An Edge Router Based Protocol for Fault Tolerant Handling of Advance Reservations

Sudeept Bhatnagar and Badri Nath

Dept. of Computer Science

Rutgers University

110 Frelinghuysen Road

Piscataway, NJ 08854-8019

{sbhatnag,badri}@cs.rutgers.edu

Abstract

To provide quality of service guarantees, resource reservation schemes have to maintain reservation states at the nodes along the path of a flow. Advance reservation schemes have to maintain these states for a long period of time. The loss of reservation state caused by node failures, makes advance reservation schemes highly susceptible to node failures. In this paper, we argue that a domain-by-domain reservation approach is inherently more fault tolerant than the traditional hop-by-hop approach. We propose a novel domain-based protocol for handling advance reservations. It requires support only at the edge routers and no changes are required at the core routers. We describe various factors which determine whether a flow's advance reservation is handled properly by a network in event of router failures. We use simulations to show that our protocol is highly fault tolerant. Our protocol allows the best effort flows to utilize the remaining bandwidth without any static partitioning of bandwidth among reserved and best effort flows.

I. Introduction

Internet in its present form, provides a single class of service, namely the best effort service. The routers do not maintain any per-flow states and forward all the packets without catering to their specific quality requirements. This simple model has lead to the robustness and scalability of the Internet. The advent of applications like video conferencing and video-on-demand has fueled the need to have quality of service (QoS) guarantees in the Internet. Over the years, researchers have proposed different schemes to the alter the existing Internet framework to provide resource reservation. The prominent resource reservation mechanism is *immediate reservation*, where an application specifies its resource requirements just prior to using it. An alternate paradigm for resource reservation is *advance reservation* where a reservation request specifies the user's intent to use a certain amount of reserved resources, at sometime in future.

New features have to be introduced in the architecture to support reservations. These include signaling mechanisms to specify requirements [1], [2], specialized flow handling [3], [4], admission control and QoS routing [5], [6]. The immediate reservation models like Integrated Services [7], [8], Stream Protocol [2] and Core-Stateless guarantees [9], [10]

require changing *all* the routers in the Internet to support QoS. The integrated services models mandate all the routers to maintain reservation states. The core-stateless model frees the core routers from the burden of state maintenance but requires the core routers to do per-packet processing. The advance reservation models [11], [12], [13] have primarily been extensions to the integrated services model. In other words, no matter what reservation models are used, all the routers in the Internet have to be modified to provide QoS guarantees and thus the deployment complexity of these approaches remains high.

Reserving resources in advance introduces another complexity factor – time. For advance reservations, the flow classification and the policing mechanisms remain similar to those for immediate reservations. The signaling mechanism to specify the resource requirement for advance reservation, should provide a way to specify *when* the reservation will be availed. [11], [12] describe extensions to the existing immediate reservation signaling mechanisms to specify advance reservation requests. For admission control and QoS routing, the immediate reservation algorithms need to be modified to look at the available resources at the time when the reservation will be used rather than the resources available at present. For this, the network elements must maintain resources available during each of the time slots in future.[1] Guerin and Orda [14] describe the mechanisms to use for QoS routing for advance reservation capable networks.

However, probably the most important source of complexity introduced in handling advance reservations is the fact that the routers have to maintain the reservation state of each flow for a long period of time. A router failure may cause the loss of all reservation state information. It has been observed in [15] that the mean-time-to-failure for an interface in a network is around 40 days and 25% of these are caused by power failure, hardware failure and software failure. Under all these circumstances, the loss of state information is highly probable. Since advance reservation requests can be issued many days or months prior to usage, fault-tolerance is a prime issue in designing any solution for providing advance reservation. The issue of fault-tolerance in advance reservation capable networks, has been left largely untouched.

Wolf and Steinmetz [13] classify the types of failures based on when the failure occur:

1. During the negotiation and reservation phase
2. After the reservation but before the usage
3. During the usage

The first and the third cases are same as those in immediate reservations. However, the second case is specific to advance reservations. If we can mitigate its effect, we shall be able to handle advance reservations in a manner no different than immediate reservations.

In this paper, we describe a novel *edge-to-edge* protocol to handle advance reservations. The prime feature of our protocol is that it is implemented entirely in edge routers. The core routers of the domain do not require *any* modifications. It follows the spirit of core-stateless networking and maintains states only at edge routers. It significantly deviates from

[1] In practice, the maximum number of time slots would be limited by the ISP.

the Dynamic Packet State based core-stateless networking by removing the burden of per-packet processing from the core routers. We believe that such a scheme is more scalable and easier to deploy than any other proposed schemes which require modifications to all the routers of each domain in the Internet. We study various parameters which contribute to a flow being mishandled, in presence of network faults. We show that our protocol is highly fault tolerant. It eliminates the effects of router failure between the negotiation phase and the usage phase of an advance reservation and thus allows us to treat immediate and advanced reservations on an equal plane.

The paper is organized as follows: In section 2 we describe the our basic network model and the terminology used in the paper. We argue the case for domain-based reservation in section 3. The EQOS protocol supporting advanced reservations is described in section 4. We evaluate various factors which determine whether or not an advance reservation flow can be handled by a network. The EQOS protocol and hop-by-hop approach are evaluated in light of these factors. Section 6 defines the EQOS-A protocol which attains both our objectives of providing strict guarantees with support required only at edge routers and achieving a high degree of fault tolerance. Section 7 is a discussion section which answers some questions about the capabilities of EQOS-A and finally we conclude by mentioning future work in section 8.

II. Model & Terminology

We assume a network to be a collection of edge and core routers. The edge routers are the ingress and egress points for the traffic using the network. The core routers switch the traffic among the routers of the network. We assume that some route-pinning mechanism like MPLS [16] or IP source routing is deployed in the network. Our protocol uses these mechanisms for intra-domain route pinning.

Although our protocol can easily be modified to work with any signaling mechanism, in this paper we use route-pinned version of RSVP as described in [12]. Our protocol provides bandwidth guarantees to flows. In case the application requires other types of QoS guarantees like delay or jitter, the request can be converted into the equivalent bandwidth guarantee at the ingress router. We believe that the ISPs would primarily be interested in providing bandwidth guarantees only. The main reason behind this belief is that pricing for bandwidth usage is easier than that for any other type of QoS guarantee. For example, it is intuitively difficult to set a price for having a guaranteed jitter value.

We focus on handling advance reservation with known duration. More specifically, the reservation request contains information about when the reservation usage starts and when it ends. Each flow is characterized by the tuple (R, S, T, I, E) where,

$R = Reserved\ Bandwidth$

$S = Flow's\ Start\ Time$

$T = Flow's\ Termination\ Time$

$I = Ingress\ Router$

$E = Egress\ Router$

We divide a flow's activity into three phases: 1)The *negotiation phase* during which the flow negotiates with the network and at the end of which its reservation state is installed in the network. 2) The *storage phase* during which the network stores the reservation state of the flow. This state may be periodically refreshed according to RSVP protocol. We assume that the frequency of updates increases as the flow's start time nears. 3) The *usage phase* when the flow uses its reservation. The usage phase starts at time $S - \delta$ and ends at $T + \delta'$ where δ and δ' are time margins provided by the ISP. A reserved flow is considered *active* during its usage phase.

Since our objective is to eliminate the effects of router failure during the *storage phase*, we consider a flow to be *handled properly*, if its reservation state is intact during the storage period. For a flow's reservation to be effective, a minimal set of nodes must have its reservation state available at the end of its storage phase. For example, for a hop-by-hop approach, this set of nodes would comprise of all the nodes on the path of the flow. In other words, a flow is handled properly if its reservation state is available at this minimal set of nodes, when it starts.

We do not consider the effect of router failures during negotiation phase or during the usage phase because these can be handled identically to immediate reservations. A soft state signaling mechanism ensures that a partial state does not remain at a router in case negotiation fails during the negotiation phase. For failure during usage phase, defining alternate routes is a simple option. Alternate routes can be done on a pro-active basis (using survival routing) or in a reactive manner. This paper's prime focus is to eliminate the effect of router failures during storage phase.

Each edge router stores the future network state information to compute routes for flows as described in [14]. In particular, for each link l in the domain, a vector $B_l = b_l[0], b_l[1], ...$ is stored where $b_l[i]$ indicates the amount of bandwidth available on link l in time slot i. Collectively, we denote the network state information by $B = \{B_i : i \in LinkSet\}$.

III. Domain-based Reservation

In this section, we argue that a domain-by-domain reservation scheme makes more sense than the traditional hop-by-hop approach to handle advance reservations. We argue that the degree of fault-tolerance inherent in a domain-by-domain scheme, can not be bettered by a hop-by-hop approach, when they operate on similar bases. By similar bases we mean that the two approaches use similar algorithms (for routing, admission control, etc.), have similar failure probabilities and the traffic parameters (like average storage phase duration, fraction of traffic between a given pair of routers) are similar.

Let the probability of router failure be uniform across all the nodes of a domain, and the probability $p(t)$ be a non-decreasing function of a router's uptime t. Also, let the set of routers along the path of a flow f inside the domain be

$r_1, r_2, \ldots, r_n$ which have stayed up from times $t_1, t_2, \ldots, t_n$ respectively. We consider the effect of node failures on the probability that f is handled properly.

In the hop-by-hop approach the flow's advance reservation request would be processed by a router using some admission control algorithm and then forwarded to the next router downstream to do a similar processing. Assuming that the reservation request is granted, each router along the path has to store the reservation information about the flow till the flow starts. This information would include the five-tuple for the flow (R, S, T, I, E), and the identity of the next hop for the flow. Now if one of the router goes down before S, the reservation information at that router is lost. The lost information includes the identity about the next hop for f and that in turn implies that the path that the flow was supposed to follow from its source to its destination is disconnected due to loss of information. In fact, if the signaling mechanism is hard-state based, the other routers will maintain the state for f when it is not likely to be able to use this reservation at all.

The probability that the state information for f remains intact is the same as the probability that no router on its path fails till time S (when the flow starts). This probability is $\prod_{i=1}^{n} [1 - p(S - t_i)]$.

An alternate approach is the domain-by-domain approach in which the reservation request is processed only once for each domain. Lets assume that when an advance reservation request arrives, some node(say a bandwidth broker node b) in the domain, processes the request for the entire domain, computes the path that the flow should take through the domain, informs the flow's ingress router its path and then passes the reservation request to the next domain. Now the required reservation state for a flow is stored at only one node in the domain. Given that the node failure probability in the domain is $p(t)$, and that the node has been up since time t_b, the probability that the node storing the reservation state does not fail before S is given by $1 - p(S - t_b)$. A simple comparison shows that for a domain based approach to be better:

$$\prod_{i=1}^{n} [1 - p(S - t_i)] < 1 - p(S - t_b)$$

Or

$$p(S - t_b) < 1 - \prod_{i=1}^{n} [1 - p(S - t_i)]$$

Ignoring higher order terms

$$p(S - t_b) < \sum_{i=1}^{n} p(S - t_i)$$

Since p(t) is a non-decreasing function of time with values ranging between 0 and 1, the domain based approach is likely to lose to the hop-by-hop approach only in those cases where the broker node has been up for a long time (and is likely to fail in near future) or when all the nodes on the path of the flow have just recently restarted. On an average the node failure probability for a single node will be much lower than the probability that no node in a given set fails. In other words, the domain-based approach is likely to fare much better than a hop-by-hop approach in handling advance

reservation state.

A common argument that is given to make a system fault-tolerant is to incorporate information backup devices. Even on that score a domain-based scheme is better because only the nodes which store reservation information need to be backed up and in the hop-by-hop approach all the nodes on the path need to be replicated. It is very likely that the number of nodes storing reservation information for the domain will not be more than the total number of routers in the domain.

Finally, a domain based approach cannot perform any worse than a hop-by-hop approach for *each* flow, if the node on which the reservation state is stored is also on the path of the flow (as suggested by the last inequality). We store a flow's state only on its ingress router (which is also on its path).

IV. EQOS Protocol

In [17], the authors have proposed a protocol called EQOS to handle immediate reservation requests entirely at the edge routers. In EQOS, the edge routers coordinate using a token-passing mechanism. The token carries the information about the available bandwidth on each link in the domain and the number of best-effort flows on each link. The protocol's mechanism to handle the immediate reservation flows and the best effort flows are based on the information in the token. We describe the relevant portions of the protocol here. For a complete discussion and analysis the reader is referred to [17].

In the previous section, we have argued that a per-domain approach is more suitable to design a fault-tolerant network to handle advance reservations than the hop-by-hop approach. EQOS being a domain-based approach is a suitable candidate for advance reservations. Certain straight forward enhancements can be made to allow it to handle advance reservation requests. We describe these modifications to the protocol which will provide bandwidth guarantees to advance reservation flows and allow dynamic bandwidth sharing among best effort flows and advance reservation flows.

A. Processing Reservation Request

When an *RSVP-path* message for advance reservation arrives from a source at an ingress router, it stamps its own address in the message and forwards it towards its destination. Since the core routers are unaware of RSVP packets, they just forward the message. The egress router receiving the path message notes the ingress router address from the path message and forwards it upstream. On receiving a *RSVP-resv* message the egress router looks at the flow specification and its stored state and sends the packet to the corresponding ingress router using some route pinning mechanism. The core routers just forward the packet toward the ingress router. On receiving the *RSVP-resv* message, the ingress router converts the QoS requested into equivalent bandwidth requirement and notes the characteristic tuple of the flow (R, S, T, I, E). The ingress router should not go ahead and compute a route for this flow immediately because it might be possible that some other edge router also received a request at the same time. The routes computed by the two routers can overlap resulting in over-allocation of bandwidth on some links. An over-allocation would lead to the guarantees to

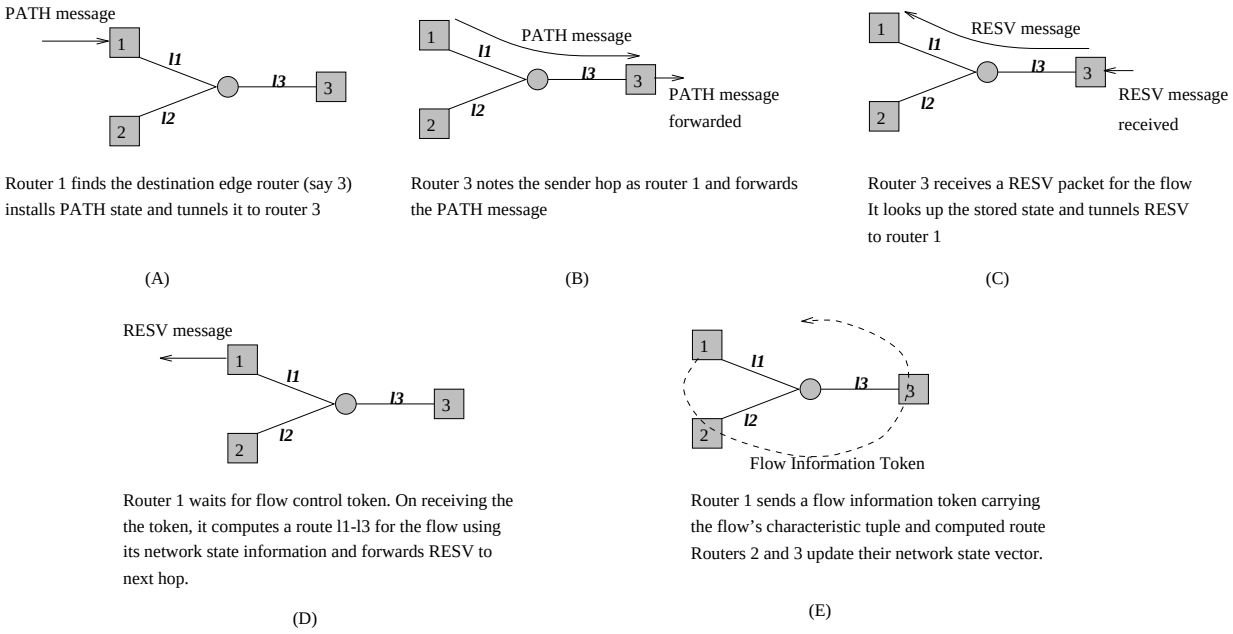

Fig. 1. Processing of an advance reservation request.

these flows not being met. So an ingress router waits for the *flow control token* (which we describe in next sub-section) before it computes route for a flow. In other words, the flow control token acts as a semaphore between edge routers to control route computation.

Once the ingress router receives the flow control token, it computes a route for the flow based on its information about the future network state stored in vector B. It then sends a *flow information token* among the other edge routers to notify them of the arrival of the new request and also sends the flow control token to the next edge router in the sequence. The circulated flow information token contains the characteristic tuple of the flow along with its computed route. On receiving the flow information token, the edge routers update their network state vectors. When the flow information token comes back to the ingress router it updates its network state vector and also installs the flow's reservation state. Thus all the edge routers have identical network state vectors after the flow information token is circulated. In other words, our reservation request handling mechanism allows that the flow state be stored only at ingress router of a flow and at the same time it keeps the network state information vectors across the edge routers consistent.

B. Flow Handling

Since our protocol does not change the core routers, they are incapable of distinguishing between packets from reserved flows and best effort flows. The only way to ensure that the reserved flows get their desired bandwidth, is to have the total traffic on each link less than or equal to its capacity. At the same time we don't want to restrict the flows so much that the network utilization suffers.

We use a *flow control token* to do exactly as mentioned above. The flow control token carries four aggregated values

for each link l in the domain. These are:

- C_l : The amount of bandwidth left over on link l after deducting the bandwidth reserved on it for the reserved flows which are *active*.
- F_l : The number of active best effort flows going through the link. An edge router knows the best effort flows for which it is ingress. It also knows the path that the flow is taking (since the flow is pinned to a path) and thus can add to the flow count on the corresponding links.
- R_l : The amount of residual bandwidth on the link. The residual bandwidth is the amount of bandwidth left over by those *best effort flows* on the link, which are bottlenecked at some other link on their path. Since an edge router knows the path of its best effort flows, it knows the bottleneck link for each flows and hence knows the unused portion of their fair share on the link. the residual bandwidth on each link on the path of the flow.
- CR_l : The claimed residual bandwidth on the link. Since residual bandwidth is the unused portion of bandwidth of some best effort flows, other best effort flows can use that bandwidth if their rates can increase by using this bandwidth. The residual bandwidth is claimed on a first-come-first-serve basis, thus allowing the flows whose ingress router is next in the token circulation cycle the first opportunity to use the residual bandwidth.

The rates of the flows are computed and the flow control token handling is done as follows: When an edge router receives the token:

- If a reserved flow has terminated, it increases the available bandwidth C_l on all links l on its path.
- If a best effort flow has terminated (timed out), it decreases the number of flows F_l on all links l on the path of the flow.
- It looks up the reservation state table and finds if any reserved flow is going to start within δ time from now. For each such flow f, it has the path information stored. On each link l on the path of f, the available bandwidth C_l is reduced by its reserved bandwidth R, if it has not already done so in the previous token cycle. If C_l has already been reduced to take into account f, the router is all set to allow packets from f into the network.
- For a new best effort flow f, it computes the path. On all the links l on path of f, it increments the flow count F_l by one. It allows packets from f to share the bandwidth with some other best effort flows which are going to the same egress router as f. This sharing is done by using some fair rate scheduler [18].
- Compute the new rate of policing of the installed fair rate schedulers.[2] Let the number of flows being policed by the fair rate scheduler be N. For each link l on the path of the flows, compute the fair share $F = \frac{N*C_l}{F_l}$. The best share is computed as $F + (R_l - R'_l) - (CR_l - CR'_l)$ where R'_l is the residual bandwidth and CR'_l is the claimed

[2] A fair rate scheduler collectively polices all the flows sharing the same path. Hence the new rate is computed for a scheduler and not individual flows.

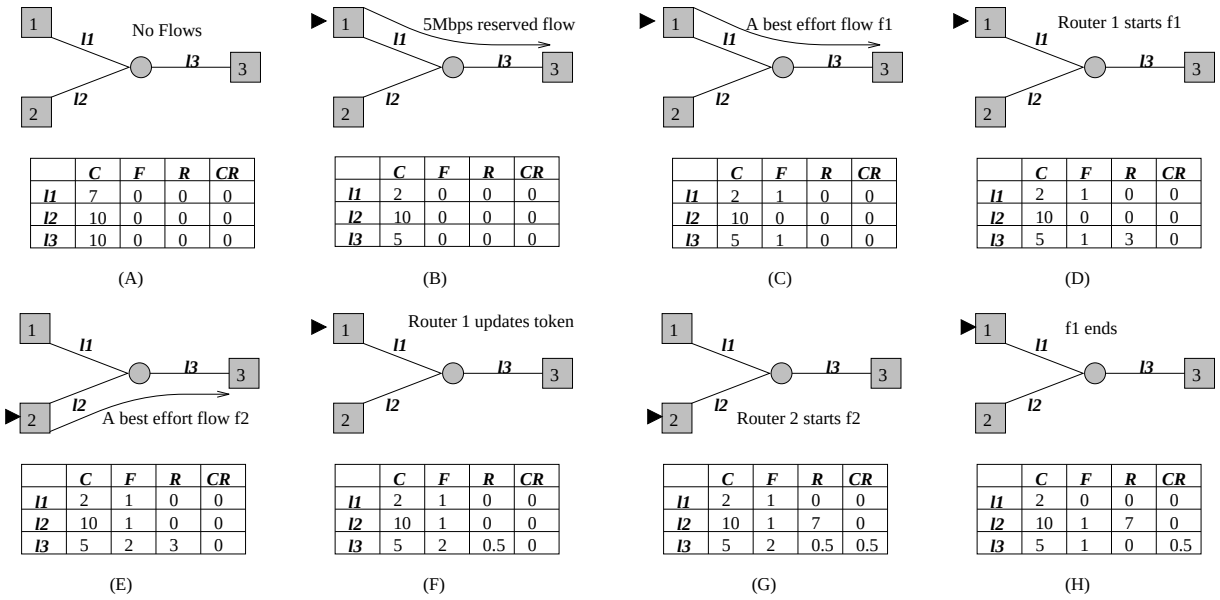

Fig. 2. An example illustrating the working of EQOS. The black arrowhead indicates the router which has just updated the token.

residual bandwidth on l by the scheduler's own flows(which it put in the token in the previous circulation cycle). The minimum of the best share over all links on the flows' path, is the new rate for the scheduler. After that the router updates the R_l and CR_l on the links l on the flow group's path.

- The router then sends the token to the next edge router in the circulation sequence.

We illustrate how these values are computed using a simple example. Consider the simple topology given in fig. 2. Assume that the token circulates among the edge routers in sequence 1,2,3. Let the links $l2, l3$ have initial bandwidth of 10Mbps, $l1$ have a initial bandwidth of 7Mbps and that initially there is no flow in the network. At this time the available capacity in all links is equal to their capacities, no best effort flows on any links and there is no residual bandwidth as in fig. 2(A). Let a reserved flow become *active* from router 1 to router 3 taking 5Mbps bandwidth. When the token arrives at router 1, it updates the token contents as shown in fig. 2(B). The available bandwidth on $l1$ is now 2Mbps and that on $l3$ is 5Mbps. Also, there is no change in any other fields as there are no best effort flows as yet. When the token comes back to router 1 after a cycle, it can start the reserved flow. Next a best effort flow $f1$ starts from router 1 to router 3. On receiving the token, router 1 increases the number of flows on links $l1$ and $l3$ by one and forwards the token as shown in fig. 2(C), but it does not start $f1$ as yet. It has to wait for 1 token cycle before it can start. Note that this one cycle wait is only there when the ingress router does not have any other best effort flows going to the same egress router as $f1$. Normally, the best effort flows can be just be added to the fair rate scheduler policing some group of flows going to the same egress router. In the next token cycle, router 1 computes the maximum rate for $f1$ as $min(\frac{2}{1}+0-0, \frac{5}{1}+0-0)$= 2Mbps which is the maximum bandwidth it can use on link $l1$. It updates the contents of the token as shown in fig. 2(D). The residual bandwidth on $l1$ is set to 0 because $f1$ is using the entire 2Mbps of the available bandwidth. However, the

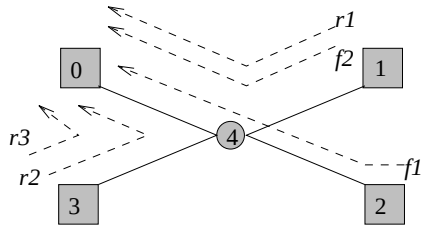

Fig. 3. Topology for the simulation depicting the working of EQOS . All links have a bandwidth of 10Mbps

residual bandwidth on $l3$ is set to 3Mbps because, $f1$ is entitled to 5Mbps on $l3$ but cannot use more than 2Mbps on it. Now a best effort flow $f2$ starts from router 2 to router 3. On receiving the token, router 2 increases the number of flows on links $l2$ and $l3$ by one as in fig. 2(E) and sends the token to the next router in sequence. When router 1 receives the token, it finds that the number of flows on $l3$ has increased and it recomputes the rate for $f1$ as $min(\frac{2}{1}+0-0, \frac{5}{2}+0-0)$ which is still 2Mbps. But now the residual bandwidth of 3Mbps that it had left on $l3$ is reduced to 0.5Mbps (which is its fair share $\frac{5}{2}$ on $l3$ minus its rate which is 2Mbps). It updates the token as in fig. 2(F) and sends it to the next router. After 1 cycle, when router 2 receives the token, it knows that the other edge routers must have corrected their flow rates and token information to take into account $f2$. It computes the rate for $f2$ as $min(\frac{10}{1}+0-0, \frac{5}{2}+0.5-0)$= 3Mbps. So it claims 0.5Mbps of the residual bandwidth on $l3$ (which is the entire residual bandwidth in this case) for $f2$. Moreover, it sets the residual bandwidth on $l2$ to 7Mbps as $f2$ is using only 3Mbps of the 10Mbps available to it. It updates the token accordingly as in fig. 2(G) and forwards the token to the next router. Lets say that $f1$ terminates now. On receiving the token, router 1 will adjust all the fields which were modified on account of $f1$ and send out the token with the flow count on links $l1$ and $l3$ and the residual bandwidth on $l3$ reduced as shown in fig. 2(H). On receiving this token, router 2 will readjust the rate of its best effort flow to $min(\frac{10}{1}+0-0, \frac{5}{1}+0-0)$ = 5, the residual bandwidth on $l2$ is set to 5, claimed residual bandwidth on $l3$ is reduced to 0 and update the token accordingly. This way the process continues.

C. Starting A Reserved Flow

The ingress router for a reserved flow knows when it is going to start. However, the flow must re-confirm its intent to use the reservation between t_{min} and t_{max} times before it starts otherwise the network may discard the reservation.[3] We describe in next section how the re-confirmation message can add to the fault tolerance. Once the flow has re-confirmed, it can start anytime after $S-\delta$. The ingress router considers the flow active at this time and accordingly reduces bandwidth on the links on its path. The EQOS protocol will ensure that after one token circulation cycle, the links on the flow's path have enough bandwidth to support it.

We simulate our protocol to verify its dynamic bandwidth sharing capability. We modified the ns-2 simulator [19] to

[3]The ISP's policy determines the values of t_{min} and t_{max}.

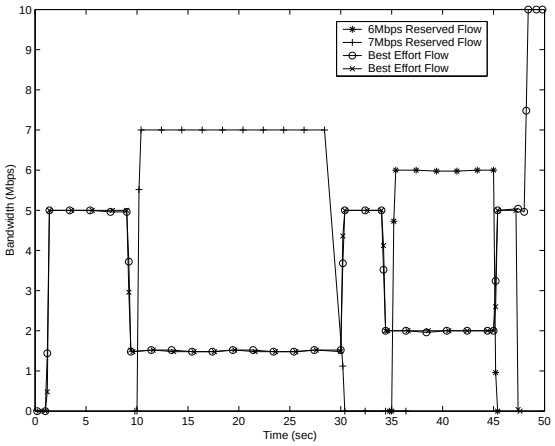

Fig. 4. Results for simulation experiment 1

incorporate the details of EQOS in it[4]. We describe the results of our simulations next.

For this experiment we used the topology shown in fig. 3. The routers labeled $0-3$ are edge routers and router 4 is the core router. A best effort flow $f1$, arriving at a rate of 10Mbps at router 2 and going to router 0 on links $2-4$ and $4-0$, runs through time 1 to time 50. Another best effort flow $f2$ runs through time 1 to time 47 on links $1-4$ and $4-0$ between edge routers 1 and 0. An advance reservation request for a flow $r1$ arrives at time 2 at ingress router 1 indicating a desired bandwidth of 7 Mbps, starting time 10, ending time 30 and destination router 0. This request is granted and the bandwidth is reserved on links $1-4$ and $4-0$ for the flow. For this simulation, the edge routers are supposed to reduce flows 1 time unit before the usage starts. Another request arrives at time 5 for an advance reservation flow $r2$ which requires a bandwidth of 6Mbps for time 25 to 35 from ingress router 3 to egress 0. Since there is not enough bandwidth available during time 25-30 seconds on link $4-0$, the request is rejected (The graph does not show it). Finally, a third request for flow $r3$ arrives at time 7 for 6Mbps of bandwidth from ingress router 3 to egress router 0 starting at time 35 end ending at time 45. This request is granted. The simulation results shown in fig. 4 indicate that during time 1-9 the best effort flows $f1$ and $f2$ share the entire bandwidth. At time 9, edge router 0, which is the ingress for flow $r1$, reduces the available bandwidth in the token. As a result, routers 2 and 0 reduce the rate of $f1$ and $f2$ to 1.5Mbps each. At time 10, the advance reservation flow starts and gets its reserved bandwidth. It continues till time 30 when it stops and then $f1$ and $f2$ take up the entire 10Mbps again. At time 34 the best effort flows back off to 2Mbps each. At time 35 $r3$ starts and gets its full bandwidth. At time 45, $r3$ terminates, allowing $f1$ and $f2$ to share the entire bandwidth again. Finally, when $f2$ times out at time 47, $f1$ takes up the entire bandwidth. Thus we see that EQOS manages the network bandwidth perfectly without over-allocating any link bandwidth at any time.

[4]A proof of correctness of EQOS is described in [17].

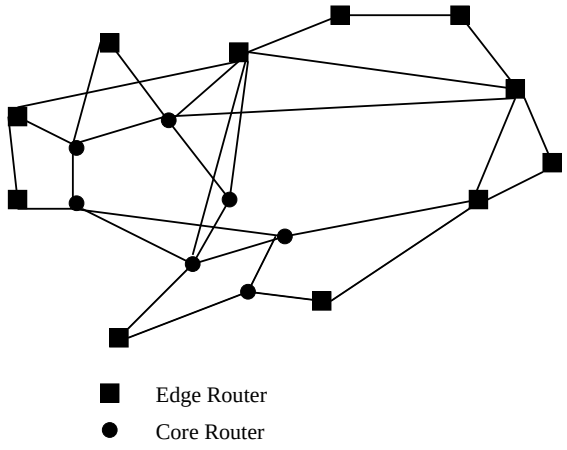

Fig. 5. The ISP topology used for experiments 1 and 2.

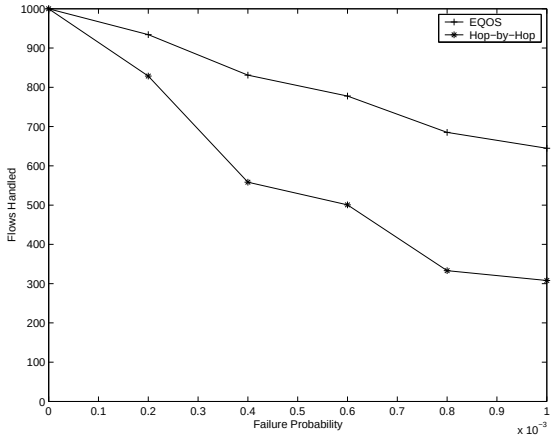

Fig. 6. Result for experiment testing the impact of constant failure probability.

V. Factors Affecting Flow Handling

We have argued that a domain-based approach is inherently more fault-tolerant than a hop-by-hop approach in handling advance reservations. In this section we describe various factors which affect the degree of fault tolerance of any network. We use simulations to show the effects of these parameters on both hop-by-hop approach and the domain based approach of EQOS (without any measures of fault tolerance). Each experiment is conducted 10 times using different flow patterns. Each point in the result is an average of these runs. For all simulations, we use seconds as a virtual unit for time. In practice, it could be minutes, hours, days or any other unit.

The first factor which determines how many reserved flows can be handled by the network is obviously the *failure probability*. We use a simple ISP topology shown in figure 5 for this set of experiments. In this set of experiments, the failure probability in a run is kept constant. The failure probability value is increased from 0 to 10^{-3} in different runs. One thousand advance reservation requests arrive at the network within the first 100 seconds. All flows start after 100 seconds and all the flows start by 1000 seconds. The simulation is run for 1000 seconds. The results are shown in

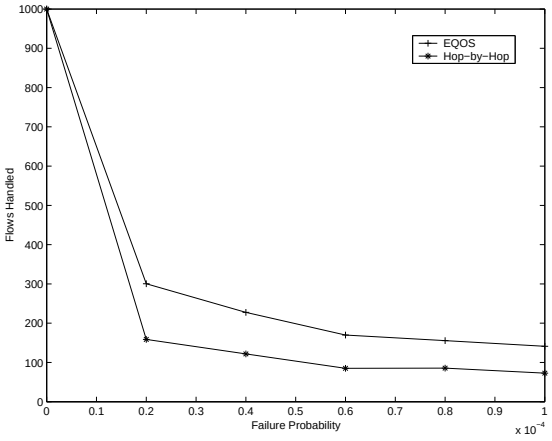

Fig. 7. Result for experiment testing the impact of a failure probability function linearly increasing in time.

figure 6. We see that the number of flows that can be handled properly by both approaches reduce as the failure probability increases. However, EQOS always outperforms the hop-by-hop approach.

We repeat the above experiment, with a different failure probability function. This time we use a linearly increasing function of time as the probability function. The node failure probability for uptime t is increased from $10^{-5}t$ to $10^{-4}t$. The results, shown in figure 7, are similar to the previous case, except that the number of flows not handled properly is higher (because the failure probability function's value increases with time thus causing failure at a higher number of nodes). Irrespective of the failure probability function, EQOS can handle more reserved flows than a hop-by-hop approach.

The next factor that affects the fault tolerance capability of a network handling advance reservations, is the *average storage phase duration*. The longer the duration, the more likely it is that the flow is not properly handled. We simulate this effect on the ISP network of figure 5. We have three set of flows which have different durations of storage phase. Again we assume that all flows in each set of 1000 requests arrive at the network before the first 100 simulation seconds and all flows start after 100 seconds. The flows in set 1 have an average storage phase duration 100 seconds, those in set 2 have average storage phase duration 1000 seconds), and those in set 3 have an average storage phase duration 10000 seconds).

First we use the constant failure probability function. Figures 8 and 9 show the number of flows handled properly for failure probability values 5×10^{-4} and 10^{-3}. The results show that the as the duration of storage phase increases, the number of flows handled properly decreases. Again, EQOS performs better than the hop-by-hop approach for all storage phase durations.

We repeat this experiment with linearly increasing failure probabilities $10^{-5}t$ and $5 \times 10^{-4}t$. The results shown in figures 10 and 11 are similar in nature to the preceding ones.

The third factor that needs to be considered is the average number of routers on path of a flow or the *average path*

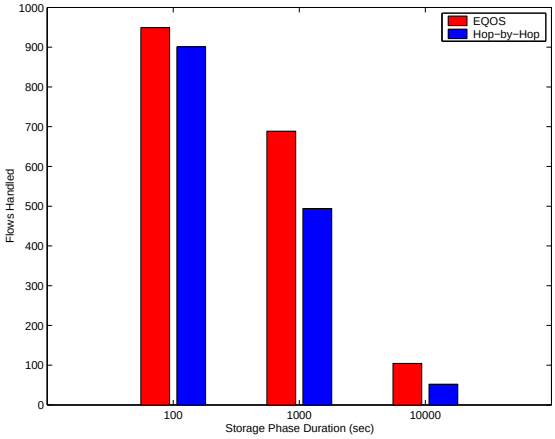

Fig. 8. Result for experiment testing the impact of storage phase duration with constant failure probability 0.0005

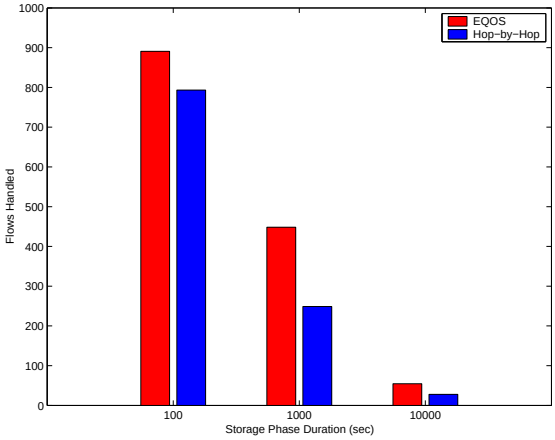

Fig. 9. Result for experiment testing the impact of storage phase duration with constant failure probability 0.001

length. For a domain-based reservation approach, an increase in the number of routers along the path does not reduce the number of flows being handled properly (since the number of nodes storing reservation state for a flow is still one). However, a hop-by-hop approach performs badly as the number of routers on the flow's path increase because of a higher cumulative failure probability. This effect is depicted in our next simulation set. We use the topologies shown in figure 12. The number of edge routers in the topologies does not change but the average number of hops between a pair of edge routers increases from 2 to 4. For all the topologies the flow patterns remain the same. We set the average storage duration to 450 seconds, i.e., all the flows start during 100 and 1000 seconds. The results for constant failure probabilities 10^{-3} and 5×10^{-4} (shown in figure 13 and 14) and linearly increasing failure probabilities $10^{-5}t$ and $5 \times 10^{-5}t$ (shown in figure 15 and 16), both show that the domain based approach is not affected by this factor, but the hop-by-hop approach deteriorates.

The above experiments verify our claim that a domain based approach based approach is inherently more fault-tolerant compared to a hop-by-hop approach. Although EQOS outperforms the hop-by-hop approach, the number of flows han-

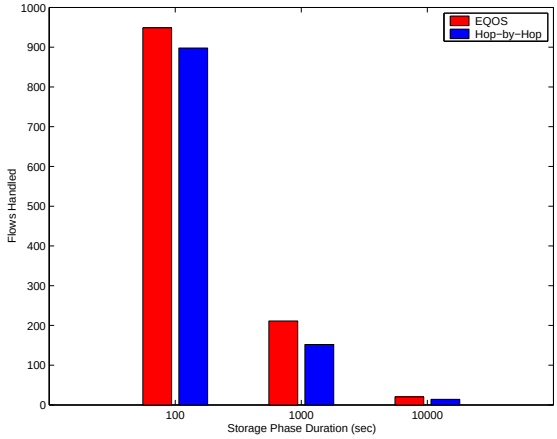

Fig. 10. Result for experiment testing the impact of storage phase duration with linearly increasing failure probability $0.00001 \times t$.

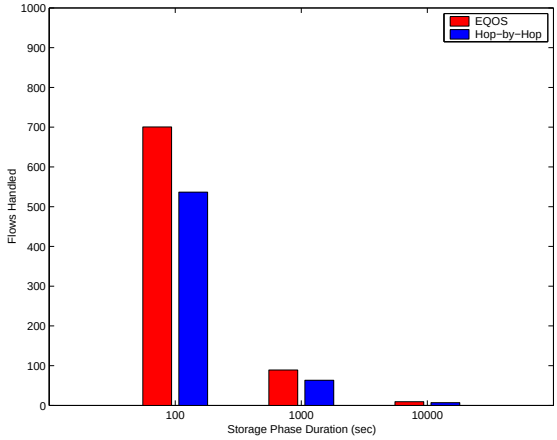

Fig. 11. Result for experiment testing the impact of storage phase duration with linearly increasing failure probability $0.00005 \times t$.

dled by EQOS show that the router failures affect it significantly. The benefits of EQOS at this point would be limited to minimizing the number of routers required to guarantee QoS. Our second objective, eliminating the effects of node failure during storage phase, is not met by EQOS in its current form. In next section we describe a simple extension which achieves this objective.

VI. The EQOS-A Protocol

We have described the EQOS protocol which handles advance reservations and does so only by using edge routers. Now we show how this simple protocol can be enhanced into EQOS-A, which provides a high degree of fault tolerance. Recall that we are trying to eliminate the effects of router failure during storage phase.

When a reservation request arrives at an edge router, it computes a route for the flow and circulates a *flow information token* among the edge routers. The other edge routers use the token to update their network state vectors. A simple enhancement that can be made in EQOS is to make the other edge routers *store* the flow information as well. Given this

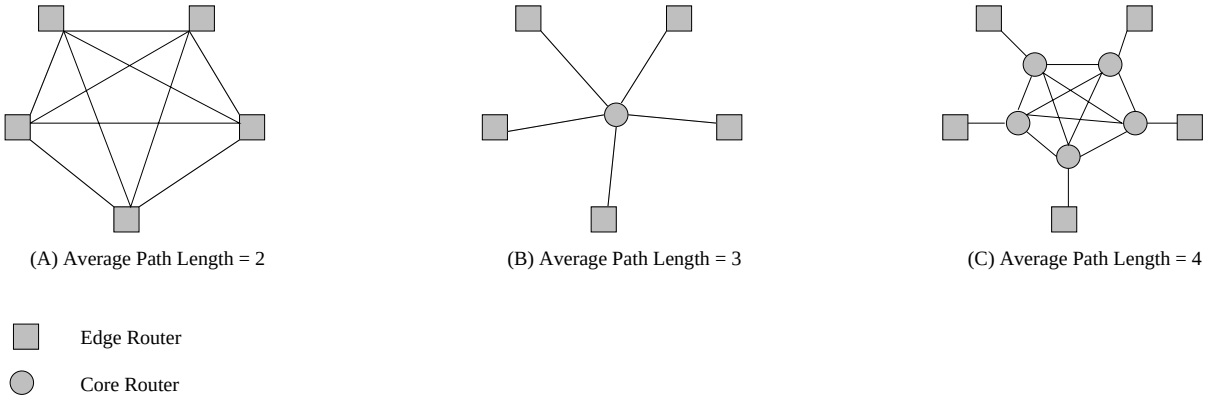

Fig. 12. Topologies with different path lengths for the same number of edge routers, used in evaluating the impact of average path length

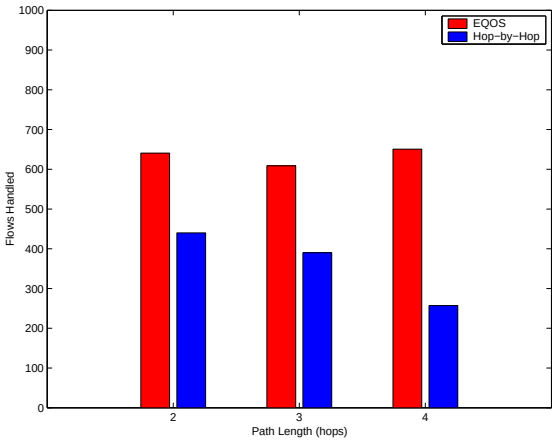

Fig. 13. Result for experiment to test the impact of average path length with constant failure probability 0.0005.

simple enhancement the states of all reserved flows will be available at all the edge routers of the domain rather than just the ingress router. In other words, this enhancement makes the flow state across the edge routers consistent along with the network state. Without this enhancement, only the network state across edge routers was consistent and not the flow state. Note that keeping state of all reserved flows at all the routers might be a significant overhead. Practically only a subset of edge routers can be designated as backups for a given edge router. However, that is more a issue of policy than being a protocol issue. We assume that the backup is made at all routers. The most significant part of this *co-operative backup* operation is that no extra backup devices for any router in the domain are required.

When an edge router fails, its state information is available at all the other edge routers. This implies that whenever the failed router comes up again, it can query some other edge router about the reservation state information stored. The queried router can then pass that information in a reliable fashion to the concerned router and thus recover from the failure. The only way we can lose state information is if all the edge routers go down simultaneously. The other case where we might not be able to handle a reservation due to loss of reservation state information is when the flow starts

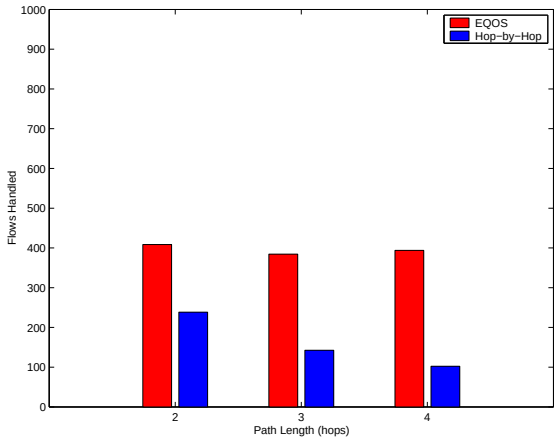

Fig. 14. Result for experiment to test the impact of average path length with constant failure probability 0.001.

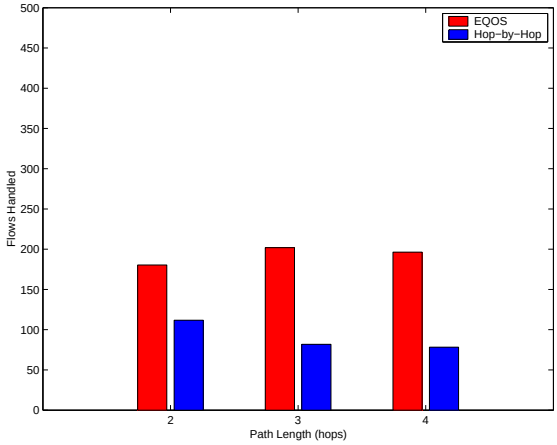

Fig. 15. Result for experiment to test the impact of average path length with linearly increasing failure probability 0.00001.

during the time when its ingress router is down.

Failure of core router does not cause any loss of state information as our approach is core-stateless unlike hop-by-hop approaches. However, this mandates that the route for any flow traversing the failed router should be recomputed. This is not different from the way immediate reservation protocols handle node failure. If the QoS routing employed for route computation has a notion of survival routing, it would have created backup paths for all reserved flows and the flows can be rerouted. In case the QoS routing algorithm does not compute backup paths, new routes should be computed for flows traversing the failed router as for immediate reservation.

Another type of failure that this design can handle is a change in topology in some upstream domain. This is possible because of the reservation re-confirmation requirement that we mentioned earlier. Suppose that a change in topology in an upstream domain causes some flow to arrive at another edge router than the one meant to be the ingress router. If the new ingress router receives the re-confirmation message for a flow it was not supposed to be an ingress to, it circulates the information about this flow around all the edge routers. The original ingress router can then clear the computed path for

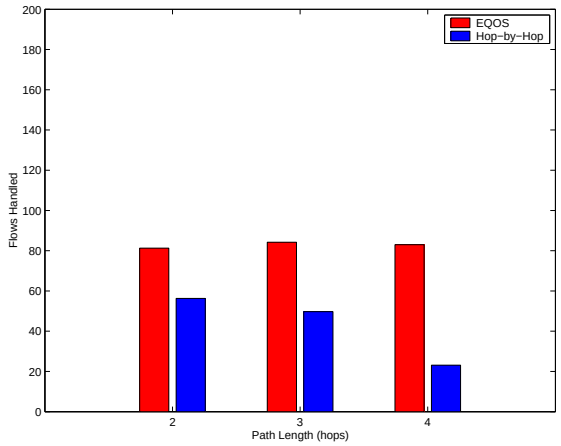

Fig. 16. Result for experiment to test the impact of average path length with linearly increasing failure probability 0.00005.

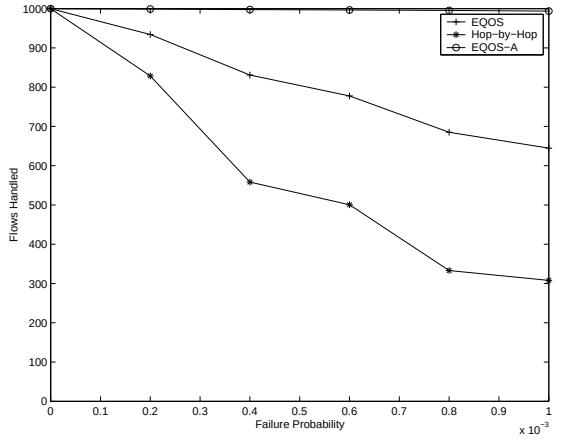

Fig. 17. Result for experiment testing the impact of constant failure probability y.

this flow and inform other routers about this cancellation. Now the new ingress router can try and find a new path from itself to the destination egress router for the flow. If it can find such a path, it becomes the ingress node for the flow and follows the same procedure as described in section 4. Note that the new ingress router might not be able to find a path for the flow. However, our protocol provides a chance for the flow to get through. The hop-by-hop approach can not handle this kind of a case gracefully. In the hop-by-hop approach, when some other edge router receives a re-confirmation, it does not have any clue about the original path for the flow. Although it can handle the request as one for a new flow, the routers on the original path will have to wait till the reservation state times out, to clear the bandwidth reserved for that flow. Till that time, the reserved bandwidth is not available for other flows. Clearly, our protocol handles such cases much more effectively than the hop-by-hop approach.

We test the flow handling capability of EQOS-A with respect to a hop-by-hop approach and pure EQOS approach. The experiments are the same as those described in section 5, except that now the results contain the figures pertaining to EQOS-A.

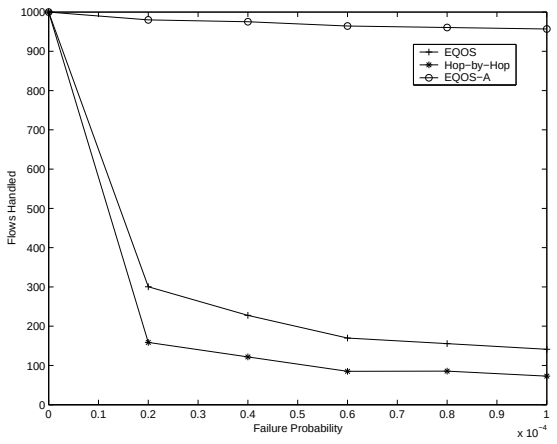

Fig. 18. Result for experiment testing the impact of a failure probability function linearly increasing in time.

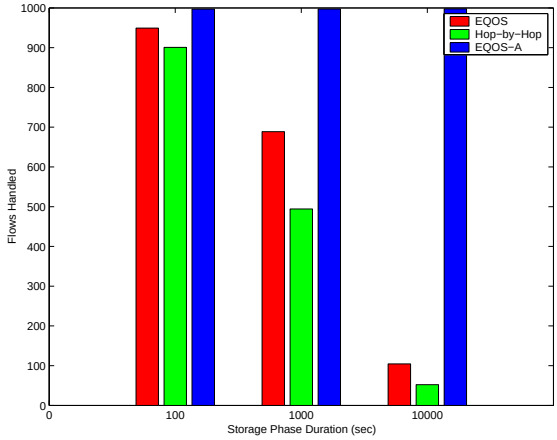

Fig. 19. Result for experiment testing the impact of storage phase duration with constant failure probability 0.0005

In the first simulation set the effect of different failure probabilities is studied. As shown in figure 17 for constant failure probabilities between 0 and 10^{-3}, the number of flows handled by EQOS-A is very high compared to the hop-by-hop and EQOS based approaches. In fact, the flow handling capability of EQOS-A is never below the 95% mark in individual simulation sets, not just on average. Figure 18 shows that under linearly increasing failure probabilities between 0 and $10^{-4} \times t$ as well, EQOS-A outperforms the hop-by-hop approach and EQOS approach significantly.

The second simulation set tests the impact of average storage phase duration on the flow handling. Both for constant fault probabilities (results shown in figure 19 and 20) and linearly increasing fault probabilities (results shown in figure 21 and 22), EQOS-A is handling more than 96% flows where as the flow handling capability of the hop-by-hop approach and pure EQOS approach go down significantly with increase in storage phase duration.

The third set of simulations checks the impact of increase in average path length through a domain. Figures 23, 24, 25 and 26 show that like EQOS, EQOS-A also remains unaffected by the increased path length over all error probabilities. The flow handling of EQOS-A is more than 95%.

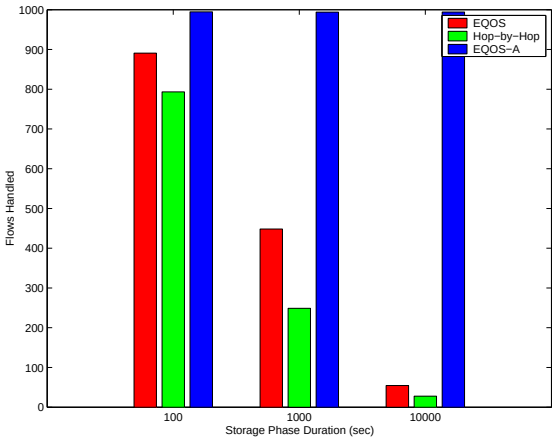

Fig. 20. Result for experiment testing the impact of storage phase duration with constant failure probability 0.001

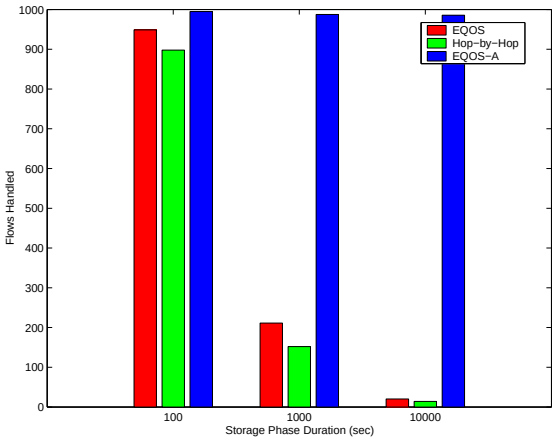

Fig. 21. Result for experiment testing the impact of storage phase duration with linearly increasing failure probability $0.00001 \times t$.

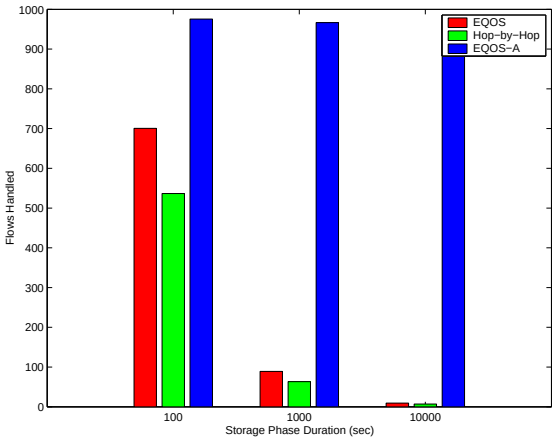

Fig. 22. Result for experiment testing the impact of storage phase duration with linearly increasing failure probability $0.00005 \times t$.

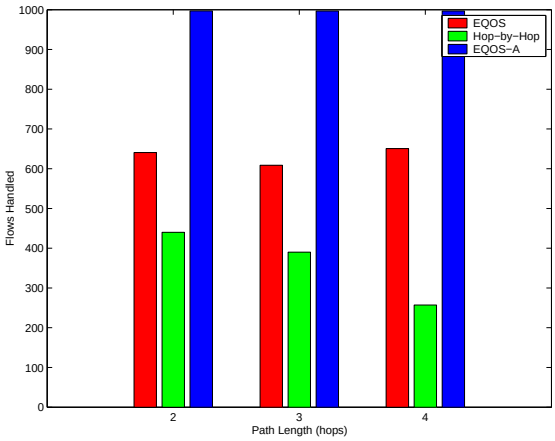

Fig. 23. Result for experiment to test the impact of average path length with constant failure probability 0.0005.

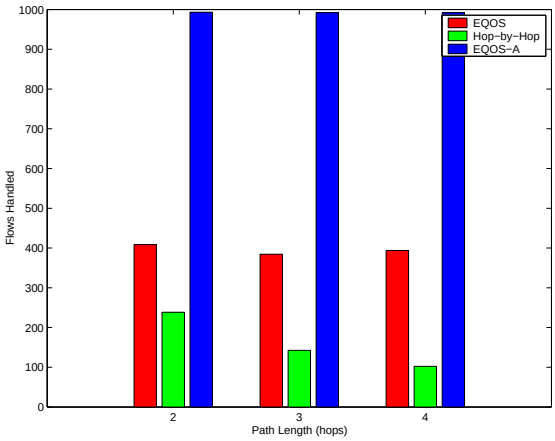

Fig. 24. Result for experiment to test the impact of average path length with constant failure probability 0.001.

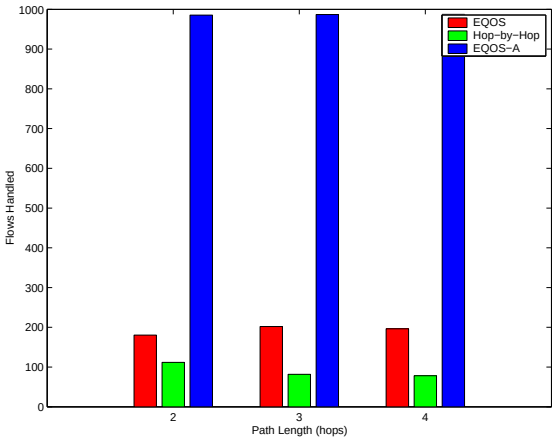

Fig. 25. Result for experiment to test the impact of average path length with linearly increasing failure probability $0.00001 \times t$.

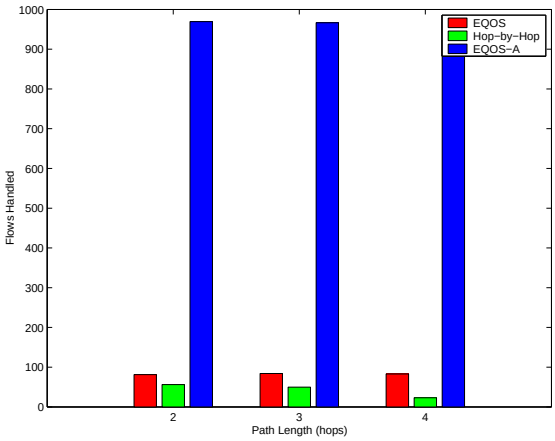

Fig. 26. Result for experiment to test the impact of average path length with linearly increasing failure probability $0.00005 \times t$.

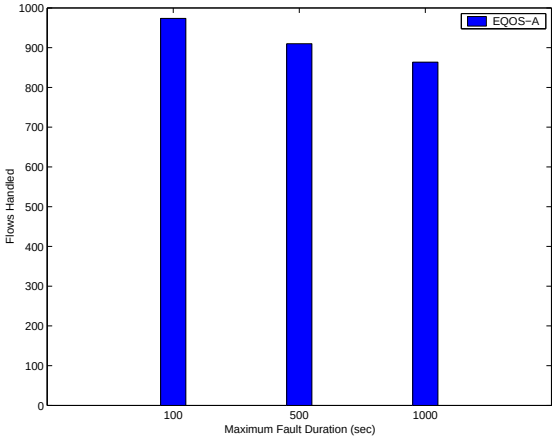

Fig. 27. Result for experiment testing the impact of average fault duration with constant failure probability 0.0005

Finally, another factor which can affect flow handling in a system with back up, is the *average fault duration*,i.e., the average amount of time a node is down before it comes up again. We simulate the effect of this factor on the ISP topology of figure 5. We again have 1000 flows with the requests arriving within first 100 seconds. The flows start anytime between 100 and 1000 seconds. The minimum fault duration is 1 second. Figure 27 shows the number of flows handled by EQOS-A when the maximum fault duration is set to 100 sec, 500 sec and 1000 sec. The flow handling percentage of EQOS-A goes down as the duration increases. However, it is still performing quite well.

In summary, the only factor that can affect EQOS-A's flow handling is the average fault duration. All other parameters have virtually no impact on the fault handling of EQOS-A. As long as the fault duration is not a high percentage of the average storage phase duration, EQOS-A is likely to perform very well. In fact, the flow handling capabilities of EQOS-A remain so high that the effects of router failure during storage phase is virtually eliminated.

VII. Discussion

A. Why not a centralized bandwidth broker?

As we have described, all the edge routers see the same network state vector. If we are going to store the backup information at all edge routers, then the reserved flow state across the routers will also be the same. An obvious question to ask at this point is that why not have a single bandwidth broker for the entire domain?

A centralized broker approach is much more susceptible to failures than even the pure EQOS approach. A centralized bandwidth broker has to maintain the reservation state of all the flows. If the centralized broker fails, the reservation state of all the flows in the domain is lost. The EQOS approach requires that the state of a flow be stored at its ingress router only. In case of an edge router failure, the lost reservation states are of only the flows for which it is the ingress router. In other words, failure of one node could mean a loss of all flows' reservation states in a centralized approach, whereas in EQOS approach, only a fraction of flows may lose their reservation state in case a router fails.

Another reason for this is that just having a centralized bandwidth broker will not allow dynamic bandwidth sharing between reserved flows and best effort flows without the token passing mechanism of EQOS. To the best of our knowledge, there is no protocol other than EQOS, which allows dynamic sharing of bandwidth between reserved and best effort flows. To utilize the network resources as much as possible, it is desirable to have the best effort flows take up all the remaining bandwidth left by the reserved flows. Hence, if we are using the token passing mechanism anyway, the edge routers can themselves act as distributed bandwidth broker and act as backup for one another.

A common argument provided for making a node fault tolerant is to introduce backup devices. Even on that count a centralized broker does not compare favorably with EQOS-A approach because the amount of backup that our approach provides(each edge router for every other edge router) is not likely to be there for a special node acting as a centralized bandwidth broker. Hence we believe that our approach will perform much better than a centralized bandwidth broker both in terms of network utilization and fault tolerance.

B. Handling Immediate Reservations

An immediate reservation is effectively an advance reservation with a zero-duration storage phase. An important deviation is that the termination time of the flow is not known. To handle advance reservation and immediate reservation flows with unknown duration, the network should estimate the amount of time that the flow could take. This can be done by collecting duration statistics of reserved flows over a period of time. Also the application seeking reservation can be asked to provide an estimate of its duration. For example, an application streaming a movie to a viewer, is likely to know the duration of the movie. A complete description of handling unknown duration immediate reservations with advance reservations is given in [20].

VIII. Conclusion

Advance reservation schemes have to maintain a flow's state for a long period of time. We showed that for advance reservations, the traditional hop-by-hop approach is significantly less fault tolerant that a domain-based approach. We presented a protocol called EQOS-A for fault tolerant handling of advanced reservations. EQOS-A being implemented entirely in edge routers, minimizes the deployment complexity of the protocol. We have shown that our protocol eliminates the effect of router failure between the time the resources were reserved and the time the resources are used. This reduces the complexity due to router failure in advance reservation capable networks, to a level similar to immediate reservation capable networks.

In future, we plan to implement our protocol on a testbed and evaluate its performance in a practical scenario. We will also like to test the interaction between advance reservations and immediate reservations.

References

[1] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource reSerVation Protocol (RSVP) version 1, Functional Specification," RFC 2205, IETF, September 1997.

[2] C. Topolcic, "Experimental Internet Stream Protocol: Version 2 (ST-II)," RFC 1190, IETF, October 1990.

[3] J. Wroclawski, "Specification of the Controlled-Load Network Element Service," RFC 2211, IETF, September 1997.

[4] S. Shenker, C. Partridge, and R. Guerin, "Specification of Guaranteed Quality of Service," RFC 2212, IETF, September 1997.

[5] G. Apostolopoulos, R. Guérin, and S. Kamat, "Implementation and Performance Measurements of QoS Routing Extensions to OSPF," in *Proc. of IEEE Infocom*, March 1999.

[6] R. Guérin and A. Orda, "QoS Routing in Networks with Inaccurate Information: Theory and Algorithms," *IEEE/ACM Transactions on Networking*, vol. 7, no. 3, pp. 350–364, June 1999.

[7] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A new Resource ReSerVation Protocol," *IEEE Network Magazine*, pp. 8–18, September 1993.

[8] J. Wroclawski, "The Use of RSVP with IETF Integrated Services," RFC 2210, IETF, September 1997.

[9] I. Stoica and H. Zhang, "Providing Guaranteed Services Without Per Flow Management," in *Proc. of ACM SIGCOMM*, September 1999, pp. 81–94.

[10] Z. Zhang, Z. Duran, L. Gao, and Y. Hou, "Decoupling QoS Control from Core Routers: A Novel Bandwidth Broker Architecture for Scalable Support of Guaranteed Services," in *Proc. of ACM SIGCOMM*, October 2000, pp. 71–83.

[11] W. Reinhardt, "Advance Reservation of Network Resources for Multimedia Applications," in *Proc. of the 2nd International Workshop on Advanced teleservices and High-Speed Communication Architectures*, 1994.

[12] M. Degermark, T. Kohler, S. Pink, and O. Schelen, "Advance Reservations for Predictive Service in the Internet," *ACM–Springer Journal of Multimedia Systems*, vol. 5, no. 3, pp. 177–186, 1997.

[13] L. Wolf and R. Steinmetz, "Concepts for Resource Reservation in Advance," *Kluwer Journal on Multimedia Tools and Applications*, vol. 4, no. 3, May 1997.

[14] R. Guerin and A. Orda, "Networks with Advance Reservations: The Routing Perspective," in *Proc. of IEEE Infocom*, March 2000.

[15] C. Labovitz, A. Ahuja, and F. Jahanian, "Experimental study of internet stability and wide-area backbone failures," Tech. Rep. CSE-TR-382-98, University of Michigan, 1998.

[16] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," RFC 3031, IETF, January 2001.

[17] S. Bhatnagar and B. Vickers, "Providing Quality of Service Guarantees Using Only Edge Routers," ftp://www.cs.rutgers.edu/pub/technical-reports/dcs-tr-421.ps.Z, DCS-TR-421, Dept. of Computer Science, Rutgers University, September 2000.

[18] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a Fair Queueing Algorithm," in *Proc. of ACM SIGCOMM*, 1989, pp. 3–12.

[19] LBNL Network Research Group, *UCB/LBNL/VINT Network Simulator - ns (version 2)*, http://www-mash.cs.berkeley.edu/ns/, September 1997.

[20] O. Schelen and S. Pink, "Sharing Resources through Advance Reservation Agents," in *Proc. of International Workshop on Quality of Service*, May 1997.