A Study of Reservation Dynamics in Integrated Services Packet Networks

Danny J. Mitzel* mitzel@catarina.usc.edu
Deborah Estrin† estrin@usc.edu
Scott Shenker‡ shenker@parc.xerox.com
Lixia Zhang‡ lixia@cs.ucla.edu

Abstract

The Integrated Services Packet Network (ISPN) architecture proposed within the Internet community [2] incorporates a resource reservation mechanism for those applications requiring Quality of Service (QoS) guarantees. Resource reservation introduces a new form of resource contention that can lead to reduced network throughput and thrashing. We establish several necessary conditions to induce thrashing. We also look at the effects several different reservation models and user behaviors can have on network stability. Our work is unique from previous network resource reservation investigations in that we consider the effects of reservations for multipoint-to-multipoint applications. We conclude with examples of how simple modifications to user behavior can result in significant increases in system stability.

1 Introduction

The Internet, and other similar packet-switched network architectures, offer best-effort service. Best-effort service requires no admission control and involves no resource reservation; that is, sources need not notify the network before transmitting data, and no resources are set aside for any particular flow. This architecture has supported a wide variety of data applications, such as remote login (e.g., Telnet), file transfer (e.g., FTP), and electronic mail, in an extremely efficient manner.

However, there are application requirements that this service model does not handle efficiently. For instance, applications like interactive video and voice cannot tolerate the wide variations in delay and bandwidth present in best-effort service. Consequently, most proposals for Integrated Services Packet Network (ISPN) architectures – networks designed to support the full gamut of offered applications – include some services that require admission control; see [2, 5, 7, 8, 10] and references therein.

Notice that admission control, or equivalently resource reservation, introduces a new form of resource contention for the shared network resources. Previous work in other domains incorporating resource allocation (e.g., database systems) has uncovered phenomena – usually called thrashing – having great detrimental effects on overall system performance and throughput. In this paper we study thrashing in the context of an ISPN architecture such as that proposed in [2].

This work represents a first study of what we believe to be an important phenomena; we establish several necessary conditions to induce thrashing, including inter-reservation resource dependencies, and allowable reservation setup and teardown delays. With reservations, admission control will deny access if there are not sufficient unreserved resources available. Once reservation blocking occurs, the end-user/application may exhibit several different styles of behavior in regard to if and how reservation requests are requeued. We look at the effects different user behavior can have on system performance. We then look at more complex multicast reservations and multipoint-to-multipoint applications, which have not been studied in previous network reservation setup investigations [1, 14]. Finally we propose end-user/application behavioral characteristics for reservation request retry backoff which result in significant improvements in system stability.

This paper is organized in 7 parts. We first, in Section 2 define the model of thrashing we consider and introduce our intuition as to its cause. Next we introduce the network model, topologies studied, and our simulation methodology in Section 3. We then, in Sections 4 thru 6 evaluate reservation system throughput using simulations. We begin by establishing the underlying principles of the thrashing phenomena and demonstrate its existence using a simple uni-directional point-to-point reservation model (Section 4). In Section 5 we look at the effects of more complex reservations and application styles, and finally at methods to improve system stability (Section 6). Our current results represent a simple progression through several initial thrashing scenarios; it is far from a complete understanding of the entire network resource reservation thrashing space. In Section 7 we summarize our findings from the current scenarios studied, introduce several additional scenarios currently under investigation, and outline a number of outstanding issues for future investigation.

2 Thrashing

Thrashing has been observed in a number of different domains where there is contention for a set of shared resources [1, 9, 15]. Tay [15] showed that, in databases with a fixed transaction length and total resources, scaling up the resource demand results in an

*Sponsered by a fellowship from Hughes Aircraft Company.

†Sponsored by the Advanced Research Projects Agency (DoD), Computer Systems Technology Office under Ft. Huachuca contract #DABT63-91-C-0001, entitled "Gigabit Network Communications Research". The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

‡Sponsered in part by the Advanced Research Projects Agency, monitored by Fort Huachuca under contract DABT63-94-C-0073. The views expressed here do not reflect the position or policy of the U.S. government.

initial increase in throughput to a point and then a steady decrease due to contention. We believe that this model is most similar to the allocation of hop-by-hop reservations in the ISPN architecture.

Our rationale as to why thrashing may be exhibited by the reservation mechanism of the ISPN architecture is based upon the fact that resources are wasted when they are reserved but not used, we call these resources *provisional*. Provisional resources can be accumulated in several ways: such as during the setup process due to long end-to-end propagation and admission control delays, or due to long teardown delays after a request has been blocked or a receiver has completed service. Accumulation of any provisional resources increases the system usage level while not contributing to throughput, and in turn increases the probability of blocking for other independent reservation requests. In essence, blocking begets more blocking.

In addition to the accumulation of provisional resources due to initial blocking, we also believe that circularity in the resource dependencies of independent reservations can have a detrimental effect. We mean, looking at the set of network resources required to establish a set of n independent reservations $\{r_1, r_2, ..., r_n\}$ then r_1 may require resources held by r_2, r_2 may require resources held by r_3, ..., r_n may require resources held by r_1. This is the familiar *circular wait* condition described in the deadlock detection literature [12], and can in fact lead to deadlock among the set of reservations involved in the cycle.

Our thesis is that the accumulation of significant levels of provisional resources can occur, with the effect of significant throughput degradation as the resource demand is increased. We follow a strategy of first finding the basic phenomena in simple if perhaps unrealistic settings and then studying its dependence on various factors. In later sections we show that thrashing can arise in more realistic, but more complicated, situations. All results presented rely on reservation teardown delay as the primary source of delays introduced into the system, we have performed some analysis of the effects of propagation and admission control delays but much work is left to future research.

3 Network model

In this work we assume an Integrated Services Packet Network (ISPN) architecture such as that proposed in [2]. Critical components of this architecture include: (1) a flow specification defining the source traffic stream and receiver service requirements; (2) a routing protocol supporting QoS and multicast data distribution; (3) a reservation protocol to create and maintain resource reservations; (4) an admission control algorithm to maintain network load at a proper level; and (5) a packet service algorithm to schedule packet transmissions in an order that maintains service guarantees for individual data streams.

We consider a reservation to be a uni-directional point-to-multipoint stream using a source-rooted multicast distribution tree. Reservation requests are receiver initiated as in the RSVP reservation protocol [3, 17], the request is merged with the multicast distribution tree at the first branch where sufficient resources are already allocated for the requested stream. We arbitrarily set the amount of bandwidth requested for each reservation to be the unit bandwidth, that is each independent reservation consumes one unit of bandwidth.[1] We also arbitrarily select a 60 second holding time for all successful reservation requests. The effect of varying the reservation holding time is to perturb the total network resource demand for a specific request arrival rate, but this does not affect whether thrashing occurs.

The underlying building block in our investigations is the individual resource reservation, however we also consider more complex scenarios that include multicast data distribution and multipoint-to-multipoint applications. When discussing these complex scenarios we often find it useful to refer to the grouping of all reservations associated with the application. We use the term *session* throughout the paper when referring to a group of related source and receivers and the set of related individual reservations. In addition, a session typically exhibits a specific behavior in regard to the coordination of establishment of all the component reservations. We define the details of these behavior in Section 5 where we consider the session models in detail.

All simulations reported in this paper were performed using a discrete event simulation package implementing a receiver-initiated soft state reservation protocol similar to that specified in [3]. The term soft-state was first used by Clark in [4] and, in our context, refers to reservation state maintained at each network switch which is periodically refreshed by end applications; in the absence of refresh messages, such as in case of route changes or end host crashes, the reservation state times out and removes itself. The soft-state approach can add both adaptivity and robustness to reservation protocols, however at the added overhead of periodic refreshing messages. Therefore to keep the overhead low the refresh period should not be too short, and the timeout period also needs to be set accordingly. In our simulation a "teardown" delay is introduced to model the reservation removal delay. Explicit reservation teardown requests result in zero teardown delay; absence of explicit requests leads to a teardown delay greater than zero seconds.

We consider two distinct classes of network topology, cyclic and acyclic. The cyclic network is composed of four switching nodes each connected to two neighbors, forming a simple square topology. Each of the interconnection links in the cyclic network is provisioned with sufficient capacity to accommodate 23 simultaneous reservations.[2] We assume minimal link propagation and node processing delays of 1 millisecond each, thus our current work focuses exclusively on the effects of teardown delays. Source and receivers were placed at each of the four switching nodes with all reservations being between a source and receiver

[1]Note that we are using a rather primitive model of reservations, using only bandwidth to describe the reservation. In practice, the flow specification [11, 16] will likely be somewhat more complex.

[2]We have also performed preliminary simulations on networks with higher degrees of multiplexing and we found similar thrashing behavior.

at opposite diagonals of the network, thus all reservations are of length two hops. For the acyclic network we consider a binary tree topology of depth four, again all links have a capacity of 23 simultaneous reservations. Source and receivers are placed only at the 16 leaf nodes of the tree with reservations between source and receiver at each pair of leaf nodes, this results in a mix of 2-, 4-, 6-, and 8-hop paths.

When studying a single network and homogeneous reservations the resource demand can be quantified by session arrival rate alone. In this study we look at multiple network topologies, heterogeneous numbers of sessions and path lengths, multipoint-to-multipoint sessions and multicast distribution trees. Assuming all reservation requests are for the unit bandwidth, we can calculate a normalized loading metric for the network as the product of the session arrival rate and the number of link reservations required to successfully allocate the session distribution mesh. Calculating the normalized load for a specific simulation scenario results in an estimate of the average number of new link reservation requests injected into the network per unit of time. We use the normalized load measure throughput this paper when contrasting results that are dependent on network loading level.

The normalized loading metric tells us that for a larger session size fewer arrivals are required to maintain a fixed loading level, for this reason we scale the length of our simulation runs for scenarios with larger session sizes. All of our simulation runs are for 10,000, 20,000, or 40,000 simulated seconds depending on the session size. We found these simulation lengths to be sufficient to ensure consistent simulation results. When reporting any results we discard the data from the first half of the simulation run and calculate all statistics from the data following the warmup period.

4 Point-to-point reservations

The basic reservation model of the ISPN architecture is that of a uni-directional point-to-multipoint stream. However, we begin our investigation by looking at the simplest case, that of uni-directional point-to-point reservations. When a reservation request is blocked the end-user/application does not attempt to requeue the request; that is, there are no reservation request retry attempted.

Our intuition tells us that in a system with acyclic ordering of resources there should be a guaranteed level of successes; as resource demand is increased we expect system throughput to be strictly non-decreasing, and if throughput asymptotes we expect the value to be non-zero. We believe that in order to obtain dependencies among independent reservations that lead to thrashing requires a circularity in the resource dependencies of the individual reservations. Note that under the simple uni-directional point-to-point reservation model this circularity in inter-reservation resource dependencies can occur only if the network itself contains cycles.[3]

[3]We note that in fact in the real world most topologies are likely to contain cycles to avoid the single point of failure problems of a purely acyclic network, however the cyclic resource dependency may still be rare although the possibility is always there.

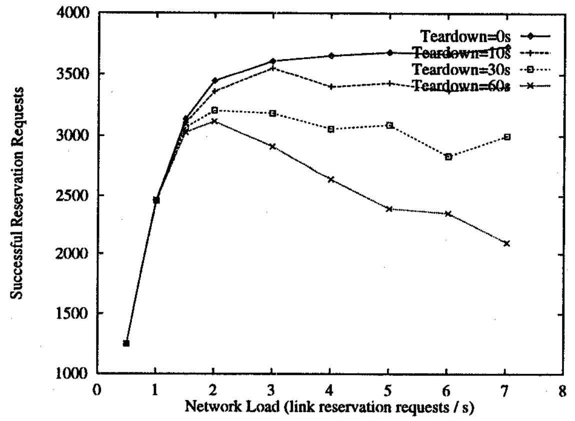

Figure 1: Number of successful reservation requests on the square topology for uni-directional point-to-point reservations.

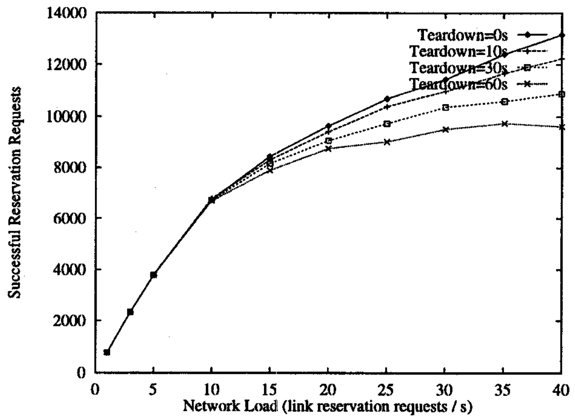

Figure 2: Number of successful reservation requests on the binary tree topology for uni-directional point-to-point reservations.

Figure 1 and Figure 2 present the number of successful reservation requests for simulations of the uni-directional point-to-point reservations on the square and binary tree topologies. The larger size and capacity of the binary tree topology required much higher loading to induce blocking, however the important distinction is in the different shapes of the plots in the two figures. We see that as predicted the cyclic resource dependencies introduced by the cyclic topology can result in a decline in network throughput as the load is increased, while the acyclic topology always shows increasing throughput.

The results in Figure 1 do establish that the thrashing phenomena can occur within the simple uni-directional point-to-point reservation scenario investigated, however the degradation was only observed for extremely long delays and network overload. For systems within reasonable operational ranges it seems quite stable. One situation where these long delays may actually be encountered is within a soft state protocol that requires large timer values to control protocol overhead. The lesson to be learned here is that a soft state mechanism should not rely on timers alone, explicit messages should be incorporated to effect state changes. Soft state timers should only be relied upon to maintain consistency in exceptional cases such as when messages are lost or systems crash. We note that in fact the RSVP soft state reservation protocol does employ this model with explicit teardown messages.

4.1 Point-to-point reservations with blocked reservation retry

In the simple reservation scenario explored in Section 4 we noted that whenever a reservation request was blocked no further action was taken by the end-user/application. An extension to this scenario is to recognize that a common mode of operation might be for the end-user/application to retry its reservation setup request after a short delay in the hope that network conditions have changed in the interim. In this section we investigate the effect of adding reservation request retries to the uni-directional point-to-point reservation model. We arbitrarily selected a reservation request retry interval of one second.[4]

We assume that each reservation request retry attempts to build upon the provisional resource allocation obtained during earlier requests if the reservation is still in place; this leads to two distinct regions of operation. Whenever the teardown delay is greater than the reservation request retry interval, each retry can build upon the previously established partial path reservation. If the teardown delay is less than the retry interval, then the provisional resources have already timed out and each new request must once again contend for resources along every link in the end-to-end path.

Figure 3 presents the number of successful reservation requests for simulations on the square topology for uni-directional point-to-point reservations with a one second reservation request retry interval. It might seem logical that the performance in the region where retries attempt to build upon earlier provisional allocations (i.e., the teardown delay is greater than the retry interval) would be superior to that of the region where each request must contend for new resources at every link, however this is obviously not true. The problem with attempting to build upon the earlier partial path reservation is that once a sufficient blocking level is reached every retry continues to block and the provisional allocation is held forever. Thus we see that system performance for all teardown delays greater than the retry interval is exactly identical, the entire system deadlocks and throughput immediately drops to zero. Note that because the allowable teardown delay is directly dependent on the retry interval, system instability can be induced for arbitrarily small teardown delays by aggressive end-users/applications.

[4]The effect of the selection of the reservation request retry interval is shown to partition the system performance into two distinct operational regions, however it does not affect our results in terms of whether the network can be made to exhibit thrashing.

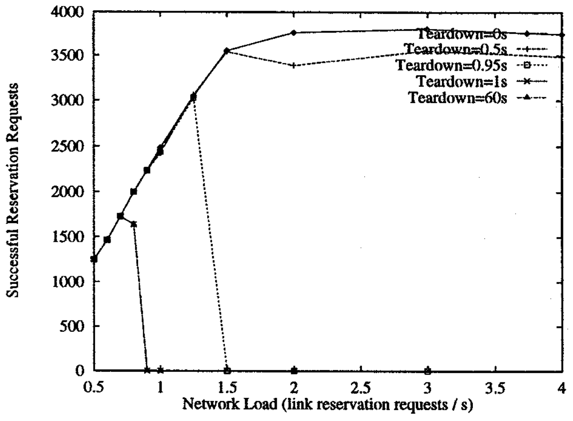

Figure 3: Number of successful reservation requests on the square topology for uni-directional point-to-point reservations with a one second reservation request retry interval.

Network Load	Total Requests	Blocked Requests	Successful Requests
0.7	1725	0	1725
0.8	43929	42293	1636
0.9	577949	577949	0

Table 1: Summary of reservation success and failures on the square topology for uni-directional point-to-point reservations with a one second reservation request retry interval and 60 second teardown delay.

For the operational region where earlier provisional allocations are released the deadlock condition can be avoided. We observe for teardown delays only slightly less than the retry interval thrashing can still occur, while smaller delays result in strictly non-decreasing throughput. This transition from thrashing to non-decreasing throughput is dependent on a number of factors including network topology, capacity and delays, and resource demands. We have not completely modeled this transition phenomena at the present time.

Table 1 and Table 2 summarize the total numbers of reservation request success and failure on the square and binary tree topologies for uni-directional point-to-point reservations with a one second reservation request retry interval and 60 second teardown delay. We see that the total reservation requests behavior is similar under both network topologies. As the net-

Network Load	Total Requests	Blocked Requests	Successful Requests
4	1528	0	1528
5	6057	4146	1911
6	163105	161007	2098

Table 2: Summary of reservation success and failures on the binary tree topology for uni-directional point-to-point reservations with a one second reservation request retry interval and 60 second teardown delay.

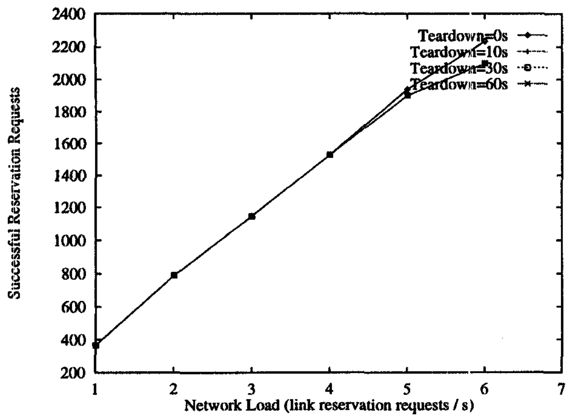

Figure 4: Number of successful reservation requests on the binary tree topology for uni-directional point-to-point reservations with a one second reservation request retry interval.

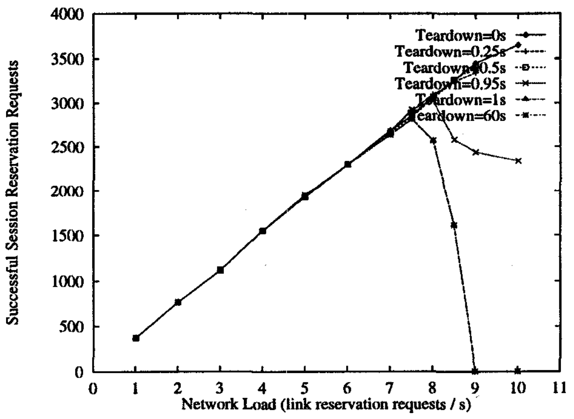

Figure 5: Number of successful session reservation requests on the binary tree topology for 2-way sessions with a one second reservation request retry interval and retry-all-receivers session retry policy.

work load is increased and blocking begins to occur the total number of reservation requests injected into the system begins to increase exponentially. Figure 4 presents the number of successful reservation requests for simulations on the binary tree topology for uni-directional point-to-point reservations with a one second reservation request retry interval. Note that even with the much greater reservation request rate introduced by the retry policy, the system throughput is still non-decreasing. These binary tree results differ significantly from the throughput performance observed for the cyclic topology (see Figure 3) where thrashing was observed as the network load increased. This further re-enforces our initial intuition that the inter-reservation resource dependency circularity is a necessary condition to induce thrashing.

5 Multipoint session models

In Section 4 we looked at scenarios where each source and receiver represented an independent uni-directional point-to-point reservation. In fact, the ISPN reservation model directly supports more complex point-to-multipoint reservations associated with multicast data distribution. Additionally we believe that multipoint-to-multipoint applications may become quite common and these may require coordination in establishment of multiple reservations. We call the set of source and receivers composing a multipoint-to-multipoint application a *session* and assume the existence of a *session controller* element which applies specific policies to the coordinated resource reservation requests.[5]

In the remainder of this section we look at the effects of the session model on the reservation system performance. We assume an N-way conferencing session model with a session controller that requires all reservation requests to succeed before session establishment is completed. We believe this model is most appropriate in capturing the effects of small video teleconferencing sessions.[6] We will quantify the effects of the more complex resource dependencies inherent in the session model and the effects of increased session sizes.

5.1 Bi-directional session model

The simplest extension to the reservation model is to pair two uni-directional point-to-point reservations to form a bi-directional session. That is, if there is a reservation requested from A to B then there must also be a reservation requested from B to A and both requests must successfully complete for the session to be established. This session model is likely to be quite common (e.g., telephone conversations). Note the major implication inherent in this simple session model extension, now for any two independent sessions traversing a link there is an inter-session resource dependency circularity. Our conjecture is that thrashing is now possible even on the acyclic network topology.

Figure 5 presents the number of successful session reservation requests on the binary tree topology for 2-way sessions with a one second reservation request retry interval. We assume that when a session is blocked and provisional resources are to be released (i.e., the teardown delay is less than the retry interval) that the session controller causes both end-points of the session to simultaneously release their provisional allocations. System performance is now similar to that observed earlier in Figure 3 for the uni-directional point-to-point reservations on the cyclic topology. The session model has introduced resource dependency circu-

[5]Note that the multipoint application and session controller issues are actually independent. One could theoretically imagine a point-to-multipoint application with a session controller that required all end-points to succeed; however, such applications appear to be less common.

[6]We recognize that there are other classes of applications, particularly ones with very large membership, that do not have this strict model of success. Analysis of these session models is another area for future investigation.

Network Load	2-way Session	4-way Session	8-way Session
1	0.00	0.00	0.01
5	0.00	0.07	0.17
10	0.13	0.32	0.42
15	0.27	0.51	0.59
20	0.38	0.60	0.65

Table 3: Blocking probability for various N-way session sizes on the binary tree topology for no blocked reservation retry and immediate blocked reservation teardown.

larities, which in turn induces thrashing and system deadlock for teardown delays greater than the retry interval. This is quite significant in the fact that with the introduction of the session model it is now possible to induce thrashing on any network topology, not just those with physical cycles.

5.2 Scaling session size

The simplest session model, as presented in Section 5.1, is the combination of two uni-directional point-to-point reservations to form a bi-directional session, however we expect larger N-way sessions to be quite common too. A consequence of these larger session models is that each source must now establish a multicast distribution tree to each of the other $(N-1)$ session members, and N of these reservations must be obtained before session establishment is completed. Even with the increased efficiency in multicast distribution over multiple point-to-point connections, the effect of the larger session sizes is to greatly increase the total number of link reservations required for session establishment. For example, for all combinations of 2-way sessions on the binary tree topology we find that the average number of link reservations required for session establishment is 13, for 4-way sessions this increases to 54, and for 8-way sessions 172 link reservations are required.

The net effect of larger session sizes is that each session arrival: 1) represents a larger independent resource demand and 2) involves many more admission control decisions, failure of any one can result in blocking of the entire session. For these reasons we believe that as the session size is increased the probability of session blocking will also increase for a fixed network loading level. Table 3 shows the blocking probability for various N-way session sizes on the binary tree topology when there are no blocked reservation retry attempts and immediate blocked reservation teardown. As expected, the blocking probability is significantly increased for fixed load level as the session size is increased.

In the remainder of this section we look at the effects on system performance of this higher blocking probability for larger sessions.

5.2.1 Retry-all-receivers session retry policy

As noted in Section 5.1 during simulation of the 2-way sessions, one possible session retry policy is for the session controller to force all receivers to simultaneously release their provisional resource allocations after a session block. We call this session retry policy *retry-all-receivers* since the retry policy applies to all session receivers independent of whether their individual requests succeeded.

Figure 6 presents the number of successful session reservation requests on the binary tree topology for 4-way sessions with a one second reservation request retry interval and retry-all-receivers session retry policy. We see that the shape of the plots are similar to those observed for the 2-way sessions in Figure 5, that is both 0.95 and 1 second teardown induce thrashing while the smaller teardown delays do not. However, note the effect of the higher blocking probabilities in the 4-way sessions, the onset point for thrashing has been significantly reduced. For a one second teardown delay we see that the maximum network loading has been reduced from a load level of 8 down to a load level of 4.

An interesting observation was discovered after histogramming the tree depths at which blocking was occurring. As the session size is increased the total number of flows traversing the network backbone links significantly increases. This would lead one to believe that in a homogeneous network, such as the one we simulated, the backbone becomes more of a bottleneck as the session size is increased. Further, one might as-

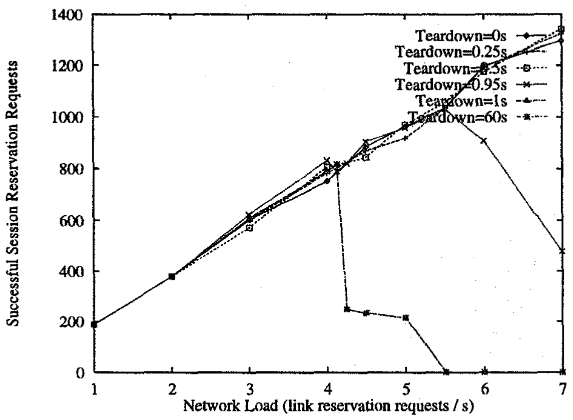

Figure 6: Number of successful session reservation requests on the binary tree topology for 4-way sessions with a one second reservation request retry interval and retry-all-receivers session retry policy.

Tree Depth	2-way Session	4-way Session	8-way Session
1	0.16	0.00	0.00
2	0.45	0.20	0.07
3	0.33	0.61	0.72
4	0.06	0.19	0.21

Table 4: Session blocking probability at thrashing onset point for various tree depths and N-way session sizes with a one second reservation request retry interval, one second blocked reservation teardown delay and retry-all-receivers session retry policy.

sume that scaling the capacity of links closer to the backbone would be beneficial in maintaining uniform utilization levels throughout the network. In fact, we found that the opposite is true. Table 4 shows the percentage of session reservation requests blocked at the various tree depth levels (level 1 is at the root of the tree while larger values of tree depth are towards the leaves) for different N-way session sizes at the network loading level that induced thrashing. We see that as the session size is increased each receiver requests a larger aggregate reservation on its local access links, pushing the bottleneck and associated blocking out towards the leaves, and thereby reducing blocking on the backbone links.

5.2.2 Retry-blocked-receivers session retry policy

An alternative to the retry-all-receivers session retry policy investigated in Section 5.2.1 is to incorporate a session controller that coordinates reservation retry for only those session receivers that had their previous reservation request blocked, all successful receivers maintain their resource allocations while waiting for the other receivers to complete session establishment. We call this session retry policy *retry-blocked-receivers*. The retry-blocked-receivers session retry policy might appear beneficial to developers of multipoint applications because, as we noted earlier, as the session size increases it becomes much more difficult to establish a complete session as a single request. Retry-blocked-receivers eliminates the need for all session receivers to re-contend for resources for every retry request, while incrementally adding receivers to complete session establishment.

Figure 7 presents the number of successful session reservation requests on the binary tree topology for 2-way sessions with a one second reservation request retry interval and retry-blocked-receivers session retry policy. Although the retry-blocked-receivers session retry policy intuitively seemed better suited to establishing multipoint sessions than the retry-all-receivers policy, we see this is not true. Similar to the effect observed in the uni-directional point-to-point retry model (see Section 4.1) where teardown delays greater than the retry interval led to system deadlock, we see that the effect of successful receivers holding onto their resource allocations is to significantly increase the level of provisional resources, resulting in further system degradation. In fact, now we see that even if there are no delays in blocked reservation teardown for those receivers performing retries, the system deadlocks. The exact same effects were observed in the 4- and 8-way session simulations with the incorporation of the retry-blocked-receivers session retry policy.

The performance results presented in Figure 7 show that the retry-blocked-receivers session retry policy is very detrimental to the network. However, the retry-blocked-receivers session retry policy is very beneficial to the individual sessions that adopt it. The incremental addition of session receivers always results in fewer retries, and therefore lower session establishment delays, than retry-all-receivers.

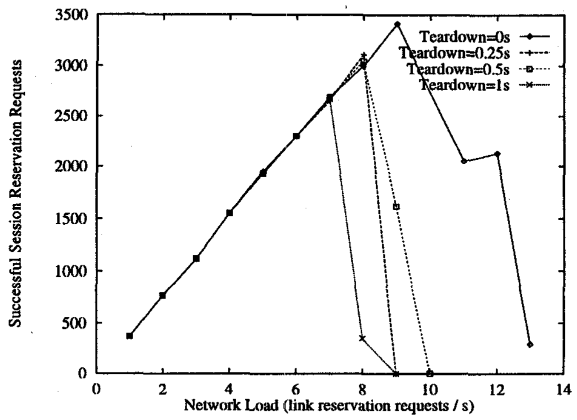

Figure 7: Number of successful session reservation requests on the binary tree topology for 2-way sessions with a one second reservation request retry interval and retry-blocked-receivers session retry policy.

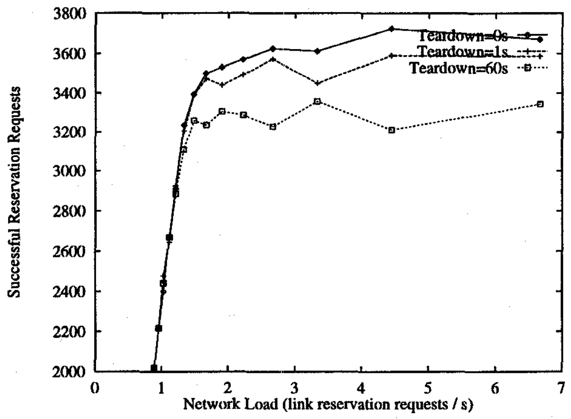

Figure 8: Number of successful reservation requests on the square topology with uni-directional point-to-point reservations with exponential retry backoff.

6 Reservation retry backoff

In the previous sections we have seen that reservation thrashing can occur due to the backlog of users retrying their reservation requests. Previous work on arbitrating access to shared resources (e.g., Ethernet CSMA/CD, TCP congestion control) has shown an exponential retry backoff policy to be highly effective in improving system stability.

In this section we investigate the effects on reservation system stability when an exponential retry backoff policy is added to the reservation retry scenarios explored in earlier sections. Our model for incorporating exponential reservation retry backoff is to assume a set of cooperating end-applications each of which doubles its retry interval timer each time a blocked reservation request indication is received.

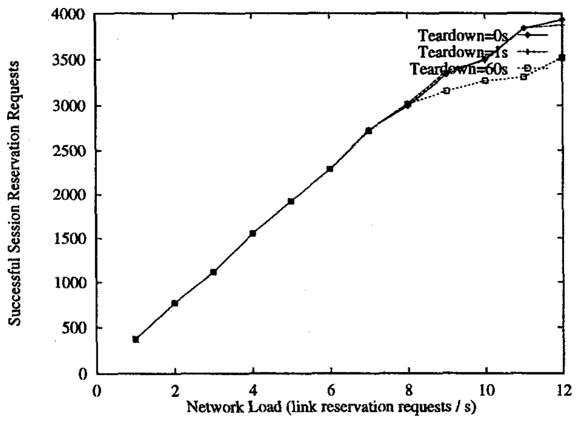

Figure 9: Number of successful session reservation requests on the binary tree topology for 2-way sessions with exponential retry backoff and retry-all-receivers session retry policy.

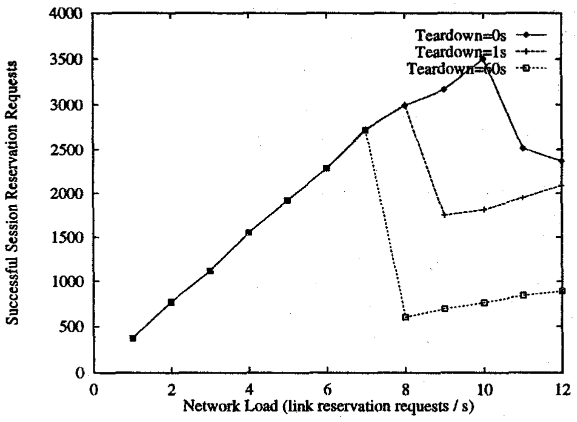

Figure 10: Number of successful session reservation requests on the binary tree topology for 2-way sessions with exponential retry backoff and retry-blocked-receivers session retry policy.

6.1 Unidirectional point-to-point reservations with exponential retry backoff

Figure 8 presents the number of successful reservation requests for simulations on the square topology with uni-directional point-to-point reservations and the exponential retry backoff. Comparing the results to those found earlier for the similar scenario with fixed retry interval (see Figure 3) we see that the retry interval backoff has a dramatic effect on improving system stability. With the retry backoff in place, once a reservation request begins to block the retry interval is quickly pushed to a level greater than the teardown delay, thus eliminating the deadlock problem. Interestingly, measurements of the average setup delays showed both scenarios to be nearly equivalent. The total number of reservation request retries was significantly lower with the backoff, however the exponential increase in retry interval resulted in virtually identical elapsed times.

6.2 Session reservations with exponential retry backoff

Figure 9 presents the number of successful session reservation requests for simulations of the binary tree topology with 2-way sessions, the exponential retry backoff and retry-all-receivers. Once again when compared to the earlier results for the similar scenario with fixed retry interval (see Figure 5) we see a substantial improvement in system stability. Throughput is now strictly non-decreasing even for very long teardown delays. In fact, for reasonable teardown delays (e.g., less than 1 second) we see that throughput is almost identical to the immediate teardown case in Figure 5. We also saw similar results for the larger session sizes.

Figure 10 presents the number of successful session reservation requests for simulations of the binary tree topology with 2-way sessions, the exponential retry backoff and retry-blocked-receivers. We see that in this case the exponential retry interval backoff is insufficient to stabilize the system. The problem here is that the backoff policy is only applied to those receivers performing retries. Once a receiver has successfully obtained its reservation request it continues to consume resources while any remaining session members are still attempting to obtain their requested reservations. This accumulation of provisional resources is sufficient to interfere with the requests of additional session arrivals.

7 Summary

We wish to stress once again that the current results presented are just an initial study. However, we think that our simple progression of reservation scenarios uncovers a number or interesting dynamics introduced into the ISPN architecture by resource reservations and multipoint applications.

Our earliest results establish that even the simplest reservation scenarios with no blocked reservation retries can exhibit throughput degradation. Because of the excessively long delays required this result most likely pertains only to those developers of soft state protocols, who should heed the advice to incorporate explicit state change messages. We also found that the larger resource demand and number of admission control decisions as N-way conference session size is increased results in significant decreases in maximum throughput.

The primary findings of our study is related to the effects the end-user/application behavior can exert on the network stability. It is reasonable that an application be allowed to retry its reservation request after a failure, however we found that the retry interval selected has a direct influence on the allowable delays before entering the thrashing region. Aggressive applications can induce thrashing in a system with arbitrarily fast reservation setup and teardown. The retry-blocked-receivers session retry policy provides another example of a behavior that is advantageous to an application but quite detrimental to the network. We showed that retry-blocked-receivers also induced thrashing for arbitrarily small delays.

We assumed an environment with cooperative applications employing an exponential reservation request retry backoff and found significant improvement. Exponential backoff resulted in improved system stability, maximized throughput, and setup delays consistent with the non-backoff scenario. We did find that the exponential backoff was not sufficient to overcome the negative effects of every possible user behavior, the retry-blocked-receivers still showed significant degradation.

We see a number of directions in which this work can be extended to expand our understanding of the subject. A few of the topics we are considering for future investigation include:

- In the current work all significant delays are attributed to the teardown delay. In large networks end-to-end propagation and admission control may also introduce significant delays. We would like to quantify the effects these other delays have on network throughput. We have looked at several scenarios exhibiting significant reductions in throughput, but no deadlock. Are there scenarios that lead to thrashing?

- We were surprised by earlier results showing very similar performance for the 4- and 8-way session models. We believe this may be an artifact of the dense distribution of session members across the 16 leaf nodes in the current network. We plan to look at larger networks with more sparse membership distributions to see if this differentiates the performance of the session models. We also recognize that the N-way success session model does not scale to very large groups, and there are many applications that require only a subset of members to succeed. We would like to investigate the effects of further scaling session size and the partial success model.

- Provided the retry-blocked-receivers session retry policy is beneficial from an applications point of view, it may be useful to consider a hybrid retry policy incorporating both the exponential backoff and a session idle threshold. Under this scheme after a specified number of failed reservation request retry attempts, all session receivers would initiate retry in an attempt to preempt any deadlock conditions. What is the effect of this hybrid scheme on system stability, throughput, and session setup duration?

We have investigated the problem of thrashing from a technical perspective; however, there is an important underlying incentive issue. If we were looking at this as a unified design problem, where we could design the behavior of end users as well as the network, then there is little question that we could easily prevent thrashing. When sessions use the retry-all-receivers policy, and the retry interval is significantly longer than the teardown delay, we never observed thrashing. The problem is that sessions can determine their own session retry policy, and own retry timing, and these decisions need not be taken with the overall health of the network in mind. The retry-blocked-receivers session retry policy yields lower delays for the individual sessions adopting it. Reducing the retry interval also lowers delays. The result of each session optimizing its own performance is to send the network into a thrashing state. The ultimate question, therefore, is whether it's possible for the network to actually proscribe specific user behavior by employing incentive mechanisms, or by isolating individual sessions from each other similar to that done by Fair Queueing [6] for transport data streams.

References

[1] J. Akinpelu. *The Overload Performance of Engineered Networks With Nonhierarchical and Hierarchical Routing,* In **AT&T Bell Labs Tech. Journal**, Vol. 63, No. 7, 1984.

[2] R. Braden, el al. *Integrated Services in the Internet Architecture: an Overview,* In RFC-1633, June 1994.

[3] R. Braden, et al. *Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification,* In **Internet Draft draft-ietf-rsvp-spec-08.txt**, Nov. 1995.

[4] D. Clark. *The Design Philosophy of the DARPA Internet Protocols,* In **Proceedings of ACM SIGCOMM '88.**

[5] D. Clark, S. Shenker, and L. Zhang. *Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism,* In **Proceedings of SIGCOMM '92**, pp. 14-26, 1992.

[6] A. Demers, S. Keshav, and S. Shenker. *Analysis and Simulation of a Fair Queueing Algorithm,* In **Internetworking: Research and Experience**, Vol. 1, pp. 3-26, 1990.

[7] D. Ferrari and D. Verma. *A Scheme for Real-Time Channel Establishment in Wide-Area Networks,* In **IEEE JSAC**, Vol. 8, No. 3, pp. 368-379, April 1990.

[8] J. Hyman, A. Lazar, and G. Pacifici. *Real-Time Scheduling with Quality of Service Constraints,* In **IEEE JSAC**, Vol. 9, No. 9, pp. 1052-1063, September 1991.

[9] M. Maekawa, et al. **Operating Systems Advanced Concepts**, Benjamin/Cummings, pg. 139, 1987.

[10] A. Parekh. *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks,* In **Technical Report LIDS-TR-2089**, MIT, 1992.

[11] C. Partridge. *A Proposed Flow Specification,* In **Internet Request for Comments**, RFC-1363, September 1992.

[12] J. Peterson and A. Silberschatz. **Operating System Concepts**, Addison-Wesley, pg. 261, 1983.

[13] S. Shenker, D. Clark, and L. Zhang. *A Scheduling Service Model and a Scheduling Architecture for an Integrated Services Packet Network,* preprint, 1993.

[14] S. Sibal, and A. DeSimone. *Controlling Alternate Routing in General-Mesh Packet Flow Networks,* In **Proceedings of ACM SIGCOMM '94**, pp. 168-179, October 1994.

[15] Y.C. Tay. **Locking Performance in Centralized Databases**, 1987.

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

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