Impact of Resource Reservation on the Distributed Multi–path Quality of Service Routing Scheme

Yuan Zhong Xin Yuan
Department of Computer Science
Florida State University
Tallahassee, FL 32306
{zhong,xyuan}@cs.fsu.edu

abstract

This paper studies the impact of resource reservation on the multi–path Quality–of–Service (QoS) routing schemes that use global network state to make routing decisions. Incorporating resource reservation into multi–path QoS routing algorithms greatly changes the communication characteristics in a network system and affects the performance of the routing algorithms. In this paper, we develop a QoS routing protocol that combines resource reservation with the ticket–based distributed multi–path QoS routing scheme, evaluate the new routing protocol through extensive simulation, and study the impact of other network components, such as the network size and the link state update mechanisms, on the performance of multi-path QoS routing schemes with resource reservation.

1 Introduction

The migration to integrated networks for voice, data and multimedia applications introduces new challenges in supporting predictable communication performance. Multimedia applications require the communication to meet stringent requirement on delay, delay–jitter, cost and/or other quality of service (QoS) metrics. To support such applications, the network must be able to provide communication channels with QoS guarantees. Two key issues, *QoS routing*, which identifies paths that meet the QoS requirement and selects the one that leads to high overall resource efficiency, and *resource reservation*, which reserves the resources along the path, must be addressed to support communication with QoS guarantees.

Although QoS routing and resource reservation [15] are two closely related network components, traditionally, these two tasks are separated into two steps. First, a route is selected, then the route is set–up and the resources are reserved along the route. Separating routing and resource reservation simplifies the protocol design. However, in B–ISDN, resource availability may change rapidly and the route information may be outdated. In such environment, a route that was computed in the first step may lack the resources in the second step. Combining the two steps was suggested to overcome this problem [5, 8]. Combining resource reservation and QoS routing can greatly change the communication characteristics in the network system and affect the performance of the routing algorithms.

Many QoS routing algorithms have been designed to determine the route for a connection request [3, 5, 6, 7, 9, 10, 12, 13, 14]. Among these algorithms, multi–path QoS routing schemes [3] that use global network information to make routing decisions are promising for future networks for the following reasons. First, by using the global network information, these protocols incur less messaging overhead compared to the flooding based protocols [5, 9, 12] that do not use the global network information to make routing decisions. Second, by exploring multiple paths simultaneously in search of the path that satisfies the QoS requirement of a connection, multi–path QoS routing algorithms are more effective, in terms of both the blocking probability and the path establishment time, than single–path QoS routing schemes [6, 7, 14] that probe one path at a time.

When combining resource reservation with QoS routing, resource reservation affects the performance of the multi–path QoS routing algorithms that use global network state information more than it affects the performance of the single–path QoS routing algorithms or that of the flooding based algorithms. In multi–path routing, multiple paths are probed simultaneously, which requires reserving resources on multiple paths for each connection request. This problem is called the *over reservation problem*. Furthermore, reserving resources on multiple paths can greatly change the resource availability characteristics in the network system and decrease the precision of the global network state information. Since the routing algorithms rely on the global

network state to make effective routing decisions, resource reservation may greatly decrease the routing performance by degrading the precision of the global network state information. Thus, incorporating resource reservation into multi–path QoS routing algorithms not only requires the design of new efficient protocols that combine resource reservation and QoS routing, but also requires the re–study of the performance issues for the routing algorithms.

In this paper, we develop a new QoS routing protocol that combines resource reservation and a variation of the ticket–based QoS routing scheme [3], which is a distributed multi–path QoS routing scheme designed to deal with the imprecise global network state information. We evaluate the performance of the new protocol through extensive simulation. One unique feature of the ticket–based QoS routing scheme is that the number of paths to be probed in parallel is controlled by the number of tickets generated for each connection request. By manipulating the number of tickets for each connection request, the ticket–based routing scheme can emulate a wide range of QoS routing schemes including flooding based routing schemes and single–path routing schemes. Thus, introducing resource reservation into the ticket–based QoS routing enables us to study the impact of resource reservation on a wide range of QoS routing algorithms.

Our results show that multi–path QoS routing schemes with resource reservation are more effective in finding paths that satisfy the QoS requirement of a connection than single–path QoS routing schemes when the network is under light load. When the network is under heavy traffic, multi–path routing is better than the single–path routing when the resource requirement of each connection is low and the network is not saturated. Single–path QoS routing schemes are more efficient when the network is close to saturation. We conclude that it is possible to explore multiple paths in search of the paths that satisfy the QoS requirement without introducing excessive protocol overhead and that the ticket–based QoS routing algorithm with resource reservation is an effective QoS routing protocol.

The rest of the paper is organized as follows. Section 2 summarizes the related work. Section 3 presents the ticket based QoS routing with resource reservation. Section 4 reports the performance evaluation. Section 5 concludes the paper.

2 Related work

QoS routing has attracted much attention recently. An extensive survey can be found in [4]. Most existing QoS routing schemes decouple the routing issue from the resource reservation issue [2, 3, 6, 7, 10, 13, 14]. Few schemes [5, 8, 9, 12] combine routing with resource reservation. These routing schemes are either flooding based [5, 9, 12], where the precision of the global network information does not affect the routing performance, or single–path routing schemes [8], where the over reservation problem is not severe. This paper studies the impact of resource reservation on the multi–path routing schemes that use global network state information to make routing decisions. Without resource reservation, multi–path QoS routing appears to have some advantages over both the flooding based schemes and the single–path routing schemes. However, the impact of resource reservation on multi–path routing is unclear and has not been studied. This paper aims at filling in this gap. Studies on other network components, such as the link state update mechanisms and network topology, are reported in [1, 11]. Resource reservation adds another dimension and requires most of the performance issues to be revisited. In [3], the authors proposed a ticket–based routing algorithm to solve the delay–constrained least–cost routing without considering resource reservation. We incorporate resource reservation into a variant of this algorithm and study the impact of resource reservation.

3 Ticket–based distributed QoS routing with resource reservation

This section presents the QoS routing protocol that combines resource reservation with the ticket–based QoS routing scheme [3]. The protocol assumes that the global network state information is maintained by a link state algorithm. We will first briefly introduce the ticket–based QoS routing algorithm without resource reservation, then describe the protocol that combines resource reservation with the algorithm. A detailed description of the ticket–based routing algorithm (without resource reservation) can be found in [3]. We will use the bandwidth constraint as the QoS metric to illustrate the protocol. The protocol can be extended to deal with other QoS metrics, such as delay, delay jitter and cost.

3.1 Ticket–based distributed QoS routing algorithm

When a connection request arrives at the source node, a certain number (t) of tickets are generated and a reservation packet with the t tickets is sent to the destination in search of paths that satisfy the QoS constraints. Each reservation packet carries one or more tickets. When an intermediate node receives a reservation packet, it determines the next hops that can potentially establish connections for the request, calculates the number of tickets to be distributed for each of the next hops, and sends a reservation packet with its share of the tickets to each of the next hops. The algorithm to determine the next hops and to distribute tickets to each

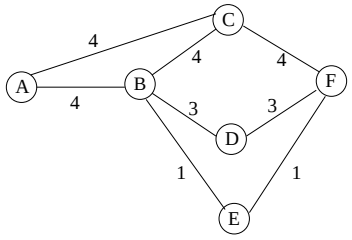
of the next hops will be discussed in the next subsection. The algorithm is called the *routing and ticket distribution* algorithm. Along the path, a reservation packet with more than one ticket may split into multiple reservation packets to explore multiple paths. Since intermediate nodes only distribute the tickets and do not generate any new tickets, the maximum number of reservation packets at any time is bounded by the total number of tickets, t, which was determined at the source node when the connection request arrived. Since each reservation packet probes a path, the maximum number of paths probed is also bounded by t.

The ticket–based QoS routing limits the messaging overhead by controlling the number of tickets for each request. It can deal with the imprecise state information since it explores multiple paths instead of only the "optimal" path that is computed based on the imprecise global network state information. In addition, the ticket–based QoS routing algorithm can emulate other QoS routing schemes by properly generating the number of tickets for each request. For example, the ticket–based QoS routing algorithm can realize the flooding based QoS routing schemes by generating an infinite number of tickets for each request. It can also realize the single–path QoS routing schemes that rely on the global link state information to make routing decisions by generating one ticket for each request.

3.2 Routing and ticket distribution

In large networks, the global network state information maintained by the link state algorithm is imprecise. The ticket–based algorithm includes methods in the routing and ticket distribution algorithm to deal with the imprecise global state information. On the one hand, the algorithm explores multiple paths instead of only the 'optimal' path computed based on the imprecise global state information. This offsets the impact of the imprecise state information. On the other hand, the algorithm attempts to explore better routes by distributing more tickets to probe the routes that have better chance of satisfying the QoS constraints.

In our algorithm, for each destination node in the network, a router ranks its neighbors using the global state information based on their capability to provide QoS paths to the destination node. In the case of a tie, the hop counts of the paths are used to break the tie. If two paths can provide the same bandwidth with the same hop counts, their ranks are assigned randomly. In other words, our algorithm uses *widest–shortest path* routing algorithm [13] to rank its neighbors for each node in the network. This ranking information is stored in the routing table in each router. Consider the example in Figure 1 (a). The numbers in the figure represent bandwidth. The ranking of the neighbors of router B to destination F is shown in Figure 1 (b). Nodes C and A can both provide paths to node F with bandwidth 4, they



(a) An example

neighbor	rank	bandwidth
A	2	4
C	1	4
D	3	3
E	4	1

(b) Routing table entry from node B to node F

Figure 1: Routing and ticket distribution

are ranked higher than nodes D and E. Since the path from nodes B to F needs 3 hops when the path goes through node A and 2 hops when the path goes through node C, node C is ranked No. 1 and node A is ranked No. 2.

Every time a link state packet is received, the ranking information in the routing table is recomputed. This information is used to guide ticket distribution. Specifically, when an intermediate router N receives from its neighbor N_0 a reservation packet for a connection request from source src to destination dst with the QoS requirement equal to $bandwidth$, router N first determines all the candidates that the reservation packet can be forwarded. For a neighbor N_k to be a candidate, link $N \rightarrow N_k$ should be able to support the $bandwidth$ requirement and there must exist a path from N_k to dst that can support the $bandwidth$ requirement. The router maintains the candidates in a list called the *next hops list*. All neighbors except node N_0 are considered as potential candidates. The router then performs the ticket distribution for the candidate based on the ranking information. How to distribute tickets in order to achieve the best performance is hard to determine. One simple heuristic used in our performance evaluation is as follows. The first ranked next hop node gets 50% of the tickets, the second ranked next hop node gets 25% of the tickets and so on. The lowest ranked next hop node gets the remaining of the tickets. The rational behind this ticket distribution heuristic is that more tickets will go towards the paths that have better chance of satisfying the QoS constraints. Other tickets will be distributed to explore other not so good paths to allow the

algorithm to adapt to the imprecise global state information.

Consider the example in Figure 1 (a). When a reservation packet requesting a connection with bandwidth 3 from node A to node F arrives at node B with 4 tickets. Node B will determine that the next hops list includes nodes C and D. Node A is not included in the next hops list since it is where the packet came from. Node E is not included in the next hop list since it can only provide bandwidth 1 which is less than the bandwidth requirement. Node B will send two reservation packets toward nodes C and D. Using the simple ticket distribution heuristic discussed earlier, the packet towards node C will get 50% of the tickets since it is ranked No. 1. The other packet towards node D will get the rest of the tickets.

3.3 Incorporating resource reservation into the ticket–based QoS routing scheme

This section describes the protocol that combines resource reservation with the ticket–based QoS routing scheme. The following control packets are used in the protocol.

- *REQ* packet. The REQ packet is used to probe the paths and to reserve resources along the paths. The information in this packet includes the connection identification, number of tickets, the QoS requirement and the current QoS. The connection identification is specified as a triple (src, dst, seq), where src is the source node number, dst is the destination node number and seq is a sequence number assigned by the source node. The current QoS is the QoS that can be satisfied by the partial path, that is, the QoS that can be satisfied from the source node to the router that receives the REQ packet. A REQ packet will be denoted as $REQ((src, dst, seq), ticket, QoS, cQos)$.

- *REJECT* packet. The $REJECT$ packet is used to release the reserved resources and inform the source the failure of the path establishment when the reservation attempt failed. The information in the $REJECT$ packet includes the connection identification. This packet travels in the opposite direction along the path of the corresponding REQ packet. A $REJECT$ packet will be denoted as $REJECT((src, dst, seq))$.

- *ACCEPT* packet. The $ACCEPT$ packet is used to inform the source and the intermediate nodes that the path has been established. This packet contains the connection identification information, the granted QoS and the path for the connection. An $ACCEPT$ packet will be denoted as $ACCEPT((src, dst, seq), QoS, path)$.

- *RELEASE* packet. The $RELEASE$ packet is used to release the resources for a connection when the communication is done. This packet contains the connection identification and the path for the connection. The $RELEASE$ packet will be denoted as $RELEASE((src, dst, seq), path)$.

Each router maintains precise state information for all outgoing links from the router. Since our algorithm focuses on bandwidth, each router will maintain for each outgoing link the amount of available bandwidth, the amount of bandwidth that is currently used by some connections, and the amount of bandwidth that is reserved but is not currently used. We will use notation $AVAILABLE$, $USED$ and $LOCKED$ to denote the three states of the bandwidths. The router also maintains a routing table. For each node in the network, routing table contains the ranking information of each neighbor and the QoS that can be provided when a connection to the destination goes through that neighbor. The router also records the seq of the latest connection that have been established for each source/destination pair whose connection went through the router. This information is used to deal with the strayed REQ packets. In addition, a router maintains the following information for each connection request that passes the router.

- The parent node and a counter to count the number of REQ packets received from the parent node. The parent node of the connection indicates where the REQ packet came from. The $REJECT$ packet and the $ACCPET$ packet will follow the parent node pointer from the destination to the source. The counter allows that multiple REQ packets for the same request to be sent over the same link without reserving multiple copies of resources.

- Current QoS of the connection. This information is used to compare with the current QoS in a REQ packet to determine whether the new REQ packet can provide better QoS.

- The number of QoS packets sent over each of the outgoing links and the associated resource reserved. This information is used to determined whether all REQs from a node or over a link have been rejected.

- The path for each connection starting from the router that has been established.

The operations in the source nodes, the intermediate nodes and the destination nodes are different in the protocol. Next, we will describe the operations for the three different types of nodes.

- *Source node*. When the source node wants to establish a connection, it constructs a REQ packet with

proper number of tickets and the QoS requirement of the connection. It then sends the REQ packet towards the destination. After that, the source waits for the $ACCEPT$ and/or the $REJECT$ packets. If all the REQ packets for the connection request are rejected, the source may start another round, trying to establish the connection for the request at a later time. If the source node receives an $ACCPET$ packet for the request, the connection has been established and the source can start sending data. In this case, the source records the path for the connection from the $ACCEPT$ packet. Once the source node finishes sending the data, it sends a $RELEASE$ packet to release the resources for the connection.

- *Intermediate nodes.* The following operations are performed at intermediate nodes when they receive different control packets.
  - $REQ((src, dst, seq), ticket, QoS, cQos)$. If (src, dst, seq) has been established, the router rejects the request since there is no meaning to reserve resources for an established connection. If resources have not been reserved for the connection (the REQ packet is the first one that reaches the router), the router will check whether it has sufficient resources to satisfy the connection using the routing and ticket distribution algorithm. If next hop candidates exist, the router will distribute the tickets for the candidates and send REQ packets to the candidates. If there is no next hop candidate, the router rejects the request. If the current REQ packet is not the first one to reach the router, the router checks whether the current REQ can provide better QoS for the connection. If the current REQ carries better $cQoS$, the router would then forward the REQ. Otherwise, the router rejects the REQ packet. When a router forwards the second REQ packet, it needs to change the parent node for this connection, reject the old REQ if the new parent node is not the same as the old parent node, and update the counter that records the number of REQ packets for the connection using the link. When the second REQ is sent through a link, the router will just increase the counter of the number of REQ packets sent over the link, but not reserve extra resources.
  - $REJECT((src, dst, seq))$. When an intermediate node receives a $REJECT$ packet, the intermediate node checks the number of REQ packets sent through that link. When the all REQ packets sent through that link have been rejected, the router releases the resources locked for the connection over that link. Otherwise, the router decreases the counter for the number of REQ packets sent through the link. When all REQ packets from this router (all REQ packets for all outgoing links) are rejected, $REJECT$ packets are sent to the parent node for this connection in the network. The number of $REJECT$ packets to be sent to the parent node is equal to the number of REQ packets it received from the parent node.
  - $ACCEPT((src, dst, seq), QoS, path)$. When an $ACCEPT$ packet is received, the intermediate node changes the state of the resources from the $LOCKED$ state to the $USED$ state, releases the extra resources reserved but not granted and forwards the $ACCEPT$ packet to the parent node for this connection. The router also records the $path$ for the connection in the $ACCEPT$ packet.
  - $RELEASE((src, dst, seq), path)$. The router releases the resources used by the connection and forwards the packet to the next hop according to the $path$ field in the packet.
- Destination node. When the destination node receives a REQ packet, it checks whether the connection has been established. If the connection has been established, the REQ packet is rejected. Otherwise, an $ACCEPT$ packet is constructed is sent to the source. The destination will not receive $REJECT$ or $ACCPET$ packets.

A number of optimizations are incorporated in this protocol. First, to alleviate the over–reservation problem, the protocol ensures that the resources on one link are reserved at most once for each connection. Second, the protocol enforces an early–release policy. That is, when a reservation packet reaches a node that another reservation packet reached before, the second reservation packet is rejected if it cannot provide better QoS. This allows the resources reserved by the second reservation packet to be released at the earliest time. This policy also ensures that the QoS paths found by the protocol are loop–free. When the loop is formed, a reservation packet will reach a node two times. When the second time a reservation packet is processed at a node, it will be rejected since it cannot provide better QoS than the first time it arrived at the node. Third, the protocol minimizes the messaging overhead by sending a reject packet towards the source only after all out–going reservation packets from the node are rejected.

4 Performance evaluation

To evaluate the performance of the protocol and to study the impact of resource reservation on the multi–path QoS routing, we develop a cycle–by–cycle network simulator that simulates the ticket–based distributed QoS routing protocol with resource reservation. We use bandwidth as the QoS metric. The major features of the network simulator are the followings.

- The simulator simulates the QoS routing protocol that combines resource reservation with the ticket–based QoS routing scheme. By manipulating the number of tickets for each request, the simulator can simulate the flooding based routing schemes that do not use the global network state information to make routing decisions, multi–path QoS routing schemes that probe multiple paths for each connection request based on the global network state information, and single–path QoS routing schemes that probe one path at a time.
- The control network is logically separated from the data network. The control network is used to exchange control packets and link state packets while the data network is used to exchange data messages.
- The simulator simulates various link state update mechanisms including the periodical link state update mechanism and the sudden–change link state update mechanism [1] which broadcasts link state information when the state of a link changes drastically.
- The simulator allows the user to specify the following parameters:
  - *network load*. The network load is specified using three parameters, the connection request generation rate (r), the connection duration (d) and the bandwidth requirement (b). The connection requests arrive at each node following a Poisson distribution with an average of r messages generated every time unit. When a request is generated at a node, the destination of the request is generated randomly among the other nodes in the system. A request is discarded when it is blocked.
  - *network topology*. The network topology is independent of the protocol and is loaded from a file at the beginning of the simulation. We use 4×4, 6×6, 8×8 and 10×10 meshes in our simulation study. The link capacity is assumed to be 100 for all the studies.
  - *link state update frequency* (uf) in the periodical link state update mechanism. This parameter specifies the time span that the link state packets are generated.
  - *number of tickets* (t) for each request.
  - *control packet processing speed* (ct). This parameter specifies how fast a router can process the control packets it received.

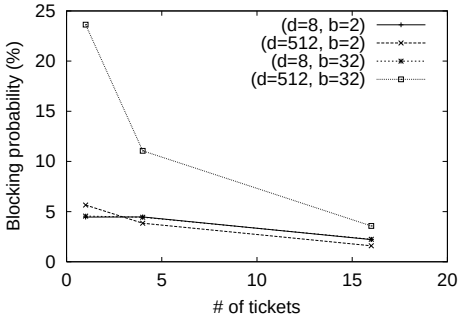
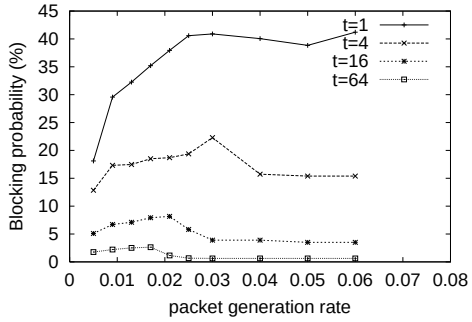


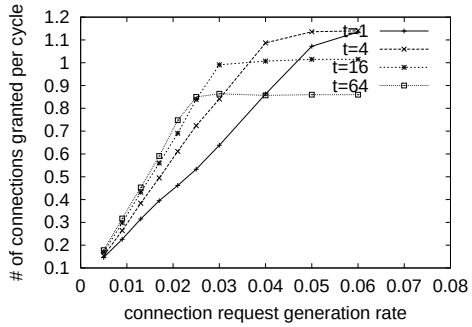
Figure 2: Blocking probability for light traffic

We use two performance metrics, the blocking probability and the average number of connections granted, to evaluate the protocols. The blocking probability is the ratio of the number of connection requests that are rejected (blocked) and the total number of connection requests processed. Under light traffic, both the blocking probability and the number of connection requests granted are good indicators of the performance of the protocols. Under heavy traffic, the number of connection requests granted is the better indicator of the performance. The simulation time is measured in time cycles, which is the basic unit of the network activities. All other network activities, such as the control packet processing time and packet propagation time over a link, take multiples of the time cycle to complete. In the simulations, we assume that the propagation delay in each link 1 cycle and the control packet processing time (ct) in each router is also 1 cycle. All the simulation results are obtained with 95% confidence level and 5% confidence interval.

Figure 2 shows the blocking probability as a function of the number of tickets for each connection request when the network is under light traffic. This experiment is done on 6×6 meshes and assumes $r = 0.001$ and $uf = 500$. Four cases are considered, small connection duration and small bandwidth requirement ($d = 8, b = 2$), small connection duration and large bandwidth requirement ($d = 8, b = 32$), large connection duration and small bandwidth requirement ($d = 512, b = 2$), and large connection duration and large bandwidth requirement ($d = 512, b = 16$). These four cases represent different traffic patterns when the network is under light load. As shown in Figure 2, the blocking probability decreases as the number of tickets increases, which indicates that under light load, the protocols that probe more

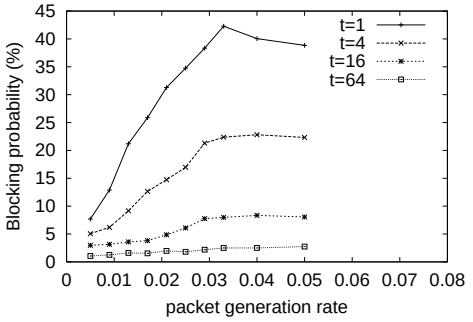


(a) Blocking probability

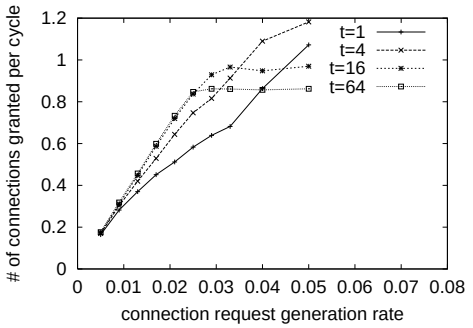


(b) No. of connections granted

Figure 3: Performance of the protocols $(d = 8, b = 2)$



(a) Blocking probability



(b) No. of connections granted

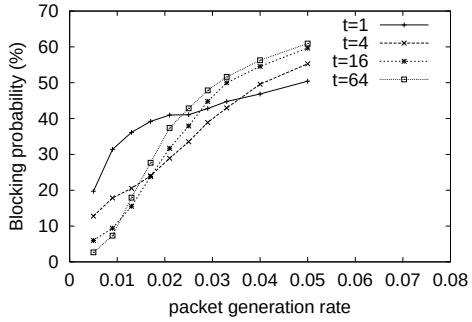
Figure 4: Performance of the protocols $(d = 512, b = 2)$

paths perform better than the protocols that probe less number of paths. This result is consistent for all the four traffic patterns. When the network load is low, the over reservation problem for probing more paths is not significant since the network has sufficient resources to support both the offered traffic and the overly reserved resources. Thus, the protocols that probe more paths and have higher path establishment probability perform better than other protocols.

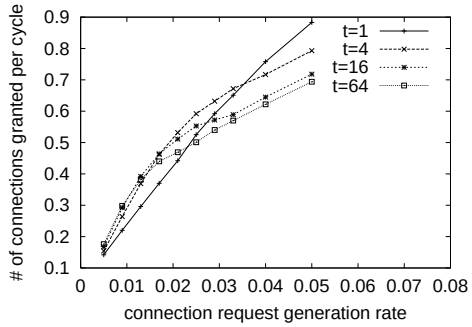
Figures 3, 4, 5, 6 show the performance of the protocols when the network load is higher. We also consider the four types of traffic patterns, small connection duration and small bandwidth requirement $(d = 8, b = 2)$ in Figure 3, large connection duration and small bandwidth requirement $(d = 512, b = 2)$ in Figure 4, small connection duration and large bandwidth requirement $(d = 8, b = 32)$ in Figure 5, and large connection duration and large bandwidth requirement $(d = 512, b = 16)$. These experiments are done on 6×6 meshes with $uf = 500$.

Figure 3 and Figure 4 have similar trend. When the bandwidth requirement is small $(b = 2)$, probing more paths always results in lower blocking probability. However, when the network load becomes higher, although the protocols with larger number of tickets still have smaller blocking probability, the throughput is also smaller. Two factors contribute to this observation. First, it takes longer time for a protocol with larger number of tickets to reject a request since it must wait until all tickets are rejected. Thus, although the protocols with larger number of tickets have higher probability to establish each individual connection, these protocols may degrade the overall system throughput by spending more time to reject connection requests. Second, larger number of tickets also results in heavier traffic in the control network, which increases the connection establishment time for each connection. This also affects the system throughput.

Figure 5 shows the case when the bandwidth requirement is high $(b = 32)$ and the duration requirement is small (d=8). In this case, the global link state information in each router is extremely imprecise. As seen in the figure, protocols with larger number of tickets performs well only

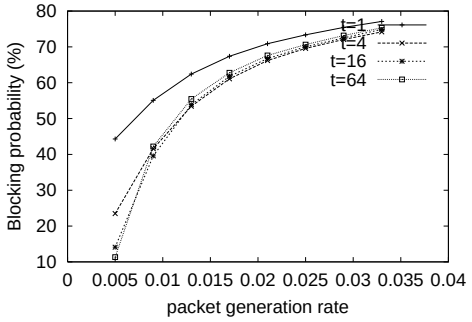


(a) Blocking probability

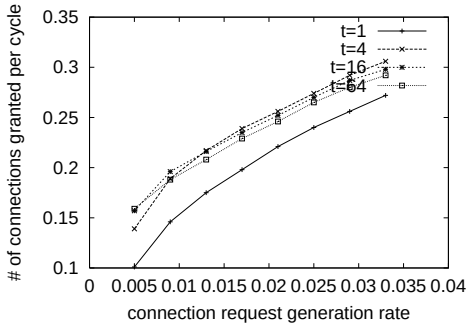


(b) No. of connections granted

Figure 5: Performance of the protocols ($d = 8, b = 32$)



(a) Blocking probability



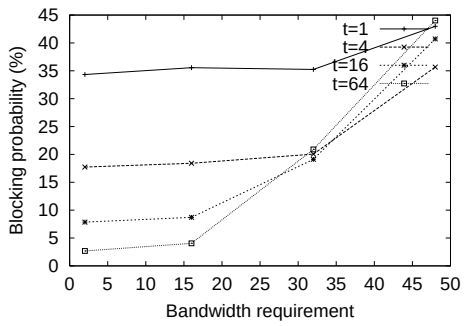
(b) No. of connections granted

Figure 6: Performance of the protocols ($d = 512, b = 16$)

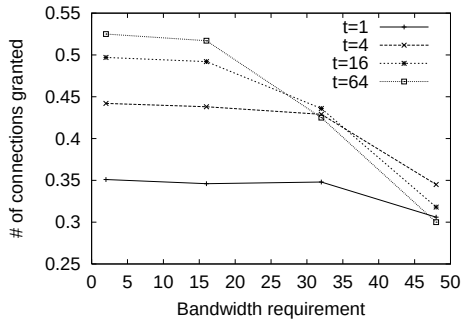
when the network load is very small. When the network load becomes higher, even before the network is saturated, the protocols with smaller number of tickets have both lower blocking probability and higher throughput. The over reservation problem dominates the overall performance in this case. Figure 6 shows the case when the bandwidth requirement is high ($b = 32$) and the duration requirement is high (d=512). In this case, the global link state information in each router is not as imprecise as the case in Figure 5 since the connection duration is larger. The performance of the single path routing is noticeably worse than the performance of multi–path routing both in terms of the blocking probability and the system throughput. Probing multiple paths is more efficient than probing a single path when the path that satisfies the QoS constraints is difficult to find.

Figure 7 shows the performance of the protocols with different number of tickets for different bandwidth requirements. This experiment is done on 6×6 meshes with $r = 0.015$, $d = 8$ and $uf = 500$. When the bandwidth requirement is small, the performance of the protocols with large number of tickets is much better than that of the protocols with small number of tickets. However, as the bandwidth requirement of each request increases, the over reservation problem starts to have more impacts and the performance of the protocols with large number of tickets decreases drastically. As shown in the figure, the performance of the multi–path routing is quite sensitive to the bandwidth requirement.

Figure 8 and Figure 9 study the impact of the link state update frequency. The impact of link state update frequency is two folds. First, higher link state update frequency results in more precise global network state, which improves the routing performance. Second, higher link state update frequency also results in higher control network traffic, which may slow down the path establishment. Figure 8 shows the impact of link state update frequency on a 4-ticket routing algorithm. This experiment is done on 6×6 meshes and assumes $r = 0.017$. Since the network is not saturated, we only compare the blocking probability. As can be seen in the figure, the frequency of link state update has little im-



(a) Blocking probability



(b) No. of connections granted

Figure 7: Performance of the protocols for different bandwidth requirement

pact when the connection duration is small ($d = 8$). When the duration is so small, the range of the link state update frequency studied (250 to 4000) does not make much difference in the precision of the global network state. For larger duration ($d = 512$), the routing performance improves when the update frequency is sufficiently large ($uf = 250$).

Figure 9 shows the impact of link state update frequency on different multi–path QoS routing algorithms. This experiment is done on 6×6 meshes and assumes $r = 0.017$, $d = 512$ and $b = 2$. As shown in the figure, the link state update frequency greatly affects the performance of the single path routing. However, it has little impact on the protocols with large number of tickets ($t = 16$). This result shows that the multi–path routing algorithm can deal with imprecise global network state information effectively even when resource reservation is incorporated.

Figures 10 show the impact of network size on the protocols when the network is under light load (r=0.009). Under light load, protocols with large number of tickets perform

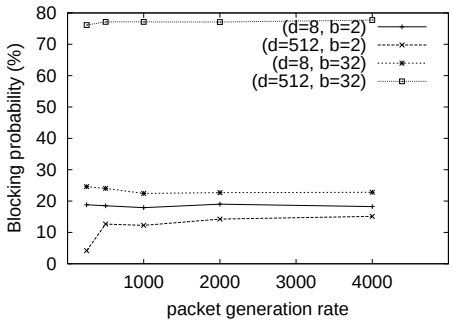


Figure 8: Impact of link state update frequency on the 4–ticket protocol

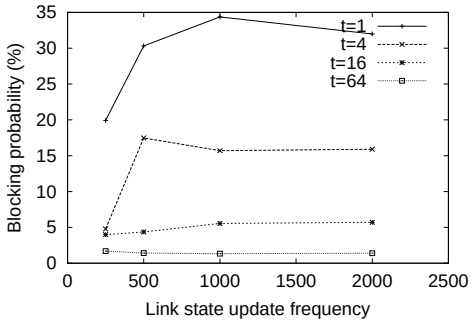


Figure 9: Impact of link state update frequency on different protocols

better than the protocols with small number of tickets for all the network sizes. Our other experiment shows that protocols with large number of tickets do not scale with respect to the network size. A 16–ticket protocol will saturate the control network for an 8×8 meshes when $r = 0.02$ while protocols with 1 or 4 tickets will not saturate the network.

5 Conclusion

In this paper, we develop a new QoS routing protocol that combines resource reservation with the ticket–based distributed multi–path QoS routing scheme, evaluate the new routing protocol through extensive simulation, study the impacts of other network components on the performance of the multi-path QoS routing with resource reservation. Our major conclusions are the followings:

- When the network is under light load, probing more paths results in better performance for all types of traffic patterns.
- When the network is close to saturation (very high load), probing more paths performs worse than probing less paths. Protocols with larger number of tickets have, in most cases, lower maximum throughput.

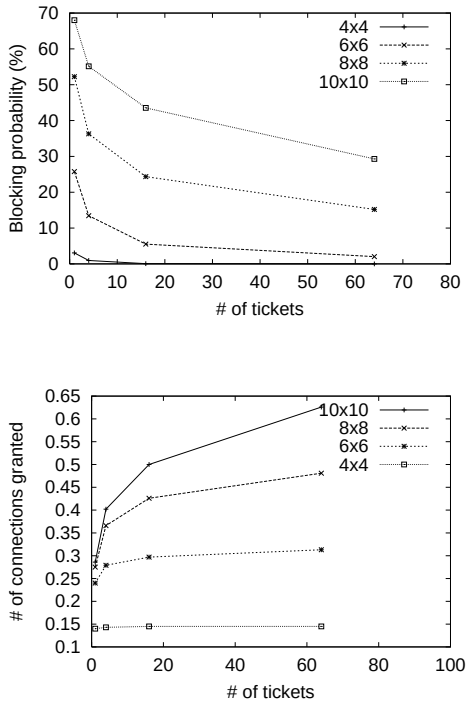


Figure 10: Impact of network size

There are three causes for the network to be under heavy load, (1) high request generation rate, (2) long connection duration, and (3) large bandwidth requirement. Among the three causes, multi–path QoS routing is more sensitive to the bandwidth requirement and less sensitive to the request generation rate and the connection duration.

- Multi–path QoS routing handles the imprecise global network state information more effectively than the single path QoS routing scheme even when resource reservation is incorporated. Protocols with large number of tickets are not very sensitive to the link state update frequency, while the single path routing is very sensitive to the link state update frequency. Although probing more paths may affect the precision of the global network state information, the overall routing performance of multi–path routing is, in most cases, better than that of single–path routing.

- Protocols with large number of tickets are not scalable. However, when the network is under light load, protocols with large number of tickets offer low blocking probability for reasonably large networks.

References

[1] G. Apostolopoulos, R. Guerin, S. Kamat, and S. Tripathi, "Improving QoS Routing Performance Under Inaccurate Link State Information." *Proceedings of the 16th International Teletraffic Congress*, Edinburgh, United Kingdom, June 7-11, 1999.

[2] S. Chen and K. Nahrstedt, "On Finding Multi–Constrained Paths", *IEEE International Conference on Communications*, June 1998.

[3] S. Chen and K. Nahrstedt, "Distributed QoS routing with Imprecise State Information." *IEEE International Conference on Computer and Communication Networks*, 1998.

[4] S. Chen and K. Nahrstedt, "An Overview of Quality–of–Service Routing for the Next Generation High–Speed Networks: Problems and Solutions", *IEEE Network*, Special Issue on Transmission and Distribution of Digital Video, Nov./Dec., 1998.

[5] I. Cidon, R. Rom and Y. Shavitt, "Multi–Path Routing Combined with Resource Reservation", In *IEEE INFORMCOM'97*, April 1997.

[6] R. Guerin and A. Orda, "QoS–based Routing in Networks with Inaccurate Information: Theory and Algorithms." *IEEE INFOCOM'97*, April 1997.

[7] Q. Ma and P. Steenkiste, "Quality–of–Service Routing with Performance Guarantees." *4th International IFIP Workshop on Quality of Service*, May 1997.

[8] *ATM Forum, Private network network interface (PNNI) v1.0 specifications*, June 1996.

[9] H.K. Pung, J. Song and L. Jacob, "Fast and Efficient Flooding Based QoS Routing Algorithm", *IEEE International Conference on Computer Communications and Networks*, October 1999.

[10] H.F. Salama, D. S. Reeves and Y. Viniotis, "A Distributed Algorithm for Delay–Constrained Unicast Routing." *IEEE INFOCOM'97*, April 1997.

[11] A. Shaikh, J. Rexford and K. Shin, "Dynamics of Quality–of–Service Routing with Inaccurate Link–state Information." *University of Michigan Technical Report CSE–TR–350–97*, November 1997.

[12] K. G. Shin and C.C. Chou, "A Distributed QoS Routing Using Bounded Flooding", *University of Michigan CSE technical report, CSE–TR–388–99*, 1999.

[13] Z. Wang and J. Crowcroft, "QoS Routing for Supporting Resource Reservation." *IEEE Journal on Selected Areas in Communications*, September 1996.

[14] X. Yuan, "On the Extended Bellman–Ford Algorithms to Solver Two–Constraints Quality–of–Service Routing Problems", *IEEE International Conference on Computer and Communication Networks*, November 1999.

[15] L. Zhang, S. Deering, D. Estrin, S. Shenker and D. Zappala, "RSVP, A New Resource ReSerVation Protocol", *IEEE Network*, September 1993.