Resource Reservation Mechanisms for Distributed Multi–path Quality of Service Routing

Xin Yuan Hui Ding Yuan Zhong Jie Zhang
Department of Computer Science
Florida State University
Tallahassee, FL 32306
{xyuan,ding,zhong,zhang}@cs.fsu.edu

Abstract

This paper studies resource reservation mechanisms that can be incorporated into distributed multi–path Quality–of–Service (QoS) routing algorithms. Three resource reservation schemes, the forward reservation scheme, the backward reservation scheme and the hybrid reservation scheme, are investigated. Protocols that combine these resource reservation schemes with a ticket–based distributed QoS routing algorithm are developed. Simulations are carried out to compare the performance of the protocols and to study the impact of system parameters on the performance of the protocols. Our results show that, in most of the cases, backward reservation protocols are more efficient than their forward reservation and hybrid reservation counterparts.

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 in order to achieve QoS guarantees, must be addressed to support communications 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 However, in B–ISDN, resource availability may change rapidly and the route information may be outdated.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 resources in the second step. Combining the two steps was suggested to overcome this problem [4, 7, 8, 11, 16]. 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 [2, 4, 5, 6, 8, 9, 11, 12, 14]. Among these algorithms, multi–path QoS routing schemes [2] that use global network information to make routing decisions are promising for future networks for a number of reasons. First, by using the global network information, these protocols incur less messaging overhead compared to the flooding based protocols [4, 8, 11] 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 [5, 6, 9, 12, 14] that probe one path at a time.

When combining resource reservation with the QoS routing schemes, 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 our previous work [16], we have developed a routing protocol that combine the *forward* reservation scheme with a ticket based distributed QoS routing scheme [2]. In forward reservation, network resources are reserved when the routing algorithm probes the paths from the source to the destination. The main limitation of this scheme is that the protocol tends to reserve more resources than needed for a connection request due to the imprecise global network state information. In this paper, we study two resource reservation schemes, namely the backward reservation scheme and the hybrid reservation scheme, that attempt to overcome the over–reservation problem in the forward reservation scheme. In the backward reservation scheme, the imprecise global network information is only used to make routing decisions, more up–to–date network state information along the paths that are probed on demand for each connection request is used to make reservation decisions. By doing this, the backward reservation scheme greatly improves the resource reservation efficiency and alleviates the over–reservation problem. The hybrid reservation scheme tries to combine the advantages of both forward reservation and backward reservation and overcome their limitations by allowing forward reservation along a limited number of paths while probing other paths for backward reservation.

In this paper, we develop protocols that combine a ticket–based QoS routing algorithm [2] with the backward reservation scheme and with the hybrid resource reservation scheme, compare the effectiveness of the newly designed protocols with that of the forward reservation protocol and study the impact of system parameters on the resource reservation schemes. Our results show that, in most of the cases, backward reservation protocols are more efficient than their forward reservation and hybrid reservation counterparts, although backward reservation is slightly more complicated than forward reservation.

The rest of the paper is organized as follows. Section 2 summarizes the related work. Section 3 introduces the ticket based QoS routing scheme with forward reservation. Section 4 presents the backward reservation protocol. Section 5 describes the hybrid reservation scheme. Section 6 reports our performance study and Section 7 concludes the paper.

2 Relative work

QoS routing has attracted much attention recently. An extensive survey can be found in [3]. Most existing QoS routing schemes decouple the routing issues from the resource reservation issues [2, 5, 6, 9, 12, 14]. Few schemes [4, 7, 8, 11, 16] combine routing with resource reservation. The routing schemes in [4, 8, 11] are flooding based where the global network information does not affect the routing performance. The scheme in [7] is a single–path routing scheme where the over reservation problem is not severe. Zhong [16] describes a protocol that combines the forward reservation scheme with a multi–path QoS routing algorithm. All the protocols [4, 7, 8, 11, 16] employ the forward reservation scheme to reserve resources. The idea of the backward reservation scheme was proposed for path establishment in multiplexed all optical networks [13]. Routing and resource reservation in all optical networks is different from routing and resource reservation for establishing a path that satisfies QoS constraints. The backward reservation scheme and the hybrid reservation scheme for QoS routing have not been studied before. This paper develops protocols that combine the backward and hybrid resource reservation schemes with a multi–path QoS routing algorithm and compares the performance of these two resource reservation schemes with that of the forward resource reservation scheme. Studies on other network components, such as the link state update mechanisms and network topology, are reported in [1, 10]. Resource reservation adds another dimension to the network components and requires most of the performance issues to be revisited. In [2], the authors proposed a ticket–based routing algorithm to solve the delay–constrained least–cost routing without considering resource reservation. We use a variant of this algorithm as the base QoS routing algorithm and study resource reservation mechanisms.

3 Ticket–based distributed QoS routing with forward reservation

This section presents the QoS routing protocol that incorporates forward reservation into a ticket–based QoS routing scheme. We will first introduce the ticket–based QoS routing algorithm then briefly describe the protocol that combines the ticket–based routing algorithm with forward reservation. A detailed description of the ticket–based routing algorithm for the delay–constrained least–cost routing problem can be found in [2]. The protocol that combines the ticket–based routing algorithm with the forward resource reservation scheme can be found in [16]. In the protocol, we assume that a link state algorithm is used to maintain the global state information. In the rest of the paper, we will use bandwidth as the QoS metric to illustrate the protocols. However, the protocols can easily be modified to deal with other QoS metrics, such as delay, delay jitter and cost.

3.1 Ticket–based distributed routing algorithm

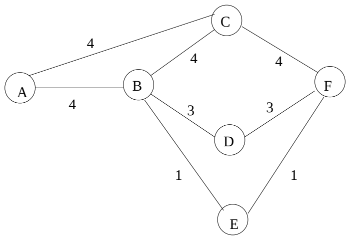
When a connection request arrives at the source node, a certain number (t) of tickets are generated and a probe packet with the t tickets is sent to the destination in search of paths that satisfy the QoS constraints. Each probe packet carries one or more tickets. When an intermediate node receives a probe 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 probe 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 probe packet with more than one ticket may split into multiple probe packets to explore multiple paths. Since intermediate nodes only distribute the tickets but not generate any new tickets, the maximum number of probe 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 probe 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 ticket–based 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 uses the global state information to rank its neighbors based on the capability to provide QoS routes 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 [12] to rank its neighbors for each of the nodes in the network. This ranking information is stored



(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

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 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 ranking information is used to guide the ticket distribution. Specifically, when an intermediate router N receives from its neighbor N_0 a probe 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 probe 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 will be considered. The router then performs the ticket distribution for the candidates. The ticket distribution is based on the ranking information. How to distribute tickets in order to achieve the best performance is hard to determine. One simple heuristic that is 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 imprecision of the global state information.

Consider the example in Figure 1 (a). When a probe 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 probe 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 Forward reservation

The protocol that combines the ticket–based routing scheme with forward reservation works as follows. When a new connection request arrives at the source node, the source node sends a reservation packet with t ticket to the destination. The reservation packet goes through the network, separates into multiple reservation packets if necessary, and reserves resources along the path. A reservation packet fails when there is no available resource to support the QoS requirement of the connection. In this case, the resources reserved by the reservation packet in the partial path are released if no other reservation packets for the same connection request share the resources. When the reservation packet reaches the destination, the destination knows that the resources along the path have been reserved for the connection and sends an accept packet to inform the source and the intermediate nodes that this connection has been established. When the accept packet reaches the source, the source can start sending data. Once the source finishes sending data, it sends a release packet to release the resources along the path for the connection.

A number of optimizations are incorporated in the protocol [16]. First, to alleviate the over–reservation problem, the protocol ensures that the resources in 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 has been reached by another reservation packet for the same request, 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 final paths found by the protocol are loop–free. When a loop is formed, a reservation packet will reach a node two times. When the second time the 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 Backward Reservation

The forward reservation protocol relies solely on the global network state information maintained by the link state algorithm to make both routing and resource reservation decisions. Due to the imprecise global network state information, the forward reservation protocol tends to overly reserve resources for a connection request. The backward reservation scheme overcomes this limitation by postponing resource reservation until the protocol probes the network and obtains more up–to–date network information (than the information maintained by the link state information) about the paths from the source to the destination. In the backward reservation protocol, the imprecise global network information maintained by the link state algorithm is only used in making routing decisions, more accurate network state information along the paths which are probed on demand for each connection request is used to make reservation decisions. Hence, the backward reservation scheme greatly improves the resource reservation performance and alleviates the over–reservation problem.

The ticket–based routing algorithm combined with backward reservation works as follows. When a connection request arrives at the source node, a certain number (t) of tickets are generated and a probe packet with the t tickets is sent to the destination in search of paths that satisfy the QoS constraints. When an intermediate node receives a probe packet, it determines the next hops that can potentially establish a connection for the request, calculates the number of tickets to be distributed for each of the next hops, and sends a probe packet with its share of the tickets to each of the next hops. The probe packet carries at least one ticket, records the path that it traveled and maintains the QoS metric that the path can support. A probe packet may be rejected by an intermediate router when the router determines that there is no path that can reach the destination while satisfying the QoS requirement. Once the probe packet reaches the destination, the destination knows that the resources on the path that the probe packet traveled can satisfy the QoS requirement and sends a reservation packet to the source along the path in the opposite direction. The reservation packet reserves resources along the path to the source. Once the source node receives the reservation packet, it can start the data transmission. Note that the reservation packet may still be rejected due to the lack of resources since the resource availability may change. However, since the network state information carried in the probe packet is much more precise than the global network state information maintained by the link state algorithm, resource reservation in the backward reservation scheme has much higher successful probability than that in the forward reservation scheme. Note also that since the protocol explores multiple paths, the over reservation problem can still exist in the backward reservation scheme. In our protocol, we use a protocol parameter, bn, to limit the number of outstanding backward reservations for each request to control the over–reservation problem. At a given time, there are at most bn reservation packets sent from the destination for a given request, that is, at most bn paths can be reserved at the same time for any given connection request. Backward reservation can be partitioned into two phases. The first phase is the *probing* phase where the global network state information maintained by the link state algorithm is used to decide which paths to probe and more accurate state information about the paths is gather by the probe packets. The second phase is the *reservation* phase where the probing results are used to decide which paths to reserve.

Next, we will describe the control packets used in this protocol, discuss the information that each router must maintain for this protocol, and present the operations in the source, the intermediate and the destination nodes. The backward reservation scheme uses the following control packets.

- *PROB* packet. This packet collects the up–to–date network state information and finds paths that have sufficient resources for a connection request. The information in this packet includes the connection identification, the number of tickets, the QoS requirement, the partial path that the packet traveled and the current QoS which is the QoS that can be supported by the partial path. A PROB packet will be denoted as $PROB((src, dst, seq), ticket, QoS, cQos, PATH)$, where (src, dst, seq) is the connection identifier, $ticket$ is the number of tickets carried by the packet, QoS is the QoS requirement of the connection, $cQoS$ is the current QoS and $PATH$ is the partial path that the packet traveled.

- *RES* packet. This packet performs the resource reservation from the destination to the source along the path that the corresponding PROB packet traveled. The information in this packet includes the connection identification, the number of tickets, the QoS requirement, the current QoS and the path. A RES packet will be denoted as $RES((src, dst, seq), ticket, QoS, cQoS, PATH)$. Here, $PATH$ contains the full path from the source to the destination, and $cQoS$ records the QoS metric that can be provided by the partial path from the current node to the destination.

- *CONFIRM* packet. This packet informs the intermediate nodes and the destination that the connection has been established. The information in this packet includes the connection identifier and the path. A CONFIRM packet will be denoted as $CONFIRM((src, dst, seq), PATH)$.

- *REJECT* packet. This packet informs the source that the path establishment fails. The information in this packet includes connection identifier, the number of tickets and the path. A REJECT packet will be denoted as $REJECT((src, dst, seq), ticket, PATH)$.

- *FAIL* packet. This packet informs the intermediate nodes that a reservation attempt has failed and the corresponding locked resources can be released. The information in this packet includes connection identifier and the path. A FAIL packet will be denoted as $FAIL((src, dst, seq), PATH)$.

- *RELEASE* packet. This packet releases the network resources when the transmission 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 notion $AVAILABLE$, $USED$ and $LOCKED$ to denote the three states. The router also maintains a routing table. For each node in the network, the 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 connections went through the router. This information is used to determine whether a connection has been established. In addition, a router maintains the following information for each connection request that passes the router.

- The number of RES packets sent over each of the outgoing links and the associated resources. This information is used to determined whether all RESs from a node or over a link have been rejected. This counter enables multiple RES packets for the same connection to share one copy of resources on each link.

- Current QoS for the connection. This is the QoS that can be supported by the partial path from the source to this router. This information is used to reject the PROB packets that arrive at the router later and carry worse $cQoS$.

- The number of tickets rejected so far. This information is maintained by the source node and is used to determine whether all tickets for a connection request have been rejected.

- The path for each connection starting from the router that has been established and has not been released. This information is used to generate the $RELEASE$ packet at the source node when the communication finishes.
- The number of RES packets sent for each connection whose destination is the router (the destination maintains the number of RES packets sent for each connection). This information is used together with the maximum number of outstanding backward reservations (a protocol parameter), bn, to control the over reservation problem.

Knowing all the control packets used in the protocol and the information maintained by each router, we will now present the operations on the source, the intermediate and the destination nodes.

- *Source node.* When the source node wants to establish a connection, it constructs a PROB packet with t tickets, where t is a protocol parameter, and the QoS requirement of the connection. The source sends the PROB packet towards the destination. The source then waits for either RES or REJECT packets for the connection request. When a RES packet reaches the source node, the path is established. The source changes the state of the connection to be established (by updating the largest seq number of the (src, dst) pair), sends a $CONFIRM$ packet to inform the intermediate nodes and the destination node about the establishment of the connection, and starts sending data. When the source finishes sending data, it sends a RELEASE packet to release the resources for the connection. When the connection has not been established and a REJECT packet is received by the source node, the source node calculates the total number of tickets rejected (each REJECT packet carries the number of tickets rejected by the packet). When the total number of tickets rejected is equal to t, the source node knows that the path establishment attempt has failed and it can try the request at a later time.
- *Intermediate nodes.* An intermediate node rejects all PROB packets and RES packets if the connection has been established. To reject a PROB packet, a REJECT packet with the same number of tickets as the PROB packet is sent to the source. To reject a RES packet, A REJECT packet with the same number of ticket as the RES packet is sent to the source and a $FAIL$ packet is sent to the destination. The following operations are performed at intermediate nodes when they receive different control packets and the connection has not been established. Figure 2 shows the pseudo–code for the operations at intermediate nodes.
  - PROB packet. When an intermediate router receives a PROB packet, the router checks whether the network has sufficient resources to satisfy the connection using the routing and ticket distribution algorithm. If the router decides that the network does not have sufficient resources, it rejects the PROB packet. The router also rejects the PROB packet if other PROB packets with better $cQoS$ for the same connection request have reached the router. If the PROB packet is not rejected, the router distributes the tickets for all the outgoing links that may satisfy the QoS constraints, records the current node in the partial path, updates the current QoS for each of the outgoing links and forwards the PROB packets.

Let the intermediate node number be N.
IF (PROB((src, dst, seq), ticket, QoS, cQoS, $PATH = \{N_1 \rightarrow N_2 \rightarrow ... \rightarrow N_i\}$) is received) **THEN**
  IF ((src, dst, seq) has been established)) **THEN**
    send REJECT((src, dst, seq), ticket, PATH) to node N_i
  ELSE IF (other PROB has arrived and carried better cQoS) **THEN**
    send REJECT((src, dst, seq), ticket, PATH) to node N_i
  ELSE
    IF (no proper next hops) **THEN**
      Send REJECT((src, dst, seq), ticket, PATH) to node N_i
    ELSE
      Let nodes M_1, ..., M_j be all the possible next hops.
      Let t_1, ..., t_j be the corresponding ticket distribution for each node
      Compute $cQoS_1$, ..., $cQoS_j$ for each of the next hops.
      FOR all $k \in \{1, 2, ..., j\}$
        send PROB((src, dst, seq), ticket, QoS, $cQoS_k$, $PATH' = \{N_1 \rightarrow N_2 \rightarrow ... \rightarrow N_i \rightarrow N\}$) to M_k.
      END FOR
    END IF
  END IF
ELSE IF (RES((src, dst, seq), ticket, QoS, cQoS, $PATH = \{src \rightarrow ...N_0 \rightarrow N \rightarrow N_1... \rightarrow dst\}$) is received)
  IF ((src, dst, seq) has been established)) **THEN**
    Send FAIL((src, dst, seq), PATH) to node N_1.
  ELSE
    IF (connection (src, dst, seq) has reserved resources on link $N \rightarrow N_1$) **THEN**
      Increase counter for (src, dst, seq) on link $N \rightarrow N_1$ by 1
      Calculate new cQoS'
      Send RES((src, dst, seq), ticket, QoS, cQoS', PATH) to N_0
    ELSE IF (Link $N \rightarrow N_1$ does not have sufficient resources for the connection) **THEN**
      Send REJECT((src, dst, seq), ticket, PATH) to node N_0
      Send FAIL((src, dst, seq), PATH) to node N_1.
    ELSE
      Reserve the resources on link $N \rightarrow N_1$ for the connection (src, dst, seq)
      Initialize the counter for (src, dst, seq) on link $N \rightarrow N_1$
      Calculate cQoS'
      Send RES((src, dst, seq), ticket, QoS, cQoS', PATH) to N_0
    END IF
  END IF
ELSE IF (CONFIRM((src,dst,seq), PATH) is received) **THEN**
  Change the link state from $LOCKED$ to $USED$
  Record the establishment of the connection(update seq), forward the packet
ELSE IF (REJECT((src,dst,seq), ticket, PATH) is received) **THEN**
  Forward the packet to the next hop towards the source.
Else IF (FAIL((src, dst, seq), $PATH = \{... \rightarrow N \rightarrow N_1 \rightarrow ...\}$) is received) **THEN**
  Decrease the counter for connection (src, dst, seq) on link $N \rightarrow N_1$
  IF ($counter == 0$) release the locked resources
  Send FAIL((src, dst, seq), PATH) to node N_1.
ELSE IF (RELEASE((src, dst, seq), PATH) is received) **THEN**
  Release the resources used by (src, dst, seq)
  Forward the packet along $PATH$ towards the destination
END IF

Figure 2: Operations at intermediate nodes for the backward reservation scheme

- RES packet. When an intermediate router receives a RES packet, it does the followings. If an earlier RES packet has been processed and forwarded, the router will forward the current RES packet and increase the counter for the connection for using the resources on the link. Otherwise, the router will first check whether the link has sufficient resources for the connection. If the link does not have the resources, the RES packet is rejected. If the link has the resources, the router reserves the resources on the link for the connection, sets the counter for the connection to be 1, and forwards and the RES packet.
- CONFIRM packet. The router changes the resources reserved for this connection from the $LOCKED$ state to the $USED$ state, records that the connection has been established and forwards the packet towards the destination.
- REJECT packet. The router forwards each REJECT packet following the PATH field in the packet.
- FAIL packet. The router decreases the counter for the connection using the corresponding link. If the $counter = 0$, the router releases the resources for the connection. The router will then forward the FAIL packet towards the destination using the PATH field in the packet.
- RELEASE packet. The router releases the resources for the connection and forwards the RELEASE packet to the destination following the PATH in the packet.

- Destination node. When the destination node receives a PROB packet, it checks whether the connection has been established. If the connection has been established, the PROB packet is discarded. If the number of RES packets sent for this connection is larger than bn, the PROB packet is rejected. Otherwise, a RES packet is constructed and sent to the source following the path that the PROB packet traveled. When the destination node receives a FAIL packet, it decrease the counter for the number of RES packets sent for the connection. When the destination node receives a RELEASE packet, it discards the packet.

discussion

Loop–free path. When a $PROB$ packet reaches a node that another $PROB$ packet for the same connection request arrived before, the second $PROB$ packet is rejected if it cannot provide better QoS. This early–reject policy ensures that PROB packets will not loop forever and that the QoS paths found by the protocol are loop–free. Specifically, When a loop is formed, a PROB packet will reach a node two times. When the second time the PROB packet is processed at the node, it will be rejected since it cannot provide better QoS than the first time it arrived at the node.

Maximum number of outstanding backward reservations. This number, bn, is an important parameter for the backward reservation protocol. When $bn = 1$, the protocol does not overly reserve. When $bn > 1$, the over–reservation problem may occur when the destination sends two or more RES packets for one connection request. Large bn may improve the blocking probability for each individual connection request.

Advantages and limitations. The backward reservation protocol uses the imprecise global network state information to probe the network on demand for a connection request and uses the precise network state information obtained from the on–demand probing for resource reservation. This greatly improves the resource reservation efficiency and reduces the over–reservation problem comparing to the forward reservation protocol. In addition, by reducing the over–reservation problem, the global network state information maintained by the link state algorithm is more precise since resources are less likely to be in the transient state. This further improves the routing efficiency. The backward reservation protocol also has limitations. First, since resource probing and resource reservation are done at different time, there are chances that resource availability changes and resource reservation fails after the path is probed. Second, on average, it takes the backward reservation protocol longer time to reject a request than it takes the forward reservation protocol. Both the probing phase and the reservation phase in the backward reservation scheme can result in failures. The failures in the probing phase is similar to the failures in the forward reservation scheme except that no resources are reserved. However, failures in the reservation phase mean that it will take the source at least the round–trip delay time to notice that the path establishment attempt has failed. When the network is under heavy load, this can occur frequently and the system throughput will suffer. Third, the backward reservation scheme incurs more messaging overheads, which affects the path establishment time since the load in the control network in higher. All these factors will affect the performance of the protocol, the overall effect will be studied through simulation in the evaluation section.

5 Hybrid scheme

Both the forward reservation protocol and the backward reservation protocol have their advantages and limitations. To combine their advantages and overcome their limitations, we develop a hybrid resource reservation scheme. The idea of the hybrid resource reservation scheme is to perform resource reservation in the forward pass along a limited number (ft) of paths that have the highest successful reservation probability and to probe the rest $t - ft$ paths, where t is the number of tickets for each request. If the forward reservation successfully establishes the path, the destination node can just accept the connection and ignore the results of the probing. In this case, both the problems in the backward scheme and the over reservation problem in the forward scheme are under control. If the forward reservation fails, the hybrid scheme resorts to the results of the probing and tries to reserve resources following the backward reservation protocol. In this case, the hybrid scheme behaves similar to the backward scheme. Thus, the hybrid scheme uses backward reservation as its base and tries to explore the advantages of the forward reservation scheme. Next, we will discuss the control packets used in the hybrid scheme and the operations in different nodes. Our hybrid protocol requires the routers to maintain the same information as the backward reservation protocol discussed in the previous section. The hybrid protocol uses the following control packets.

- FRES (forward reservation) packet. The FRES packet probes and reserves resources along the path towards the destination. It contains the connection ID, the number of tickets, the number of tickets for forward reservation, the QoS requirement, the current QoS and PATH information. The FRES packet is denoted as

$FRES((src, dst, seq), ticket, rticket, QoS, cQoS, PATH)$.

- PROB packet. This control packet is the same as the PROB packet in the backward reservation scheme. It travels towards the destination probing the paths that satisfy the QoS requirement. The PROB packet is denoted as $PROB((src, dst, seq), ticket, QoS, cQoS, PATH)$.

- BRES (backward reservation) packet. This packet is the same as the RES packet in the backward reservation scheme. This packet performs resource reservation from the destination to the source. A BRES packet is denoted as $BRES((src, dst, seq), ticket, QoS, cQoS, PATH)$.

- *CONFIRM* packet. This packet informs the intermediate nodes and the destination that the connection has been established. The information in this packet includes the connection identifier and the path. A CONFIRM packet will be denoted as $CONFIRM((src, dst, seq), PATH)$.

- REJECT packet. This is the same as the REJECT packet in the backward reservation scheme. It informs the source that a reservation attempt has failed. A REJECT packet is denoted as $REJECT((src, dst, seq), ticket, PATH)$.

- FAIL packet. This packet is the same as the FAIL packet in the backward reservation scheme. It informs the intermediate nodes that the reservation attempt has failed, and that the resources locked may be released. A FAIL packet is denoted as $FAIL((src, dst, seq), PATH)$.

- REJECT/FAIL packet. This packet combines the functionality of the REJECT packet and FAIL packet. It travels from the intermediate nodes towards the source to inform the source and the intermediate nodes of the failure of the reservation. The intermediate node may release the resources reserved for the connection when receiving the REJECT/FAIL packet. The REJECT/FAIL packet is denoted as $REJECT/FAIL((src, dst, seq), ticket, PATH)$.

- ACCEPT packet. This packet accepts forward reservation and informs the source about the path with all resources reserved. In this protocol, ACCEPT packets do not change the link state to be USED since ACCEPT packets can be rejected. The ACCEPT packet is denoted as $ACCEPT((src, dst, seq), PATH)$.

- *RELEASE* packet. This packet releases the network resources when the transmission is done. This packet contains the connection identification and the path for the connection. The RELEASE packet will be denoted as $RELEASE((src, dst, seq), PATH)$.

Next, we will describe the operations in the source, the intermediate and the destination nodes. Most of the control packets used in the hybrid scheme are processed in the same way as their corresponding control packets in the forward [16] and backward schemes. Rather than describing all the operations, we will focus on the differences when processing these packets.

- *Source node.* When the source node wants to establish a connection, it constructs a FRES packet with t tickets, ft tickets for forward reservation and the QoS requirement of the connection. The source sends the FRES packet towards the destination. The source then waits for ACCEPT, BRES, REJECT, and REJECT/FAIL packets. When an ACCEPT packet or a BRES packet reaches the source node, the path is established. The source changes the status of the connection to be established, sends a CONFIRM packet to the destination to inform the establishment of the connection, and starts sending data. When the source node finishes sending data, it sends a RELEASE packet to release the resources for the connection. Once a connection for a request has been established, the source node will reject all ACCEPT, FRES, BRES and PROB packets for the request. When receiving a REJECT packet, the source computes the number of tickets rejected for the request. When receiving a REJECT/FAIL packet, the source computes the number of tickets rejected and releases the resources locked if necessary. When the total number of tickets rejected is equal to the number of tickets issued, the source node knows that the path establishment attempt has failed and it can try the attempt at a later time.

- *Intermediate nodes.* The processing of the control packets at the intermediate nodes in this protocol is mostly the combination of the processing of the control packets in the forward protocol [16] and that in the backward protocol. However, two types of packets, the FRES packet and the ACCEPT packet, are processed differently. When an intermediate router receives a FRES packet, it first checks whether this packet should be rejected. Among other reasons, when the router finds that the connection has been established or that the network does not have sufficient resources to support the QoS requirement of the connection, the router will reject the FRES packet. If the FRES packet is not rejected, the router will use the routing and ticket distribution algorithm to distribute tickets for all the outgoing links that may satisfy the QoS requirement and generate FRES packets or PROB packets for different next hops based on the number tickets (*rticket*) for forward reservation in the FRES packet and the ranking of the next hops. The next hops with higher probability of successful path establishment will be sent FRES packets, while the next hops with lower probability of successful path establishment will be sent $PROB$ packets. Details about this process are shown in Figure 3. When the router receives an ACCEPT packet, the router only forwards the packet towards the source without changing the link state since the ACCEPT packet may be rejected in this protocol.

When node N receives $FRES((src, dst, seq), ticket, rticket, QoS, cQoS, PATH = \{N_1 \rightarrow N_2 \rightarrow ... \rightarrow N_i\})$
    Let us assume that the FRES packet is not rejected.
    Let nodes M_1, ..., M_j be all the possible next hops in the decreasing probability order.
    Let t_1, ..., t_j be the corresponding ticket distribution for each node
    Compute $cQoS_1$, ..., $cQoS_j$ for each of the next hops.
    Let l be the smallest integer such that $\sum_{a=1}^{l} t_a \geq rticket$ and $\sum_{a=1}^{l-1} t_a < rticket$
    FOR k=l+1 **TO** j
        send PROB((src, dst, seq), ticket, QoS, $cQoS_k$, $PATH' = \{N_1 \rightarrow N_2 \rightarrow ... \rightarrow N_i \rightarrow N\}$) to M_k.
    send FRES((src, dst, seq), t_l, $rticket - t_1 - ... - t_{l-1}$, QoS, $cQoS_l$, $PATH'$) to M_l.
    FOR k=1 **TO** $l-1$
        send FRES((src, dst, seq), t_k, t_k, QoS, $cQoS_k$, $PATH' = \{N_1 \rightarrow N_2 \rightarrow ... \rightarrow N_i \rightarrow N\}$) to M_k.

Figure 3: Processing a FRES packet

- Destination node. When the destination node receives a PROB packet, it checks whether the connection has been established. If the connection has been established, the PROB packet is rejected. If the number of BRES packets sent for this connection request is equal to bn, the router rejects the PROB packet. Otherwise, a BRES packet is constructed and sent to the source. When the destination node receives a FRES packet, it checks whether the connection has been established. If the connection has been established, the FRES packet is rejected. Otherwise, an ACCEPT packet is sent to the source.

One important parameter for the hybrid protocol is the number of tickets for forward reservation (ft). This parameter determines the number of paths that will be reserved in the forward reservation manner. Let t be the total number of tickets for a request. When $ft = t$, the hybrid protocol is reduced to the forward reservation protocol. When $ft = 0$, the hybrid protocol becomes the backward reservation protocol. However, due to the overhead when combining the two reservation schemes into one protocol, the reduced hybrid protocols perform slightly worse than the pure forward or backward reservation protocols.

6 Experiments

To evaluate and compare the performance of the protocols that combine different reservation schemes with the ticket–based QoS routing scheme, we develop a cycle–by–cycle network simulator that simulates all three protocols. We use bandwidth as the QoS metric for the experiments. The major features of the network simulator are the followings.

- The simulator simulates the QoS routing protocols that combine three resource reservation schemes with the ticket–based QoS routing algorithm. By manipulating the number of tickets for each request, the simulator can simulate flooding based routing schemes that do not use the global network state information to make routing decisions, multi–path QoS routing schemes which 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 parameterizes the following protocol, network and traffic parameters:

  - *reservation scheme.* The user can choose to simulate forward reservation protocols, backward reservation protocols or hybrid reservation protocols.
  - *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) of each connection. The connection requests arrive at each node following

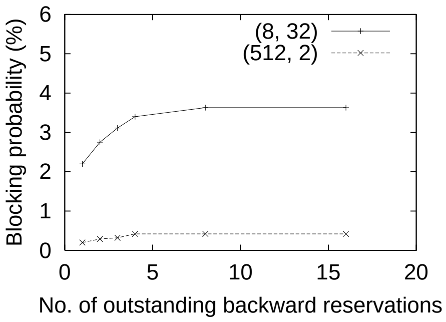
a Poisson distribution with an average of r messages generated every time unit. When a request is generated at a node, the destination of the request is generated randomly among the other nodes in the system. A request is discarded when it is blocked.

- *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 and determines the precision of the global network state information used by the routing algorithm.
- *number of tickets* (t) for each request. This number decides the maximum number of paths to be probed.
- *forward reservation tickets* (ft) for each request. This parameter is used only in the hybrid reservation scheme to decide the number of forward reservations for each request.
- *the number of outstanding backward reservations allowed* (bn). This parameter is used in the backward and hybrid protocols. This parameter limits the number of outstanding backward reservations for each connection request.
- *control packet processing speed* (ct). This parameter specifies how fast a router can process the control packets it received.

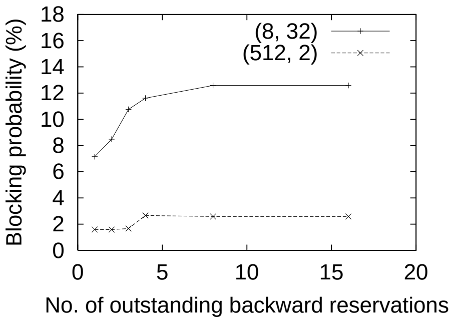
We use two performance metrics, the blocking probability and the average number of connections granted, to evaluate the protocols. The blocking probability is the ratio of the number of connection requests blocked and the number of connection requests processed. Under light traffic, both the blocking 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.

Many factors can affect the performance of the protocols, we tried to isolate the factors by using the following four different traffic patterns to compare the performance of the protocols.

- *Long connection duration with small bandwidth requirement.* In this traffic pattern, the global network state information is relatively accurate while the over reservation problem is not as severe as other cases. We generate such traffic pattern by letting $d = 512$ and $b = 2$ and denote such pattern as $(512, 2)$.
- *Short connection duration with large bandwidth requirement.* This traffic pattern represents the cases where the global network state is extremely imprecise and the over–



(a) Low load ($r = 0.007$)



(b) Medium load ($r = 0.021$)

Figure 4: The impact of the number of outstanding backward reservations

reservation problem is very severe. We generate such traffic pattern by letting $d = 8$ and $b = 32$ and denote such pattern as $(8, 32)$.

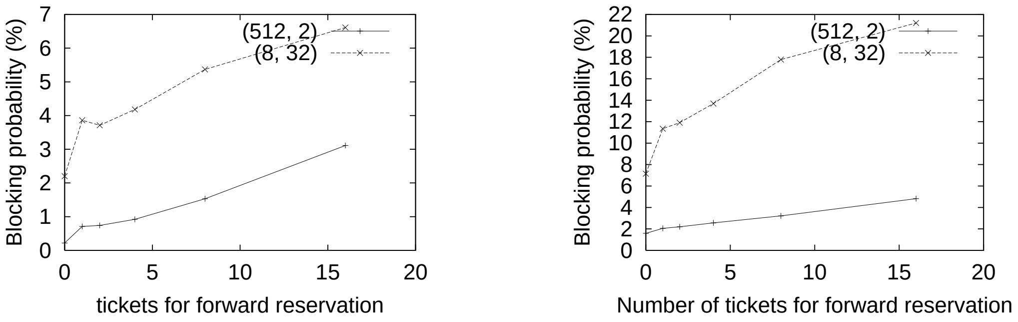
- *Long connection duration with large bandwidth requirement.* This patterns can easily result in high network load and severe over–reservation problems. The precision of the global network state information is also affected by the large amount of bandwidth in the transient state. We generate such traffic pattern by letting $d = 512$ and $b = 32$ and denote such pattern as $(512, 32)$.

- *Short connection duration with small bandwidth requirement.* In this pattern, the over–reservation problem is not very severe while the global network state is imprecise. We generate such traffic pattern by letting $d = 8$ and $b = 2$ and denote such pattern as $(8, 2)$.

Among the four traffic patterns, patterns $(512, 2)$ and $(8, 32)$ are two extremes and patterns $(512, 32)$ and $(8, 2)$ are in between the two extremes. Next, we first study the protocol parameters and identify the protocol parameters in the backward and hybrid protocols that can result in the best performance of the protocols, then compare the performance of the protocols.

Protocol parameters

Figure 4 studies the impact of the maximum number of outstanding backward reservations, bn, on the performance of the backward reservation protocol. This experiment is done on 6×6 meshes and assumes $uf = 500$ and $t = 16$. The figure shows the results for the two extreme traffic patterns $(512, 2)$ and $(8, 32)$ with different network load. We have also studied other traffic patterns and other network load, the results exhibit similar trend. From the figure, we can see that for all the cases, allowing more backward reservations decreases the performance. There are two reasons for this. First, the probability that the first backward reservation will be successful is very high since the chance that the resource availability changes during the round trip delay is small. Thus, allowing more outstanding backward reservations creates the



(a) Low load ($r = 0.007$)

(b) Medium load ($r = 0.021$)

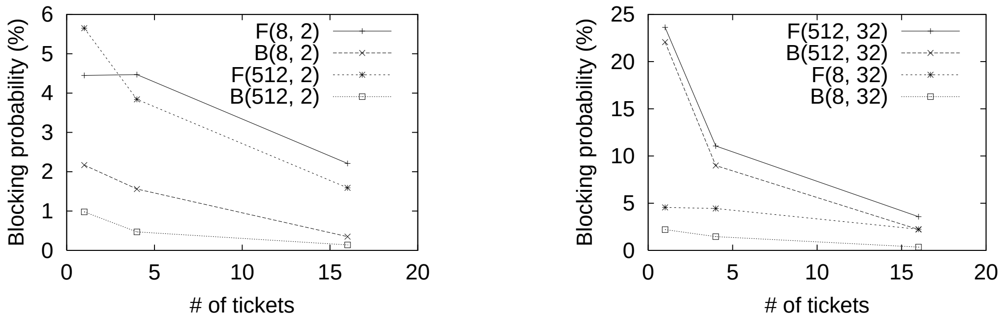
Figure 5: The performance of the hybrid schemes

over-reservation problem without improving the successful reservation probability for a given request. Second, limiting the number of backward reservations to one reduces the amount of resources reserved but not used, which improves the precision of the global network state information and the routing efficiency. Since $bn = 1$ always results in the best performance for the backward reservation schemes, we will set $bn = 1$ for the rest of the evaluations.

Figure 5 shows the performance of the hybrid protocols with different number of tickets for forward reservation. This experiment is done on 6×6 mesh and assumes $uf = 500$ and $t = 16$. The figure shows the results for the two extreme traffic patterns $(512, 2)$ and $(8, 32)$ with different network load. We have also studied other traffic patterns and other network load, the results exhibit similar trend. From the figure, we can see that for all the cases, allowing more tickets for forward reservations increases the blocking probability. Allowing 0 ticket for forward reservation, which effectively reduces the hybrid protocol to the backward reservation protocol achieves the best performance. The result is consistent for all the settings in the experiment, which indicates that our intuition in the design of the hybrid reservation protocol was wrong. Although the hybrid reservation scheme overcomes some problems in the backward reservation scheme, it also introduces other inefficiencies in resource reservation. For example, due to the imprecise global network state, the likelihood that the forward reservations successfully establish the path is small. The failed forward reservations result in the over reservation problem and decrease the precision of the global network state information. All these factors can affect the performance of the routing algorithm. As shown in Figure 5, the overall effect is that allowing forward reservation decreases the routing performance in comparison to the pure backward reservation scheme.

Forward reservation .vs. backward reservation

The study of the protocol parameters establishes that the backward schemes are more efficient than the hybrid schemes with the same number of tickets and that the backward scheme



(a) Load bandwidth requirement ($b = 2$)

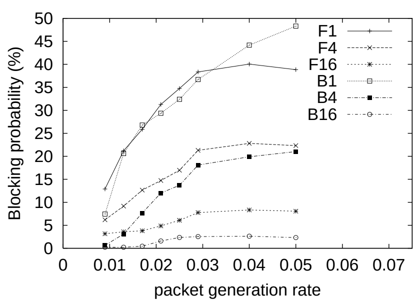
(b) High bandwidth requirement ($b = 32$)

Figure 6: The performance of the protocols when the network is under low load

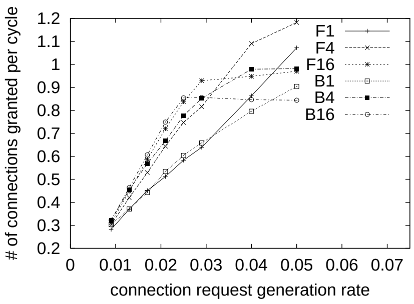
achieves the best performance when $bn = 1$. In the rest of the section, we will compare the two representative resource reservation schemes, the forward reservation scheme and the backward reservation scheme since the forward reservation scheme is a well–established resource reservation scheme [4, 7, 8, 11, 16]. We will use the letters B and F to represent the backward reservation protocol and the forward reservation protocol respectively. When other simulation parameters are fixed, we will use $B\ t$ and $F\ t$ to represent the backward and forward schemes with t tickets respectively, $B\ (pattern)$ and $F\ (pattern)$ to represent using the backward scheme and the forward scheme to deal with the traffic pattern $(pattern)$ respectively, and $B\ network$ and $F\ network$ to represent running the backward scheme and the forward scheme on $network$ respectively. For example, $B4$ represents the backward scheme with 4 tickets. $B(512, 2)$ represents using the backward scheme to deal with traffic pattern $(512, 2)$. $B10 \times 10$ represents running the backward scheme on a 10×10 mesh.

Figure 6 compares the performance of the protocols when the network is under light load. This experiment is done on 6×6 mesh and assumes $uf = 500$ and $r = 0.001$. For all the four traffic patterns, backward reservation schemes result in significantly better blocking probability than their forward reservation counterparts. Reducing the over–reservation problem and improving the precision of the global network state information greatly improve the communication performance when the network is under light load.

Figures 7 and 8 show the cases when the network load is higher. These experiments are done on 6×6 mesh and assumes $uf = 500$. For different traffic patterns, the backward and forward protocols behave differently. Figure 7 shows the case for large connection duration and small bandwidth requirement. In this case, the global network state is relatively precise. The behave of the backward reservation protocol is similar to that of the forward reservation protocol. When the network is not saturated, the backward protocols offer lower blocking probability and higher throughput since they avoid the over-reservation problem and have more precise global network state information. When the network is approaching saturation, the forward protocols grant more connections than the corresponding backward protocols do,

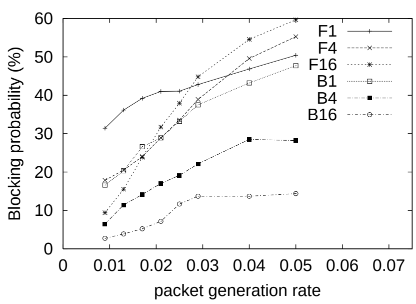


(a) Blocking probability

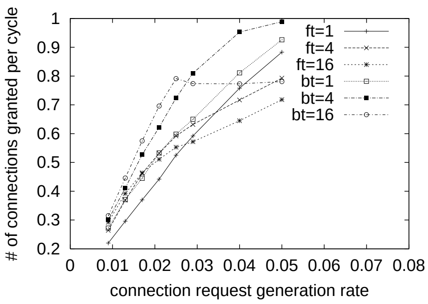


(b) Number of connections granted per cycle

Figure 7: Performance of the forward and backward protocols for $(d = 512, b = 2)$



(a) Blocking probability



(b) Number of connections granted

Figure 8: Performance of the forward and backward protocols for $(d = 8, b = 32)$

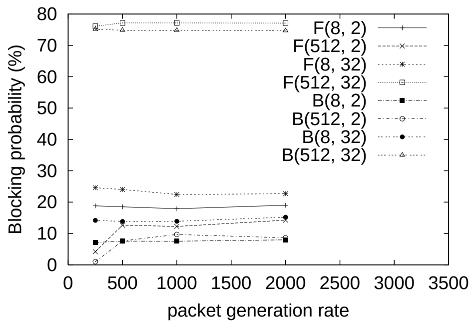


Figure 9: Impact of link state update frequency on the 4–ticket protocol

even though they suffer from the over–reservation problem. Before we discuss the reasons for this phenomenon, let us first consider another observation in this experiment. From the figure, we can see that for both forward reservation and backward reservation, protocols with a larger number of tickets always have lower blocking probability and lower maximum throughput when the network is close to saturation. The reason is that protocols with a larger number of tickets take longer time to process each request. Such protocols explore more paths to increase the probability of successful path establishment for each individual request at the cost of longer path establishment time especially when the network is under heavy load. The reason that backward reservation protocols have lower blocking probability and lower system throughput in comparison to the corresponding forward reservation protocols is similar. The backward reservation protocols also process each request slower. The backward reservation protocols reject a request slower since they allow the PROB packets to reach the destination without successfully establishing the path. This happens quite frequently when the network is close to saturation. In addition, the backward reservation protocols also result in more control packets in the control network, which slows down the path establishment. Thus, when the bandwidth requirement is small and the over–reservation problem in the forward reservation schemes is not very severe, the backward reservation schemes have a lower blocking probability and a lower system throughput when the network is close to saturation as shown in this experiment.

Figure 8 shows the case when the connection duration is small and the bandwidth requirement is large. In this case, the reduction of the over–reservation problem by the backward reservation schemes has great impact on the performance. As can be seen from the figure, the backward schemes achieve both lower blocking probability and higher system throughput comparing to their forward reservation counterparts. Notice that for this setting, the behavior of the forward reservation schemes and that of the backward reservation schemes are different. Forward reservation protocols with a larger number of tickets have lower blocking probability when the request generation rate is low and higher blocking probability when the request generation rate is high. The over–reservation problem in the forward protocols with large numbers of tickets manifests itself in this experiment. In contrast, backward reservation protocols with a larger number of tickets always have lower blocking probability. Since we set $bn = 1$, over–reservation is not a problem in the backward reservation schemes. Increasing number of tickets only increases the load in the control network and the time to process each request. It does not incur the over–reservation problem.

Figure 9 and Figure 10 study the impact of the link state update frequency. The impact of

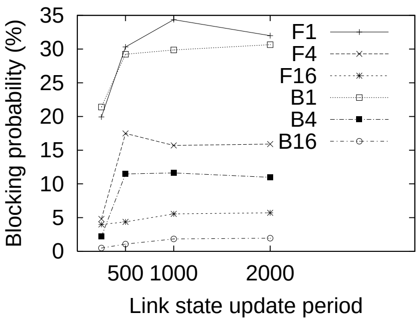


Figure 10: Impact of link state update frequency for pattern $(512, 2)$

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 9 shows the impact of link state update frequency on 4–ticket routing protocols. 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. The trend for the backward reservation protocols is very similar to that for the forward reservation protocols. The link state update frequency makes difference only for pattern $(512, 2)$. However, backward reservation consistently has lower blocking probability for all patterns and all link state update frequencies.

Figure 10 further studies the pattern $(512, 2)$ and shows the impact of link state update frequency on protocols with different number of tickets. This experiment is done on 6×6 meshes and assumes $r = 0.02$. In the experiment, the trend for the backward reservation protocols is very similar to that for the forward reservation protocols. For both forward and backward protocols, the link state update frequency greatly affects the performance of the single path routing ($t = 1$) and 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 with either forward reservation or backward reservation. The results also show that the backward protocols provide better performance than their forward counterparts for different link state update periods.

Figures 11 shows the impact of network size on the protocols when the network is under light load (r=0.009). This experiment assumes $d = 128$ and $b = 2$. For both forward reservation and backward reservation, under light load, protocols with more tickets perform better than the protocols with less tickets for all the network sizes. Our other experiment indicates that protocols with large number of tickets do not scale with respect to the network size. A 16–ticket protocol with either forward reservation or backward reservation will saturate the control network for an 8×8 mesh when $r = 0.02$ while protocols with 1 or 4 tickets will not saturate the network. However, as can be seen from the figure, backward reservation provides better performance for different network sizes when the networks are not saturated.

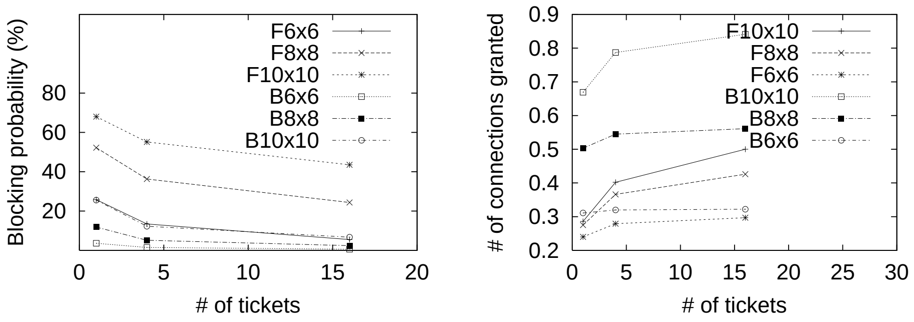


Figure 11: Impact of network size

7 Conclusion

In this paper, we develop multi–path QoS routing protocols that combine the ticket–based routing algorithm with the backward reservation scheme and with the hybrid reservation scheme, evaluate the protocols through extensive simulation, and compare these newly developed protocols with a more established forward resource reservation scheme. Our major conclusions are the followings. First, the backward reservation protocols achieve the best performance when the maximum number of outstanding backward reservations is equal to 1. Second, the backward reservation schemes perform better than the hybrid reservation schemes with the same number of tickets. Third, while the backward reservation schemes behave similar to their forward reservation counterparts for different network settings, the backward schemes consistently offer lower blocking probability when the network is not saturated.

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 (ITC'16)*, Edinburgh, United Kingdom, June 7-11, 1999.

[2] S. Chen and K. Nahrstedt, "Distributed QoS routing with Imprecise State Information." *ICCCN'98*, 1998.

[3] 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.

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

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

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

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

[8] H.K. Pung, J. Song and L. Jacob, "Fast and Efficient Flooding Based QoS Routing Algorithm", *IEEE International Conference on Computer Communications and Networks (IC3N'99)*, October, 1999.

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

[10] 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*, Nov. 1997.

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

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

[13] X. Yuan, R. Melhem, R. Gupta "Distributed Path Reservation for Multiplexed All-Optical Networks", *IEEE Transaction on Computers*, Volume 48, No. 12, pages 1355-1363, Dec. 1999.

[14] X. Yuan, "On the Extended Bellman–Ford Algorithms to Solver Two–Constraints Quality–of–Service Routing Problems", *IEEE IC3N*, 1999.

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

[16] Y. Zhong and X. Yuan, "Impact of Resource Reservation on the Distributed Multi–path Quality of Service Routing Scheme", *The Eighth International Workshop on Quality of Service (IWQoS2000)*, Pittsburgh, PA, June 2000.