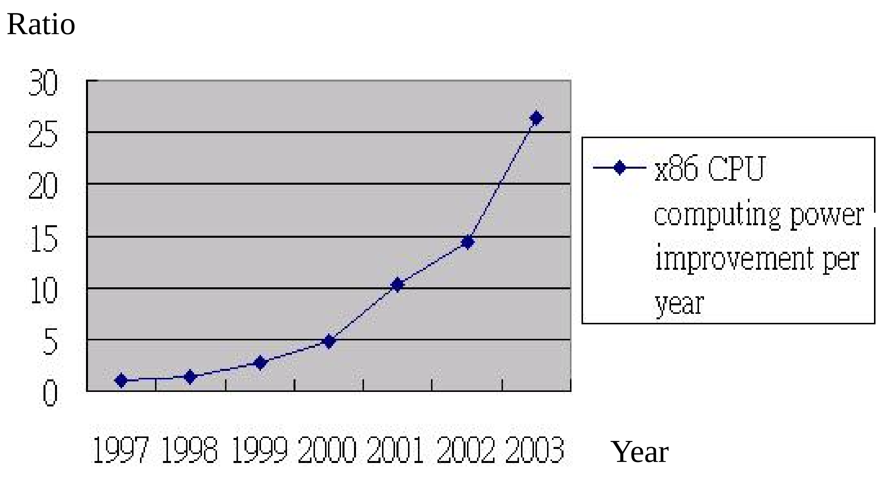
End-host delay is defined as the delay occurred at the hosts of both ends, including codec delay, packetiztion delay, and etc. Although it is a component of total delay time, it is independent of the path it travels. Thus, it is generally ignored in the design of routing algorithms.

1.2.2.3.1 Packetization Delay

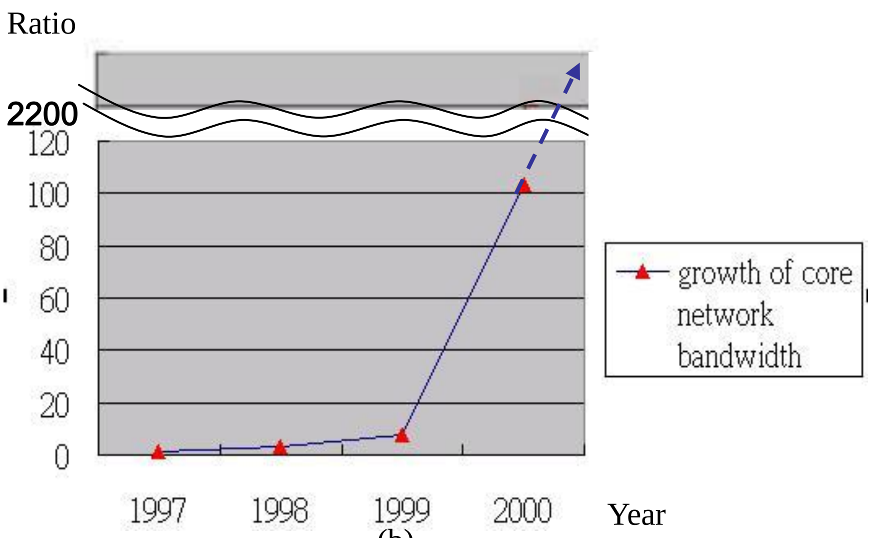
Packetization delay is the time to construct and format a packet. Some real time applications sends packets periodically. For example, some VoIP applications use G.729.1[4] as their communication protocol, sending packets every 30ms. That implies that there is an inevitable 30ms packetization delay at both sender end receiver end. The packetization delay can’t be reduced unless the communication protocol or the periodical parameters are changed.

1.2.3 The Myth of Bandwidth

In the beginning of router algorithm development, the tasks performed by a router are simple and the power of the processor within a router is much faster than the links in terms of processing or transmitting packets. The link delay was the major concern in designing a routing algorithm. In 1998, fiber optic networks with DWDM technique started to be deployed. It causes a dramatic increase in network bandwidth. Figure 1.4 shows the growth in bandwidth and node processing capacity. This faster growth in bandwidth makes link bandwidth closer to the node processing capacity and results in the increase of relative weigh of node delay. Therefore, both link and node delay time must be taken into account in designing a delay sensitive routing algorithm.



(a)



(b)

Figure 1.4: The growth rate in past few years:

(a) x86 CPU power capacity and (b) fiber bandwidth.

1.2.4 Possible Delay in Routers

The basic function of a router are routing and forwarding packets. As dot com fever outbroke in last decade, new data-communication applications and routing technologies emerge like waves. Being a critical component on the network, a router has to carry much more tasks than before, such as packet filtering and flow management. Hence, a router may not work as efficiently as it claims and may become a potential bottleneck in Internet.

1.2.4.1 MPLS Traffic Engineering (MPLS TE)

Label switching becomes a new approach to reduce node delay. A label switching router forwards packets according to labels instead of packet headers. This will increase processing speed in forwarding packets. The hottest example is MPLS Traffic Engineering(MPLS TE)[5, 6, 7]. It is originated from Cisco Tag-Switching. It coordinates routers to switch packets by labels, which is a 20bit-length header added to the front of a packet. Routers transmit packets by packet labels. A path selected by labels in MPLS network is called Label Switched Paths(LSP), shown in Figure 1.5.

The corresponding routing protocol routes packets based on a single end-to-end routing decision uniformly applied to all packets bound for the same destination. A single physical path between source and destination that delivers streaming packets in-sequence with uniform latency.

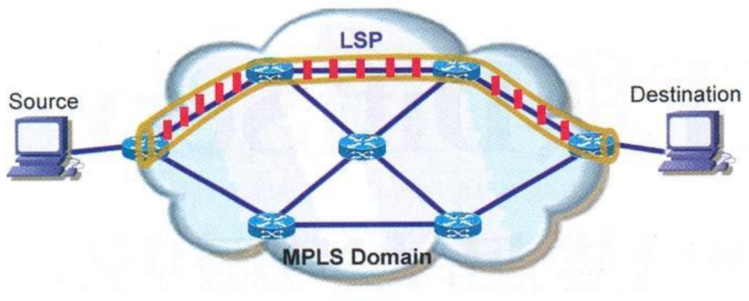


Figure 1.5: A Label Switched Path(LSP) in MPLS.

LSP technique reduces the routing table look up time as compared to the traditional header processing. Traffic Engineering (TE) is concerned with performance optimization of operational networks. In general, it measures, models, characterizes and controls the applications run on a specific network to achieve specific performance objectives. MPLS Traffic Engineering is the way to implement TE within MPLS domain. It aggregates all possible resources and prepares for potential failures to achieve the optimization descried above. The disadvantage of MPLS TE is the real-life of MPLS. In practice, routers that claim being with MPLS functions usually only support IP Label, but not other functions. These poor supports in functions may not speed up the routing when applying MPLS, but will certainly decrease the efficiency, just as the extra load balancing jobs do.

1.3 Motivation and Research Objectives

Routing is a critical task of a packet-switching network to decide the paths to deliver a packet. Most traditional routing algorithms do not treat the delay time as their major concern, because most packet-switching networks are not designed to support time sensitive services. To maximize the utilization of link bandwidth, the delay time is usually sacrificed.

Only a few are designed for time sensitive services. However, these time sensitive routing algorithms are designed at the time when the link bandwidth is the scarcest resource. As the link bandwidth grows rapidly in recent years due to the advance of optical communication technologies, link bandwidth is no longer the only scarce resource. The processing speed of nodes, e.g. routers, becomes another critical source of delay time. Thus, we hypothesize that node delay is a significant part in time sensitive routing for high-speed packet-switching networks. In this paper, we will show that considering with node delay time in routing algorithm for high-speed packet-switching network could provide a better performance to time sensitive services as compared to those only consider link delay.

1.4 Solution Approaches

In chapter 3, we will model the problem as a flow-based routing problem with link and node delay dependent on their loads. An iterative approach is taken to cope with the problem of load dependent delay time on links and nodes. We use a transformation to convert node delay and link delay such that each intermediate problem in each iteration can be solved by using traditional routing algorithm. The above algorithm is called the “KLONE” algorithm.

1.5 Performance Evaluation

To evaluate the performance of KLONE algorithm, we use the average path delay time and goodput ratio as our performance metrics and compare with OSPF algorithm in Chapter 4.

Chapter 2

Related Work

The main function of the network layer is to route packets form the source machine to the destination machine. In most subnets, packets require multiple hops to make the journey. The only notable exception is for broadcast network, but even here routing is an issue if the source and destination are not on the same network. The algorithms that choose the routes and the data structures that they use are a major area of network layer design. The categories of routing algorithms are illustrated in this chapter.

2.1 Routing Approaches

Variable routing approaches[8] are summarized in this section.

2.1.1 Shortest Path Routing

Shortest path routing is to build up a shortest path tree to present the network topology, so then routes each of request traffics to its destination. The Dijkstra's shortest path algorithm[9] is a very famous example of shortest path routing, and it discovers a node's shortest paths to other nodes in O(n log n) with maintaining complete information about the network topology. For the knowledge of whole network topology, it is centralized in nature. Resulting from this centralized nature, it has a privilege, the loop-free feature.

2.1.2 Flooding

Another routing algorithm is flooding, in which every incoming packet is sent out on every outgoing line except the one it arrived on. It obviously generates vast numbers of duplicate packets, in fact, and infinite number unless some measures are taken to damp the process. It is not practical in most applications, but it does have some uses. For example, in military applications, where large numbers of routers may be blown to bits at any instant, the tremendous robustness of flooding is highly desirable.

2.1.3 Flow-Based Routing

Unlike other categories of routing approaches, flow-based routing considers both the network topology and load. The basic idea is that for a given line, if the capacity and average flow are known, it is possible to compute the mean packet delay on that line from queueing theory. From the mean delays on all the lines, it is straightforward to calculate a flow-weighted average to get the mean packet delay for the subnet. The routing problem then reduces to finding the routing algorithm that produces the minimum average delay for the subnet.

2.1.4 Distance Vector Routing

Distance vector routing estimates the distance from source to destination by certain approaches. The distance vector protocols are often referred to as “Bellman-Ford” protocols because they are based on a shortest path computation algorithm. Distance vector routing protocols have been widely used since the early ARPANET[10] to nowadays Internet. For example, hop count is generally used for this issue. RIP[11] is a routing protocol in the family. Distance vector routing protocols periodically send information to its neighbor nodes. Each node could estimate the distance to other nodes according to the number of intermediate passed nodes. If the network is fixed, the nodes will compute a converged result. The updated messages are only periodically sent to their neighbors, and the update messages do not cause a heavy load to the network compared to other protocols. A weakness of distance vector routing is the convergence time. The convergence is delayed of the period update message, so RIP is constrained in a network under 16 nodes. However, RIP is still widely used in small-scaled network because of its very little configuration, management overhead and easy to implement.

2.1.5 Link State Routing

On the other hand, link state routing focuses on the states of links. Link state routing transforms the link states into some mathematical expressions to choose proper paths. Open Shortest Path First (OSPF[12]) is a widely used example of link state routing, and is parallelly used with RIP, which is a distance vector routing.

2.1.6 Hierarchical Routing

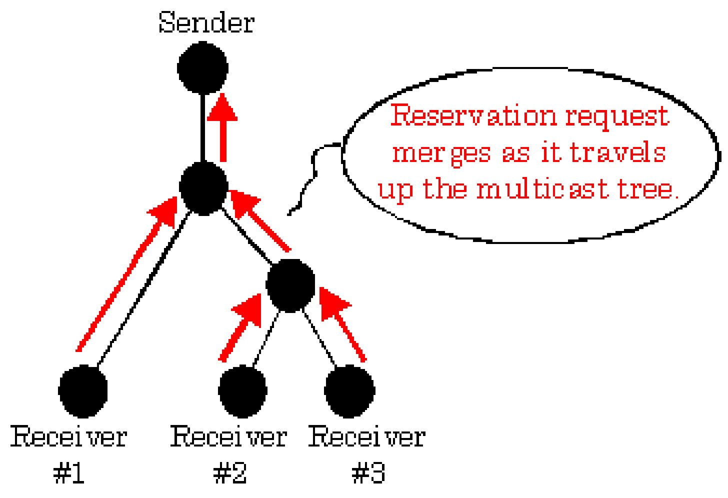
As networks grow in size, the router's routing tables grow proportionally. Not only is router memory consumed by ever increasing tables, but more CPU time is needed to scan them and more bandwidth is needed to send status reports about them. At a certain point the network may grow to the point where it is no longer feasible for every router to have an entry for every other router, so the routing will have to be done hierarchically, as it is in the telephone network.

2.2 Delay Sensitive Routing

To perform delay sensitive routing, people apply these following methods: resource reservation, probes flooding, classified queues, delay estimation and miscellaneous.

2.2.1 Resource Reservation

Resource reservation is an intuitional approach, delay time naturally decreased with bandwidth increases. Therefore, to deliver traffics from sender to receiver that is behind several routers, source host asks the intermediate routers to reserve resource for it. In practice, RSVP[15] is a famous example of resource reservation and it carries the request through the network, visiting each node the network uses to carry the stream. At each node, RSVP attempts to make a resource reservation for the stream backward from receiver to sender, as shown in Figure 2.1.



Sender
Reservation request merges as it travels up the multicast tree.
Receiver #1
Receiver #2
Receiver #3

Figure 2.1: RSVP reserves resources backward from receiver to sender.

2.2.2 Probes Flooding

An approach to deal with unknown delay time in networks is sending probes through the network and measures its traveling time to estimate the delay time of a path. Christophe Beaujean[13] proposed an method that floods probes to deicide the shortest delay path, as shown in Figure 2.2. However, the characteristics of the probe may be much different from the request traffic, for example: size, and it might lead to incorrect information and then sender may make a sub-optimal route.

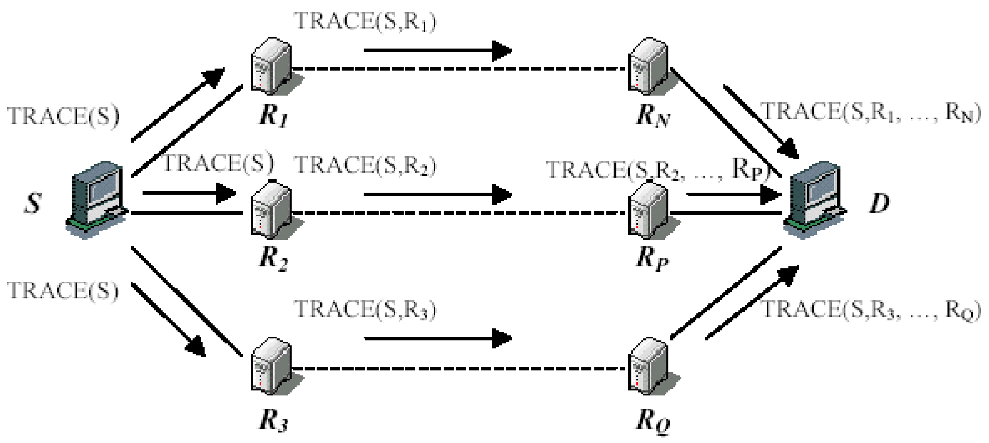


Figure 2.2: Collecting path-delay information by flooding probes.

2.2.3 Classified Queues

Classified queues are used in routers to provide QoS in transmission. Routers with classified queues forward packets according to the class information on the packets. Type of Service(ToS) field is used for a simple classification in the past, and DiffServ[16] provide more complex classification to provide QoS, as shown in Figure 2.3.

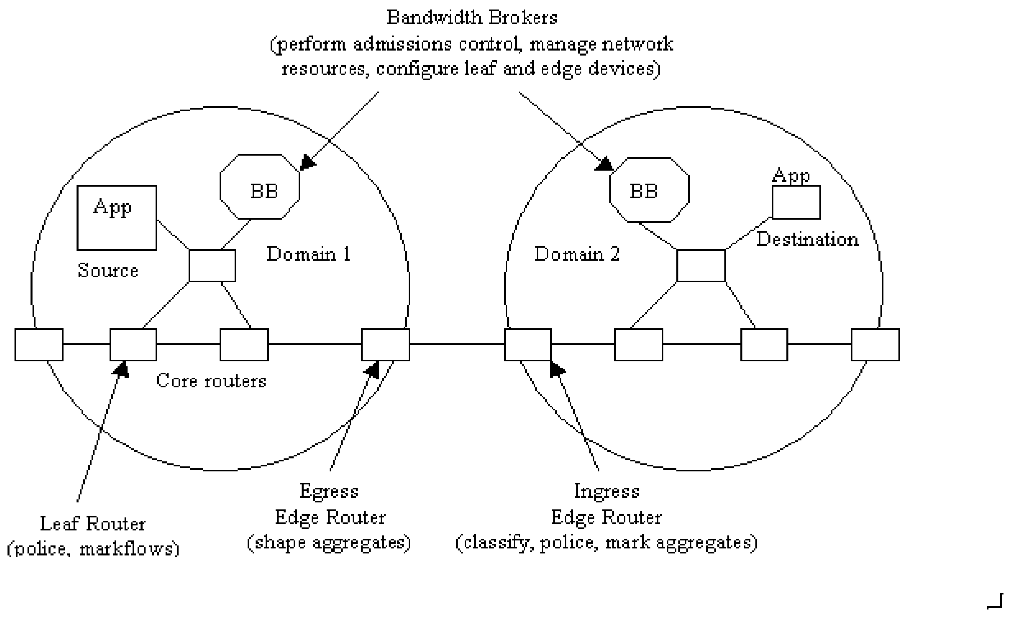


Figure 2.3: A DiffServ network.

2.2.4 Delay Estimation

People also could know the delay time via estimation. Networks are treated as queues, once people know the operating mechanisms of links and routers, they could estimate delay time. This approach is related with the flow-based problem, the delay on links and routers are dependent to not only the topology but also the load. The estimation may be difficult until the load is clearly known.

2.2.5 Miscellaneous

Douglas S. Reeves and Hussein F. Salama[14] proposed a distributed algorithm to get a path that below the delay bound while the link delay time is reported by a node connected on it. However, their approaches neither consider node delay time nor propose a mechanism to measure the delay time.

2.3 Summary

Therefore, we developed an algorithm that consider node delay time, which is important in a high-speed packet-switching network, and propose a mechanism to measure the delay time on links and nodes. The detail of our algorithm is illustrated in the next chapter.

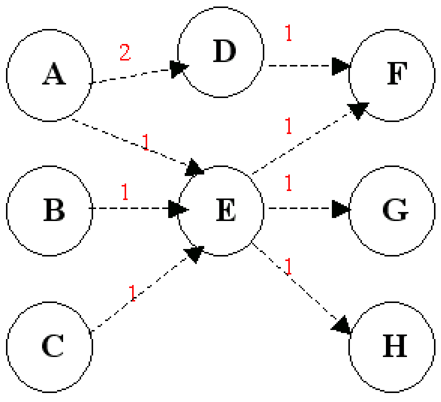
Chapter 3

Routing With Node Delay

3.1 An Illustration Example

To maintain the QoS of an All-IP network, the delay time of packet transmission must be controlled. If node delay time were not taken into account in network management such as routing, it may take longer time to transmit packets. In the following example, discovering a minimal delay path with and without taking node delay into account will be compared. The network presented is composed of eight nodes, A, B, C, …, H, and eight directed links, $\overrightarrow{AD}, \overrightarrow{AE}, \overrightarrow{BE}, \overrightarrow{CE}, \overrightarrow{DF}, \overrightarrow{EF}, \overrightarrow{EG}, \overrightarrow{EH}$. The delay time of link $\overrightarrow{AD}$ is 2, and all the other links is 1. We assume that the delay time caused by a node is proportional to the traffic passing that node. There is a unit traffic demand form A to F, from B to G and from C to H. Without considering node delay, the best routing algorithm will route all three traffic flows through node E, as shown in Figure 3.1(b). This will result the delay time of 6 for each flow.

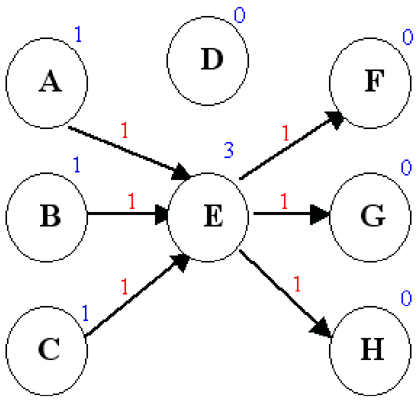
Figure 3.1(c) shows the result of another possible routing that takes node delay time into account. One traffic flow will pass node D, instead of node E. The delay time is then 5 for each flow.



(a)

Source	Destination	Volume
A	F	1
B	G	1
C	H	1

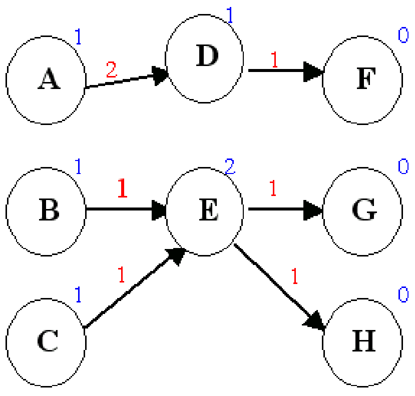
Request Traffic Demand



(b)

Source	Destination	Path	Delay
A	F	AEF	6
B	G	BEG	6
C	H	CEH	6

Path Table



(c)

Source	Dest.	Path	Delay
A	F	ADF	5
B	G	BEG	5
C	H	CEH	5

Path Table

Figure 3.1: Reducing total delay time by considering node delay in routing. (a) an example network, (b) routing result without node delay consideration, and (c) routing result with node delay consideration.

Above example shows that, for high-speed networks, a routing algorithm that takes node delay time into account may obtain a better result as compared to the traditional routing algorithms that consider link delay time only.

3.2 Routing Problem Model

Given a directed graph *G(V, E),* with |*V*| nodes and |*E*| links, the propagation delay time and bandwidth of each link, and the processing capacity of each node, the problem is to find a set of paths for a given set of traffic demands such that the total delay time is minimized. Given and derived parameters are listed in Table 3.1 and 3.2 respectively.

3.2.1 Traffic Model

Most traditional algorithms for graph related problems assume the weights of links are all fixed constants. However, the delay time occurred in the components of a real network really depends on the stochastic behavior of the traffic and routing process. In reality, it is extremely difficult to solve a routing problem that takes stochastic behavior into account. Therefore, we take a compromised approach by relaxing the stochastic property of traffic and routing process as follows.

To reduce the stochastic behavior, we assume all traffics are of Constant Bit Rate (CBR) type and all resources (processing and link bandwidth) are proportionally shared by all traffic flows passing through. In this way, the load of each resource can be computed based on the total amount of traffics passing through that resource.

Although this is a compromised model, it is more realistic as compared to the traditional fixed weight model. We hope it is a good approximation of a real network.

Table 3.1: Notation of Input Parameters.

$G(V,E)$	a directed graph, G, containing $\lvert V\rvert$ nodes and $\lvert E\rvert$ directed links; V denotes the set of all nodes, and E denotes the set of all links
v_i	a node; $v_i \in V$
e_k	a directed link $e_k = (\mathrm{v}_x , v_y) \in \mathrm{E}$, v_x is the start node, v_y is the end node of link e_k.; also denoted as e_{xy}
λ_{ij}	traffic requests volume from v_i to v_j
Λ_k	traffic requests volume from v_k to all other nodes. $\Lambda_k=\{ \lambda_{ki} ,\mid i=1,\ldots,\lvert v\rvert\}$
Λ	set of λ_{ij} , $\Lambda=\{ \lambda_{ij}, \mid v_i, v_j \in \mathrm{V}\}$
D	allowable delay time of a packet transmitted from a source node to a destination node
$b(e_k)$	bandwidth of link e_k
$t(e_k)$	propagation delay of link e_k
$p(v_k)$	processing capacity of node v_k

Table 3.2: Notation of Derived Parameters and Routing Results.

M_k	$\sum_{1}^{\lvert V\rvert}\lambda_{ki}$, summation of requested traffics starting from node v_k; the traffic volume of $\lvert\Lambda_k\rvert$
$S_k^{(n)}$	the selected routing path set(slice) of iteration n, corresponding to the request Λ_k
$P^{(n)}$	the selected routing path set(pasta) of iteration n, set of $S_k^{(n)}$. $P^{(n)} = \{ S_k^{(n)} \mid k=1,\ldots,\lvert V\rvert\}$
μ_h	volume of traffics passing through link e_h, starting from v_x, and ending at v_y, also denoted as μ_{xy}
σ_k	volume of traffic passing trough node v_k
U	set of μ_h, $U=\{\mu_h\}$
φ_{ij}	the path for λ_{ij}, selected by the routing algorithm; $\varphi_{ij}=v_i,e_{ii+1},v_{i+1},e_{i+1\ i+2},\cdots,e_{j-1j},v_j$
Φ	set of φ_{ij}
$d(v_k)$	delay time caused by node v_k
$d(e_h)$	delay time caused by link e_h
$d(\varphi_{ij})$	total delay time along path set φ_{ij}, $d(\phi_{ij}) = \sum_{e\in\phi_{ij}} d(e) + \sum_{v\in\phi_{ij}} d(v)$

3.2.3 Objective Function

For a given traffic pattern and a network, the problem is set to find a set of routing paths for the requested traffic demand, with the delay time of each traffic flow bounded to D, such that the total delay time is minimized:

Find Φ

$$\ni \min \sum_{\phi_{ij} \in \Phi} d(\phi_{ij}),$$

s.t. $\quad d(\phi_{ij}) < D, \forall \phi_{ij} \in \Phi$. (3-7)

The delay bound of each traffic flow, D, could be variant without incurring a significant impact to the model.

In E.q.3-7, *d(φ ij)* is the total delay time for a traffic flow; it is an accumulation of the delay time on all links and nodes along all the selected paths.

The problem can be reduced to a 0-1 knapsack problem; therefore it is a NP-Hard problem, and we do not expect to find a polynomial-time optimal algorithm for it. Instead, we designed a heuristic algorithm to find sub-optimal solutions.

Furthermore, both objective and constraints are not simple functions of given parameters (delay time). Instead, they are result dependent variables. This makes the problem much more complicated.

3.3 Iterative Solution Approach

Because delay times of nodes and links are not constant, and are dependent on the traffic passing through that node or link, traditional algorithms are not appropriate to solve the problem. Therefore, we choose to use iterative approach. In an iteration, the delay time of all network components can be fixed and computed based on the result obtained in the previous iteration. Initially, the delay time of all nodes(links) are set to zero and propagation delay time respectively to compute the input for next iteration.

For convenience, the result obtained in an iteration is called a *pasta*. In each iteration, the problem is still too complicated to solve. Thus, we divide the problem into some number of sub-problems, and solve each of them incrementally. Theoretically, each routing solution (pasta) can be divided into a number of single root flow trees, named a *slice*. In such a tree, the root node is any node and the tree presents the flows generated from that root node and are forwarded to all other nodes. In each incremental step within an iteration, a new single root flow tree is recomputed to replace the old one rooted at the same node and was computed in the previous iteration.

After some number of iterations, hopefully, the delay time of each network component will be stabilized, and the routing solution obtained will be a good solution. When the results obtained in two consecutive iterations are close enough or the number of iterations exceeds a given number, the process stops.

3.3.1 Intra Iteration Procedures

Theoretically, if each of $|\Lambda|$ requested flows travel along only one path, there are $|\Lambda|$ corresponding paths needed. However, it is not necessary to solve an independent routing problem for each of the $|\Lambda|$ requests. Instead, we group all those requests started from the same node, says v_k, into a subset; then solve the sub-problem as a single-source shortest path routing problem. If we solve the sub-problem using an algorithm similar to Dijkstra's, we can obtain a single root flow tree (slice) rooted from v_k. A pasta, the result of an iteration, is recomputed incrementally slice by slice. In each incremental step, a slice corresponding to a request set $|\Lambda_k|$ is removed from the pasta; the delay time of all network components is then estimated based on the remaining of the pasta.; a new slice for $|\Lambda_k|$ is then computed, and is superimposed back to the pasta. A pasta is actually the superposition of all slices obtained in an iteration. The iterative procedure is summarized in the followings. We denote the result (pasta) obtained in the n-th iteration as $P^{(n)}$. A single root path tree (slice) corresponding to the Λ_k in the n-th iteration is denoted as $S_k^{(n)}$, and $P^{(n)} = \{S_1^{(n)} \oplus S_2^{(n)} \ldots \oplus S_k^{(n)}\}$, where $\oplus$ denotes a superposition.

```
(I) Initial condition
    for all nodes and links, $\sigma=0$, $\mu=0$, $d(v)=0$, $d(e)=t(e)$;
    $S_1^{(0)} = S_2^{(0)} = S_3^{(0)}, \ldots, = S_{|V|}^{(0)} = \{\}$;     //empty set
    $P^{(0)} = S_1^{(0)} \oplus S_2^{(0)} \oplus \cdots S_{|V|}^{(0)}$;
//$\oplus$ denotes superimposing a traffic flow tree into a pasta
//$\ominus$ denotes removing a single root flow tree from a pasta

(II) First iteration
  $P^{(1)} = \{\}$;
  route $\Lambda_1$ based on ($P^{(0)} \ominus S_1^{(0)}$), to obtain $S_1^{(1)}$;
    $P^{(1)} = P^{(1)} \oplus S_1^{(1)}$;
```

```
route $\Lambda_2$ based on ( $P^{(0)} \ominus S_1^{(0)} \ominus S_2^{(0)} \oplus S_1^{(1)}$ ), to obtain $S_2^{(1)}$;
  $P^{(1)} = P^{(1)} \oplus S_2^{(1)}$;
  .
  .
  .
route $\Lambda_{|V|}$ based on ( $P^{(0)} \ominus S_1^{(0)} \cdots \ominus S_{|V|}^{(0)} \oplus \cdots \oplus S_{|V|-1}^{(1)}$ ), to obtain $S_{|V|}^{(0)}$;
  $P^{(1)} = P^{(1)} \oplus S_{|V|}^{(1)}$;
(III) On the k-th iteration:
$S^{(k)}$ ={};
for j←1 to |V|
{
    route $\Lambda_j$ based on ( $P^{(k-1)} \ominus \cdots S_2^{(k-1)} \cdots \ominus S_j^{(k-1)} \oplus \cdots \oplus S_{j-1}^{(k)}$ ), to obtain $S_j^{(k)}$;
    $P^{(k)} = P^{(k)} \oplus S_j^{(k)}$;
}
( IV ) Termination Conditions
    when $P^{(M)} \approx P^{(M+1)}$ or number of iteration >$|\Lambda|$, terminate;
```

3.3.2 Termination Conditions

Termination is triggered in these two conditions: when average path delay of two consecutive iteration are close within the predetermined value ε; or the number of iterations is greater than the number of sets, $|\Lambda|$. In the first condition, ε is defined as the difference of two consecutive iteration divided to the total path delay time of previous iteration. ε is defined in Eq. 3-8:

$$\varepsilon = \left|[\sum d(\phi_{ij}) - \sum d(\phi_{lm})]\right| \Big/ \sum d(\phi_{ij}) \Big| \phi_{ij} \in S_k^{(n)}, \phi_{lm} \in S_{k+1}^{(n)}. \qquad (3\text{-}8)$$

On the other hand, we terminate the iteration process after $|\Lambda|$ PASTAs(result of an iteration) are computed.

3.3.3 Estimation of Path Delay Time

The delay time of a path is the accumulation of the delay time occurred on all network components along that path, the delay time of path φ_{ij} , $d(\varphi_{ij})$, consists of the delay time on all nodes and links in a path, which is $d(v_i)+d(e_{i\ i+1})+d(v_{i+1})+d(e_{i+1\ i+2})+ \cdots +d(e_{j-1\ j})+d(v_j)$.

μ_h and σ_k, are defined as the total volume of traffic flows passing through a link e_h and a node v_k, respectively in E.q. 3-1, and 3-2:

$$\mu_\mathrm{h} = \sum_{\substack{\phi_{ij} \in \Phi \\ e_h \in \phi_{ij}}} \lambda_{ij} \text{ , and} \qquad (3\text{-}1)$$

$$\sigma_k = \sum_{\substack{\phi_{ij} \in \Phi \\ v_k \in \phi_{ij} \\ v_k \neq v_j}} \lambda_{ij} \ . \qquad (3\text{-}2)$$

3.3.3.1 Estimation of Link Delay Time

$d(e_h)$ is the delay time of a flow of packets passing through link e_k, including transmission delay and propagation delay. The propagation delay $t(e_h)$ is a given constant, determined by the distance and the type of transmitting media, e.g. fiber or satellite. μ_h is the total flows passing through e_h. As mentioned in Section 3.2.1, we assume every traffic flow is a CBR and the bandwidth of a link is shared by all the traffic flows passing through that link. The queuing delay on the link, thus, can be ignored. Therefore, the transmission time of a link for a flow is approximately the total traffic flows divided by the bandwidth of that link, as shown in E.q. 3-3.

The transmission delay time of link e_h is then

$$\mu_h / b(e_h) = \Big(\sum_{\substack{\phi_{ij}\in\Phi \\ e_h\in\phi_{ij}}} \lambda_{ij}\Big) / b(e_h)\,. \tag{3-3}$$

After adding the propagation delay time, the delay time for a flow of traffic passing through a link is then

$$d(e_h) = \mu_h / b(e_h) + t(e_h) = \Big(\sum_{\substack{\phi_{ij}\in\Phi \\ e_h\in\phi_{ij}}} \lambda_{ij}\Big) / b(e_h) + t(e_h)\,. \tag{3-4}$$

Notice that the delay time of a link, $d(e_h)$, is independent of the size of the flow passing that link. All traffic flows passing a link are delayed by the same amount of time.

3.3.3.2 Estimation of Node Delay Time

d(v_k) is the delay time caused by a node, v_k. Again, to simplify the delay time model, we assume all traffic flows passing a node are processed in each node in time-sharing fashion, such that the *d(v_k)* can be estimated as the total volume of traffics divided by the processing capacity of that node, as shown in E.q. 3-5:

$$d(v_k) = \sigma_k / p(v_k) = (\sum_{\substack{\phi_{ij}\in\Phi \\ v_k\in\phi_{ij} \\ v_k\neq v_j}} \lambda_{ij}) / p(v_k)\,. \qquad (3\text{-}5)$$

Thus, the delay time of a path φ_{ij} is

$$d(\phi_{ij}) = \sum_{e_h\in\phi_{ij}} d(e_h) + \sum_{v_k\in\phi_{ij}} d(v_k)\,. \qquad (3\text{-}6)$$

Each node and the traffics passing through that node can be treated as a closed network, which does not generate or absolve traffics, as shown in Figure 3.2.

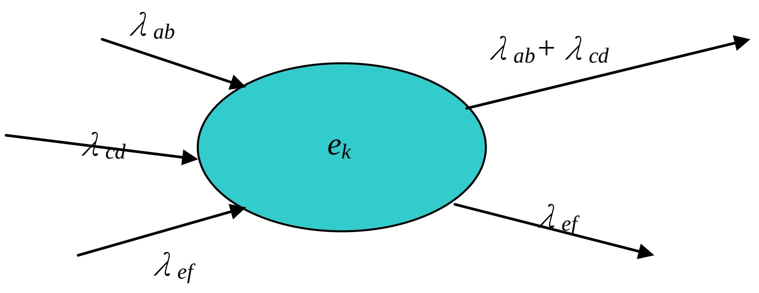


Figure 3.2: Outgoing traffic equals to incoming traffic in a closed network.

3.3.4 Node Delay to Link Delay Conversion

We need an efficient algorithm to solve a single-source shortest path routing problem to obtain a *slice*. Unfortunately, current shortest path algorithms all assume zero weight on nodes such that they are not adequate for this problem even though the delay time of network components are all constant within each iteration.

There are two approaches to solve this problem. The first one is to develop a new algorithm that considers both node and link delay together; the second one is to convert node delays into link delays, and then apply a conventional shortest path algorithm to solve this problem. Because to develop a new algorithm that considers node and link delays together may take much time, we choose the second approach such that we can solve this problem using existing shortest path algorithms.

Dijkstra and Bellman-Ford are two famous algorithms for the shortest path problem. Both algorithms only take link delay time into consideration. Dijkstra's is centralized while Bell-Ford's is distributed. Because we are designing a centralized algorithm, we choose to use Dijkstra shortest path algorithm.

We show how to convert the node weight into link weight. The transformed graph will be equivalent to the given graph in the sense of path delay time.

The delay time of a node is computed based on E.q. 3-5 in Section 3.2.2.2, where the total traffic passing through a node is obtained by the summation of its outgoing traffic flows. Node delay time is added to propagation delay time of each incoming link. By doing so, we obtain another graph that has weights on its links only and is equivalent to the original graph with respect to the path delay time, as shown in the remaining of this section.

Given a node with a weight of *m*, who has two incoming links of weight w_1 and w_2, as well as two outgoing links of weight w_3 and w_4, as shown in Figure 3.3(a). We can transform the graph by connecting the incoming links to the outgoing links, with four internal links of weight *m*. When a traffic flow passes this node, no matter which incoming link it comes from or which outgoing link it selects to leave, it should suffer from the delay caused by a delay time (weight) of *m*, as shown in Figure 3.3(b)

Consider a traffic flow passing through the node, the total weight sum is either $w_1 + m+w_3$, $w_1 + m+w_4$, or w_2+m+w_4, $w_2 + m+w_3$. Since weight *m* appears in all possible paths, it can be treated as a common link, where the incoming and outgoing links connect. This is shown in Figure 3.3(c). Finally, we shift the node weight (m) to incoming links. The weights of incoming links are then changed to w_1+m and w_2+m respectively. The node is then transformed into Figure 3.3(d), where the weight of node is shifted to links. The minimum weight paths can be obtained by applying the original Dijkstra’s shortest path algorithm.

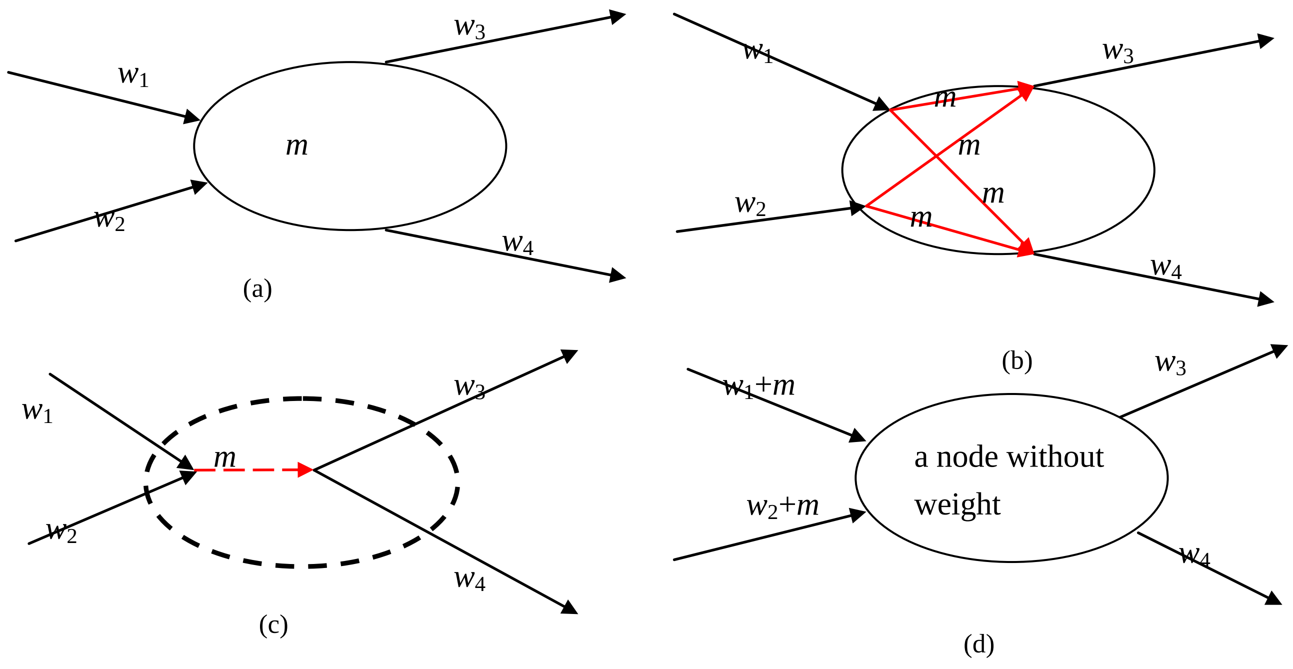


Figure 3.3: Transformation of node delay to link delay.

(a) a node in original graph,
(b) have new links from cross connecting the incoming and outgoing links,
(c) use a single link to present, since any path suffer the same *m*, and
(d) shift *m* to the each of the incoming links.

3.4 KLONE Algorithm

KLONE algorithm is an abbreviation of "Kenex aLgorithm Obtaining Node-delay Estimation". In KLONE, we compute slices, and we compose each consecutive $|\Lambda|$ slices into a pasta. Traffic volume on each node and link are estimated within a pasta. and A slice is computed with fixed amount of traffic on links and nodes. The estimation of node and link delay is introduced in Section 3.2.2.2 and we modify the update method in Dijkstra's shortest path algorithm, to Eq. 3-9:

when each routing path started from different node is chosen once, and propose temporal nodes and links delays to precede the next pasta. While pasta computing is being processed, we estimate the influence caused by traffic initiating from previous node and convert the nodes delays into links delays. Therefore, we can apply Dijkstra's shortest path algorithm to find a shortest path tree for current target node. When all nodes are visited once in a slice, we say a slice is computed. With the computation of every slice, we have a set of paths to be the result set, and we hope it could be closer to the optimal solution iteration by iteration. We take it as a process of making a statue by clay. Each time the slice creates, like we shape the statue to be closer to the result we expect. The slice creation (statue making process) will be terminated when the results converge.

3.4.2 Pseudo Codes

```
KLONE(G, $\Lambda$)
{
for k = 1 to |V|
{
    $P^{(k)} \leftarrow NIL;$
    for j $\leftarrow$ 1 to |V|
    {
        $P._{tmp} \leftarrow P^{(k-1)} \ominus S_1^{(k-1)} \cdots \ominus S_j^{(k-1)}, and \oplus S_1^{(k)} \oplus S_2^{(k)} \oplus \cdots \oplus S_{j-1}^{(k)};$
        $U \leftarrow$TVC($P._{tmp}$, $\Lambda_j$); //traffics volume calculation on nodes and links,
                                 //$P._{tmp}$ contains the propositional traffics on network,
                                 //while $\Lambda_j$ is the new incoming traffic
        $G_{tmp}$ ← NLC($U$); //Node-Link Conversion, get weight of each link
            Dijkstra.shortest.path($G_{tmp}$, $v_j$); // get shortest path tree $S_j^{(k)}$;
                $P^{(k)} \leftarrow P^{(k)} \cup \{S_j^k\}$;
        }
        if $P^{(k-1)} \approx P^{(k)}$
                return $S^{(k)}$;
}
return $S^{(|V|)}$;
}
```

```
TVC($S$, $\Lambda_i$, $G$) // Traffic Volume Calculation
{
for all $\lambda_{ij}$ in $\Lambda_i$
        Get $\varphi_{ij}$ (corresponding to $\lambda_{ij}$) from $S$;
        Add $\lambda_{ij}$ to all nodes and links that compose $\varphi_{ij}$, get $\sigma$-s and $\mu$-s of all nodes and links;
      Get and $\mu$-s;
return;
}
```

```
Dijkstra.shortest.path($G$, $\Lambda_k$)
{
Initialized.Single.Source($G$, $k$)
$S$ <-- NIL;
$Q$ <-- All vertex in $G$;
predecessors ← NIL;
while $Q$ is not $NIL$
    do $v_x$ <-- Extract-Min(Q);
            $S$ <-- $S$ union $\{v_x\}$;
            for each vertex $v_y$ <-- Adj[$v_x$]
                    do relax($v_x$,$v_y$,U) //  $\delta$-s and  $\mu$-s are stored in $G$;
return predecessors; //return the shortest path tree;
}
```

```
Initialize-Single-Source($G$, $s$)
{
for each vertex $v_k$ <-- V[G]
    do distance[$v_k$] <-- unlimited; //d[$v_k$]: the distance from $v_s$ to $v_k$;
        predessors[$v_k$] <-- NUL;
 distance[$v_k$] <-- 0;
}
```

```
Relax(x,y,G)
{
 if   distance[y] > distance[x]+ Weight($v_x$,$v_y$)
       then distance[v] $\leftarrow$ distance[u] + Weight($v_x$,$v_y$);
        predecessors[v]  $\leftarrow$ u;
}
```

```
Node-Link Conversion(U,  $\Lambda_k$)
```

```
for all links $e_{xy}$ in $U$
{
    $Weight(v_x,v_y)=d(e_{xy})+d(v_y)=(\mu_x+M_k)/b(e_x)+t(e_x)+(\sigma_y+M_k)/p(v_y)$;
}
```

3.4.3 Complexity Analysis

KLONE algorithm groups the $|\Lambda|$ traffic requests into $|V|$ sets each set is corresponding to one shortest path tree. The shortest path algorithm we apply is the Dijkstra's shorstest path algorithm, so the time complexity of this part is $N^2 \log N$. Within an iteration (pasta), we create $|V|$ path trees (slice), so it takes $N*N^2 \log N = N^3 \log N$. And at most we repeat $|V|$ iterations, the time complexity will be bounded under $N*N^3 \log N=N^4 \log N$ in worst case. On the other hand, in the intensive evaluation process in Chapter 4, we know the complexity could be $2*N^3 \log N$, that is , $O(N^3 \log N)$ in general cases.

3.5 Summary

The influence of considering nodes delay in a delay sensitive routing is illustrated in Section 3.1. We modeled the routing problem into a flow-based routing problem and proposed KLONE algorithm to solve the variable nodes and links delay time problem iteratively. In Section 3.2 and 3.3, the detail of KLONE algorithm is explained. And then we analyzed the time complexity and found that it could run at $O(N^3 \log)$, as shown in Section 3.4. We will evaluate the performance of KLONE algorithm and OSPF algorithm in Chapter 4.

Chapter 4

Performance Evaluation

We used numerical simulation to evaluate KLONE algorithm including its convergence, average path delay time, and goodput ratio. We demonstrate that node delay time in time sensitive packet routing for high-speed packet-switching network is important by comparing our algorithm with the traditional OSPF routing algorithm.

4.1 Performance Evaluation Metrics

The followings are performance evaluation metrics:

1. Convergence speed: evaluated by two different values: the number of iterations when the convergence occurs divided by the total number of nodes. and the total number of slices when the convergence occurs divided by the total number of nodes.
2. Average path delay time: the average time for a unit of request traffic passing through the network.
3. Goodput ratio: The ratio of satisfied traffic requests.

4.1.1 Convergence of KLONE Algorithm

Since KLONE algorithm is an iterative algorithm, we first evaluated its convergence speed, and observed its behaviors within the iteration process. The convergence of KLONE algorithm occurs when the average path delay time, which will be defined later, of two consecutive PASTAs(result of an iteration in KLONE algorithm) differ by a predefined value, ε. The convergence speed is evaluated by K_1/N and K_2/N, where the convergence occurs at the K_1-th iteration and the K_2-th slice and N is the number of nodes respectively.

4.1.2 Performance of KLONE Algorithm

We compared the performance of KLONE and OSPF, in terms of average path delay time and goodput ratio.

4.1.2.1 Average Path Delay Time

Both algorithms select paths for the test instances, and in each evaluation process, we forced all traffic pass through the paths selected by either KLONE or OSPF routing algorithm. The delay time of each path is computed by accumulating the delay time on all links and all nodes composing that path. Finally, the average path delay time is computed by the summation of the size of a traffic multiplied by the delay time on the selected path and then divided by the size of total traffic, as

$$\left. \sum \left[|\lambda_{ij}| * d(\phi_{ij}) \right] \middle/ \sum |\lambda_{ij}| \right. .$$

4.1.2.2. Goodput Ratio

Goodput is defined as the total satisfied traffic requests that can find a corresponding path with a delay time less than the given upper bound. Goodput ratio is defined as $L/|\Lambda|$ where L is the number of goodput and $|\Lambda|$ is the number of request traffics. The higher the ratio is, the better the routing algorithm will be. In general, larger delay bound, D, allows higher goodput ratio under the same routing algorithm.

4.2 Design of Experiments

We compared KLONE algorithm and OSPF algorithm in 64,000 different test instances, in the combinations of different number of nodes, network connectivity ratio, and different link bandwidth/processing capacity ratio. This section explains the experiments we conducted.

4.2.1 Test Instance Generation

Test instances include random networks and random traffic. The parameters of a network instance include number of nodes, link bandwidth, connectivity, link propagation delay time, and node processing capacity. The parameters of a request traffic instance include the traffic requests and the delay bound, D.

4.2.1.1 Network Instances

The range of link bandwidth was set from 0 to 400 Gbps, and propagation delays stayed below 20 ms. Number of nodes was set from 10 to 100 with a processing capacity in the range of Gbps. Connectivity is defined as $P / N * (N-1)$, where *P* is the number of links, and *N* is the number of nodes. The range of connectivity was set from 0 to 100 percents. The *BP ratio* is defined as *b(e)/p(v),* where *b(e)* is the link bandwidth and *p(v)* is the node processing capacity and we varied it from 1/300 to 1/1.

4.2.1.2 Traffic Request Instances

The traffic coming into an edge node is assumed in an aggregated form. For a graph of N nodes, there are N*(N-1) requests, one from each node to every other node. The upper bound of delay time of all paths is set to D and D varies from 100 to 2000 ms.

Table 4.1: Parameters and Ranges for Test Instances.

Parameters	**Range of values**
number of nodes	10,20, … , 100
link bandwidth	0~400 Gbps
node connectivity	40%, 60%, 80%, 100%
link propagation delay time	1~20 ms
node processing capacity	0~400 Gbps
traffic requests	0~1000 Mbps
delay bound (D)	100~2000 ms

4.2.2 Experiments

The followings are the experiments we conducted:

- Experiment Exp-1 is to evaluate the convergence in two aspects, the speed of convergence and the behavior within the iteration process.
- Experiment Exp-2 is to evaluate how the connectivity affects the performance of KLONE algorithm.
- Experiment Exp-3 is to evaluate how the BP ratio affects the performance of KLONE algorithm.
- Experiment Exp-4 is to evaluate how the number of nodes affects the performance of KLONE algorithm.

They are summarized in Table 4.2.

Table 4.2: List of Experiments.

Experiments	**Objective**
Exp-1: Convergence Test	(a) Observing the speed of convergence (b) Studying the behaviors within the convergence process
Exp-2: Sensitivity to connectivity	Observing the performance using two metrics: (a) average path delay time, and (b) goodput ratio.
Exp-3: Sensitivity to BP Ratio	
Exp-4: Sensitivity to number of nodes	

4.3 Experiments and Results

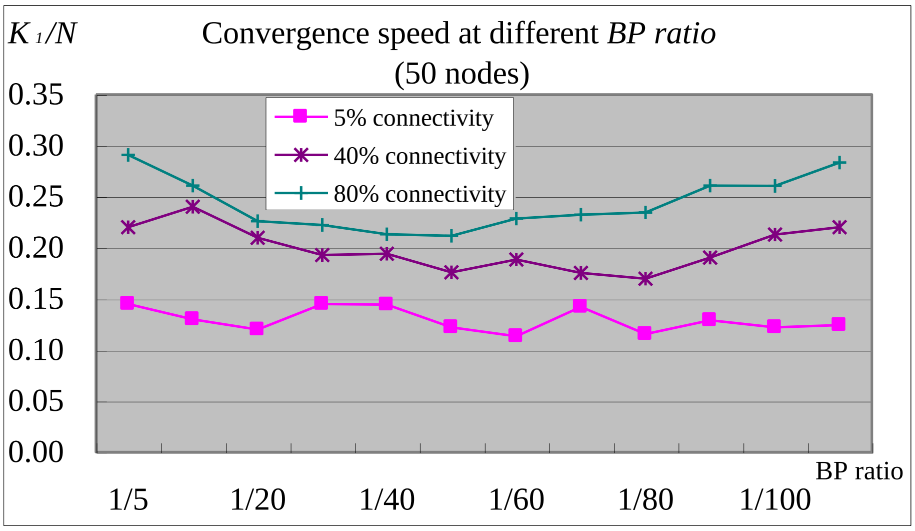
The experiments and results will be presented in this section. Due to the limit of the space, only few portions of the figures are shown. Most of the figures shown in this section are for the networks of size 50. Other parameters will be specified in the figures.

4.3.1 Exp-1: Convergence Test

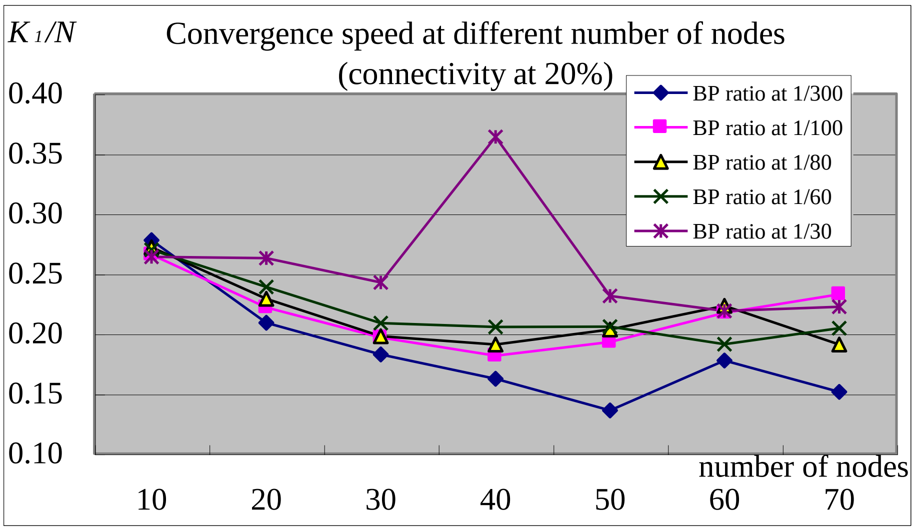
The first experiment is to evaluate the convergence. We studied the convergence speed and the behavior within the iteration process.

4.3.1.1 Convergence Speed

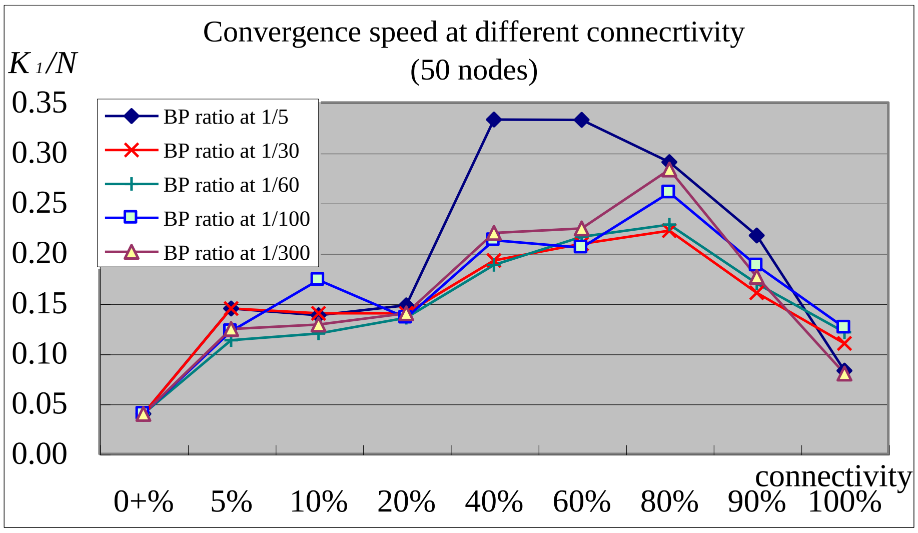
We adjusted the following three parameters in the experiment to study their impact to the convergence speed: the BP ratio, the number of nodes and the connectivity. The connectivity is defined as $P / N*(N-1)$, where P is the number of links, and N is the number of nodes. Figure 4.3(a) and (b) show that neither BP ratio nor the number of nodes has any impact to the convergence speed. On the other hand, we found that the convergence speed is dependent on the connectivity. This may be caused by two different reasons. First, higher connectivity may make a request easier to find a very good satisfied path, and then there is a higher opportunity to choose the same path in the succeeding iteration, as shown in Figure 4.3 (c). On the other hand, lower connectivity may make a request having fewer paths to choose, so that the solution domain is much smaller and thus the convergence speed is faster. The dependency between ε value and the convergence speed is shown in Figure 4.1 (d).


K_1/N
Convergence speed at different *BP ratio*
(50 nodes)
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
5% connectivity
40% connectivity
80% connectivity
BP ratio
1/5
1/20
1/40
1/60
1/80
1/100

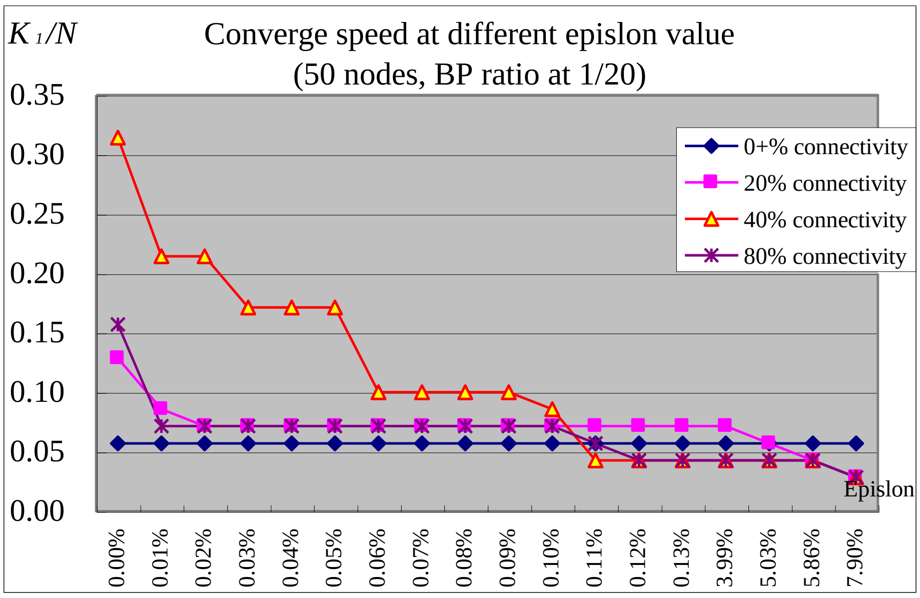
(a)


K_1/N
Convergence speed at different number of nodes
(connectivity at 20%)
0.40
0.35
0.30
0.25
0.20
0.15
0.10
BP ratio at 1/300
BP ratio at 1/100
BP ratio at 1/80
BP ratio at 1/60
BP ratio at 1/30
number of nodes
10
20
30
40
50
60
70

(b)


Convergence speed at different connecrtivity
(50 nodes)
K_1/N
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
BP ratio at 1/5
BP ratio at 1/30
BP ratio at 1/60
BP ratio at 1/100
BP ratio at 1/300
connectivity
0+%
5%
10%
20%
40%
60%
80%
90%
100%

(c)



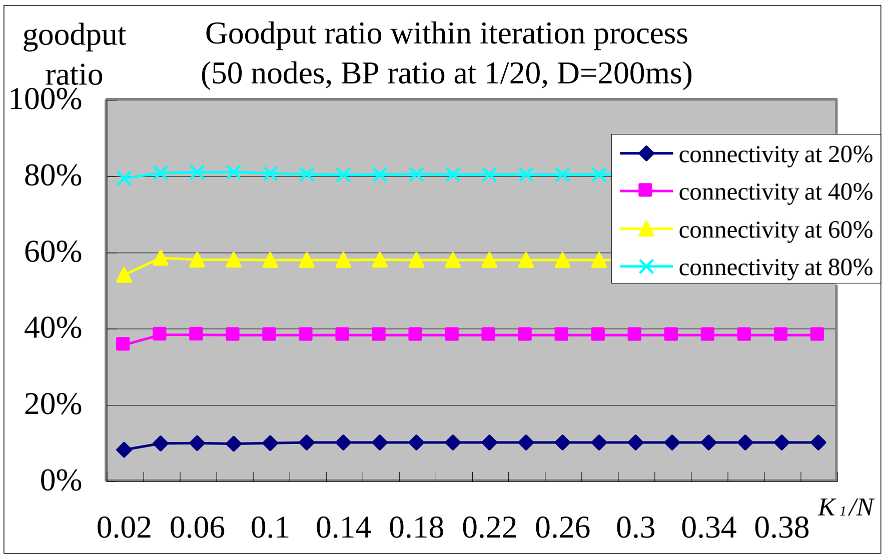
(d)

Figure 4.1: Relationship between the convergence speed and the performance metrics. (a) BP ratio , (b) number of nodes, (c) connectivity, and (d) ε value.

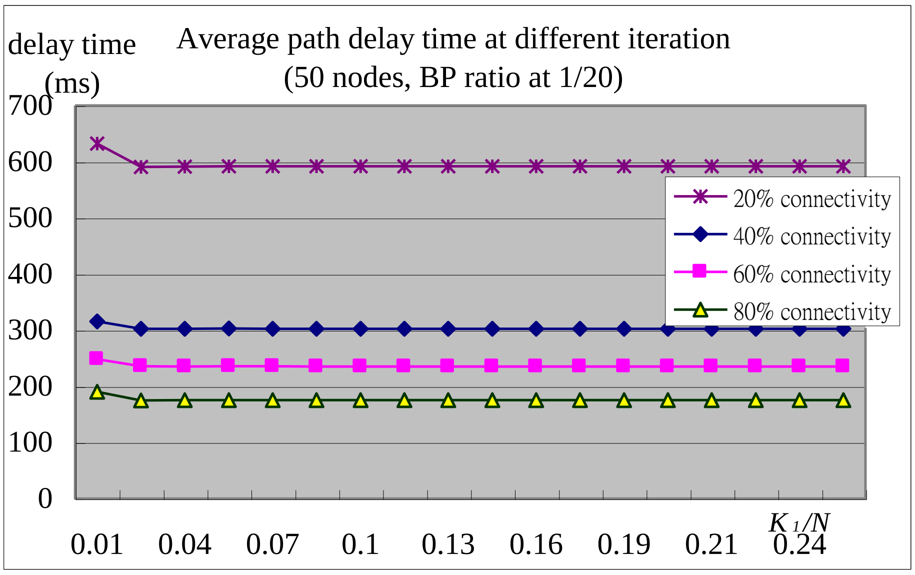
4.3.1.2 Behaviors within Iteration Process

In addition to the understanding of the convergence speed of KLONE algorithm, we also have to study the behavior within its iteration process.

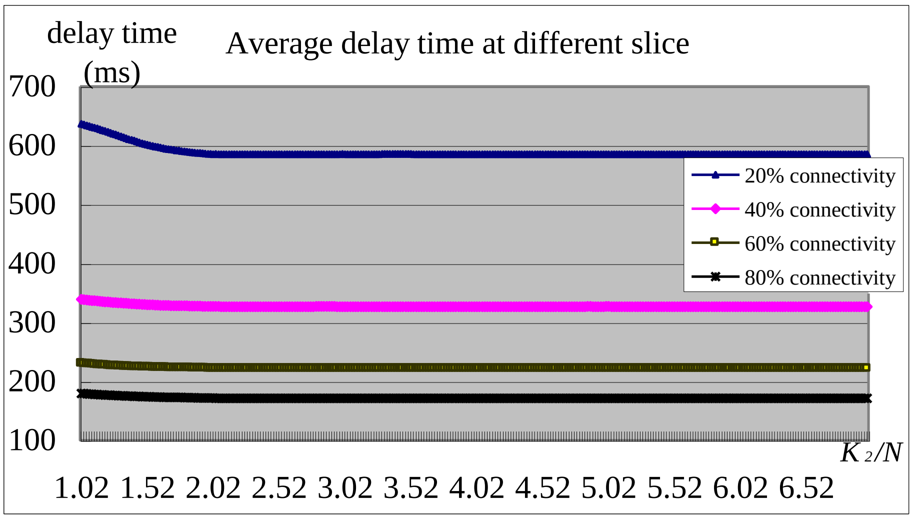
In more than 90% of the test instances, we found that the lines of both average path delay time and goodput ratio become smooth after the $K_1=2/N$ and $K_2=2$. Figure 4.2(a) and (b) shows the average path delay time and the goodput ratio at the convergence point K_1. The convergence occurs at the second iteration. On the other hand, Figure 4.2(c) and (d) shows that K_2 occurs at 2N-th iteration.


goodput ratio
Goodput ratio within iteration process
(50 nodes, BP ratio at 1/20, D=200ms)
100%
80%
60%
40%
20%
0%
0.02
0.06
0.1
0.14
0.18
0.22
0.26
0.3
0.34
0.38
K₁/N
connectivity at 20%
connectivity at 40%
connectivity at 60%
connectivity at 80%

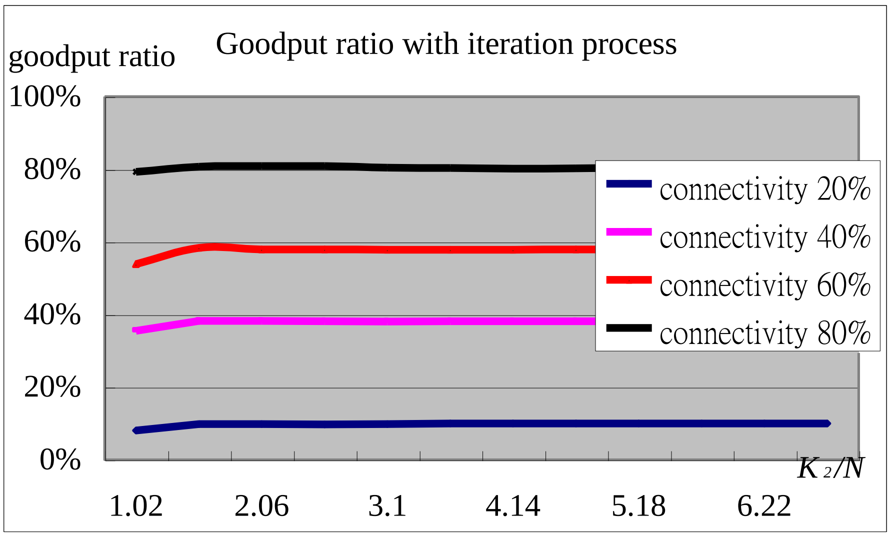
(a)


delay time (ms)
Average path delay time at different iteration
(50 nodes, BP ratio at 1/20)
700
600
500
400
300
200
100
0
0.01
0.04
0.07
0.1
0.13
0.16
0.19
0.21
0.24
K₁/N
20% connectivity
40% connectivity
60% connectivity
80% connectivity

(b)


delay time (ms)
Average delay time at different slice
700
600
500
400
300
200
100
1.02
1.52
2.02
2.52
3.02
3.52
4.02
4.52
5.02
5.52
6.02
6.52
K₂/N
20% connectivity
40% connectivity
60% connectivity
80% connectivity

(c)



(d)

Figure 4.2: KLONE behaviors within iteration processes.

(a) the goodput ratio and (b) the average path delay time at K_1, and

(c) the goodput ratio and (d) the average path delay time at K_2.

4.3.2 Exp-2: Sensitivity to Connectivity

Intuitively, higher connectivity implies more available path between nodes. But, how does connectivity influence KLONE algorithm? We studied the dependency between the connectivity and the two performance metrics: average path delay time and goodput ratio. We varied connectivity from 0% to 100% to see how average path delay time and goodput ratio are influenced.

4.3.2.1 Connectivity and Average Path Delay Time

Higher connectivity implies more available paths and more bandwidth within the network. We found that, at the same number of nodes, the average path delay time becomes smaller as the connectivity increases, as shown in Figure 4.3. We compared the average path delay in KLONE algorithm and in OSPF algorithm. The average path delay time improvement is defined as $(T_2-T_1)/T_2$, where T_2 is the average path delay time of OSPF algorithm, and T_1 is of KLONE algorithm. The larger the value, the better KLONE algorithm.

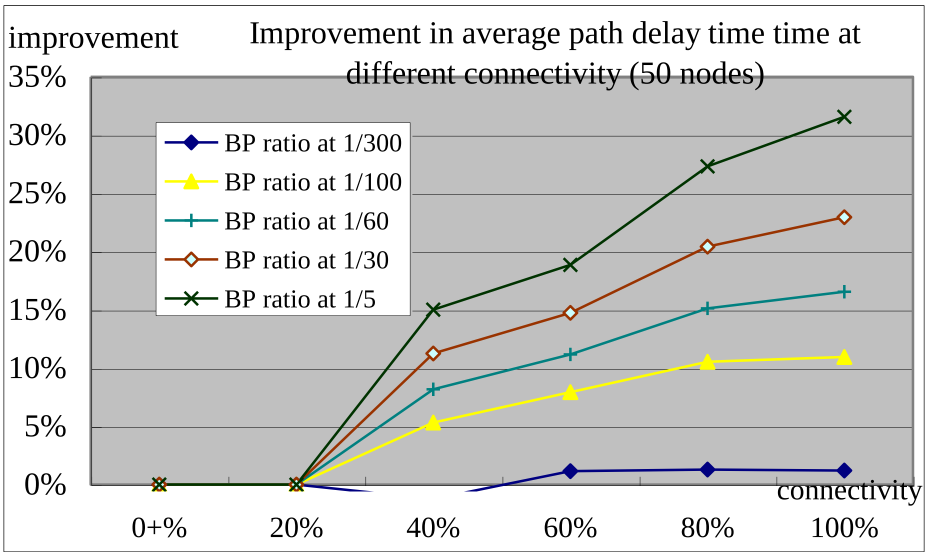


Figure 4.3: Relationship between connectivity and improvement in average path delay time.

4.3.2.2 Connectivity and Goodput Ratio

In Figure 4.4, we show that at higher connectivity, KLONE algorithm has a higher goodput ratio than OSPF algorithm.

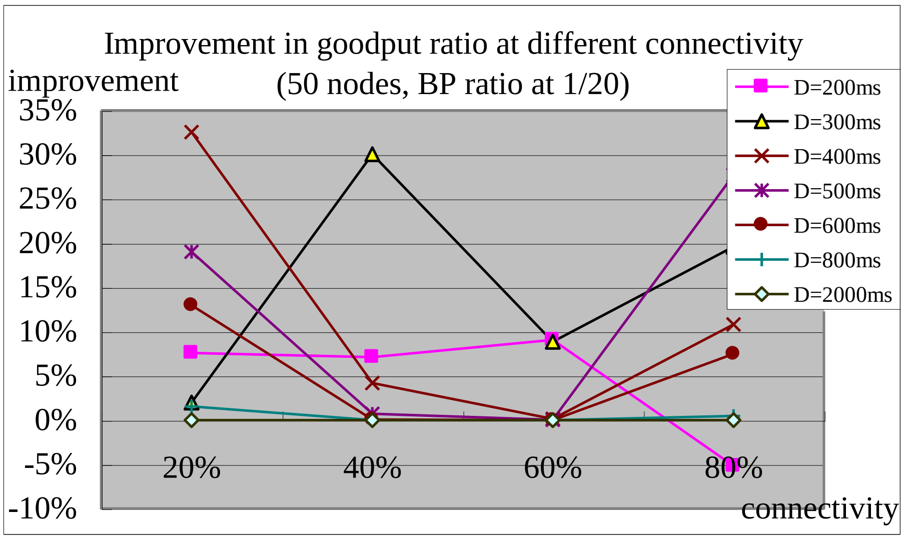


Figure 4.4: Relationship between delay bound and goodput ratio.

4.3.3 Exp-3: Sensitivity to BP Ratio

We varied the BP ratio from 1/300 to 1 to see the dependency between the BP ratio and the two performance metrics.

4.3.3.1 BP ratio and Average Path Delay Time

We found that when the BP ratio increases, the improvement of average path delay time increases, as shown in Figure 4.5. This is consistent with our hypothesis that when the speed of links increases, an algorithm that concerns both link and node delay times might have a better performance than OSPF, which only concerns links delay times.

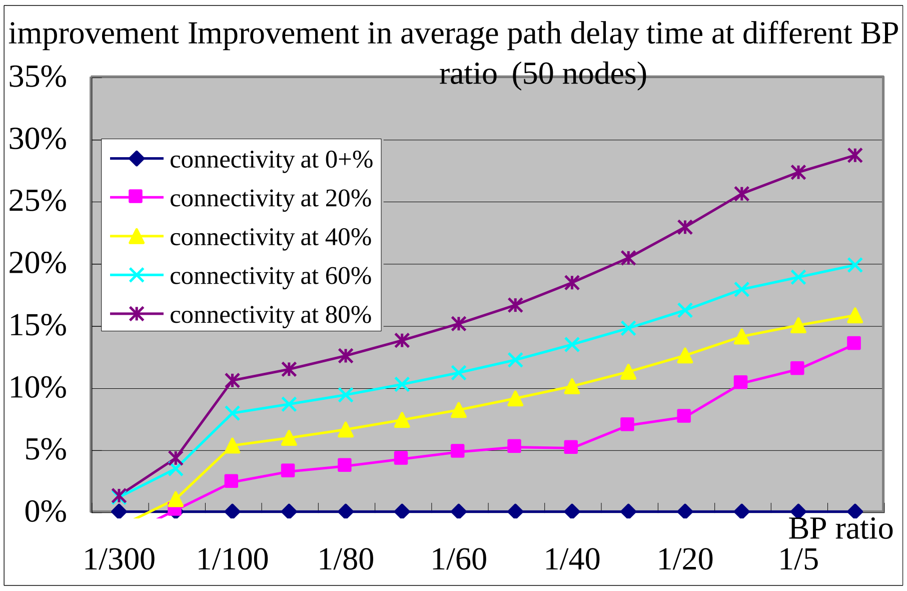


Figure 4.5: Relationship between BP ratio and average path delay time.

4.3.3.2 BP Ratio and Goodput Ratio

We also compared the goodput ratio of KLONE algorithm and OSPF algorithm. And we found that at different BP ratios, goodput ratio in KLONE algorithm usually gets better than OSPF algorithm, as shown in Figure 4.6.

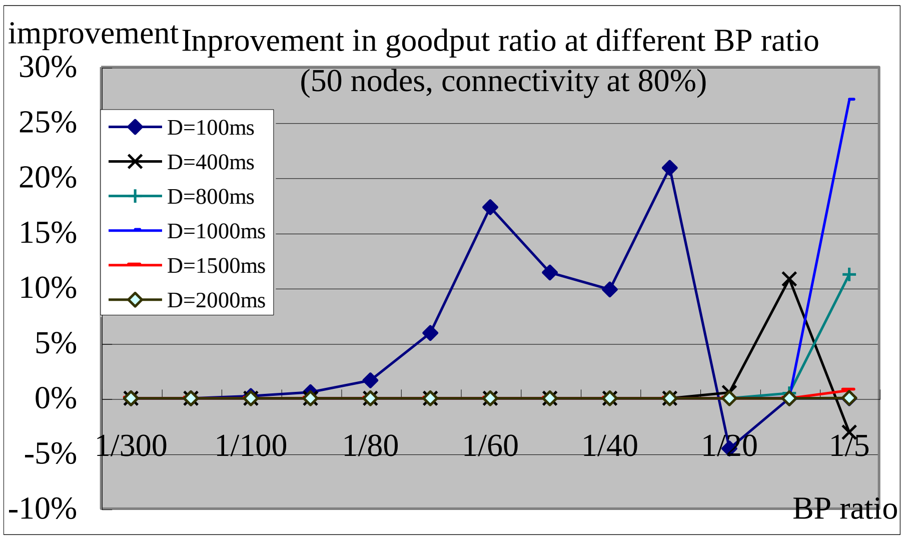


Figure 4.6: Relationship between BP ratio and goodput improvement.

4.3.4 Exp-4: Sensitivity to Number of Nodes

The number of nodes is varied from 20 to 70 in this experiment to see how it affects the performance.

4.3.4.1 Number of Nodes and Average Path Delay Time

This experiment studies the dependency between the number of nodes and the delay time improvement. The performance improvement, which is defined in Section 4.3.2.1, is shown in Figure 4.7, in which the connectivity is 20%. The improvement increases as the number of nodes increases.

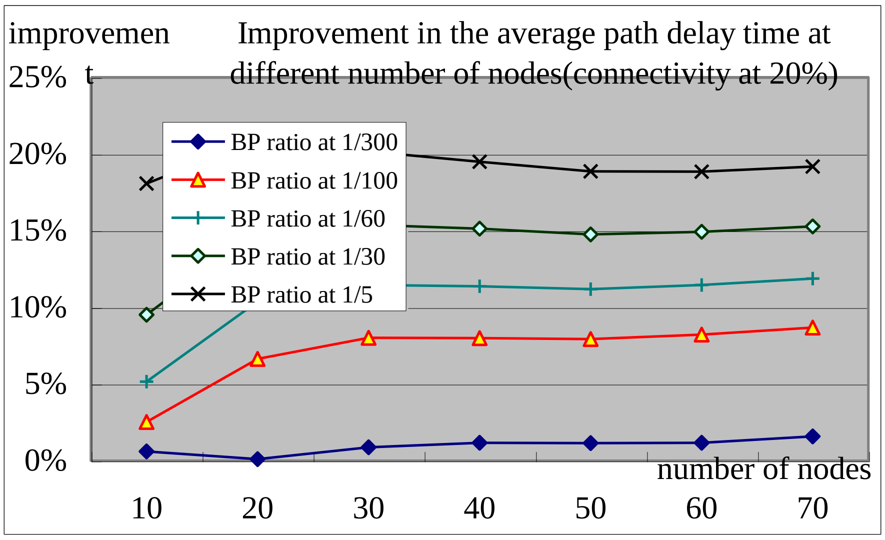


Figure 4.7: Relationship between the improvement in the average path delay time and the number of nodes.

4.3.4.2 Number of Nodes and Goodput Ratio

Increasing the number of nodes will increase the goodput ratio at same delay bound, D. We study the differece in goodput ratio between KLONE algorithm and

OSPF algorithm, as shown in Figure 4.8. At different number of nodes, KLONE algorithm has a better goodput ratio than OSPF algorithm.

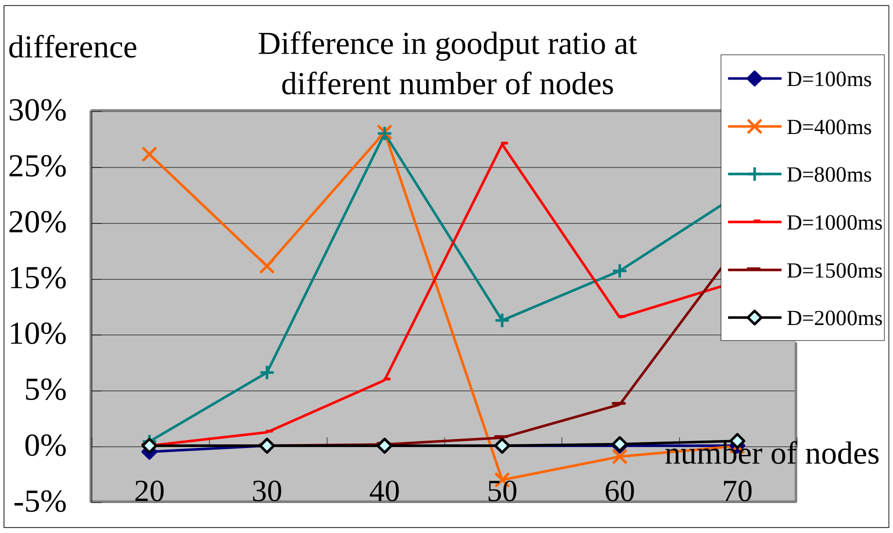
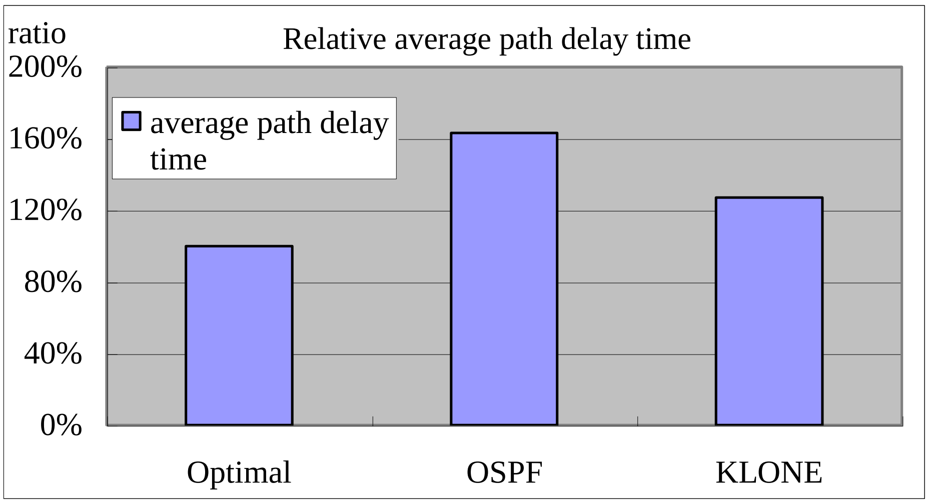


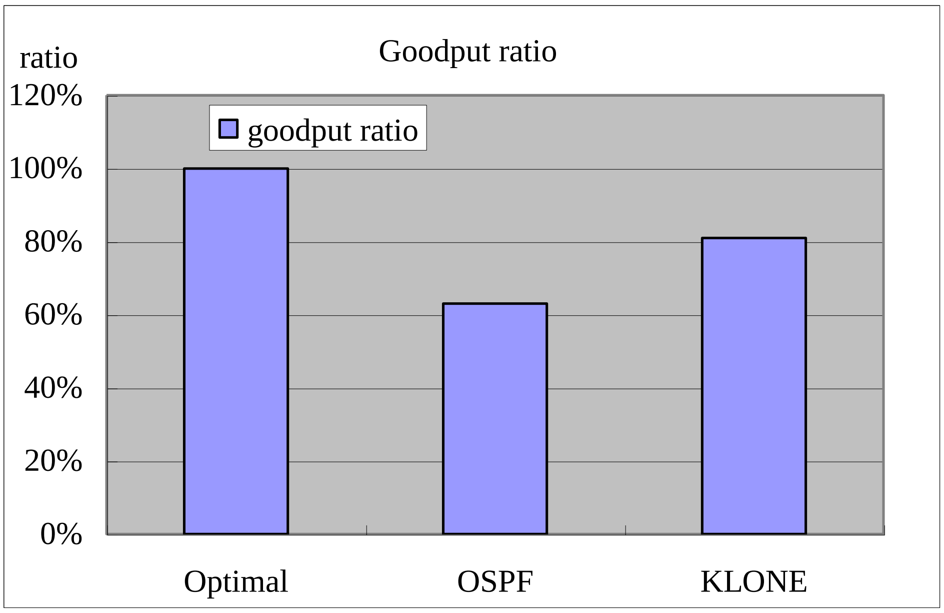
Figure 4.8: Relationship between number of node and goodput ratio.

4.3.5 Comparison with The Optimal Solution

In order to estimate the absolute performance of KLONE algorithm, we made a comparison between OSPF algorithm, KLONE algorithm and the optimal solution in a very small scale test instance, as shown in Figure 4.9, where the number of nodes is set to 10, connectivity is set to 20%, BP ratio is at 1/10. Figure 4.9 shows the comparison in the average path delay time. This toy-type study may not be a good representation of any algorithm. However, it still gives us a sense to the distance between KLONE algorithm and the optimal solution.



(a)



(b)

Figure 4.9: Comparison with the optimal solution:

(a) average path delay time, and (b) goodput ratio.

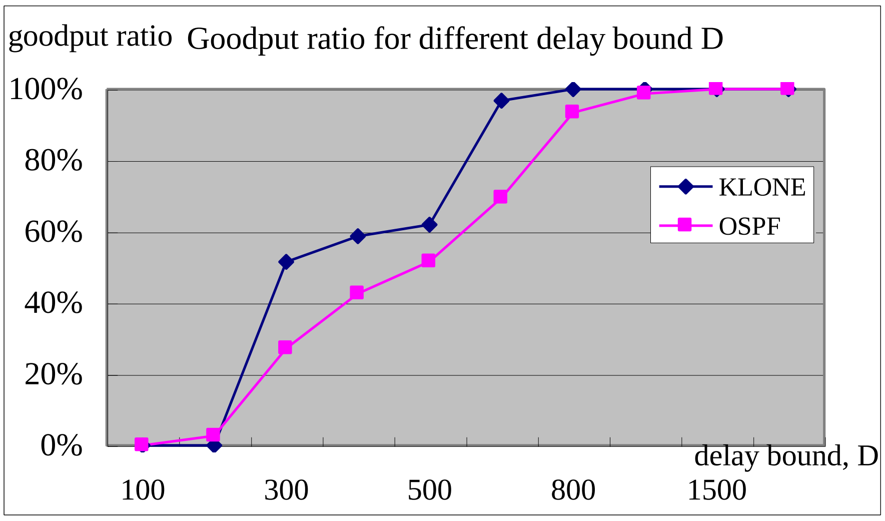
4.3.6 Weakness of KLONE Algorithm

4.3.6.1 The Complexity

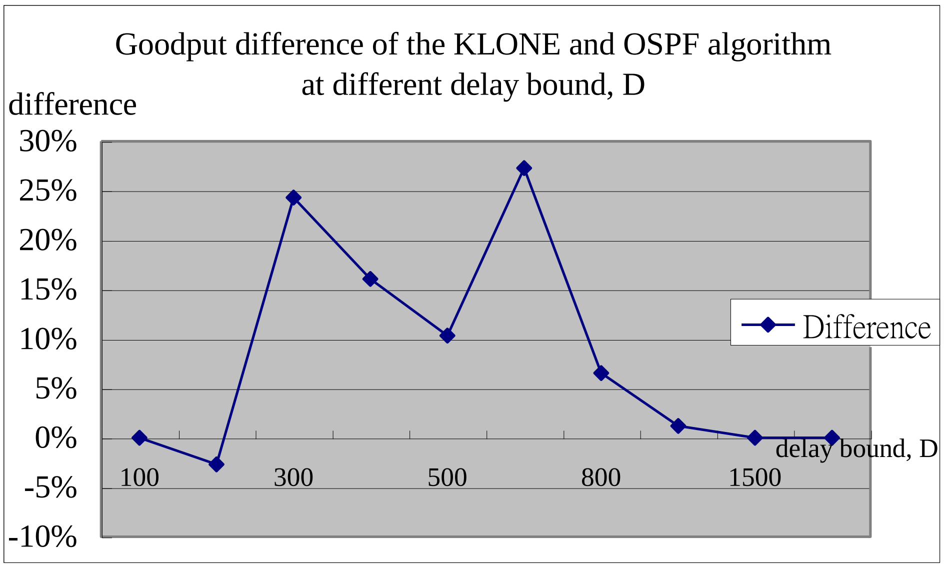
The complexity is a weakness of KLONE algorithm. The time complexity of KLONE algorithm is O($N^3 \log N$), while it may be not good enough for general use.

4.3.6.2 The Exceptions in Low Delay Bound

In the comparison of goodput ratio at different delay bound D, KLONE algorithm is generally better than OSPF algorithm. However, at some special range, such as small D, OSPF algorithm may have a better performance. It may be caused by that OSPF algorithm may gather up traffics on some specific links and nodes. The rest of traffics could be delivered on links and nodes those are slightly loaded and so they could be delivered within short delay time. However, when considering about the all paths delay time, OSPF algorithm is outperformed by KLONE algorithm. This s shown in Figure 4.10 in which the number of nodes is 30, the connectivity is 60%, and the BP ratio is 1/5. In such instances, OSPF algorithm performs better than KLONE algorithm.



(a)



(b)

Figure 4.10: An example of KLONE weakness in low delay bound (a) goodput ratio curve and (b) difference in goodput.

4.4 Analysis and Conclusion

The Exp-1, convergence test, shows us KLONE algorithm gets advances from the iteration process. Furthermore, the Exp-2, Exp-3, and Exp-4 shows us that the performance improvement between KLONE algorithm and OSPF algorithm is influenced by the two parameters, connectivity and BP ratio. And the comparison with the optimal solution shows that the performance of KLONE algorithm is between the optimal solution and OSPF algorithm.

Chapter 5

Concluding Remark and Future Work

With the intensive testing instances, we demonstrated the importance of the nodes delay in the routing path for high-speed packet-switching networks. We hypothesized that an routing algorithm considering both the nodes and links delay time could have a better performance than that only considers with links delay time in delay sensitive services. We developed a flow-based routing algorithm, KLONE algorithm, which considers both link delay time and node delay time. In our intensive evaluation, KLONE algorithm could outperform OSPF algorithm which only considers link delay time. The results of the experiments show that KLONE algorithm could have a better performance than OSPF algorithm in most cases, with only a few exceptions. Our hypothesis that considering with node delay is important in high-speed packet-switching network is thus demonstrated.

This algorithm still has some weak points. First, KLONE algorithm may have worse goodput than OSPF algorithm when the delay bound is very low. Secondly, it does not support multi-paths routing for the same traffic stream yet. On the other hand, there are some future works to be done. For example, the estimation of traffic delay time may be not precise because the under layers, MAC, and PHY, might have various approaches to transmit data. Different transmission methods may result in different delay time. Furthermore, KLONE algorithm is a centralized algorithm. If we want to apply it onto real networks, we need to develop a distributed version in the future. The traffic model should also estimated in different type, such as from CBR to VBR, and it should be able to deal with difference priorities.

Reference

【1】 3rd Generation Partnership Project, ``Technical Specification Group Services and Systems Aspects; Architecture for an All IP network", 3GPP TR 23.922 version 1.0.0., October 1999.

【2】 S. Floyd, and V. Jacobson, ``Random Early Detection Gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, vol. 1, no. 4, August 1993, pp. 397-413.

【3】 A. Demers, S. Keshav and S. Shenker, ``Design and Analysis of a Fair Queueing Algorithm", Proc. SIGCOMM'89, ACM, September 1989, pp. 1-12.

【4】 International Communication Union, ``Coding of Speech at 8kb/s Using Conjugate-Structure Algebraic-code-Excited Linear-Prediction (CS-ACELP)", ITU-T. G.729.1, March 1996.

【5】 D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell and J. McManus, ``Requirements for Traffic Engineering Over MPLS", RFC 2702, September 1999.

【6】 D. Ooms, B. Sales, W. Livens, A. Acharya, F. Griffoul and F. Ansari, ``Overview of IP Multicast in a Multi-Protocol Label Switching (MPLS) Environment", RFC 3553, August 2002.

【7】 E. Rosen and Y. Rekhter, ``BGP/MPLS VPN", RFC 2547, March 1999.

【8】 A. S. Tanenbaum, ``Computer Networks, Third Edition", Prentice Hall, March 1996, pp. 345-366.

【9】 Dijkstra, E.W., ``A Note on Two Problems in Connection with Graphs", Numerische Math, vol. 1, March 1959, pp. 269-271.

【10】C. Hedrick, ``Routing Information Protocol", RFC 1058, June 1988.

【11】J. Moy, ``OSPF version 2", RFC 1583, March 1994.

【12】Christophe Beaujean, ``Delay-Based Routing Issues in IP Networks'', *contact GRADIENT CR/98/148*, May 2000.

【13】Douglas S.Reeves and Hussein F. Salama, ``A Distributed Algorithm for Delay-Constrained Unicast Routing'', *IEEE Transaction on Network*, April 2000.

【14】R. Braden, L. Zhang, S. Berson, S. Herzog and S. Jamin, ``Resource Reservation Protocol (RSVP) – Version 1 Functional Specification'', *RFC 2205*, September 1997.

【15】K. Chan, R. Sahita, S. Hahn and K. McCloghrie, ``Differentiated Services Quality of Service Policy Information Base '', *RFC 3317*, March 2003.

【16】Bill Goodman, ``Internet Telephony and Modem Delay'', *IEEE Network*, May 1999, pp. 8-16.

【17】J. Garcia-Luna-Aceves and J. Behrens, ``Distributed scalable routing based on vectors of link states'', *IEEE J. Select on Communication*, October 1995.

【18】Jon Postel, ``Internet Protocol'', *RFC 791*, September 1981.

【19】Mark A. Sportack, ``IP Routing Fundamentals'', *Cisco ISBN: I-57870-071-x*, May 1999.

【20】R. Wideyono, ``The Design and Evaluation of Routing Algorithms for Real-Time Channels'', *International Computer Science Institute, Univ. of California at Berkeley, Tech Rep. ISCI TR-94-024*, June 1994.

【21】S. Rampal and D. Reeves, ``An evaluation of routing and admission control algorithms for multimedia traffic'', Proc. of the 5th IFIP Conf. on High Performance Networks, October 1995.

【22】S. Lavenberg, ``Mean Value Analysis of Closed Multichain Queuing Networks'', Journal of the Association for Computing Machinery, vol. 27, no. 2, April 1980, pp. 313-322.

【23】Z. Wang and J. Crowcroft, ``Quality of Service Routing for Supporting Multimedia Applications'', *IEEE Select on Communication*, September 1996.