TEKNILLINEN KORKEAKOULU
TEKNISKA HÖGSKOLAN
HELSINKI UNIVERSITY OF TECHNOLOGY

Comparison Between Pre-Computation and On-Demand Computation QoS Routing with Different Link State Update Algorithms

Haifeng Zhu

Master's thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Engineering

Espoo, Finland
November 2003

Supervisor Professor Raimo Kantola

Instructor Ph.D. Peng Zhang

HELSINKI UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ABSTRACT OF MASTER'S THESIS

Author: Haifeng Zhu

Title: Comparison Between Pre-Computation and On-Demand Computation QoS Routing with Different Link State Update Algorithms

Date: 23.10.2003 **Number of Pages:** 57

Department: Department of Computer Science and Engineering

Professorship: S-38 Network Laboratory

Supervisor: Professor Raimo Kantola

Instructor: Ph.D. Peng Zhang

As an extension to the current Best Effort service in the Internet, QoS routing may play an important role in providing quality guaranteed service. Extra cost is introduced by QoS routing. The cost mainly includes path computation cost and the cost for maintaining a QoS routing table.

This thesis begins with a brief introduction to the QoS routing, and then the issue of inaccurate routing information and its impact on network performance are discussed. After that, the pre-computation routing algorithm and the on-demand computation routing algorithm are presented, followed by the comparison between these two algorithms.

In order to run the simulations for this thesis, extensions to QRS simulator were made by adding new components and a Traffic Generator. Two new routing algorithms were added to simulate the pre-computation and the on-demand computation algorithms, respectively. By running simulations for these two algorithms with different link state update algorithms, the network performance is analyzed. Four different periods in PB and four thresholds in TB have been chosen for the simulations. Based on the simulation results, a conclusion is drawn: the frequency of link state update can affect the network performance and it costs significantly. Smaller link state update period or threshold can improve the network performance by reducing the inaccuracy in routing information, but the cost will also increase with more frequent link state updates. Thus, the update triggering policy should be chosen carefully.

Keywords: Quality of Service, QoS Routing, Pre-Computation, On-Demand Computation, Link State Update, Inaccurate Rouging Information

Acknowledgments

It is my fortune to have accomplished my thesis that was started in March 2003 in the Networking Laboratory of Helsinki University of Technology. This is also my latest achievement after more than two years of hard working.

I have been working hard to achieve this, but I clearly know that without the help, assistance and support from my senior, friends and family, I wouldn't have completed this thesis.

First of all, I sincerely thank my supervisor, Professor Raimo Kantola, who has been a pioneer and leader in this field of research, for giving me the opportunity in carrying out this work, and the support during the work.

My sincere thanks should also be given to my instructor, Dr. Peng Zhang, for his brilliant ideas, his giving me such a cherishing chance of doing the thesis under his direct help and instruction. I have been deeply moved by his devotion to research work and his unselfish help for others.

Finally, I would like to give my thanks to my dearest wife and my parents in China for their full support; my friends in Finland and colleagues in the Networking Laboratory for their help.

Table of Contents

Table of Figures

Acronyms

BE	Best Effort
BF	Bellman-Ford
DiffServ	Differentiated Services
DV	Distance Vector
ECB	Equal Class Based
FGTT	Flow Global Topology Table
FRT	Flow Routing Table
GTT	Global Topology Table
IntServ	Integrated Services
IP	Internet Protocol
LAN	Local Area Network
LS	Link State
LSA	Link State Advertisement
LTT	Local Topology Table
PB	Period-Based
QoS	Quality of Service
QOSPF	QoS Open Shortest Path First
QRS	QoS Routing Simulator
RM	Resource Management
RSVP	Resource Reservation Protocol
RT	Routing Table
TB	Threshold-Based
TCP	Transmission Control Protocol
UCB	Unequal Class Based

1 Introduction

1.1 Background

TCP/IP, which is widely used nowadays in networks including the Internet, was originally designed as a Best Effort (BE) transmission protocol. The best effort service seemed enough in the early stages of the Internet when most of the applications were limited in data sharing, including services as e-mail service and web browsing. With the rapid development of the Internet, more and more new applications have appeared, including online games, real-time multimedia, voice over IP, video conference etc. These new applications have a common traffic characteristic: they need something that traditional BE network can not provide, in fact, they require certain grade of Quality of Service (QoS), e.g., packet loss, delay, jitter and bandwidth guarantees [1].

On the other hand, most of the Telecom operators have both data traffic and voice traffic in their networks, and they have to maintain a packet switching network for the data traffic and a circuit switching network for the voice traffic. Maintenance of these two different networks needs more hardware, human resources and training of employees, thus it is costly. So more investments have been put into the IP network and more voice traffic has been transferred to the IP network by using the voice over IP technology. For example, packet data traffic in the AT&T network is about eight times as big as the amount of its voice traffic, and $3 billion will be invested in 2003 to transform the whole AT&T network into a pure IP-Based network [2]. Obviously, the IP network that can take over the traffic from the circuit switching network or other substitutes should provide QoS guarantees.

1.2 QoS Routing

Different kinds of Internet QoS architectures have been developed to achieve the required QoS in an IP network, e.g., Integrated Services (IntServ) and Differentiated Services (DiffServ). No matter which architecture is used, QoS routing may play an important role in the network that supports the QoS.

Routing for the Best Effort traffic, e.g. OSPF and RIP, use the shortest path to forward packets without considering the delay or bandwidth of the path. This is not enough for routing the traffic with bandwidth requirements. The QoS routing makes an extension to current Best-Effort routing. To be able to deal with the quality guaranteed traffic, QoS routing algorithm has to compute multiple paths for integrated-services, or consider the quality constrains (e.g. delay, bandwidth) when making routing decision [21].

To enable the nodes to make routing decisions, information exchange that makes it possible for the node to acquire link state information in the network is necessary. In general, there are two kinds of information exchange techniques: Link State (LS) and Distance Vector (DV). Distance vector is not discussed here.

There are different ways of advertising the link state information [3] [5] [8] [13] [16]:

In case of Period-Based (PB) Link State Updates, the link state information is broadcasted based on a predefined period. In this way, the loads on the network and routers introduced by link state updates are independent of the link state changes in the network, but other routers will not detect any significant changes until the period ends.

The following three methods trigger the link state update when the change in the link is big enough. Whether the change is big enough or not can be measured either by absolute value of

bandwidth or the percentage of change happened with the previous broadcasted bandwidth. The absolute value depends on the classes that the available bandwidth is divided into. There can be Equal Class Based updates (ECB) and Unequal Class Based updates (UCB).

These methods will make sure that significant changes to the link state will be broadcasted to the network in time, thus improving the accuracy of the link state information, but this may lead to heavier load for the routers. When the network is unreliable, for coping with the loss of link state update packets, a timer needs to be added to prevent too long periods between updates. When the network is unstable, for preventing too high frequency of updates from overburdening the network, a timer with a preset hold-time can also be added.

In Threshold-Based (TB) Link State Updates, the information is to be broadcasted when the change in the link is bigger than the predefined threshold. A certain percentage of previous broadcasted bandwidth is set as the threshold. *B0* stands for the previous bandwidth, *B1* is the new bandwidth and *th* is the threshold in percentage, if $|(B1 - B0)/B0| > th$, then the update will be triggered.

In Equal Class Based Updates, the available bandwidth has been divided into several classes of equal size: *(0, B), (B, 2B), (2B, 3B)*…When the change to the bandwidth of a certain link is big enough to make it move from one class to another, an update will be triggered.

In Unequal Class Based Updates, the available bandwidth has been divided into several classes of unequal size. Typically, exponential class size is implemented where the size of base class (*B*) and the factor (*f*) are defined, and then the bandwidth will be divided to: *(0, B), (B, (f+1)B), ((f+1)B, (f^2+f+1)B)*… like the previous one, the update is triggered when the border of the class is crossed.

1.3 Path Computation Algorithms

There are also different ways of computing the path [3]:

In the Pre-Computation Algorithm, the path from each vertex to all of the destinations is computed periodically or after certain number of link state updates. Then when a node initials a request, it just picks a path from the stored routing table. Thus the cost of computing paths for the whole network can be shared by several requests.

In the On-Demand Computation Algorithm, the path to a certain destination is computed every time a request is initiated. So computing the path to the specified destination is what needs to be done.

These two algorithms both have advantages and drawbacks. They are described further more in the next Chapter.

1.4 Inaccurate Routing Information

Non-optimal routes normally occupy more links than the optimal one. This wastes the resources of the network and introduces more delays to the packets. Thus, the utilization and performance of the network are affected by the quality and correctness of the path selection. The path selection depends not only on the algorithm and criteria of routing, but also on the accuracy of the link state information.

The link state information, representing the resource and status of the nodes and links in the network, is spread over the network by using the Link State Advertisements (LSAs). If an LSA is sent whenever a change happens to the link or node and no LSA packet is lost, then the link state information kept by each router will be accurate. But broadcasting the LSAs for all

changes is infeasible due to the huge overhead to the network, especially in a large size network or highly dynamic network that tends to have more frequent changes. Thus, as we have mentioned above, the link state information will be updated either periodically or be triggered by significant changes in a practical implementation. Between two subsequent updates, the link state information maintained by each router may not represent some changes happened to the network, and in an unreliable network, the LSA packets may also get lost, so the link state information of the network can not always be updated accordingly. So sometimes, the routing decision has to be made based on inaccurate link state information.

The accuracy of the link state information for routers depends on the frequency of link state updates, while the frequency of link state updates is decided by the sensitivity of the update triggering policy. Obviously, more accurate information can be achieved by increasing the sensitivity of triggering policy, for example, shortening the period in PB, lowering the threshold in TB, decreasing the class size in ECB and UCB. But improvements in accuracy gained from implementing the above methods will have to be paid by burdening more update traffic and more computation overhead on the network.

When the routing information is inaccurate, the path selection based on the information may be non-optimal, or even incorrect. Non-optimal path selection decreases the utilization of the network and incorrect path selection leads to more blocking that damages the network performance.

There are also some other reasons that can lead to inaccuracy of the routing information, for example, temporal conditions like congestion in the network, and information aggregation in a large network [6][7]. These are not studied in this thesis.

1.5 Purpose and scope of the thesis

Both on-demand QoS routing and pre-computation have pros and cons depending on some factors such as network size, traffic requests and so on. It is normally regarded that the on-demand QoS routing algorithm is suitable for small-scale networks while the pre-computation algorithm for large-scale networks. Also, the on-demand QoS routing algorithm is more suitable for networks where requests arise infrequently while the pre-computation algorithm for networks where requests arise very frequently. Nevertheless, there have been many on-demand QoS routing algorithms but a few pre-computation algorithms presented in recent years. Considering network diversities, it is unclear yet how to apply these two different routing algorithms into a real network, for example, whether the common claims are true, to how much frequency of requests the on-demand algorithm can work well, to how big network the pre-computation network should be used, etc.

Our study aims to investigate these problems and intends to present some observations how these algorithms can differ in terms of network performance when given various network environments, e.g., link state updating algorithms, network size, traffic, etc. Thus, our work can be used for guiding the deployment of either routing algorithms in real networks.

1.6 The structure of the thesis

The rest of the thesis is organized as follows:

In Chapter 2, we give some further discussion about the Pre-Computation and the On-Demand computation algorithms, followed by a brief comparison.

In Chapter 3, descriptions are given about the simulator we are using as well as the extension to the simulator that we have made in order to carry out the simulations for this thesis

The results from the simulations and the performance analysis are presented in Chapter 4. The simulations are carried out in the networks with different topologies. In order to find the influence of inaccurate link state information upon networks with different traffic loads, in each topology, we run the simulations with different routing algorithms, different link state update methods and different periods or thresholds.

In Chapter 5, we give the conclusions.

2 Pre-Computation vs. On-Demand QoS Routing

Depending on the path computation triggering criteria, there are two main QoS path computation algorithms, i.e. the Pre-Computation and the On-Demand computation algorithm.

For the pre-computation algorithm, the path computation can be triggered either periodically or when a path cannot be found from pre-computed results for a request. For the on-demand computation algorithm, the path computation is triggered when a new request arrives.

2.1 Pre-Computation QoS Routing

To cope with different requirements of QoS requests, there must be a QoS specific routing table for the QoS guaranteed traffic that is similar to the routing table used for the best effort traffic. When pre-computing the paths, the QoS requirement is unknown, thus the QoS specific routing table may have several paths for a specific source-destination pair due to different kinds of QoS requirements. For instance, for the traffic that has a bandwidth requirement, requests may need different amount of bandwidth, which may lead to different routing selections. While it is impossible to make an entry for any possible required bandwidth, a practical approach is to group the requests into several classes. Those requests in the same class have a similar bandwidth requirement, and then, for each class in a source-destination pair, the route is pre-computed and stored in the QoS routing table. Another approach does not classify requests. On the contrary, it records the bandwidth of route with minimum hops as well as those routes with more hops but having a larger amount of bandwidth. For an incoming request, the route with minimum hops will be checked first. And if the bandwidth is not enough, then other routes with more hops will be checked. [4]

Advertisements of the link state information and the computation of the paths consume a considerable amount of network resources, for example, the bandwidth of links for information

exchange, processing capability of routers for route computations and memory for storage of the routing table. These are all resource consuming. More frequent path computation increases the quality of path selection, but it consumes more network resources and increases the complexity of implementation. In a large backbone network, these overheads can be significant. Then controlling these overheads requires a careful trade-off between routing accuracy and complexity [9].

In general, the pre-computation algorithm has the following advantages [10]:

Firstly, the pre-computation algorithm has better scalability. When the network gets bigger, the number of nodes and links in the network will increase accordingly. More nodes and links tend to have more changes, thus the number of LSAs for keeping the link state information in each node grows more, and the processing load for path computation based on the information also increases.

For periodical pre-computation, the number of path computation in the pre-computation algorithm is independent from the number of requests in the network. Typically, networks with larger size have more requests in a unit time, the pre-computation algorithm, which can make the computation load shared by several requests, can reduce the overall computation load.

Secondly, the pre-computation algorithm is fault tolerant. There may be failures of nodes or links in networks, and a solution can be bypassing those failure parts by using alternative routes. By using pre-computation, the alternative routes can be computed in advance to handle the failures.

Thirdly, the pre-computation algorithm can improve the performance in busy time. During the busy time, there may be much more requests than on average. Since the paths have been computed and stored in the routing table in advance, comparing with computing the path on

demand, picking a ready path from the QoS routing table saves time in handling the requests. Thus, the pre-computation is more robust in dealing with bursts of traffic.

Finally, the pre-computation algorithm can improve load balancing. In the pre-computation algorithm, several routes are computed in advance for the same destination, and then the available resources can be allocated to different requests more efficiently. Thus the traffic in the network can be balanced by directing different requests to different alternative routes properly.

Above all, the pre-computation algorithm is considered as a solution to improve the scalability and the response time, to reduce the processing load to the network by sacrificing certain grade of routing quality. By computing the route for each destination in advance, the algorithm can reduce the respond time for requests. It can also reduce the overall computation load to the network in a considerable level when the number of the requests is high. Such kind of advance computation is not rather time critical, so it can be done in a background process to balance the network-processing load. [10]

But the pre-computation for all the destinations may waste lots of resources if only a few of the pre-computed paths are used. For example, in the ad hoc network, each node is more likely to communicate with only a few nearby nodes, so pre-computing paths to all destinations is unnecessary and inefficient.

2.2 On-Demand QoS Routing

The pre-computation routing algorithm trades the quality of routing decision for lower processing load to the routers and less bandwidth consumption to the links. However, with the development of computer hardware and transmission technologies, the cost for processing capability of routers and the cost for bandwidth have been becoming more affordable. Then, in

order to get better quality of route selection, the on-demand routing algorithm, which has been blamed for its high computation overhead, can now be deployed on a larger scale.

By using the latest link state information that is available in the network, the On-demand computation can make better routing decisions than the pre-computation due to the improved accuracy of routing information.

Unlike the pre-computation algorithm, the on-demand computation does not need to compute the routes to all destinations or for different QoS requirements. The on-demand path computation is carried out only on the arrival of a request. Thus, the destination is specified. As to the QoS traffic with bandwidth requirement, the required bandwidth is also known. So only one path that fits the QoS requirement to the certain destination is necessary to be computed. The computation algorithm can be simpler compared to the computation of multiple paths to different destinations in pre-computation.

According to the above description, path is only computed every time a new request is initiated. For the networks with fewer requests, the on-demand computation can be an efficient algorithm due to lower processing load by less path computations.

Thus, without the maintenance of the QoS routing table and less path computations, the implementation of the on-demand computation algorithm is much simpler compared to that of the pre-computation algorithm. The path computed is not necessary to be stored in the routing table for the future use, so the storage for the routing table can also be saved. [4]

With some extensions to the on-demand computation, the cost and the number of path computations can be reduced.

A path caching architecture is discussed in [4]. In this architecture, there is a path cache for storing the paths that are computed on demands of the previous requests. To each new request,

the on-demand computation is triggered only if the route needed can't be found from the path cache. After the computation, new path will be added into the path cache for future use. The result of simulations in [4] shows that the path cache reduces the processing cost while the storage of cache is comparable to or even more than that of the pre-computation algorithm.

A further study about path cache can be found in [11]. This paper studies the trade-off between the cost and the network performance by using different granularities for cache. The simulation results show that networks with proper selected hybrid granularity scheme for cache performs well with proper amount of storage.

2.3 The Comparison between these two algorithms

Both of these two algorithms have advantages and drawbacks. The selection of an algorithm should take into account the following: network size and network characteristics, for example, stability, reliability and number of requests, etc.

The comparison between these two algorithms is discussed below. There are extensions to on-demand computations, including path caching, which can improve the performance by using extra storage. This can be considered as an approach between these two algorithms. Yet, it is not discussed here.

The Storage: Compared to the on-demand computation, the pre-computation algorithm needs extra storage space for the storage of multiple paths.

The Complexity: For the pre-computation algorithm, paths for different destinations and different QoS requirements must be computed. But for the on-demand computation algorithm, only one path needs to be computed under the request of a certain QoS requirement and destination.

The extra storage for the routing table and more path computations make the pre-computation algorithm more complicated in implementation.

The Processing Load: In the pre-computation algorithm, the path computation overhead can be shared by several requests, so in the network with many requests, the overall computation overhead can be diminished. But in other situations, for example, in the Ad Hoc network, nodes tend to communicate only with nearby nodes, or with networks with fewer amounts of requests. Most of the pre-computed paths may never be used. Thus, the pre-computation algorithm wastes lots of processing capability.

While in the on-demand computation, path computation is needed for every request. Those networks with many requests will generate large computation overheads. This is the main obstacle for its scalability.

Thus, the selection of algorithm depends on the size and the characters of network.

The Quality of Path Selection: For the pre-computation algorithm, there is a period between a path computation and handling of a request and it depends on the frequency of the path pre-computation. Yet, changes may appear during this period, thus when handling the request, the routing information has become inaccurate. And paths stored in the routing table based on inaccurate routing information can be non-optimal or invalid.

For the on-demand computation, paths are computed based on the latest routing information, i.e. path computations are always based on more precise routing information. Thus the quality of path selection is better.

Compared with picking up path from the routing table, the on-demand computation takes more time in path selection than the pre-computation algorithm. This introduces more delay to packets.

In this thesis, we compare these two algorithms by running simulations with different link state update algorithms. Four periods in period-based and four thresholds in threshold-based are chosen to carry out the simulations.

3 The Simulation Environment

A brief introduction to the simulator we used to carry out the simulations is given here. Some extensions to the simulator that were necessary for our simulations are also introduced, including new routing algorithms and a new traffic generator for generating the traffic with QoS requirements.

3.1 Basic Information about QRS (QoS Routing Simulator)

In this thesis, we use QRS as our routing simulator. It was developed at Helsinki University of Technology for the evaluation and comparison of QoS routing algorithms in IP networks. Networks with different topologies are modeled with configuration files, and then the results exported from the simulator can be checked from the log files [8] [13].

QRS models the network as a combination of different kinds of components. For example, the component "Node" represents the practical nodes, using parameter "delay to process a packet" to represent the processing speed and "Buffer space" to represent the buffer. The "Link" component represents the links with characteristics of bandwidth and propagation delay [8].

The "Real-time Traffic" component initiates traffic with QoS Constrains, i.e. bandwidth requirement. It contains source and sink that are connected to the source nodes and the sink nodes respectively. The routing algorithms are implemented in the "QOSPF" component, and the implementations of these two new routing algorithms are added into this component. Every node has a "QOSPF" component to maintain the routing information. The functionality of signaling path setup is simulated by the "RSVP" component. The "RM" component is responsible for the resource reservation. Every node has a "RSVP" and a "RM" component [8].

With these components, the basic procedure for the set-up of a flow is like this: Node requests RSVP for flow set-up, and then RSVP inquires QOSPF for information about the next hop, after that, RSVP sends PATH message to the next hop according to the reply from QOSPF, finally, if an acknowledgement from the destination is received, RSVP requests RM to reserve the resources.

A certain number is allocated for every Main "Quality of Service Open Shortest Path First" (QOSPF) action in the simulator to simulate the practical cost in real implementation as shown in Table 1 below. During the process of simulation, the total sum comes from adding up all the costs of involved actions.

Name	Cost	Description
DEFAULT_QOSPF_PROC_COST	100	Processing of QOSPF
DEFAULT_RM_QOSPF_PROC_COST	50	Processing of QOSPF Resource Management
DEFAULT_RSVP_QOSPF_PROC_COST	10	Check message in RSVP
DEFAULT_RSVP_QOSPF_COMP_PROC_COST	150	Computation of routing tables
DEFAULT_LS_BROADCAST_COST	20	Broadcast the Link State information
DEFAULT_BROADCAST_PKT_COST	10	Broadcast the packet
DEFAULT_QOSPF_COST	100	Periodically computation of routing table
DEFAULT_QOSPF_LOOKUP_COST	50	Path lookup from BF Routing Table
DEFAULT_QOSPF_COMP_COST	100	On-Demand computation of path

Table 1 Costs for different QOSPF actions

In order to simulate the costs for the path lookup in the pre-computation algorithm and for the path computation in the on-demand computation algorithm respectively, we add in the last two parameters in the above table.

With the execution of every step in the simulation, the costs of all of the involved QOSPF actions are summed up one by one. After finishing the simulation, we can get a summed up value that is the total cost of all involved actions. And we use this value to simulate the costs

for flow set-up and for tearing down as well as other activities needed in real networks. In order to decrease the affect from the size of networks, we divide the cost by the number of nodes in a certain network, and get the value of cost per node.

3.2 New Routing Algorithms

We developed two new routing algorithms besides the previous ones in QRS. Algorithm 4 uses the Pre-Computation algorithm and algorithm 5 uses the On-Demand computation algorithm. Both of these two algorithms are based on the Bellman-Ford (BF) shortest path algorithm and the pseudo code can be found from [3]. For algorithm 5, only the path to a certain destination is computed. And as soon as the path with enough bandwidth is found, the calculation stops. [18] [19]

3.2.1 The Pre-Computation Routing Algorithm

The routing table for the BE traffic is computed periodically, so we add in the computation of the BF routing table after finishing the BE routing table computation. Then the BF routing table for this algorithm is computed at the same frequency of the normal routing table.

In every node, there are Routing Table (RT), Flow Routing Table (FRT), Local Topology Table (LTT), Global Topology Table (GTT) and Flow Global Topology Table (FGTT). The RT stores the paths to different destinations. Each entry contains a destination, “next hop” and the cost. The FRT stores the bandwidth of the link to the next hops. The LTT contains the information of neighbouring components. The GTT and the FGTT are used for maintaining the link cost and link bandwidth between different nodes respectively.

The Bellman-Ford Routing Table (BFRT) is added to the simulator for our simulation. The BFRT has an entry for each destination. And for each entry, bandwidth and "next hop" information are stored for different numbers of hops.

The Computation of the BF Routing Table:

1. Initialization: To each node in the BF routing table, set the bandwidths of its neighbouring nodes to 0 and the "next hop" to NULL.

2. Hop 0 and 1: Hop 0 is the starting node, so set its bandwidth to INFINITY. Check the bandwidth and link information from the LTT, and then set the bandwidths and "next hop" of the nodes that have direct connections with the starting node.

3. Hop 2: Set the bandwidths and "next hop" of the nodes that are directly connected to the nodes with hop count 1 according to the information in the FGTT.

4. Other nodes: Repeat step 3 to nodes with more hops count one by one until all nodes have been processed or the hop counting exceeds the maximum value--MAX_HOPS.

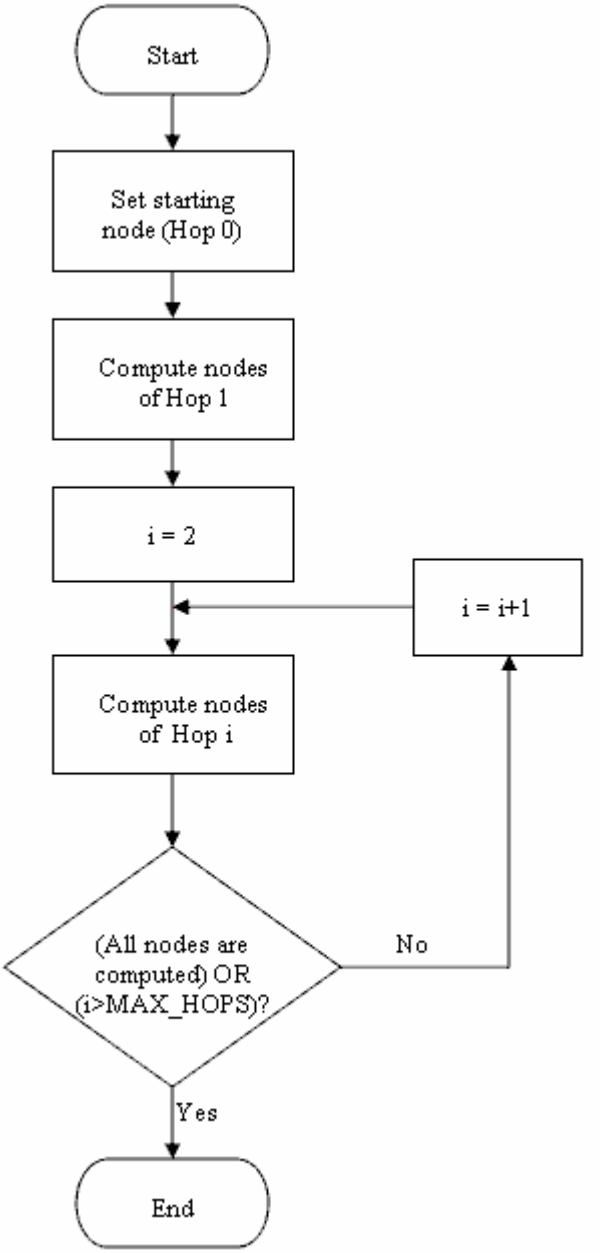

Figure 1 The Flowchart for computation of the BF routing table

Path Selection based on the BFRT goes as follows: Check the entry for the destination from the BFRT and go through the entry from hop 1, if there is a bandwidth in the entry of the destination larger than the required bandwidth, the path is found. Otherwise, finding of the path fails.

3.2.2 The On-Demand Computation Routing Algorithm

The implementation of the On-demand computation is simpler compared to the previous one. The computing step is similar to the computing of the BF routing table, but the available bandwidth is compared with the required bandwidth when computing. As soon as the path that fits the bandwidth requirement is found, the computation will stop.

The process of this algorithm is described as follows:

1. Initialization: Give initial values of "next hop" and bandwidth to each path entry.

2. Hop 1: Compute the nodes with hop count 1, and then, compare the available bandwidth with the required bandwidth of the specified destination, if the available bandwidth is larger, then the route is found and path finding stops.

3. Hop 2: First, Compute the nodes with one more hop count compared to the previous ones, and then compare the available bandwidth with the required bandwidth of the specified destination. When a path with a larger bandwidth is found, then the path needed is found and the path finding stops.

4. Other Nodes: Repeat step 3 to nodes with more hops count one by one until the path needed is found, or until the hop number exceeds the MAX_HOPS, or until all of the nodes have been checked. If the path can't be found, then the path finding fails and stops.

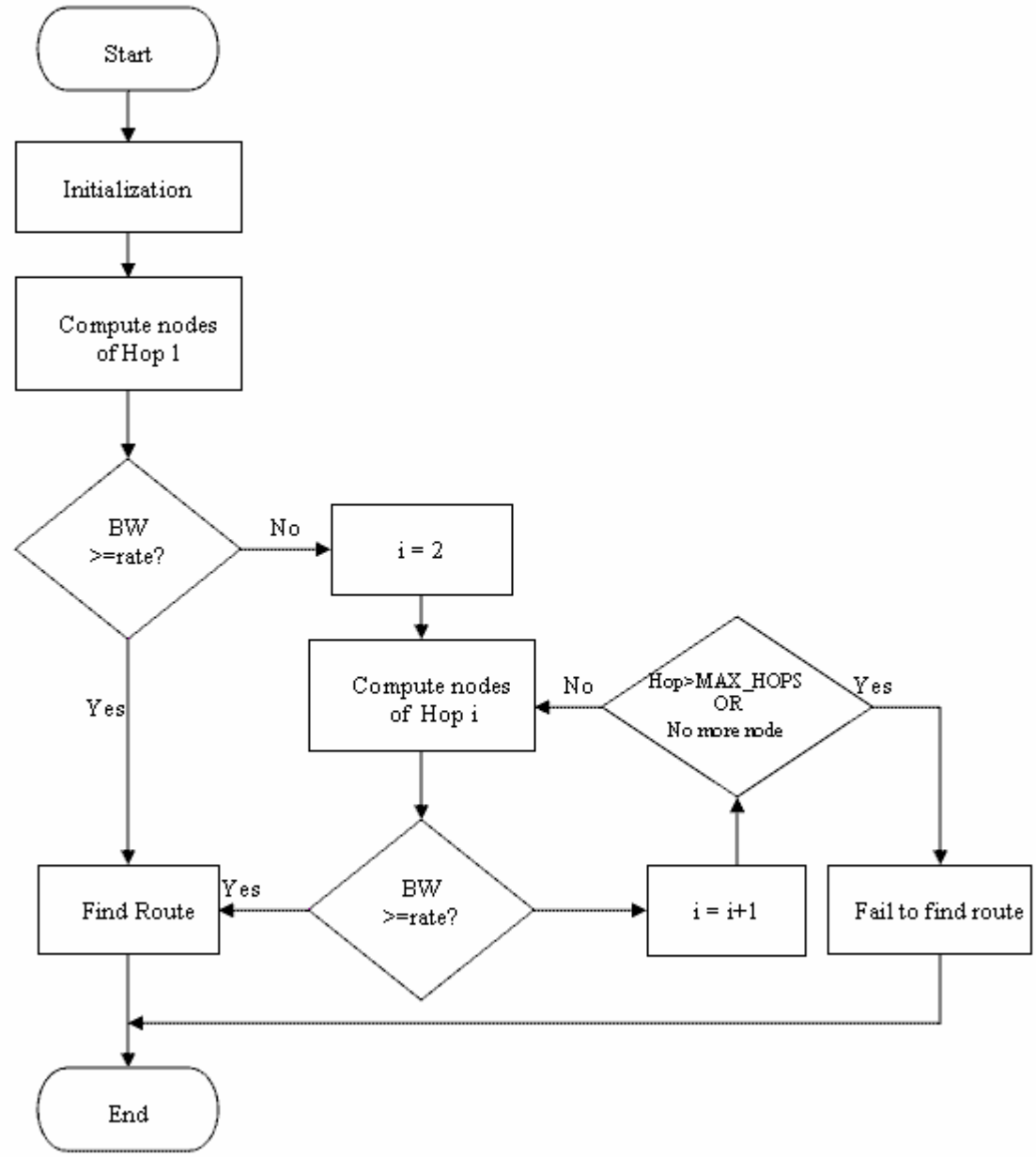

Figure 2 The Flowchart for the On-Demand Computation Algorithm

3.3 The Traffic Generator

At the beginning, in order to simulate the network performance under different traffic loads, we used the "Real-time-Traffic" component for producing different amounts of QoS constrained traffic. But soon we noticed that the results mainly depended on the request scenario of the "Real-time-Traffic". With the increasing amount of the traffic, the blocking rate rises quickly

from 0 to 100%. Then we decided to develop a new component, that is, the “Traffic Generator”, which can generate requests. By using the “Traffic Generator”, we can control the character of the generated traffic requests.

The “Traffic Generator” is developed based on the “Real-time-Traffic”, and some of the input parameters for this component are listed as follows: [8]

Average packet length specifies the size of the packet in byte. In our simulations, this parameter is always set to 512 bytes.

Average interval of requests specifies the average interval between requests in μs. In our simulations, the traffic load is adjusted by changing the value of the interval.

Standard deviation of requests specifies the standard deviation of requests. This parameter is set to 100 in our simulations.

The time of starting request specifies the time from the start of simulation to the first request produced in μs. To reserve enough time for the initialization of the simulator, this parameter is set to 1000 in our simulations.

Interval of traffic producing specifies the source active interval in μs. This parameter is set to 2s in our simulations.

Interval of traffic pausing specifies the source pausing interval in μs. This parameter is set to 1s in our simulations. Combined with *Interval of traffic producing*, we set the source to pause one second after every two second’s producing of real-time packets. Thus, we can control the percentage of the time for producing traffic.

Average delay between packets specifies the average delay between packets in μs. The traffic

load can also be adjusted by changing this parameter. For packet length is 512*8=4096 bits, to get 1 Mb/s=1 b/μs flow rate requests, the packet should be sent every 4096 μs. Thus, this parameter is set to 4096 in our simulations.

Maximum flow index and the "*Minimum flow index*" specify the range of the flow index that can be used by requests. Each request in the network should have a unique flow index.

Minimum flow index is used together with the "Maximum flow index" for defining the range of the available flow indexes.

Routing method specifies the routing method for the QoS routing. In our simulations, this parameter is set to 4 or 5, using the pre-computation routing algorithm and on-demand routing algorithm respectively.

Flow rate specifies the flow rate of the source. Its unit is Byte/s. This parameter should be synchronized with the "Average delay between packets", i.e. the traffic loads should be identical. For example, to get a packet flow with 1 Mb/s flow rate, the *Average delay between packets* is set to 4096 and the *Flow rate* should be set to 125000 corresponding to 1Mb/s.

STAN-DARD	JPEG ISO/IEC 10918-1 ITU-T T.81	H.263 ITU-T3615 LBC H.261/p x 64 ITU-T 3615	MPEG-1 ISO/IEC 11172-2	MPEG-2 ISO/IEC 13818-2 ITU-T H.262	MPEG-4 ISO/IEC 14496-2
Functionality	Compression of photographic images	Low bitrate compression for video comunication	interactive Video retrieval from CD-ROM	generic high quality video coding	generic object-based video coding
Source resolution (pixels x lines)	64 k x 64 k No fixed formats	CIF 352x288 QCIF 176x144 SQCIF88x72	SIF 352x288	CCIR 601 720x576 (MP@ML)	SQCIF to CCIR progressive & interlaced
Bit rates [bit/s]	N/A	8 kb/s to 2 Mb/s px64 kb/s	⇨1.856 M (CPS)	⇨15 M (MP@ML)	5 kb/s to 4 Mb/s
min. Data Unit	8 x 8 pels	16 x 16 pels	16 x 16 pels	16 x 16 pels	8 x 8 pels
Temporal prediction	N/A	forward/ bidirectional	forward/ backward/ bidirectional	forward/ backward/ bidirectional	forward/ bidirectional
Motion comp. range/ resolution	N/A	H.263: ± 32/ 0.5 H.261: ± 15/ 1	± 128/ 0.5	± 127 (V), ± 1023 (H) /0.5	resolution dependent; half pel
Rate control	No	Yes	Yes	Yes	Yes; multi video objects

Figure 3 International Compression Standards [22]

As we can see from Figure 3, MPEG-1, which is a standard designed for video CD, has typical bit rate of about 1.5 Mb/s. While MPEG-2, which is designed for TV broadcasts including HDTV and DVD, has typical bit rate from 4 to 8Mb/s. And MPEG-4 for interactive audio-visual communication has typical data rate from 5 kb/s to 4 Mb/s [22]. On the other hand, 10Mb/s or 100Mb/s Ethernet is common in a typical Local Area Network (LAN).

In our simulations, we tend to simulate video streams or video conferencing packets flow in a typical Ethernet LAN. Thus, we set the bandwidth of the links in the network to 20Mb/s, and the flow rate of the requests generated by our traffic generator to 1 Mb/s.

The requests are initiated and routed on a basis of flow by flow. When the flow rate is 1 Mb/s, it contains 250 packets of 512 bytes. These 250 packets are considered as a whole and are routed using the same path.

4 Simulation Results and Analysis

The simulations are carried out using different topologies. These topologies include tree topology, matrix 2*2, matrix 3*3 and matrix 4*4.

We run the simulations by using the PB and the TB routing algorithms. For the PB, the link state update periods we select are 100ms, 200ms, 500ms and 1000ms. For the TB, the link state update thresholds we select are 10%, 40%, 60% and 80%. Then for every period and threshold, simulations with different request rates are run using different algorithms.

4.1 Tree Topology

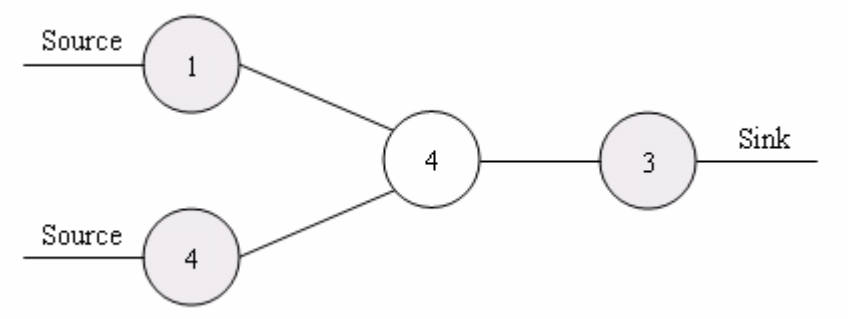

Figure 4 The Tree Topology

The new routing algorithms are tested firstly in the tree topology. Since the tree topology only has 4 nodes and three links, and also because of the random factor in the simulator, the performance of the two different algorithms only vary a little in different simulations, but no algorithm has a dominating advantage over the other one. As to the different link state update periods in the Period-Based link state update method and different thresholds in the Threshold-Based link state update method, the results are also similar. Thus, since the topology of the tree is quite simple, we cannot get conclusions about the difference between the two

routing algorithms and between different link state update methods. But by trying the tree topology, we can test the functionality of the simulator and debug the newly added routing algorithms and components.

4.2 The Matrix 2*2 Topology

4.2.1 Basic information of the topology

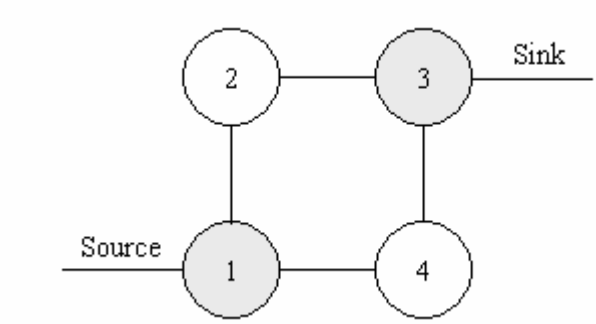

Figure 5 The Matrix 2*2 Topology

The Matrix 2*2 has four nodes with four links connecting these nodes. The bandwidth of each link is set to 20Mb/s. The links will never fail, and the flow rate of the source is 1 Mb/s. Traffic load changes are made by changing the average interval between requests. In this topology, the average interval between requests is set from 100ms to 30ms, so the corresponding number of requests per second varies from 10 to 33.33.

4.2.2 The Result and analysis

The blocking rates and costs for different periods and thresholds are shown as follows:

For the Period-Based Routing Algorithm:

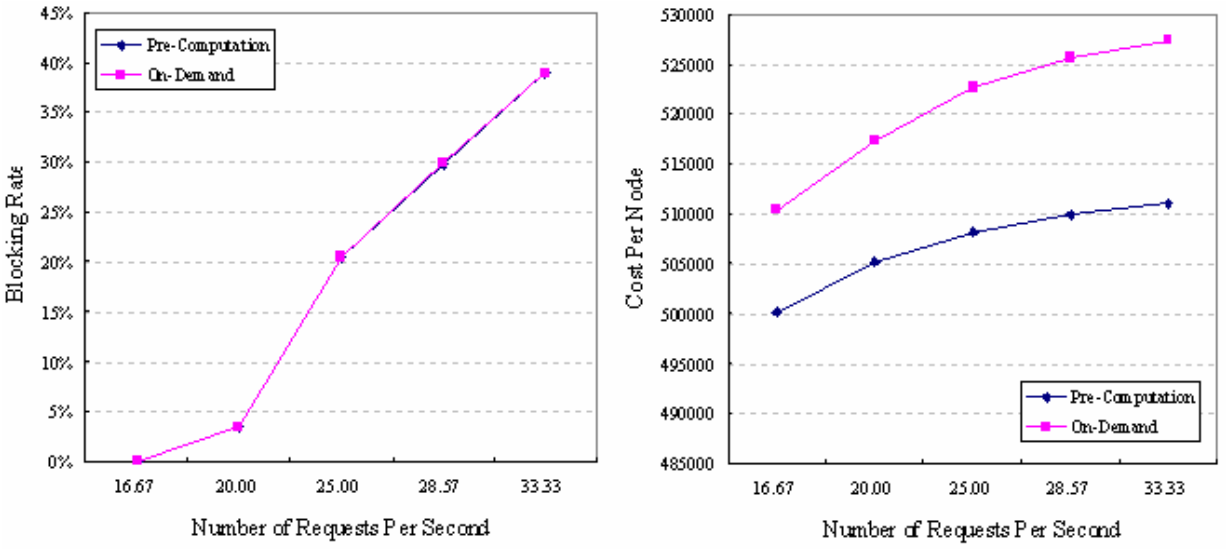

Figure 6 The Period-Based Link State Update Method, Period=100ms

Figure 6 shows the blocking rates and costs of the two algorithms under different traffic loads. We can see that the network performance of these two different algorithms is almost identical. The cost of the pre-computation is always lower than that of the on-demand computation and the gap between these two algorithms is getting larger with the increase of the request rate.

For the pre-computation algorithm, path computation is done periodically, the computation overhead does not change for different amounts of requests, but the processing overhead for handling the requests does. For the on-demand computation algorithm, more requests mean more computations. Thus, we can see that with the increase of request rate, the costs of these two algorithms grow, but the cost of the on-demand computation grows quicker due to more path computations.

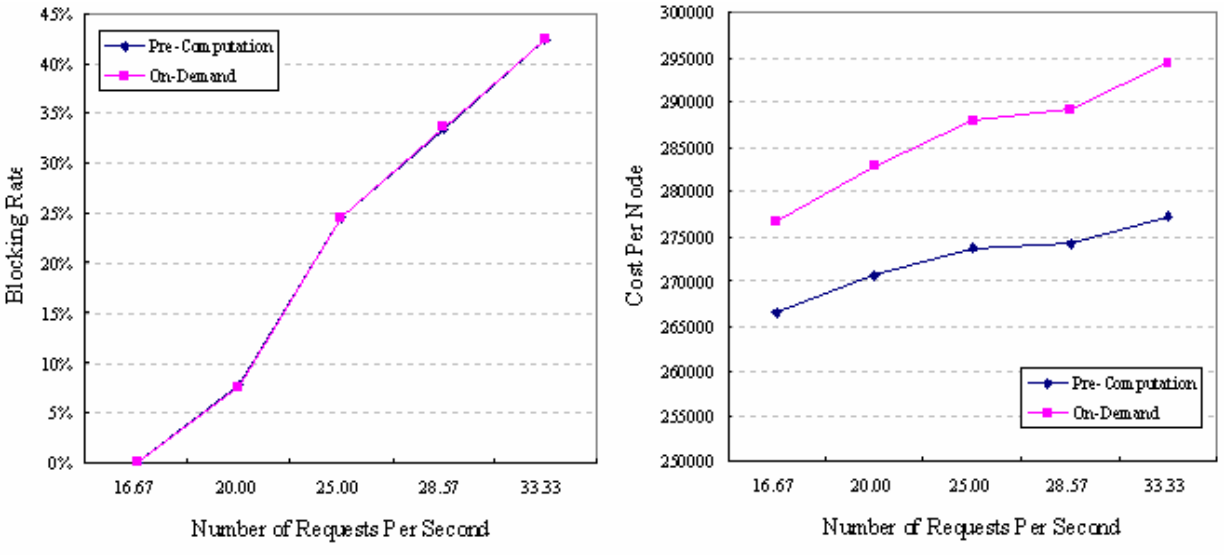

Figure 7 The Period-Based Link State Update Method, Period=200ms

Figure 7 shows that the results are similar to Figure 6, and these two algorithms do not have much difference in blocking rates. Compared to the on-demand computation algorithm, the pre-computation algorithm has lower cost, and the difference between them becomes larger with the increase of the request rate.

With the link state period of 200ms, the number of link state updates is only half of that with the period of 100ms. So we can see from this figure, the costs of these two algorithms are much smaller, they are just a little bit more than half of that in the previous one in which the link state period is 100ms.

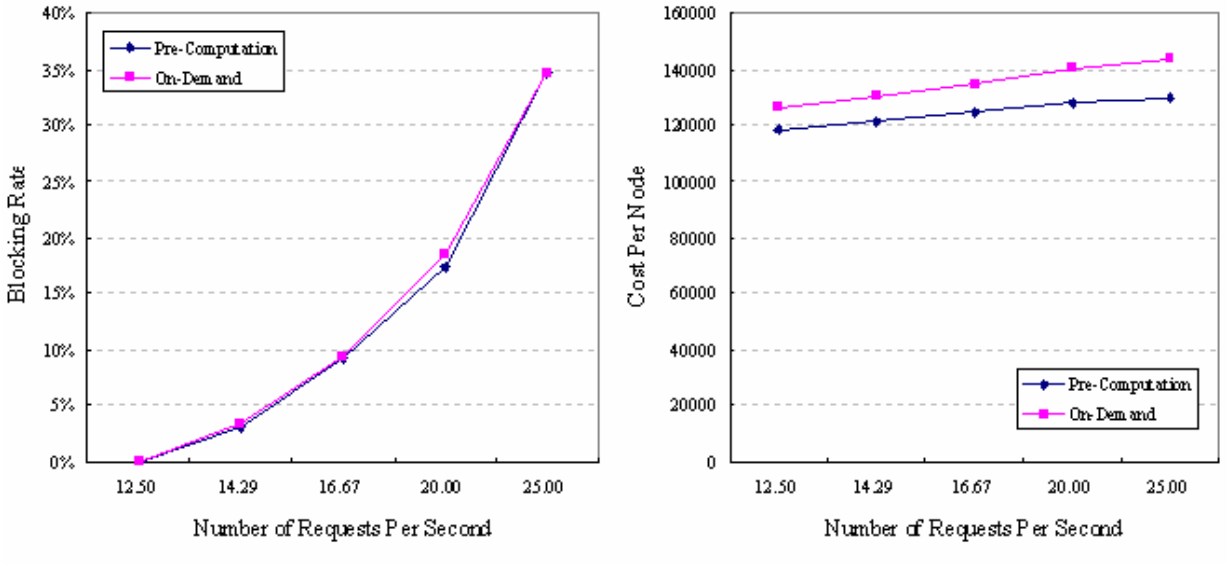

Figure 8 The Period-Based Link State Update Method, Period=500ms

Figure 8 shows that the costs of the two algorithms are about half of the costs in Figure 7 due to the same reason as previously. With period=500ms, there is still not much difference in blocking rates for these two algorithms, and the difference between the costs is smaller.

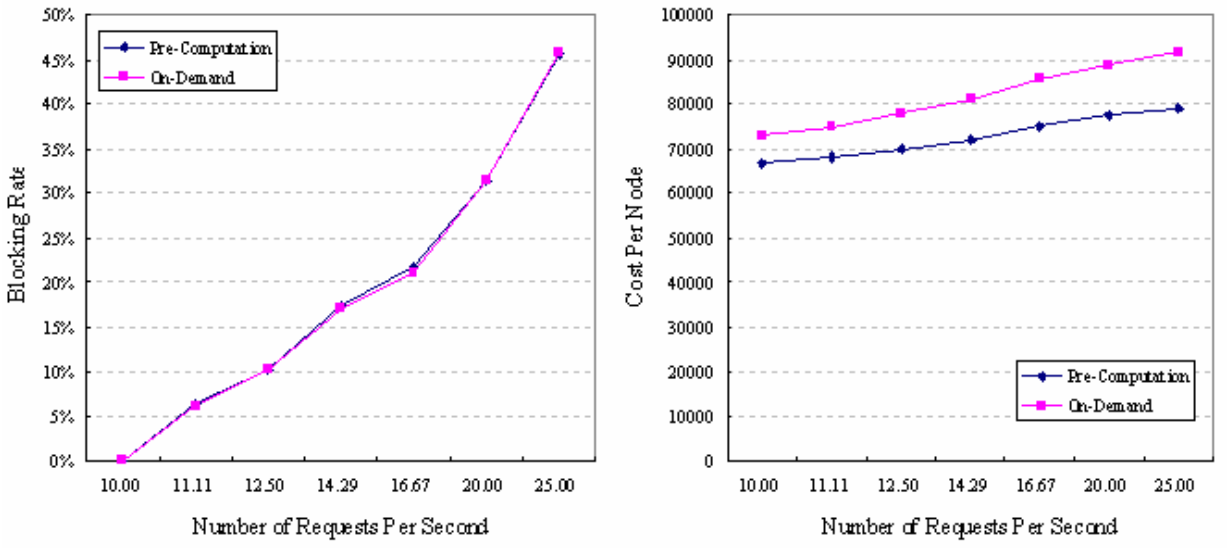

Figure 9 The Period-Based Link State Update Method, Period=1000ms

From Figure 9, we cannot tell which routing algorithm performs better from the aspect of blocking rate. The costs of these two algorithms are smaller than previously because of larger link state update periods.

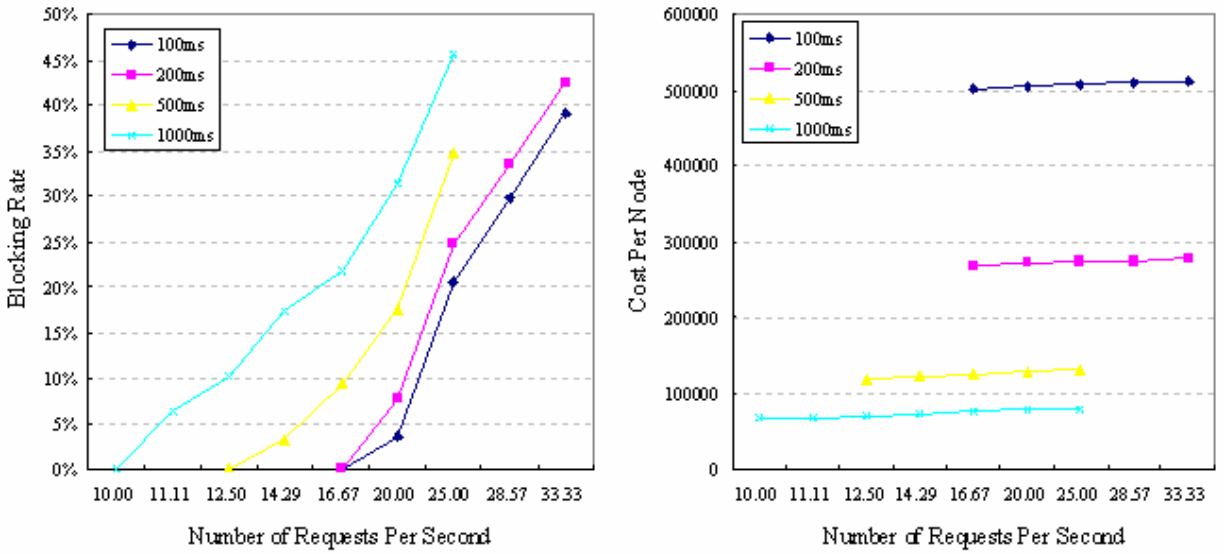

Figure 10 The Period-Based Link State Update Method with different periods (Pre-Computation)

When the request rate is lower than 16.67 per second, the blocking rates for the periods of 100ms and 200ms are both 0%. Thus these simulation results are not shown in Figure 10. Those results of the request rate lower than 12.5 per second when the period is 500ms are not showed due to the same reason.

When the link state update periods are 100ms and 200ms, the numbers of link state updates are 1000 and 500 per second respectively, we found that the blocking rates are not much different between them with different request rates. But when the link state update period is 1000ms, since the update period is much longer, and more changes may happen during the period between two subsequent updates, so the inaccuracy of link state information is more serious, thus the blocking rate is much higher than in other cases.

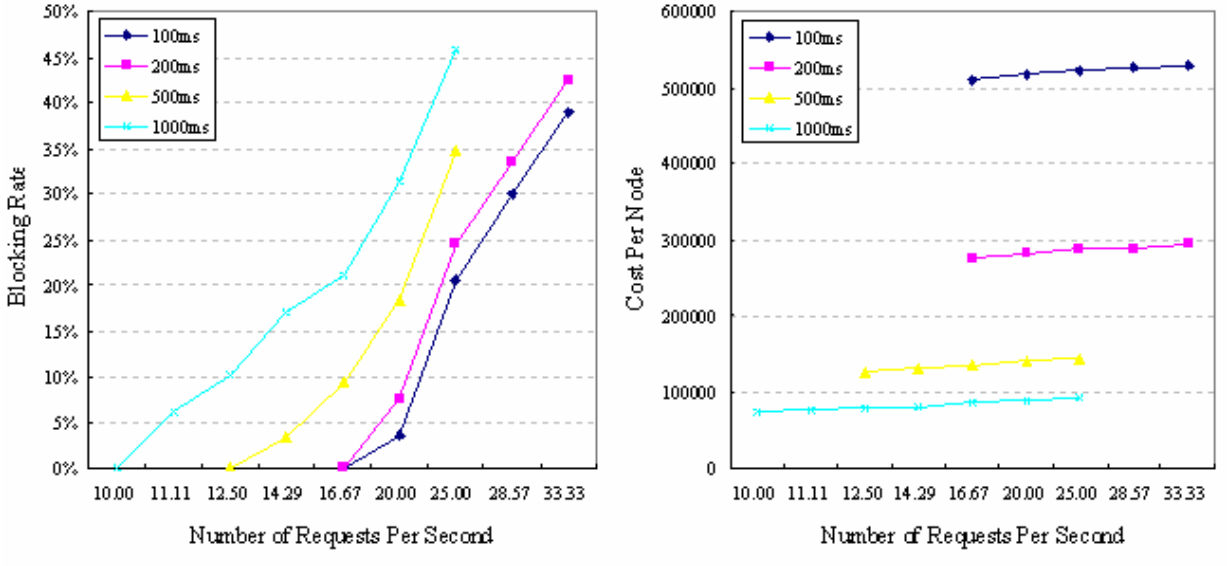

Figure 11 The Period-Based Link State Update Method with different periods (On-Demand)

As we have described above, with different link state update periods, the blocking rates are almost the same for these two algorithms. Thus, Figure 11 draws out the same conclusion as Figure 10.

For the threshold-Based Routing Algorithm:

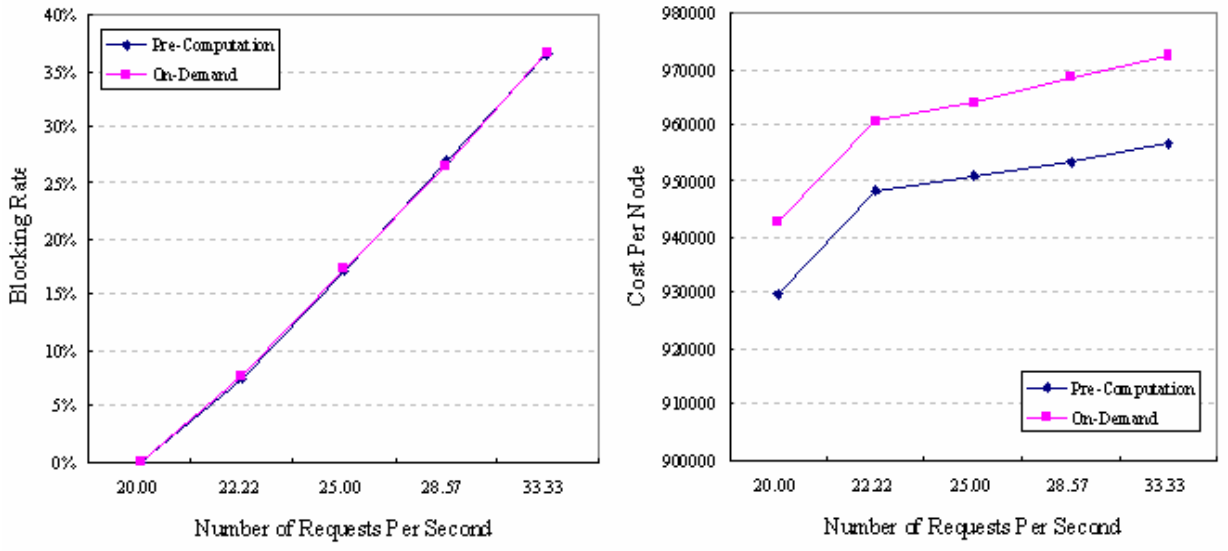

Figure 12 The Threshold-Based Link State Update Method, Threshold=10%

Figure 12 shows the blocking rates and costs of the two path computation algorithms with different traffic loads. The Threshold-Based link state update algorithm is used and the threshold is 10%. There is not much difference in the blocking rate, and the cost of the on-demand computation is higher than that of the pre-computation algorithm.

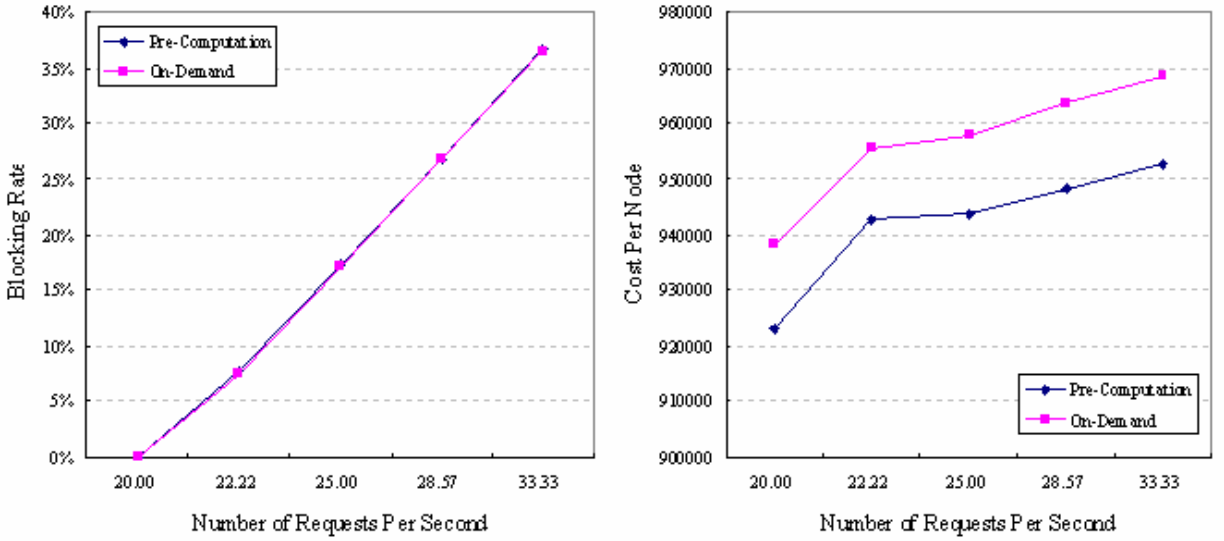

Figure 13 The Threshold-Based Link State Update Method, Threshold=40%

When threshold is 40%, the blocking rates of the two algorithms are still almost the same, and the cost of the on-demand computation is higher than that of the pre-computation algorithm.

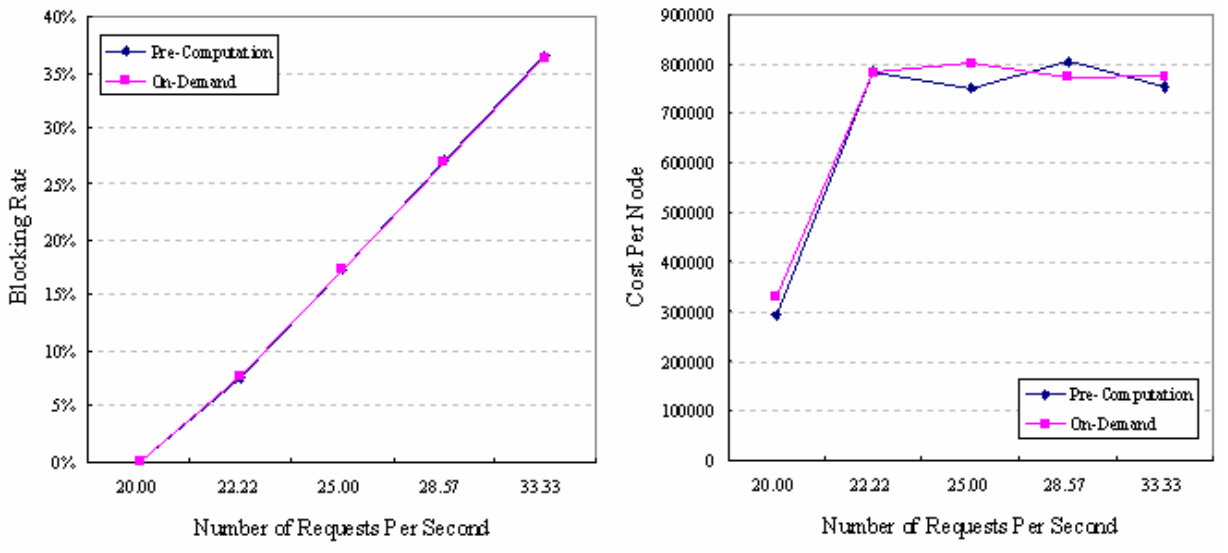

Figure 14 The Threshold-Based Link State Update Method, Threshold=60%

When the threshold is 60%, the blocking rates of the two algorithms are still almost identical, but unlike Figure 12 and 13, there is not much difference between the costs of the pre-computation and the on-demand computation.

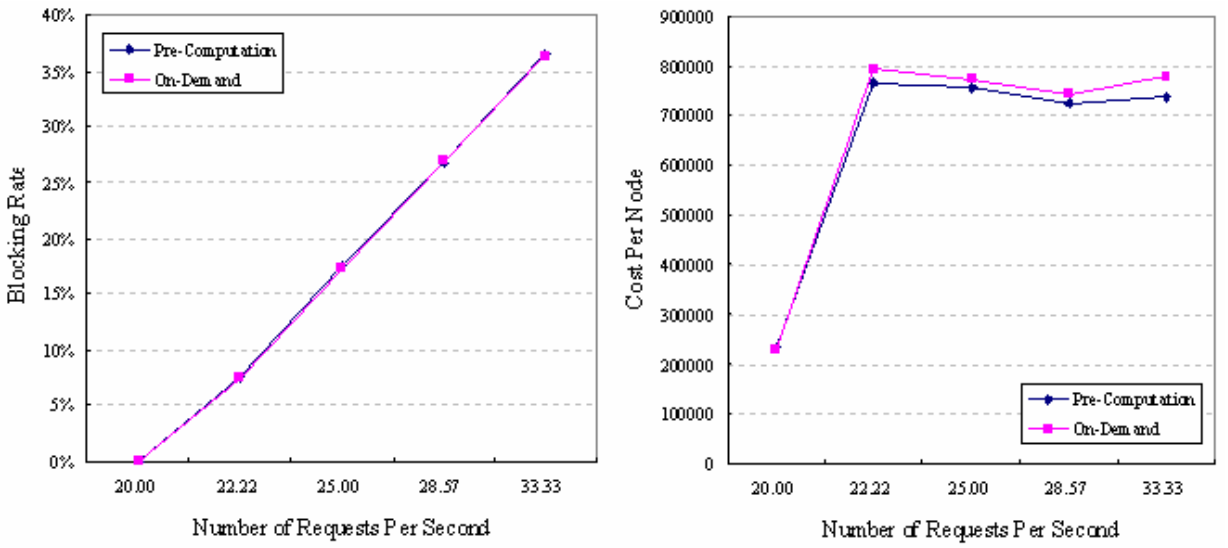

Figure 15 The Threshold-Based Link State Update Method, Threshold=80%

When threshold is 80%, both of the blocking rates and costs of the two algorithms are close.

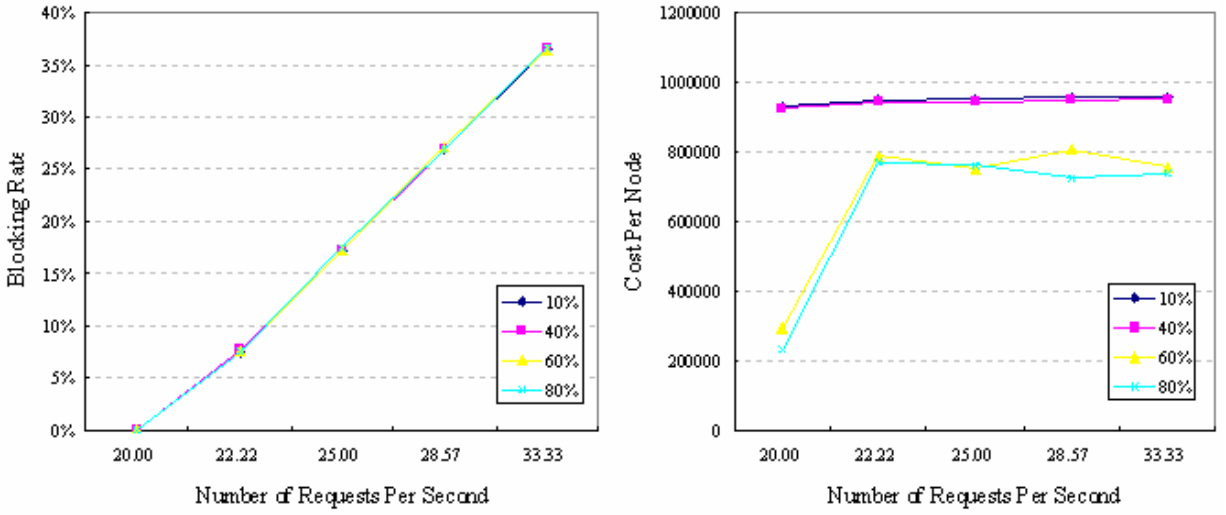

Figure 16 The Threshold-Based Link State Update Method with different thresholds (Pre-Computation)

When making comparison between different link state update thresholds, we notice that there is not much difference between the blocking rates of different thresholds. The costs of threshold=10% and threshold=40% are similar, and they are higher than the costs of threshold=60% and threshold=80%.

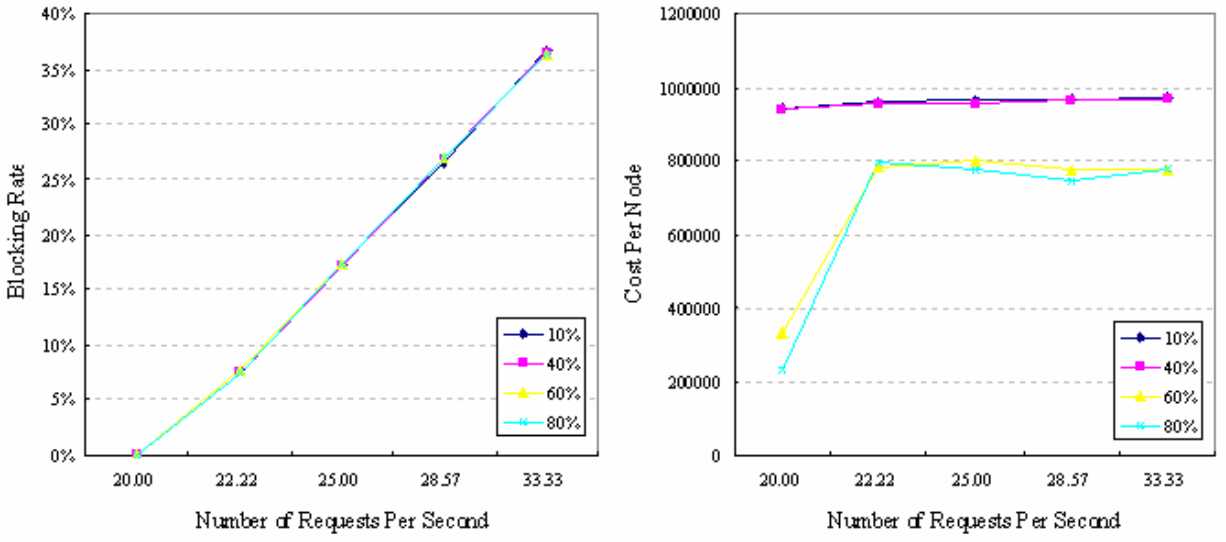

Figure 17 The Threshold-Based Link State Update Method with different thresholds (On-Demand)

As to the on-demand routing algorithm, there is also not much difference in blocking rates with different thresholds.

4.2.3 Conclusion

The matrix 2*2 topology is quite simple. It only has four nodes and four links. Fewer nodes and links tend to have fewer changes, and the changes of link state information can be spread over the whole network easily. Thus the link state information for periodically pre-computation of paths tends to be more accurate, and then the blocked requests caused by inaccurate path selection is minimized.

So in this simple topology, the on-demand computation does not obviously have better performance in blocking rate compared to that of the pre-computation algorithm. But due to more path computations, the cost of the on-demand computation is higher than that of the pre-computation algorithm, and the difference between them becomes larger with the increase of the request rate.

In [13], there is a comparison of costs and performances between the PB with period=100ms and the TB with threshold=20%, and the PB gets lower blocking rate with lower cost. From our simulation results presented above, we draw a similar conclusion that PB with period =100ms has nearly equal blocking rate with TB of threshold=10%, but PB has much lower cost.

4.3 The Matrix 3*3 Topology

4.3.1 Basic information of the topology

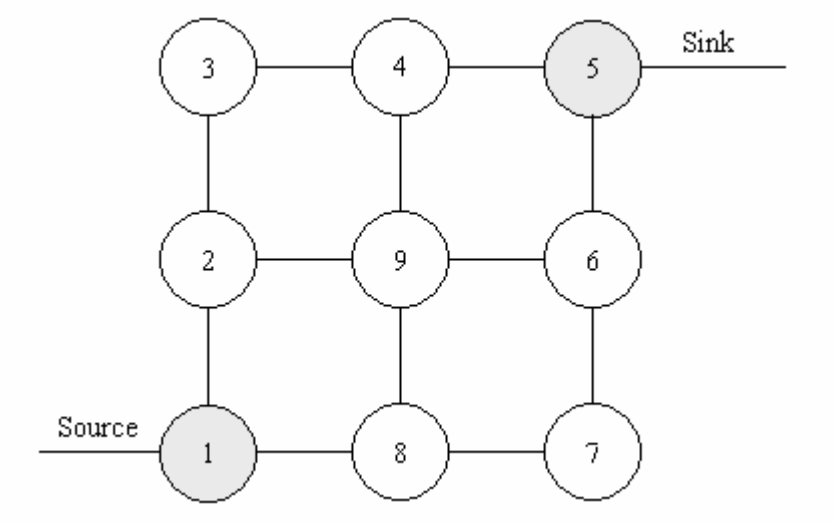

Figure 18 The Matrix 3*3 Topology

The Matrix 3*3 topology has nine nodes with twelve links connecting these nodes. The bandwidth of each link is set to 20Mb/s, the links will never fail, and the flow rate of source is 1 Mb/s. Traffic load changes are made by changing the average interval between requests.

In this topology, node 1 is the source and node 5 is the sink. The average interval between requests is set from 100ms to 30ms, so the corresponding numbers of requests per second vary from 10 to 33.33.

4.3.2 The Result and analysis

For the Period-Based Routing Algorithm:

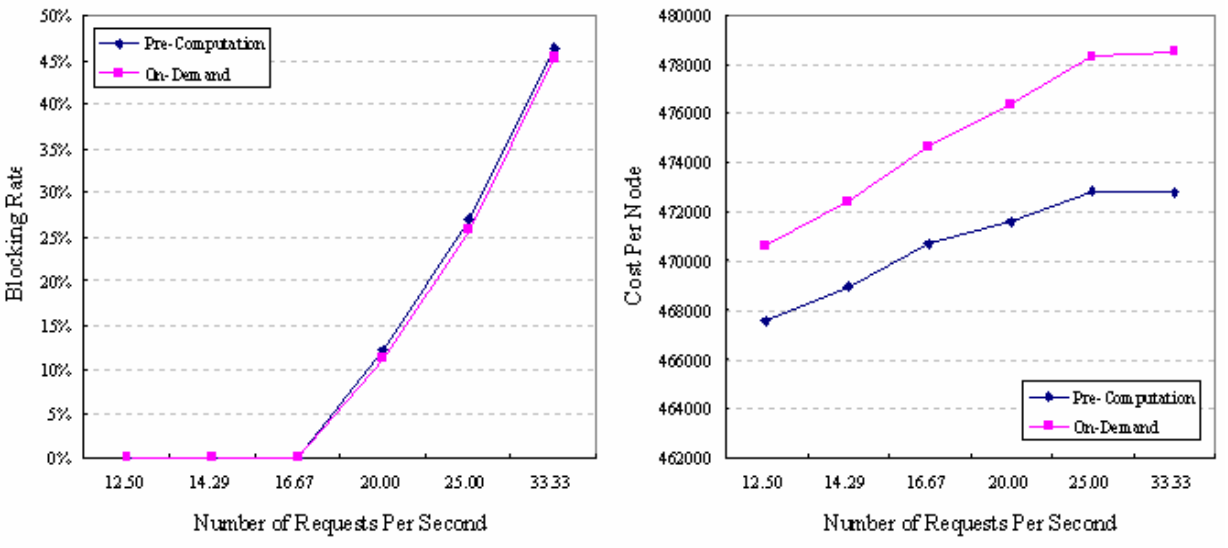

Figure 19 The Period-Based Link State Update Method, Period=100ms

From figure 19, we can see that the blocking rate of the on-demand computation is slightly lower than that of the pre-computation. The Pre-computation algorithm has lower cost compared to the on-demand algorithm, and the difference is getting larger with the increase of the request rate.

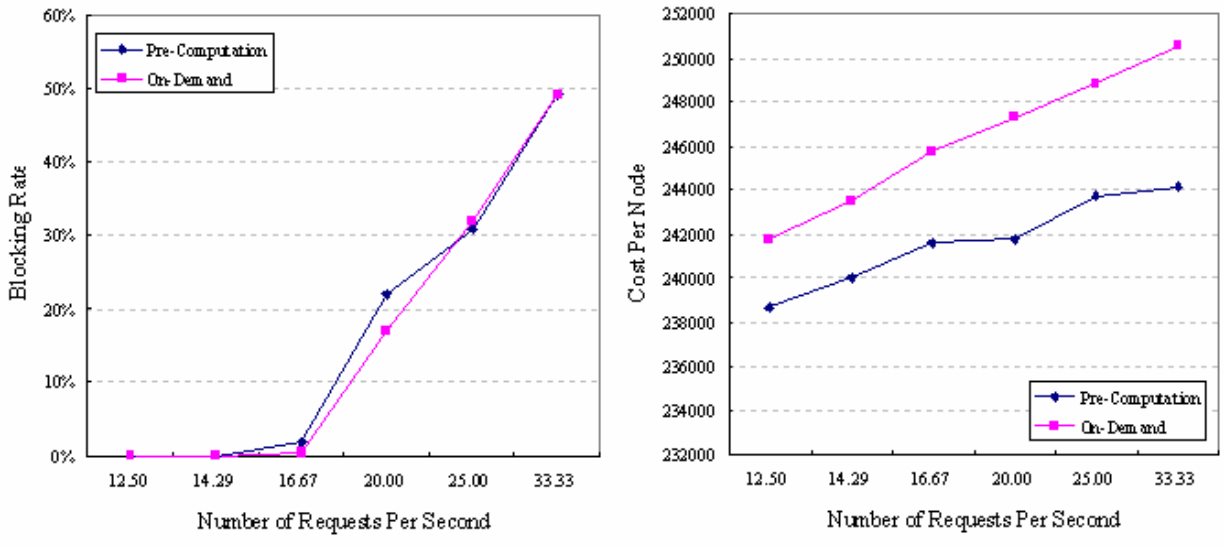

Figure 20 The Period-Based Link State Update Method, Period=200ms

When period is 200ms, the blocking rate of the on-demand computation is lower but the cost is higher compared to that of the pre-computation algorithm.

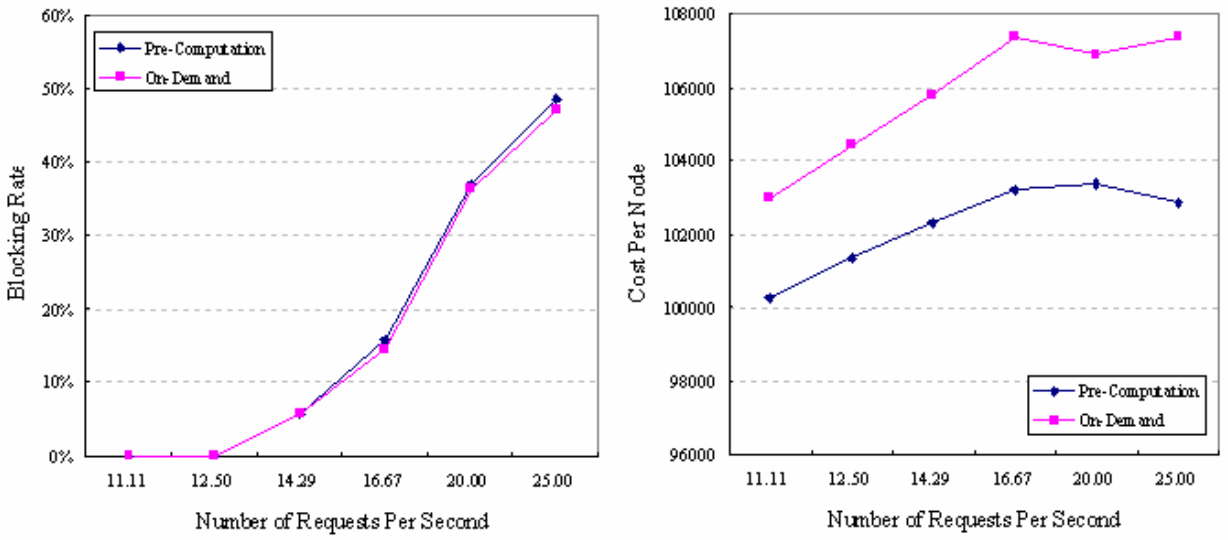

Figure 21 The Period-Based Link State Update Method, Period=500ms

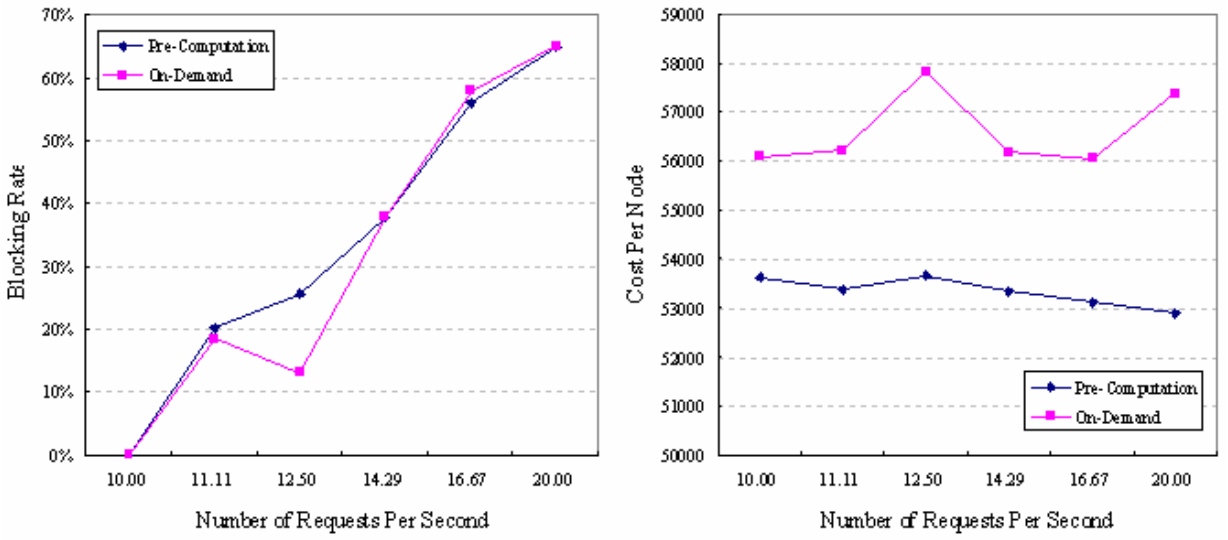

Figure 22 The Period-Based Link State Update Method, Period=1000ms

Similar to the one with period=200ms, when period=500ms and period=1000ms, the blocking rates of the on-demand computation are lower but the costs are higher compared to those of the pre-computation algorithm.

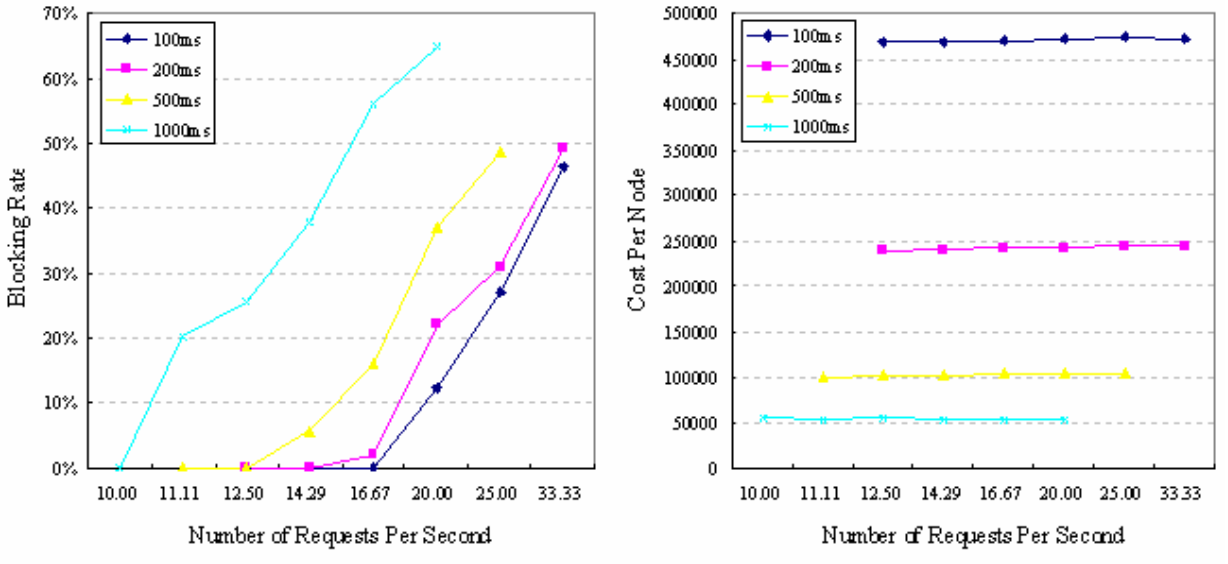

Figure 23 The Period-Based Link State Update Method with different periods (Pre-Computation)

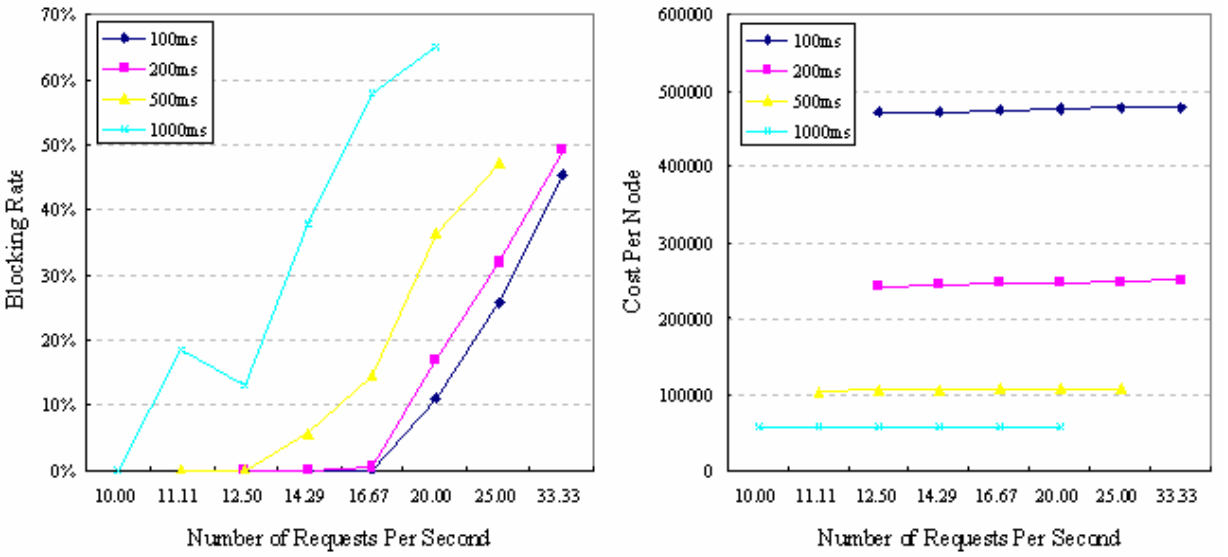

Figure 24 The Period-Based Link State Update Method with different periods (On-Demand)

From Figure 23 and 24, we find that the performance in case of period=100ms and period=200ms have obvious improvements compared to cases when period=500ms and period=1000ms. And the difference between period=100ms and period=200ms is minor, period=100ms has much higher cost.

When link state update periods are 1000ms and 500ms, the frequencies of link state updates are low. The changes to the link state information will take more time to be updated. As we have discussed in the first chapter, the path computation based on inaccurate link state information makes the path selection non-optimal or even incorrect. Non-optimal path wastes system resources and incorrect path leads to set-up failure. Thus the blocking rate increases. With the decrease of the length of the period, the link state information gets more accurate, and the blocking rate becomes lower. But this improvement is not gained free of charge, the cost also increases with the shortening of the link state update period.

For the threshold-Based Routing Algorithm:

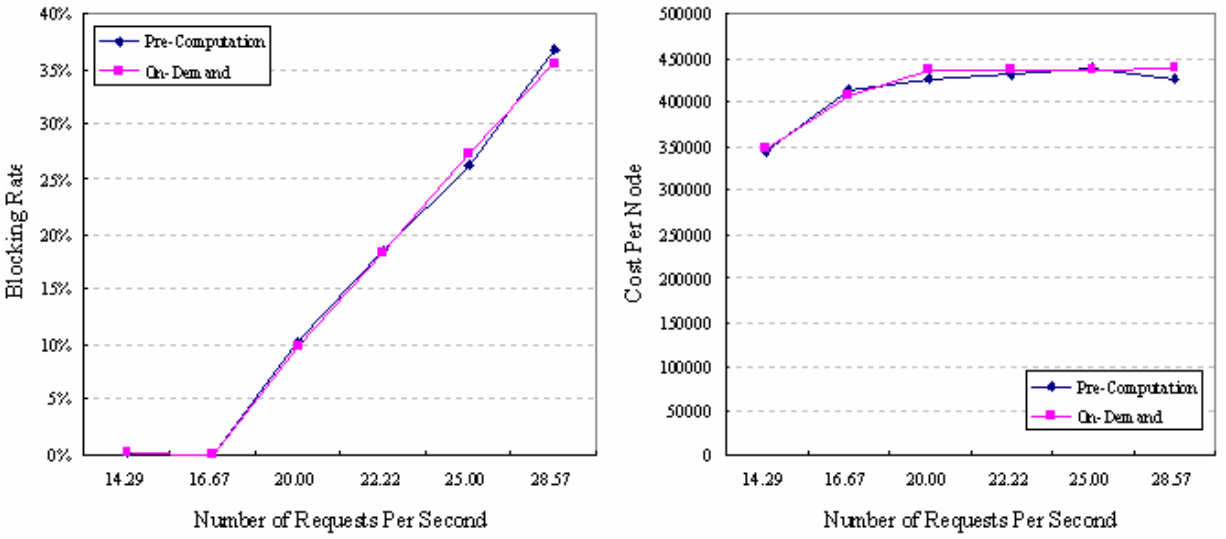

Figure 25 The Threshold-Based Link State Update Method, Threshold=10%

When threshold=10%, the blocking rates and costs of these two algorithms are almost the same.

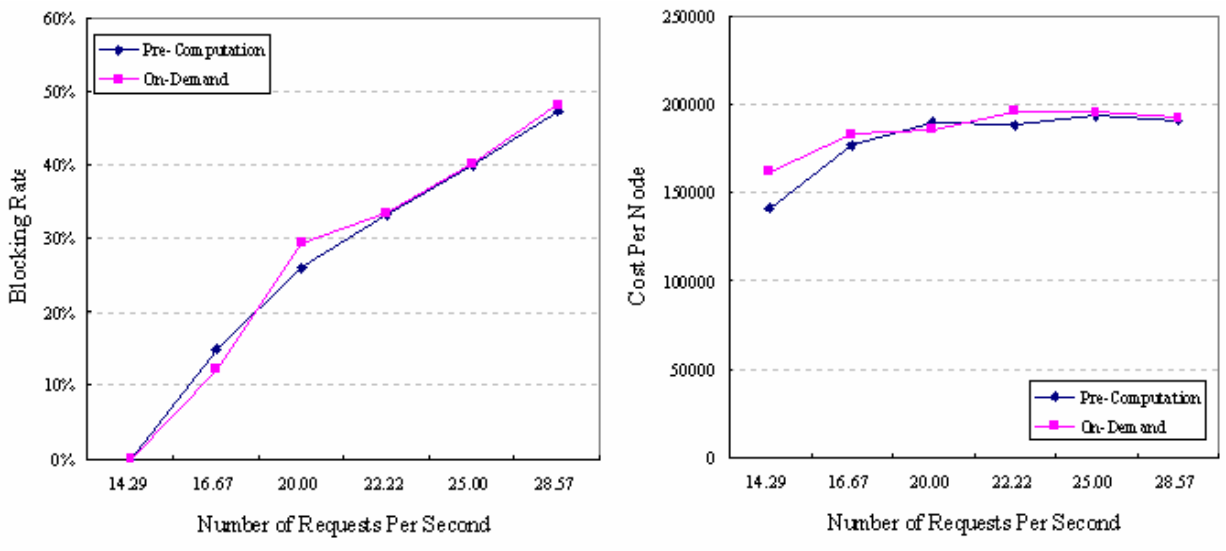

Figure 26 The Threshold-Based Link State Update Method, Threshold=40%

When threshold=40%, the blocking rates of these two algorithms are almost the same, but the on-demand computation algorithm has a little higher cost.

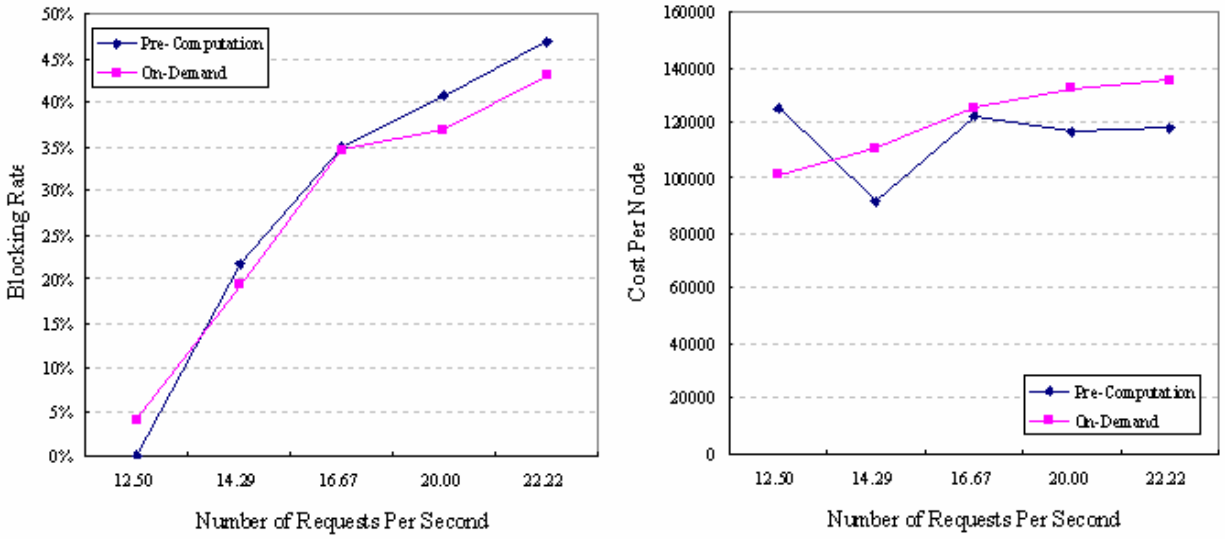

Figure 27 The Threshold-Based Link State Update Method, Threshold=60%

When threshold=60%, the on-demand computation algorithm has lower blocking rate, but its cost is higher than the pre-computation algorithm.

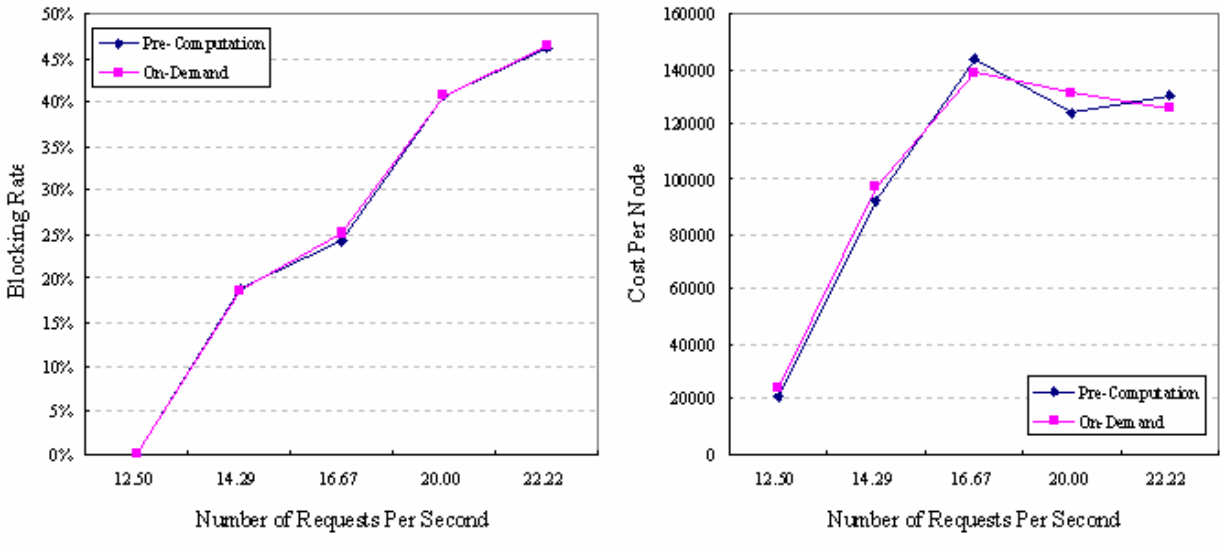

Figure 28 The Threshold-Based Link State Update Method, Threshold=80%

When threshold=80%, the pre-computation and the on-demand computation algorithm have similar performance and cost.

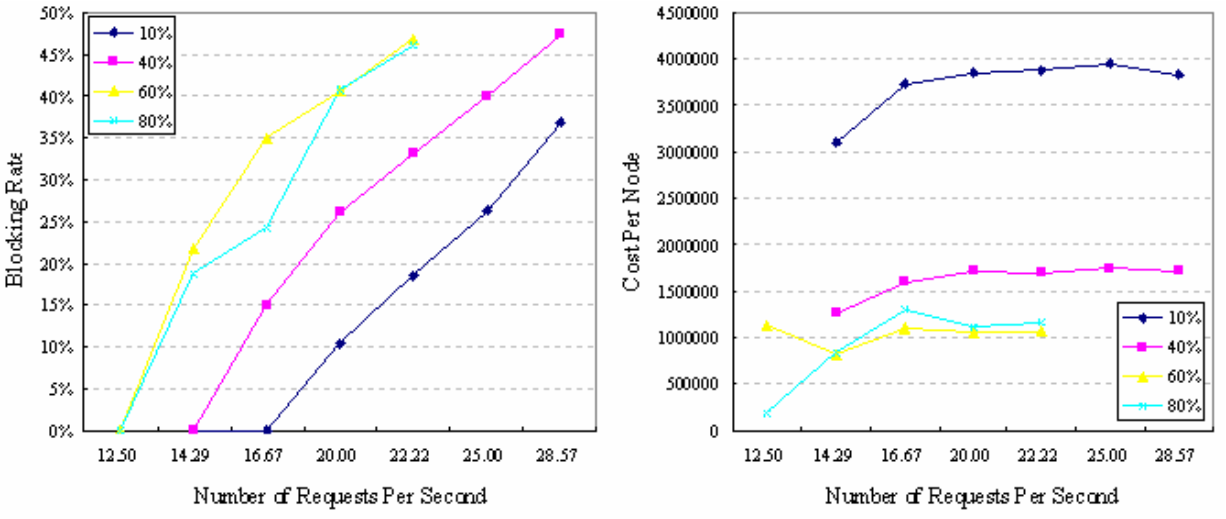

Figure 29 The Threshold-Based Link State Update Method with different thresholds (Pre-Computation)

In Figure 29, we find that with the pre-computation algorithm, when threshold=10%, the blocking rate is lower than with other algorithms, but it also has much higher cost. When threshold=60% and threshold=80%, they have similar performance in blocking rates, and the difference between their costs is also minor.

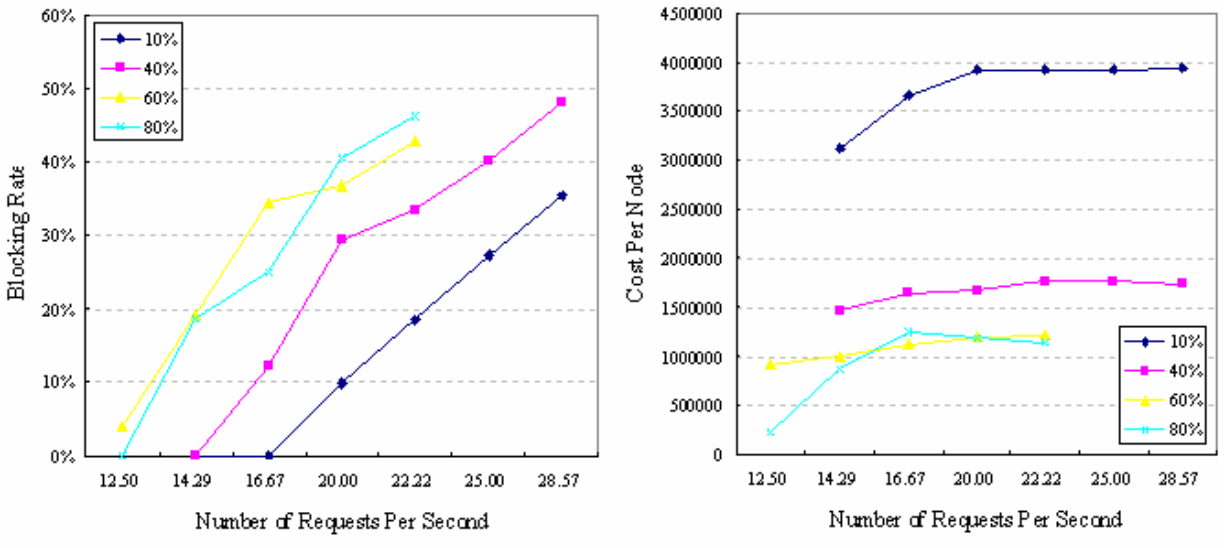

Figure 30 The Threshold-Based Link State Update Method with different thresholds (On-Demand)

Figure 30 is similar to Figure 29. When threshold=60% and threshold=80%, there is no much difference in the aspects of performance and cost.

4.3.3 Conclusion

The matrix 3*3 topology has nine nodes and twelve links. Compared to the matrix 2*2, it has more nodes and links, and more nodes and links tend to have more changes that need to be broadcasted. The broadcasting of more changes needs more processing capability and occupies more bandwidths. So the overhead is thus increased and the overall blocking rate also becomes higher. For example, in the PB algorithm, when period is 1000ms and the request rate is 20 per second, the blocking rate in the matrix 3*3 network is about twice as high as that in the matrix 2*2 network. And the cost of the matrix 3*3 is about twice as high as the cost in the matrix 2*2.

In bigger size networks, by using more accurate routing information for path computation, the on-demand computation shows small advantage in the aspect of blocking rate, though the difference in the performance between the two algorithms is still insignificant.

When the network size is getting bigger, more nodes can share the flow load between one source-destination pair, thus the average forwarding load for each node gets smaller. In our simulations, both the matrix3*3 topology and the matrix2*2 topology only have one source-destination pair, we observe that the cost per node in the matrix3*3 topology is lower than that in the matrix2*2 topology.

4.4 The Matrix 4*4 Topology

4.4.1 Basic information of the topology

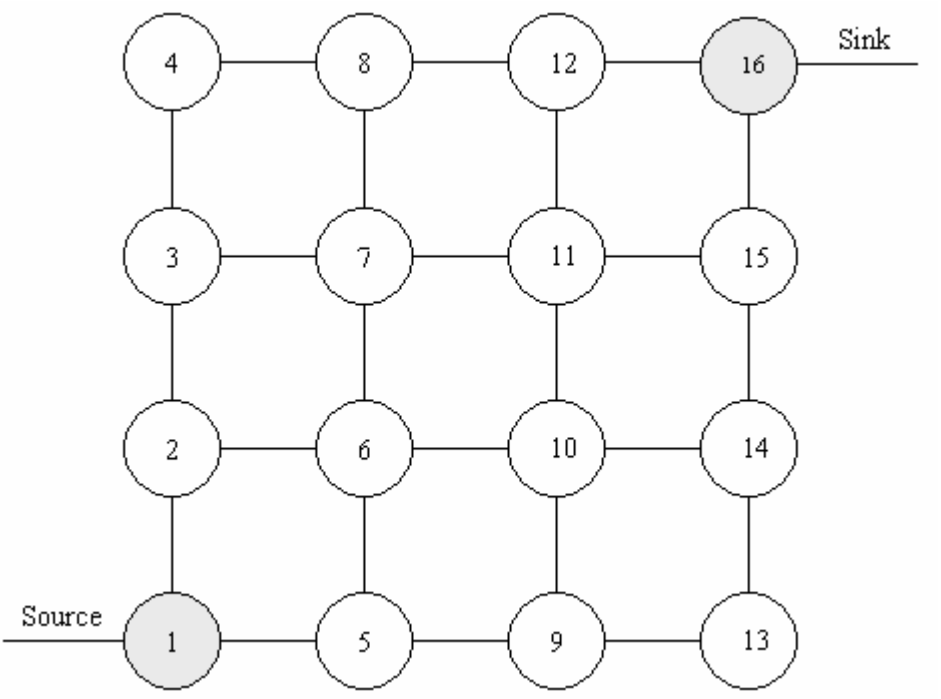

Figure 31 The Matrix 4*4 Topology

The Matrix 4*4 topology has sixteen nodes with twenty-four links connecting these nodes. The bandwidth of each link is set to 20Mb/s, the links will never fail, and the flow rate of source is 1 Mb/s. Traffic load changes are made by changing the average interval between requests.

In this topology, node 1 is the source and node 16 is the sink. The average interval between requests is set from 100ms to 40ms, so the corresponding numbers of requests per second vary from 10 to 25.

4.4.2 The Result and analysis

For the Period-Based Routing Algorithm:

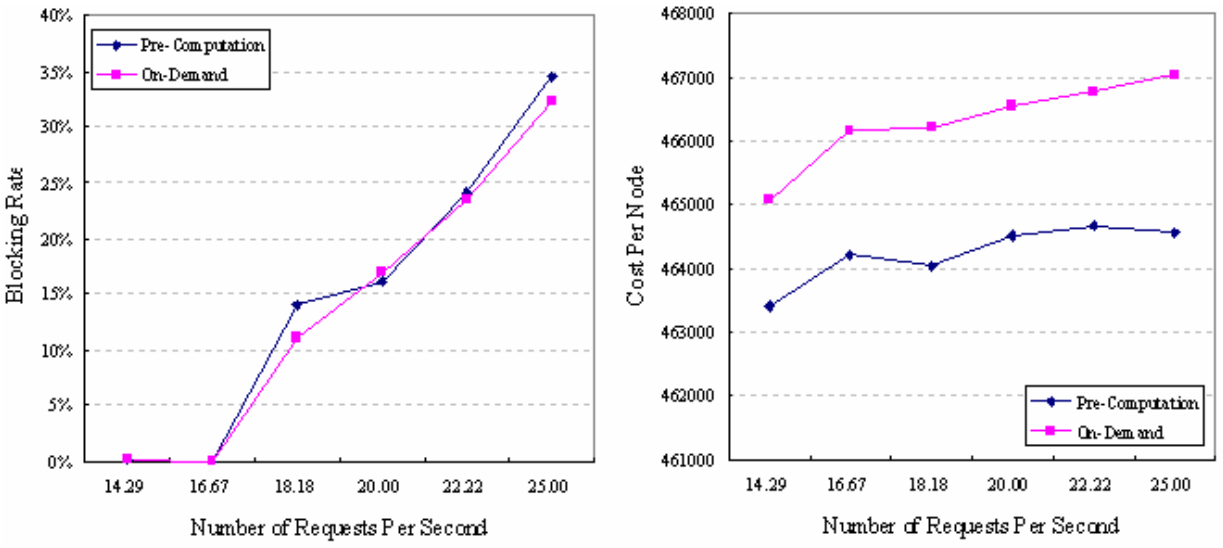

Figure 32 The Period-Based Link State Update Method, Period=100ms

From Figure 32, we can see that the on-demand computation performs better with lower blocking rate but the cost is higher compared to that of the pre-computation algorithm.

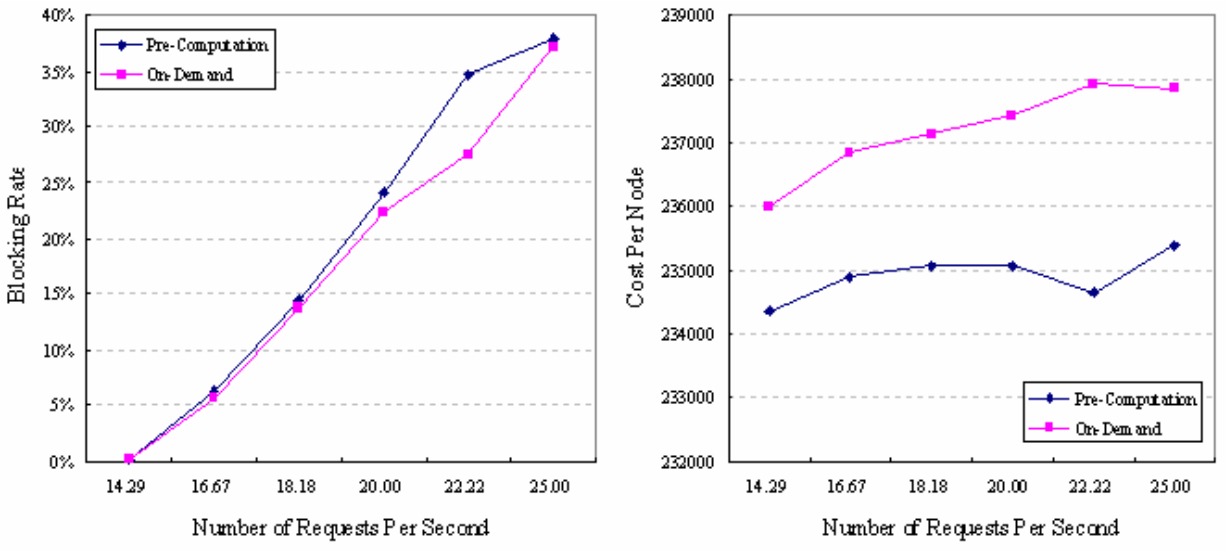

Figure 33 The Period-Based Link State Update Method, Period=200ms

Like the result in Figure 32, in Figure 33, the on-demand algorithm gets lower blocking rate while it has higher cost. Compared to the simulations with period=100ms, the increase of the length of the period significantly reduces the cost.

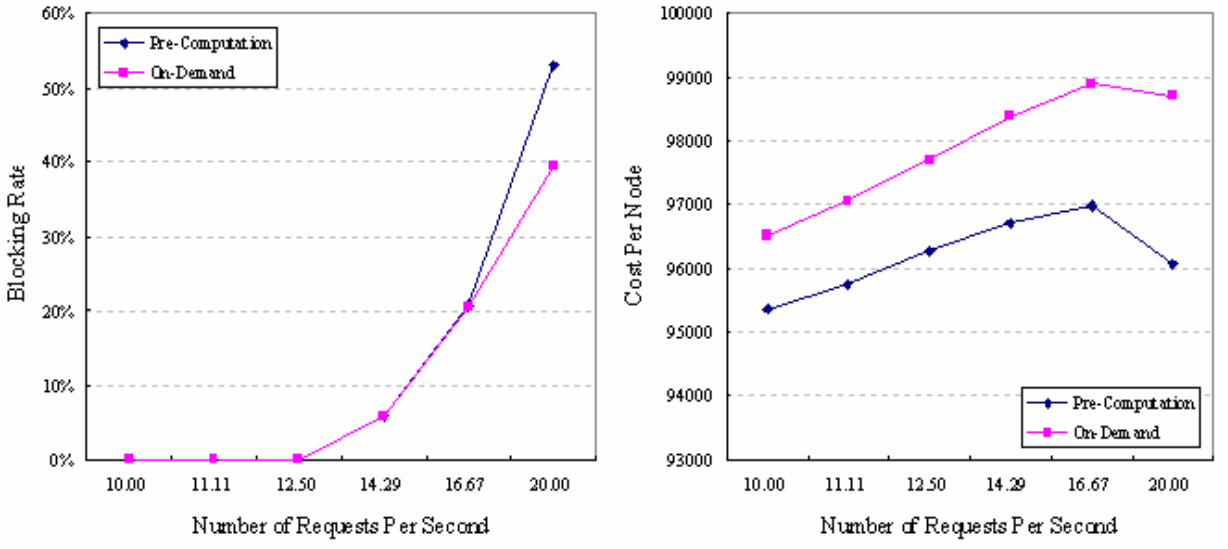

Figure 34 The Period-Based Link State Update Method, Period=500ms

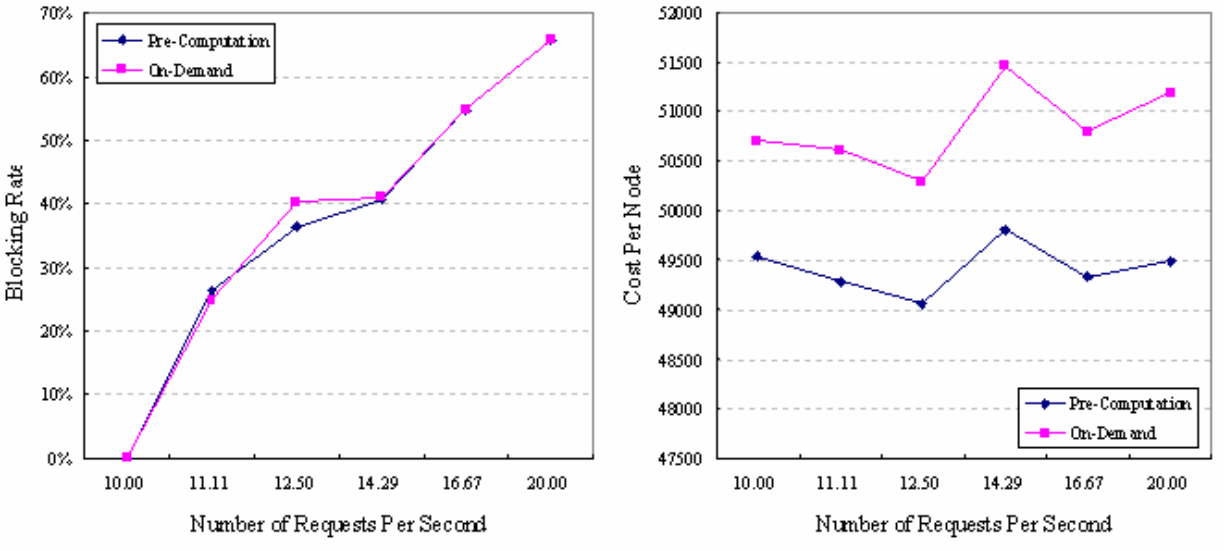

Figure 35 The Period-Based Link State Update Method, Period=1000ms

When period=500ms and period=1000ms, the results are similar, the on-demand algorithm gets lower blocking rate but higher cost.

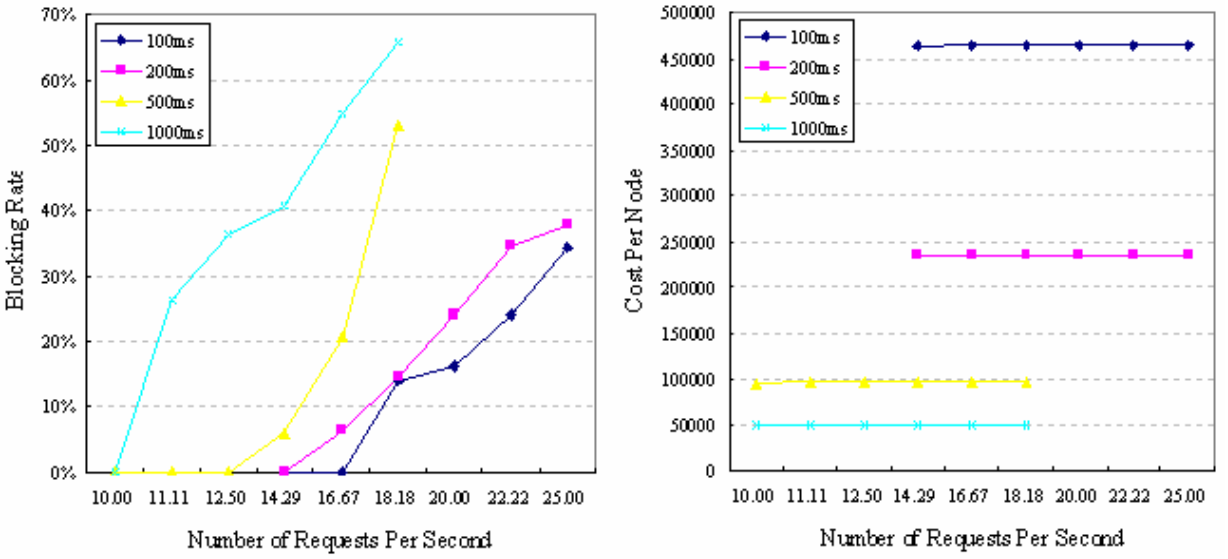

Figure 36 The Period-Based Link State Update Method with different periods (Pre-Computation)

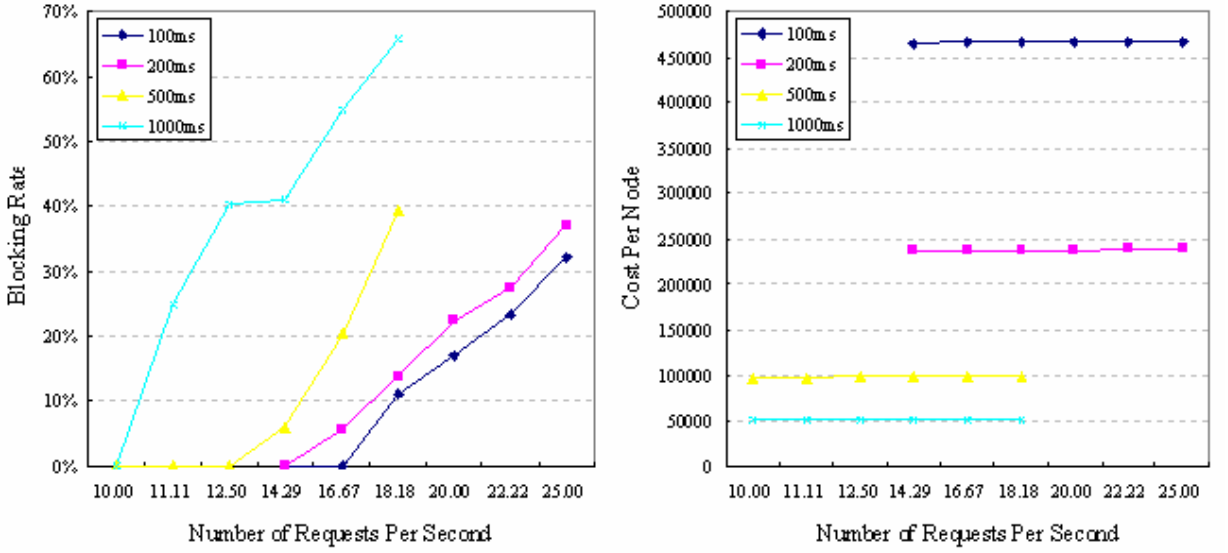

Figure 37 The Period-Based Link State Update Method with different periods (On-Demand)

From Figure 36 and 37, we can find that when period=1000ms, the performance is quite poor compared to the performance of other periods, though the cost is a little lower than when period=500ms. The difference in blocking rate between period=100ms and period=200ms is small, but when period=100ms, the cost is about twice as high as the cost when period=200ms, i.e. the small improvement in performance is paid by much higher cost.

For the Threshold-Based Routing Algorithm:

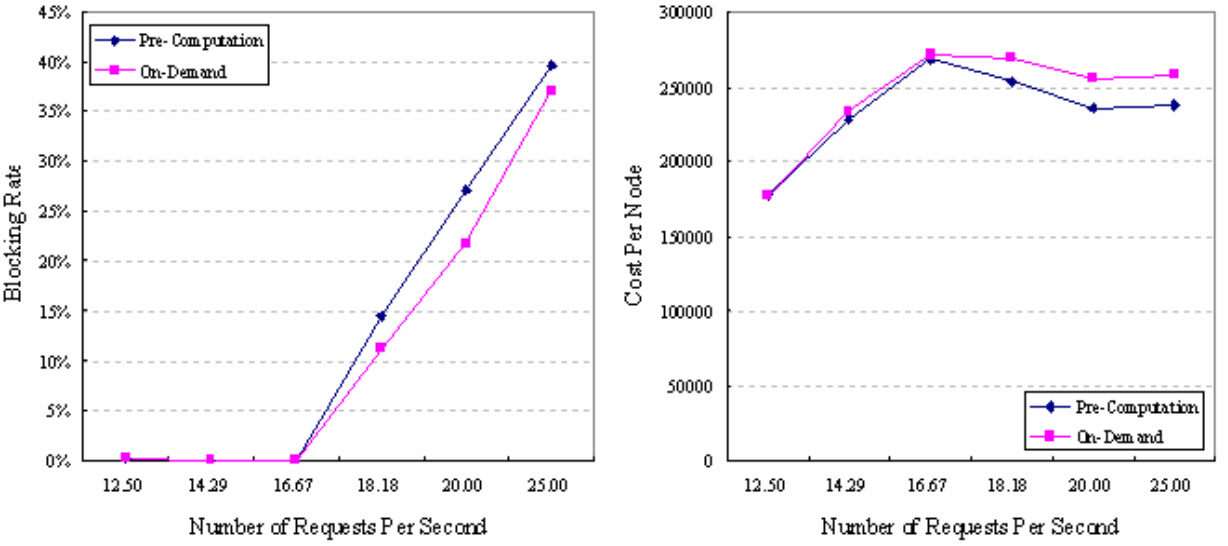

Figure 38 The Threshold-Based Link State Update Method, Threshold=10%

From Figure 38, we can find that the blocking rate of the on-demand computation is lower, and the difference between the costs of these two algorithms is minor. By using the on-demand computation, the improvement in performance is gained with a little increase in its cost.

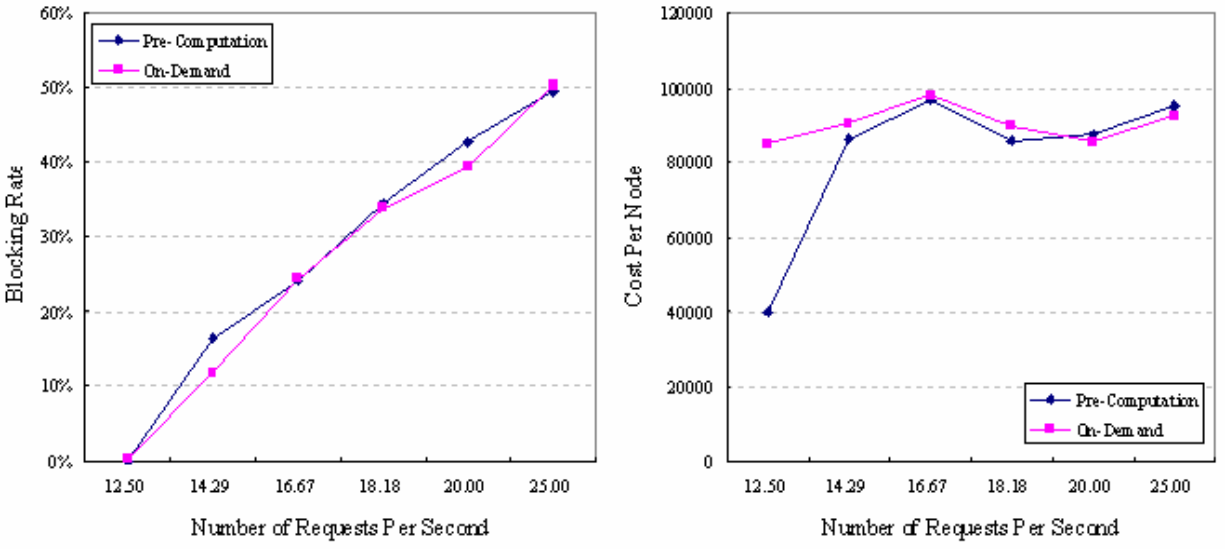

Figure 39 The Threshold-Based Link State Update Method, Threshold=40%

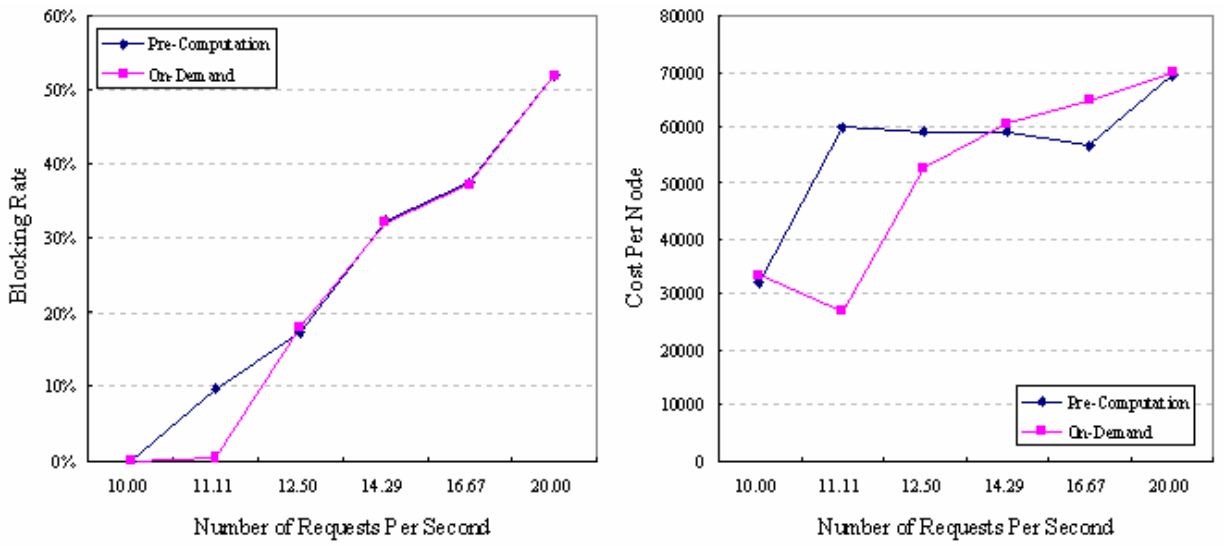

Figure 40 The Threshold-Based Link State Update Method, Threshold=60%

The above two figures show similarity with the one when threshold=10%. The on-demand computation has a minor improvement in blocking rate, and the difference between the costs is minor.

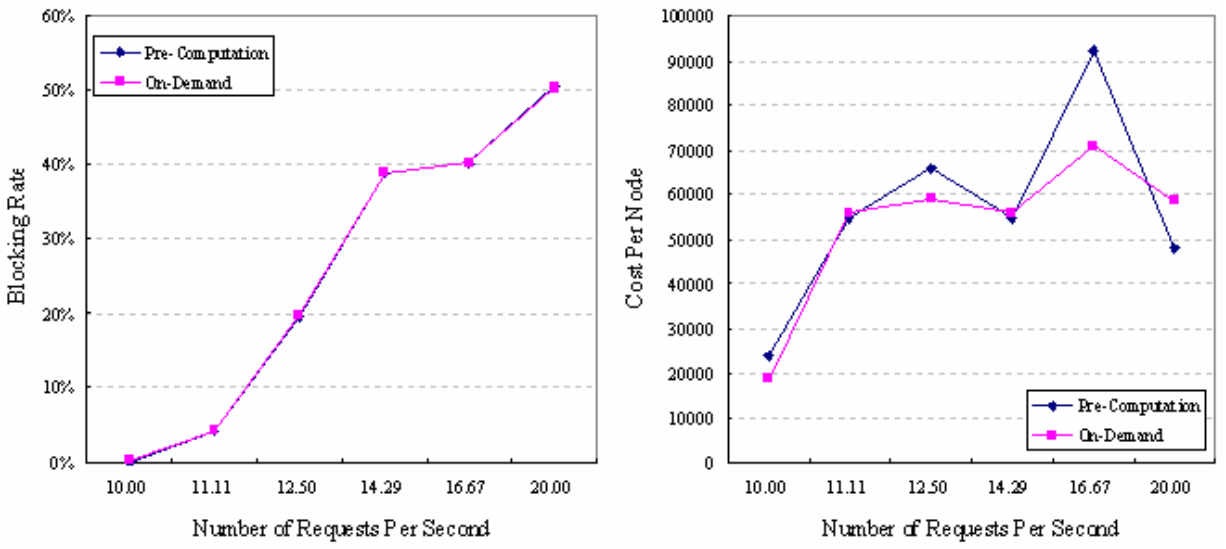

Figure 41 The Threshold-Based Link State Update Method, Threshold=80%

When threshold=80%, both of the blocking rates and the costs of these two algorithms are about the same.

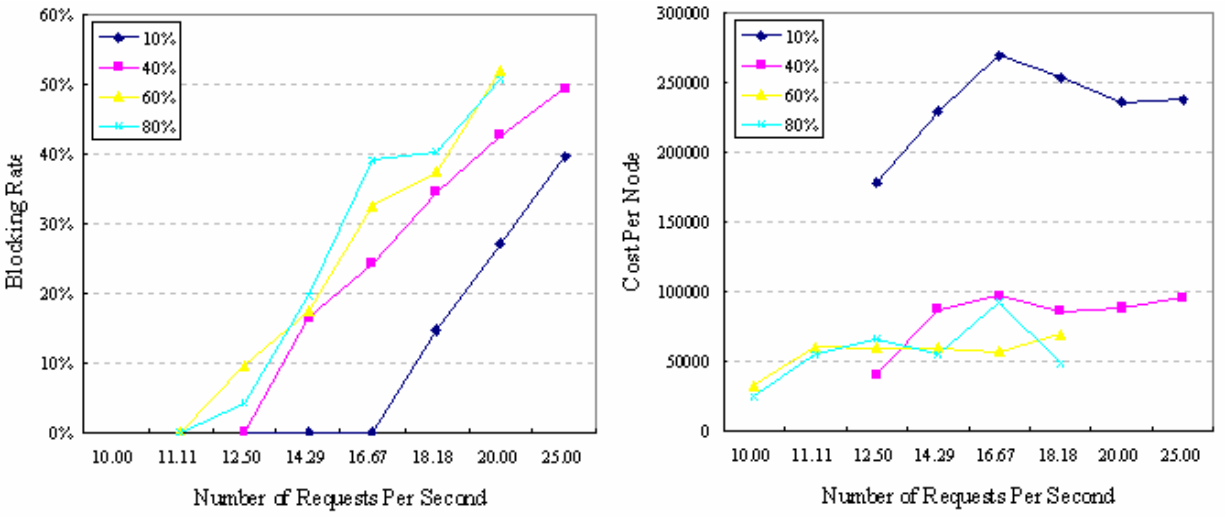

Figure 42 The Threshold-Based Link State Update Method with different thresholds (Pre-Computation)

From Figure 42, we find that when the threshold varies from 40% to 80%, the blocking rates do not change much, but when threshold=10%, there appears a high decrease in blocking rate accompanied by a significant increase of cost.

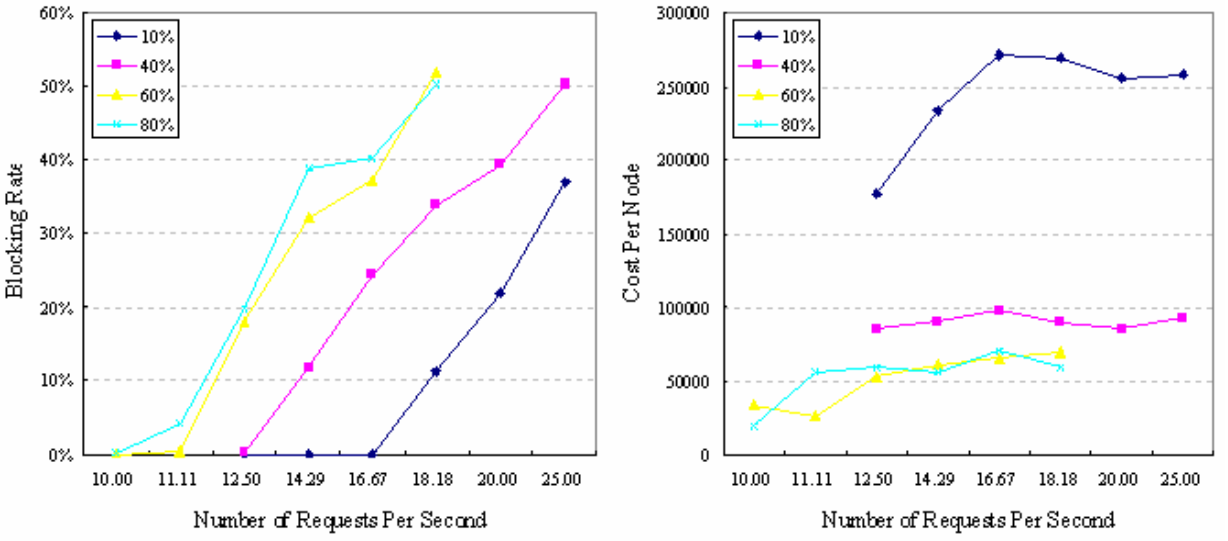

Figure 43 The Threshold-Based Link State Update Method with different thresholds (On-Demand)

From Figure 43, we can find that the performance and the cost differences between thresholds 60% and 80% are minor. When lowering the threshold to 40% and 10% respectively, the blocking rates are getting better with a corresponding increase of costs.

4.4.3 Conclusion

From the simulation results of this topology, we can see an obvious improvement in the blocking rate for the on-demand computation algorithm. The costs are also increased because of more path computations. The network is apt to have blocks compared to the previous two topologies. By using the same algorithm, the same link state update algorithm and the same frequency of link state updates, the matrix 4*4 has higher blocking rate and higher cost.

.

Compared with the matrix 2*2 and the matrix 3*3, the matrix 4*4 topology has more nodes and links. More path computations are needed and more bandwidth are used for link state updates. So the total costs are higher while the network has lower capacity for traffic. In bigger networks, the inaccuracy of routing information tends to be more serious and becomes an important factor in affecting the blocking rate.

When comparing the cost per node among these three topologies, we find that the cost per node is getting lower with the increasing of the network size. That is because we have only one source-sink pair for each topology and more nodes can share the flow load in bigger networks. For example, for the on-demand routing algorithm, when the link state update period is 100ms and the number of requests per second is 20, the cost per node is 505,285 in the matrix 2*2 topology, it is 471,612 in the 3*3 topology and it is 464,506 in the 4*4 topology.

5 Conclusions and Future Work

5.1 Conclusions

After a brief introduction to QoS routing, we discussed the sources of the inaccurate routing information and their impact upon network performance. Then we presented the pre-computation routing algorithm and the on-demand computation routing algorithm, followed by the comparison between these two algorithms.

In order to run the simulations for this thesis, we made extensions to the simulator by adding new components and a Traffic Generator. We run the simulations for these two algorithms with different link state update algorithms. Four different periods in the PB and four thresholds in the TB have been chosen for the simulations. From the simulation results, we discover that, in smaller size networks, the on-demand computation has no advantage in performance but has higher cost. The frequency of the link state update can affect the network performance and the costs significantly. Lowering the link state update period or threshold can improve the network performance by reducing the inaccuracy of routing information, but the cost will also increase with more frequent link state updates. The update triggering policy should be chosen carefully.

The size of networks is another important factor for the network performance and the costs. With the same link bandwidth and same amount of requests, the blocking rate and the cost will increase with the expansion of the size of networks.

5.2 Future Work

For networks operators, before the implementation of any QoS routing algorithm in live networks, the performance and costs of different QoS routing algorithms with different link state update triggering policies have to be analysed carefully, and then a proper link state update algorithm has to be chosen.

From our simulations, we can see that though the difference in performances is minor, the costs vary significantly for different algorithms. We have discovered that the difference between different routing algorithms and update triggering policies becomes larger with the increase of network size. We can make more simulation work with larger and more complicated networks that model the practical network better. Furthermore, the QRS simulator can be extended with more traffic models and routing algorithms.

Besides, in our future work, we also want to study other factors that might affect the usage of the routing algorithms. For example, we need to consider DiffServ networks where there might not be any requests, or MPLS networks where requests may arise frequently or infrequently.

References

[1] Cisco, “Quality of Service: QoS”,
URL: http://www.cisco.com/warp/public/784/packet/oct02/pdfs/qos.pdf

[2] Allison Taylor, “AT&T to invest $3 billion in 2003 for global network”
URL: http://www.infoworld.com/article/03/09/11/HNattnetwork_1.html

[3] G. Apostolopoulos et al., ”QoS Routing Mechanisms and OSPF Extensions”, RFC 2676, August 1999

[4] G. Apostolopoulos et al., “On reducing the processing cost of on-demand QoS path computation”, Journal of High Speed Networks, 7:77-98, 1998

[5] G. Apostolopoulos et al., ”Improving QoS Routing Performance Under Inaccurate Link State Information”, Proceedings of the 16th International Teletraffic Congress (ITC’16), Edinburgh, United Kingdom, June7-11, 1999.

[6] R. Guerin and A. Orda, “QoS-based Routing in Networks with Inaccurate Information: Theory and Algorithms”, IEEE/ACM Transaction on Networking, 7(3):350-364, June 1999

[7] D.H. Lorenz and A. Orda, “QoS Routing in Networks with Uncertain Parameters”, IEEE/ACM Transactions on Networking, vol. 6, no. 6, December 1998.

[8] P. Zhang and R. Kantola, “QoS Routing Simulator: User’s Manual (Version 2.1)”, December 2002.

[9] A. Shaikh et al., "Efficient precomputation of quality-of-service routes," Proc. Workshop on Network and Operating Systems Support for Digital Audio and Video, July 1998, pp. 15-27

[10] A. Orda and A. Sprintson, "Precomputation Schemes for QoS Routing" IEEE/ACM Transactions on Networking, Volume 11 , Issue 4, August 2003.

[11] Y. Lin et al., "QoS Routing Granularity in MPLS Networks", IEEE Communications Magazine, Vol. 40, No. 6, June 2002.

[12] G. Apostolopoulos and S. K. Tripathi, "On the effectiveness of path pre-computation in reducing the processing cost of on-demand qos path computation," in Proceedings of IEEE Symposium on Computers and Communication, June 1998.

[13] Z. Ma, P. Zhang, R. Kantola, " Influence of Link State Updating on the Performance and Cost of QoS Routing in an Intranet", 2001 IEEE Workshop on High Performance Switching and Routing (HPSR 2001), Dallas, Texas USA, May 29-31, 2001

[14] P. Zhang and R. Kantola "QoS Routing in DiffServ Networks: Issues and Solutions", Technical report, May 2001.

[15] G. Apostolopoulos et al., "Improving QoS Routing Performance Under Inaccurate Link State Information." Proceedings of the 16th International Teletrac Congress (ITC'16), Edinburgh, United Kingdom, June 7-11, 1999.

[16] P. Zhang and R. Kantola " Designing a New Routing Simulator for DiffServ MPLS Networks", 2001 SCS Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'2001). July 2001.

[17] G. Apostolopoulos et al., "Implementation and Performance Measurements of QoS Routing Extensions to OSPF", to appear in Proceedings of Infocomm '99, New York

[18] P. Zhang, R. Kantola, Z. Ma. "Design and Implementation of A New Routing Simulator". 2000 SCS Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'2K). July 16-20, 2000, Vancouver, Canada

[19] C. Alaettinoglu et al., "Design and implementation of MaRS: A routing testbed", Journal of Internetworking: Research and Experience, 5(1):17-41, 1994.

[20] S. Nelakuditi et al., "Quality-of-Service Routing without Global Information Exchange", IWQOS 1999.

[21] E. Crawley et al., "A Framework for QoS-based Routing in the Internet", RFC 2386, August 1998

[22] Dr. Sebestyen and Dr. Hundt, "Media-Video Coding: Standards", 2001
URL: http://www-itec.uni-klu.ac.at/~harald/multimedia/5-Standards-JPEG2000.pdf

[23] Zhangsong Ma, "Performance and Cost Analysis of QoS Routing in an Intranet", Master thesis at Helsinki University of Technology, October 2000