Report Number: WUCS-01-06

2001-01-01

DRES: Internet Resource Management using Deferred Reservations

Samphel Norden

In this proposal, we consider the problem of resource reservation for Integrated Services (IntServ) and Differentiated Services (DiffServ) networks. Current approaches for resource reservation in INtegrated Service Networks adopt an all-or-nothing approach, where partially acquired resources must be released if resources are not available at all routers on the chosen path. Furthermore, under high load, end-systems must retry requests repeatedly leading to inefficient allocation and increased traffic. We propose a new approach called Deferred REServation (DERS) that substantially improves performance (reduces the overall cell rejection probability and increases link utilization) over the all-or-nothing reservation approach. Call admissibility is increased... Read complete abstract on page 2.

DRES: Internet Resource Management using Deferred Reservations

Samphel Norden

Complete Abstract:

In this proposal, we consider the problem of resource reservation for Integrated Services (IntServ) and Differentiated Services (DiffServ) networks. Current approaches for resource reservation in INtegrated Service Networks adopt an all-or-nothing approach, where partially acquired resources must be released if resources are not available at all routers on the chosen path. Furthermore, under high load, end-systems must retry requests repeatedly leading to inefficient allocation and increased traffic. We propose a new approach called Deferred REServation (DERS) that substantially improves performance (reduces the overall cell rejection probability and increases link utilization) over the all-or-nothing reservation approach. Call admissibility is increased by deferring requests at routers for a limited period of time until resources are available. Simulation results confirm the performance benefits of our approach. We describe mechanisms to predict the defer time that helps to minimize the resources blocked due to partial reservations, and allows alternate routing mechanisms to further enhance performance. We present scalable QoS routing mechanisms that can be integrated with the DRES approach, and require minimum complexity. Finally, we describe an approach to design a topology that realistically approximates and ISP network over which we plan to evaluate the cost-performance benefits of DRES.

DRES: Internet Resource Management using Deferred Reservations

Samphel Norden

WUCS-01-06

April 2001

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130

DRES: Internet Resource Management using Deferred Reservations

Samphel Norden
Applied Research Lab
Department of Computer Science
Washington University in St.Louis

Abstract

In this proposal, we consider the problem of resource reservation for Integrated Services (IntServ) and Differentiated Services (DiffServ) networks. Current approaches for resource reservation in Integrated Service Networks adopt an all-or-nothing approach, where partially acquired resources must be released if resources are not available at all routers on the chosen path. Furthermore, under high load, end-systems must retry requests repeatedly leading to inefficient allocation and increased traffic. We propose a new approach called Deferred REServation (DRES) that substantially improves performance (reduces the overall call rejection probability and increases link utilization) over the all-or-nothing reservation approach. Call admissibility is increased by deferring requests at routers for a limited period of time until resources are available. Simulation results confirm the performance benefits of our approach. We describe mechanisms to predict the defer time that helps to minimize the resources blocked due to *partial reservations*, and allows alternate routing mechanisms to further enhance performance. We present scalable QoS routing mechanisms that can be integrated with the DRES approach, and require minimum complexity. Finally, we describe an approach to design a topology that realistically approximates an ISP network over which we plan to evaluate the cost-performance benefits of DRES.

1 Introduction

Continuous streaming media applications such as video-conferencing, video-on-demand and IP telephony are becoming increasingly popular in the internet. As the usage of the internet grows, packet loss, delay variations and bandwidth scarcity have made the current internet unsuitable for reliable, guaranteed delivery of multimedia services. Real-time applications require resources to be available in advance and must be shielded from transient performance degradation. The traditional best effort Internet thus must be modified to support QoS. A key mechanism to provide guaranteed services is resource reservation. A reservation protocol (e.g.: RSVP [4]) is essential for negotiating resources and for providing QoS guarantees. Integrated Service (IntServ) Networks attempt to perform per-flow resource reservation in order to satisfy the requirements of applications. However, IntServ requires packet classification and per-flow state, adding to the cost of routers. While this cost is considered acceptable for lower speed links, it currently cannot be used for the high speed links in the internet backbone.

Recently, there have been efforts to provide QoS by using Differentiated Services (DiffServ) [5]. DiffServ deals with aggregated flows instead of the per-flow resource reservation approach of IntServ. However, it is expected that in a heterogeneous environment, there will be a co-existence of both IntServ and DiffServ, with DiffServ in the backbone and IntServ at the edges where per-flow resource management can be easily performed [18]. Per-flow management using IntServ ensures QoS for flows sharing aggregated DiffServ pipes. In summary, there needs to be per-flow metering and policing at IntServ routers before they enter a DiffServ backbone. IntServ offers *Guaranteed Services* where resources are dedicated and *Controlled Load* which attempts to bound the network load. Current DiffServ approaches either over-provision (*Premium service*) where a dedicated pipe of fixed bandwidth is reserved between two edge nodes or provide statistical assurances (*Assured service*). Hence, it is possible to have an end-to-end resource reservation problem that spans both IntServ and DiffServ networks.

When using per-flow based resource reservation for QoS, traditional per-flow resource reservation schemes use a static, all-or-nothing approach to reserve resources for flows. For example, RSVP or ATM UNI [19] based signalling mechanisms either obtain resources at all routers in a route, or tear down the connection even if a single router does not have sufficient resources at the time the request is processed. Even if resources become available immediately afterward, the reservation is not admitted and the user (or the end system) is not informed of that fact. Not even information about the potential future availability is returned to the user, to give an estimate on when to retry the request. Under heavy load, this would require the end-user to retry the request repeatedly, requiring the network to propagate and process this flood of signalling message, increasing the network processing overhead. Moreover, signalling messages are typically assigned high priorities, which could potentially disrupt other services.

In this paper, we study an alternative approach to resource allocation called *Deferred REServation (DRES)*, which enables routers to defer rejection of reservation requests when there are insufficient end-to-end resources. This approach has a real world parallel in the "call-waiting" feature of telephone networks that allow a new call to be "held" until an existing call terminates. The defer period is intelligently computed by the routers so as to increase the probability of a successful reservation, taking advantage of the available knowledge of the network state. We show by simulation that our protocol performs significantly better than the all-or-nothing approach, offering a substantial improvement in terms of link utilization, and call admission probability, especially under network overload. We also propose the use of this protocol for setting up aggregate DiffServ pipes.

The rest of this proposal is organized as follows. Section 2 motivates DRES, and describes the network model and the basic DRES approach. Section 3 discusses performance of DRES for *bursty request arrivals* and shows that DRES can be especially beneficial in this case. Preliminary simulation results are presented in Section 5. Section 4 presents an analytical model for DRES that provides a closed form solution to the call rejection probability and approximates DRES at high loads, for a single link. Section 6 presents the critical issues that need to be evaluated in a distributed admission control protocol protocol based on DRES. This section investigates issues of *Deadlock, Partial Reservations, Fairness, and QoS routing.* Section 7 introduces a refinement of the basic DRES approach that allows for early rejection of deferred reservations that have a high probability of failure. Section 8 investigates the impact of DRES with early rejection on *QoS Routing.* We then present our plan for evaluating DRES on a realistic network topology intended to approximate a typical ISP network in Section 9. Section 10 presents related work in this area. We present an approximate timeline in Section 11 and conclude in Section 12.

2 Deferred Reservation

2.1 Motivation and Problem Statement

To reiterate, standard resource reservation approaches using ATM signalling or RSVP, attempt to reserve resources at all hops on a path, failing which, the reservation is rejected. We seek to improve performance by not rejecting but deferring a request when resources are insufficient, so as to allow the request to be admitted at a later time. We will motivate the concept of deferred reservations with two specific examples of *video-on-demand* and *handoff in cellular networks* that highlight the benefits of deferring.

Video-on-Demand: Consider a scenario where a Video-on-Demand session from a server site to a user. Ideally, resources are available and will be granted at the time of reservation. If the network currently is overloaded,a large fraction of requests will be rejected. Instead of not getting any access at all, the user would most likely prefer to delay the start of the session for a short period of time[1]. In conventional networks, retrying is left to the user: This is inconvenient both to the user and to the network (processing and forwarding cost associated with such redundant requests). If the current reservation models were instead upgraded to inform the user that the request could not be satisfied right now, but the network would notify the user and reserve resources as soon as resources became available, then the user and the network would be freed from the overhead of tedious retries, as we will see below.

Mobile Handoff: Besides the connection setup scenario outlined above, roaming in wireless and mobile computing environments can also benefit from deferred reservations. As users of wireless services move from one *cell* to another, there is a hand-off period during which the user connection is transferred from the fading base station, to a stronger base station. At the time a hand-off is initiated, there may be insufficient resources available in the destination cell. Normally, this would mean a dropped call. In a network supporting DRES, placing a reservation to the next cell early allows for reservation deferral, increasing the chance of successful hand-over. The *handoff* must occur within a finite time for the call to continue, and must occur before the signal fades from the original cell. Note that the user cannot determine the optimal point when the connection is handed over since the user has no knowledge of the current load at the base station. Hence, this situation is well suited to deferred reservations. Deferring is orthogonal

[1]Alternatively, the user might start immediately with a downgraded QoS (reduced bandwidth or increased delay) and try to upgrade to a better service during the session. This option, often used in adaptive applications, is still possible and is orthogonal to our approach.

to the CDMA "soft handoff"[2] feature and can be used in combination with it.

From the overall perspective of the user and the service provider, we have the following benefits of deferring:

Network-friendliness: Unlike user or end-system retries, the network is not burdened with additional requests. Also, the router can use information unavailable to the end system, such as past and current network load to improve the call admission rates, as well as reduce processing cost by only trying to admit waiting flows when resources become available, effectively upgrading from a polling-based user or end-system mechanism to an interrupt-driven network service. Deferring also smoothes out congestion or bursty request arrivals by providing resources in a phased manner.

ISP-friendliness: Due to the improved call acceptance rate for the same amount of bandwidth available, the need for ISPs to over-engineer the network is reduced, providing for improved cost-effectiveness. This is especially true for Virtual Private Networks (VPNs).

User-friendliness: Users will not need to manually retry. In addition, immediate reply from the network assures minimal setup delay compared to user or end-system polling.

Figure 1 shows a space-time diagram that highlights the essential difference between the traditional (called NDRES for No Deferred REServation) and DRES modes of reservation. Two reservation requests arrive as shown. Both DRES and NDRES allow req_1 to reserve resources. However, in NDRES, req_2 fails at hop_2 and the resources are released. With DRES, the reservation is deferred for a period and is able to obtain resources after a short delay, allowing req_2 to be completed. Specifically, deferring helped since resources that were allocated to some existing flow were released after the flow terminated. The defer period is a function of a user defined QoS constraint and a value that is predicted using the current network state. An NDRES reservation would have to repeatedly poll for resources until it succeeds. This leads to high overhead for the end-user and increased traffic on the network, as well as poor utilization. We now present the baseline DRES algorithm.

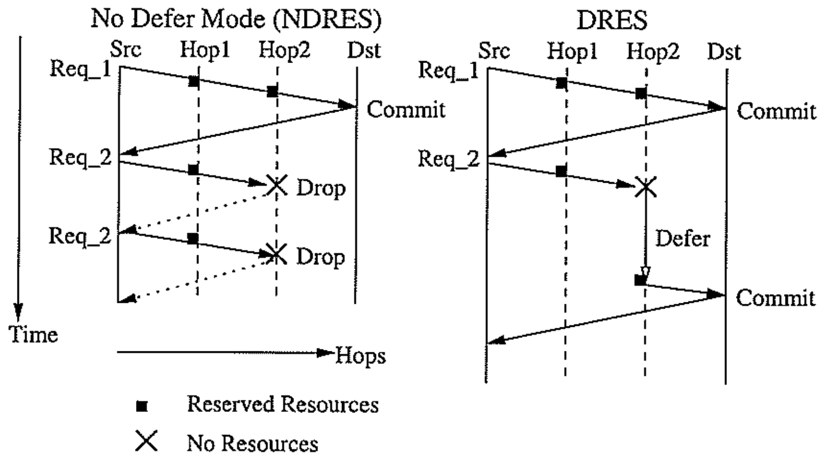


Figure 1: NDRES vs DRES

2.2 Basic DRES Protocol

DRES is a 2-phase resource reservation protocol. For the rest of this discussion, we will assume that the QoS for a flow is characterized by bandwidth. The key idea of DRES is to use deferring (delaying) as a mechanism to accommodate flows that can tolerate initial set-up delays. When a user requests a new connection, the router first locally verifies if sufficient link bandwidth or capacity is available. Typically, link bandwidth can either be in a committed (allocated) or a free state. DRES adds a third state: "reserved" where capacity is allocated at local routers but there is no end-to-end commitment.

[2]As the mobile nears the boundary of a neighboring cell, it receives transmissions from both cells. The mobile will receive some transmission from one cell, and some from the other until it has moved into one or the other cells.

```
A. Request Arrival $req_i(B_i)$
        IF $(B_i > L_{free})$ set $t^i_{defer}(See Section\ 7)$;
        ELSE
            Allocate $B_i$; Forward $req_i$
            If (Last Hop Router)
                Send $P_{ack}$ to source;
B. Timer Expiry: // $t^i_{defer} == 0$
        Send $N_{ack}$ back to source;
        Release resources at all hops along path;
C. Call Expiry:
        Release resources at all hops;
        Admit jobs from deferred queue;
```

Figure 2: Base DRES Algorithm

When the bandwidth requested (B_i) is not available at an intermediate link, a request is not rejected. Instead, it is queued and deferred at that hop. By delaying requests, DRES can improve the admission probability, since it is likely that by waiting, resources can be committed to the request due to the expiry of an ongoing reservation, or due to the freeing of "reserved" resources. Deferred flows and reserved resources are flushed after a timeout period.

The defer period (t_{defer}) is the key parameter and can be viewed from a user and a network perspective. The user could specify t_{defer} as a QoS constraint. Typically, the longer one defers a request, the better the chance of it being admitted, assuming there are no deadlocks. However, users typically will not be willing to wait for an unbounded time period and will give up on a request that is delayed for too long. There are two perspectives with respect to the defer bound t_{defer}. We can assume that the defer bound is known in advance. However, our results can be applied to situations where reservation requests are maintained until users abandon them. Alternatively, the network operator can set t_{defer} as a matter of policy. The value may vary with network loading conditions or as a function of reservation characteristics or the user's service class.

Description: Figure 2.2 defines the behaviour of routers implementing the basic DRES protocol along a given reservation path. Arrival of a request (Event A) results in verifying that the request can be admitted. If an end-to-end reservation is possible, then a positive acknowledgement P_{ack} is sent by the last hop router. If not, the request is deferred. When a router receives a P_{ack}, it forwards the request all the way to the source confirming the reservation. If a timer expires (Event B), a negative acknowledgement (N_{ack}) is sent. If an existing admitted call expires (Event C), then we perform admission control for the first deferred request. L_{free} denotes the available link capacity. The same algorithm is executed independently at every node, and information is only exchanged by the reservation request, acknowledgement, and release messages described above.

The order in which requests are processed is based on the defer time or the deadline. Thus, the most critical request is processed first. The use of *timing wheels* allows an efficient implementation approach where requests can be ordered based on the defer timer value indicating the amount of time remaining in which a decision must be made on admitting/rejecting the flow. While this prevents the Head-Of-Line[3] blocking problems of a FIFO queue, there are still certain situations where fairness is not assured. We discuss these issues in Section 6.

Discussion: If resources were always available then there would be no need for deferring. This would be the case if the network were over-provisioned. However, over-provisioning is wasteful. Deferring allows a network to be able to still provide acceptable QoS in spite of transient overloads, as can be seen in the simulation results subsequently presented. The primary contributions of DRES are: *improved admissibility, higher link usage, and lower latency (response time) by deferring in parallel across all routers*, when compared to NDRES. Admissibility is improved by deferring flows that would have been otherwise rejected by standard admission control procedures.

[3]The head of the queue (a large request) cannot be accommodated and blocks subsequent smaller requests that could be admitted

3 Bursty Request Arrivals

In this section, we describe a particular situation in which DRES could have a large impact. Specifically, we show that with bursty request arrivals, DRES can provide significantly better performance than the no deferring (NDRES) approach. An example of such a scenario is when a new music video is made available on-line creating a burst of requests. Consider Figure 3 which shows a timeline indicating the arrival of the bursts. The behaviour of DRES and NDRES are shown below. We claim that there could be as much as 50% improvement in terms of the number of admitted flows by DRES over NDRES, if a few constraints are satisfied. Essentially, we would like the bursts to be synchronized to behave like a specialised ON/OFF source.

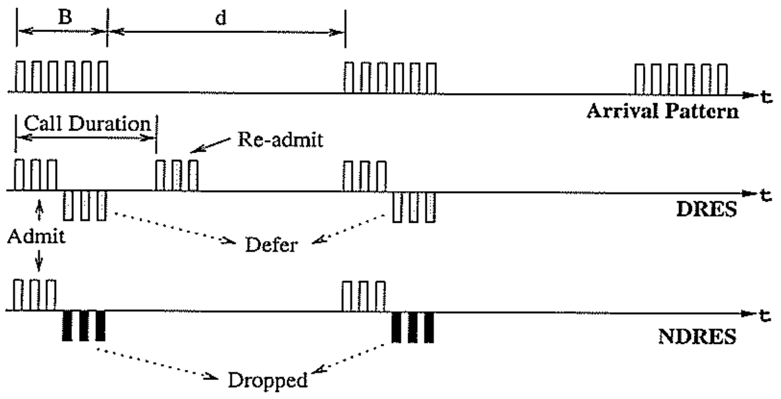


Figure 3: DRES vs NDRES: Bursty traffic behavior

In Figure 3, we note that the flow reservation requests follow the bursty pattern. Initially, on the time line, there are six such requests in a burst each requiring say 30% of the link bandwidth, such that only three can be accommodated at a time. The NDRES style reservation mode will result in three getting accepted and the other three getting rejected as shown in the third time line. This gives us a 50% *admission rate* (fraction of accepted flows over all flow arrivals). However, with DRES, we can achieve 100% *admission rate* by deferring as shown. Since only three flows can be admitted, the other three flows in the burst will be deferred as shown by the lightly shaded blocks. Once the initial three flows complete, (the duration of the flow expires), then we can readmit the three deferred flows (denoted by the darkly shaded blocks). The only point to note in this example is that the deferred flows do not occupy the link for a period exceeding *d*. If not, it would interfere with the next burst of arrivals. This could lead to the deferring of all flows of the second burst and this could have recursive effects, increasing the defer time for the subsequent flows.

4 Analytical Model for Single Link

In this section, we develop an analytical model for DRES on a single link and develop a closed formula for the call rejection probability. We then use the model to obtain numerical results for the analytical model and compare them to results from simulations. We are simulating an M/M/n/k model with exponential interarrival times and exponential holding times having a fixed number of servers with a bounded storage capacity. Deferring is captured by including a buffer which is used to hold deferred flows. The model is shown in Figure 4.

λ_i	Arrival Rate when the population of size i
μ_i	Service Rate for a population of size i (1/MHT)
p_i	Probability of being in State i
n	Number of Servers ($\frac{1}{LF}$)
k	Size of Queue (To simulate Deferring)
T_d	Target defer time

Table 1: Notation

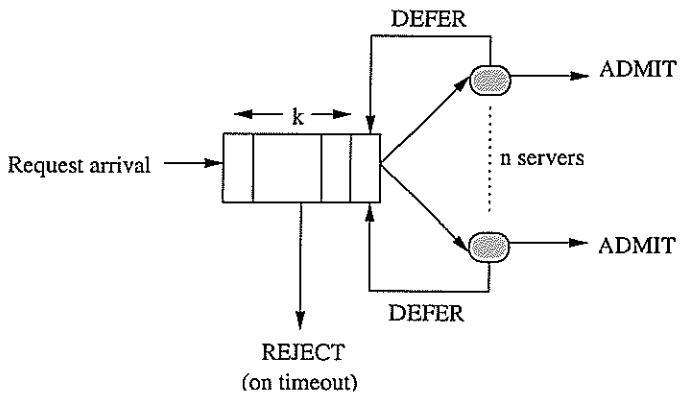


Figure 4: Queueing Model

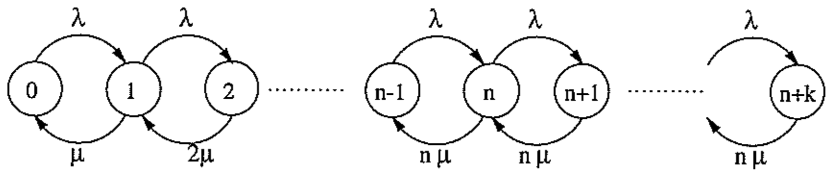


Figure 5: State Transition Diagram for M/M/n/k Markov chain

The finite M/M/n/k markov chain (shown in Figure 5) that represents the model has the following characteristics, explained in Table 1.

$\lambda = \frac{OL \cdot NC}{MHT}$ (See Equation 10) ; $\lambda_i = \lambda$; $\mu_i = \mu \times min\{i, n\}$; $\mu_{i<1} = 0, \lambda_{j<0} = 0$; $k = \lceil T_d \times n \times \mu \rceil$

There are n servers to accommodate flows, and an additional buffer of k to accommodate deferred flows. The maximum size of the buffer k is bounded by the maximum number of requests that arrive during the defer bound (T_d) for a request. Since there are n servers, the maximum rate of arrival/departure is $n \cdot \mu$. Hence, the buffer size is bounded by $T_d \cdot n \cdot \mu$. Requests are dropped when the buffer is full or reaches the state $n + k$ in Figure 5. Our goal is to develop a closed form for the Call rejection probability, which is given by p_{n+k}.

For such a model, we can derive the probability of being in state i (or having a population of size i using standard queueing theory results [13] as:

$$p_m = p_0 \cdot \prod_{i=0}^{m-1} (\frac{\lambda_i}{\mu_{i+1}}) \tag{1}$$

$$= p_0 \cdot \prod_{i=0}^{m-1} (\frac{\lambda}{\mu \cdot min\{i+1, n\}}) \tag{2}$$

Thus we get:

$$p_m = \begin{cases} p_0 \cdot (\frac{\lambda^m}{\mu^m}) \cdot \frac{1}{m!}; & m \leq n \\ p_0 \cdot (\frac{\lambda^m}{\mu^m}) \cdot (\frac{1}{n^{m-n}}) \cdot \frac{1}{n!}; & m > n \end{cases} \tag{3}$$

where:

$$p_0 = \frac{1}{1 + \sum_{m=1}^{n} \frac{(\lambda/\mu)^m}{m!} + \frac{1}{n!} \cdot \sum_{m=n+1}^{n+k} (\frac{(\lambda/\mu)^m}{n^{m-n}})} \tag{4}$$

4.1 M/M/n^* Variant

In this section, we will discuss a variant of the above which we call the M/M/n^* where infinite buffers are present, modeling exponential defer times. The state transition rate diagram is shown in Figure 6. This resembles Figure 5

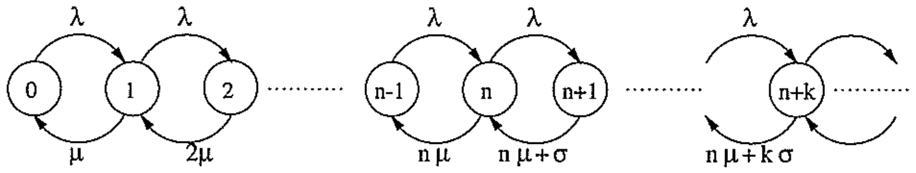


Figure 6: State Transition Diagram for M/M/n^* Markov chain

upto state n. However, once the queue occupancy exceeds n (number of flows that can simultaneously be accommodated), the processing rate for state $n+i$ increases to $n \cdot \mu + i \cdot \sigma$ where $\sigma = \frac{1}{T_d}$. Note the M/M/∞ model with infinite servers where the service rate increases linearly as $(n+i) \cdot \mu$. Since a deferred request will remain in the queue for upto T_d, the processing rate of deferred requests assuming the expiry of the defer timer is $\frac{1}{T_d}$. However, each deferred request has its own individual deferred bound. Thus, i deferred requests are processed at a rate of $i \cdot \sigma$. This is in addition to the normal processing of link requests at the rate $n \cdot \mu$. Thus, the server rate is augmented with the term $i \cdot \sigma$ to account for deferred requests that are processed on the expiry of the defer timer for other deferred requests. Hence the probability expressions are as follows:

$$p_m = \begin{cases} p_0 \cdot (\frac{\lambda}{\mu})^m \cdot \frac{1}{m!}, & m \leq n \\ p_0 \cdot \frac{(\lambda/\mu)^n}{n!} \cdot (\frac{\lambda}{\sigma})^{m-n} \cdot \prod_{k=1}^{m-n} (\frac{1}{k+\frac{n \cdot \mu}{\sigma}}), & m > n \end{cases} \tag{5}$$

where:

$$p_0 = \frac{1}{1 + \sum_{m=1}^{n} \frac{(\lambda/\mu)^m}{m!} + \frac{(\lambda/\mu)^n}{n!} \cdot \sum_{m=n+1}^{\infty} \prod_{k=1}^{m-n} (\frac{1}{k+\frac{n \cdot \mu}{\sigma}})} \tag{6}$$

We can get a more accurate expression for p_0 in Equation 6 by the following substitution:

$$\sum_{m=n+1}^{\infty} \prod_{k=1}^{m-n} (\frac{1}{k+\frac{n \cdot \mu}{\sigma}}) = \frac{(n \cdot \mu/\sigma)!}{(\lambda/\sigma)^{n \cdot \mu/\sigma}} \cdot [e^{\lambda/\sigma} - \sum_{k=0}^{n \cdot \mu/\sigma} \frac{(\lambda/\sigma)^k}{k!}]$$

The rejection probability is given in terms of the Offered Load (OL), and the Carried Load (CL) as follows:

$$\text{OL} = \frac{\lambda}{n \cdot \mu} \tag{7}$$

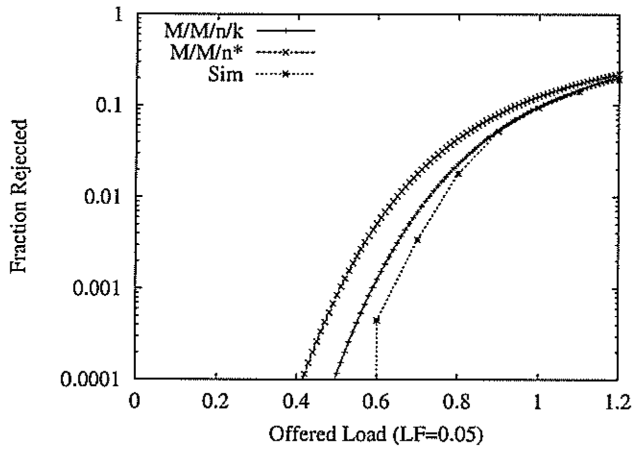
$$\text{CL} = \sum_{i=0}^{n} \frac{i}{n} \cdot p_i + \sum_{i=n+1}^{\infty} p_i \tag{8}$$

$$P_{\text{rej}} = \frac{\text{OL} - \text{CL}}{\text{OL}} \tag{9}$$

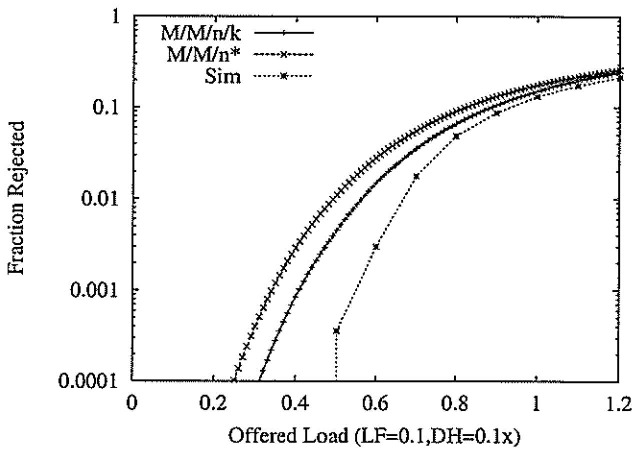
The rejection probability (Equation 9) is the fraction of the offered load that cannot be carried by the link. The carried load is composed of two parts. The first summation gives the percentage of carried load as $p_i \cdot \frac{i}{n}$ for all $i \leq n$. The second summation is for $i > n$, when the link is already carrying n flows. This term captures the deferred flows that timed out freeing up the queue for other requests. Since atleast n flows are carried, the carried load in this region [n+1,∞] is p_i. Since we do not have a closed form solution for the M/M/n^* model, we use a summation of sufficient number of terms to minimize any loss in accuracy.

Discussion: Figure 7(b) plots the comparison graphs between the analytical and the simulation models where requests have constant bandwidths (LF indicating the fraction of the link occupied), with mean call durations of 60 units, and a defer bound of 10 time units. At higher loads, the analytical and simulation models agree but there are differences at lower loads where the simulation outperforms the analytical versions. We translate this difference to the fact that the M/M/n/k analytical model has a finite buffer size. Thus, requests will be dropped in the analytical model in spite of having positive defer timers. At higher loads, this effect is mitigated since deferring has less impact. The requests dropped due to finite buffers in the analytical model can be equated to the requests that are dropped due to the defer timer expiring in the simulation especially since requests are arriving at a higher rate.

At a lower link fraction of 0.05 in Figure 7(a), the performance gap between the models decreases significantly. This is attributed to the fact that the number of servers increases (20 servers as opposed to 10) leading to less requests that need to be queued. Interestingly, the M/M/n^* variant performs slightly worse than the M/M/n/k model. With a decrease in link fraction from 0.1 to 0.05, the number of servers increases from 10 to 20, and there is more buffer space. Essentially, the first summation term in Equation 8 dominates the second summation term. Since the summation term in M/M/n/k goes upto $n + k$ as opposed to n in the summation for M/M/n^* (Equation 8), the M/M/n/k model allows a higher carried load leading to a lower rejection probability. As the link fraction is increased, the k term for the buffer space decreases, and the first summation is less, reducing the gap between the two models as shown in Figure 7(b).



(a) FR vs. OL (LF=0.05)



(b) FR vs. OL (LF=0.1)

Figure 7: Comparing the Analytical/Simulation Models: Single Link

5 Simulation Experiments

In this section, we describe the simulation experiments that were done to evaluate the performance of basic DRES. The experiments are split into two parts. The first set of experiments shows performance results with a simple network topology called the *parking lot topology.* The second part deals with performance on a more realistic IP backbone topology. Users make requests in the form of calls to the nearest active router. Each call is characterized by its *Holding time (duration of call) and requested bandwidth.* The ratio of the defer to hold time (DH) ratio is fixed at $\frac{1}{6}$. Note that each point in a graph is an average over 10 runs. Notation used subsequently is explained in Figure 2.

Load: We compute the Offered Load on a link as follows.

$$OL = \frac{MHT}{MIT \times NC} \tag{10}$$

where we denote MHT as the mean holding time, MIT as the mean packet inter-arrival time, and NC as the number of calls that can be simultaneously accommodated on the link. The key *performance metric* which is commonly used in telephone networks and major ISP's is the blocking or rejection probability which we denote as the *Fraction rejected (FR):* This represents the fraction of flows that were rejected.

Term	Description
DH	Defer to Hold time ratio
LF	Fraction of Link bandwidth allocated to call
NC	Number of calls supported by link
MHT	Mean holding time
MIT	Mean reservtion inter arrival time
OL	Offered Load
FR	Fraction Rejected
D_{end}	End-to-End Defer Bound
MAXTIME	Total Simulation Time

Table 2: Notation and Terminology

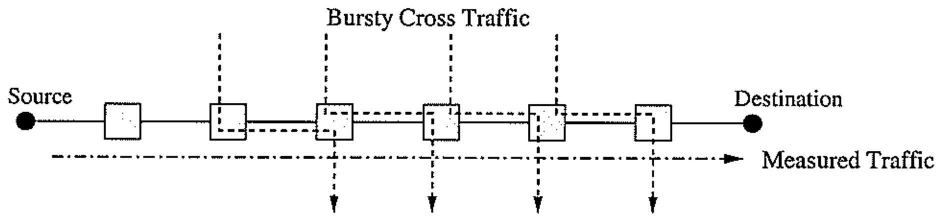


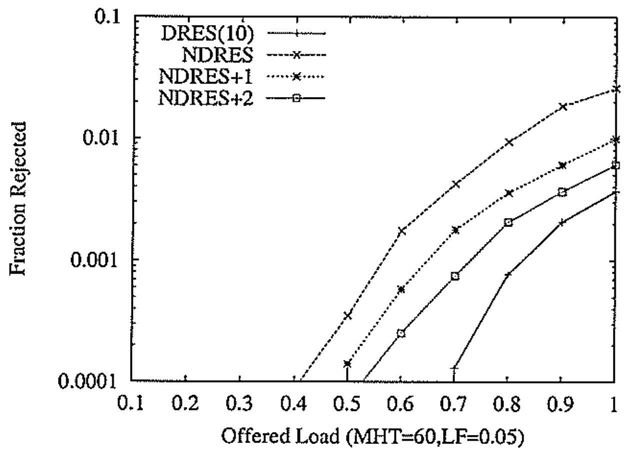
Figure 8: Parking Lot Topology

5.1 Simulation Model: Parking Lot Topology

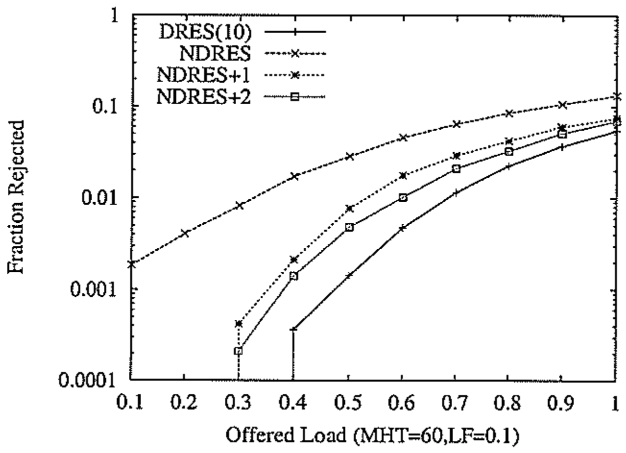
In this section, we will discuss results for the *Parking Lot* topology shown in Figure 8. The topology consists of 6 nodes and the performance is measured for requests going from the leftmost router to the rightmost router (source to destination), which we call the *main-stream requests* for the rest of this section. At each intermediate router, we introduce cross traffic that goes one hop away. The cross traffic is meant to introduce random link overloads and to study the effect of deferring in such a scenario. The holding time of the *main-stream* requests is 60 seconds. The bandwidth requested is a uniform random variable from [0,LF], where LF is the maximum requested bandwidth as a fraction of the total link capacity. The cross traffic requests are generated as in [11], where we have an ON/OFF model with exponentially distributed ON and OFF times. During each ON period, an exponential number of requests are generated with a mean N at a fixed rate of p requests per time unit. The OFF time is an exponentially distributed value with mean I. This gives an average request generation rate of $1/a = I/N + 1/p$. The values of I,N and p are 100,10 and 1 respectively. The duration of the cross traffic requests were 20% of the *main-stream* holding times and the requests bandwidths were uniformly distributed upto an LF of 0.05 or 5% of the link capacity. The choice of these values was to keep the rejection fraction low enough, even at high loads to simulate a realistic operating range with rejection fractions of [10^{-2},10^{-4}]. We compare DRES with NDRES and the NDRES+Retry mode, which we explain subsequently.

NDRES+Retry: We will first explain the NDRES+Retry model. In basic NDRES, the request is rejected if it does not get sufficient resources. However, in NDRES + Retry, the request is rejected after the request has been retried a few times. The number of times the request is retried is determined by the ratio of the D_{end} defer bound and the *retry period.* Thus, after every retry period time elapses, the user retries the request until the total time elapsed since the initial retry exceeds D_{end} which is the end-to-end bound. We chose the bound so as to provide a point of comparison for the DRES approaches and NDRES+Retry. For example, with a $D_{end} = 10$, if there is one additional retry per request, then if the original request is rejected, the user will retry the request after 10 units of time. If there are two retries per request, then the retry period is 5 units of time. The notation for the graphs is as follows:

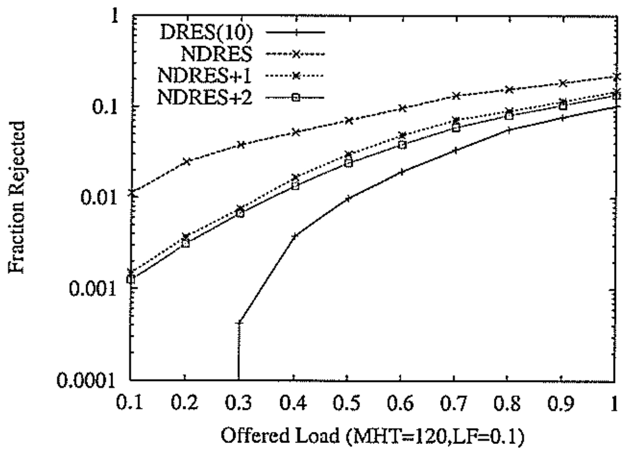
NDRES+x: NDRES with Retry where x is the number of additional requests sent after the original request.



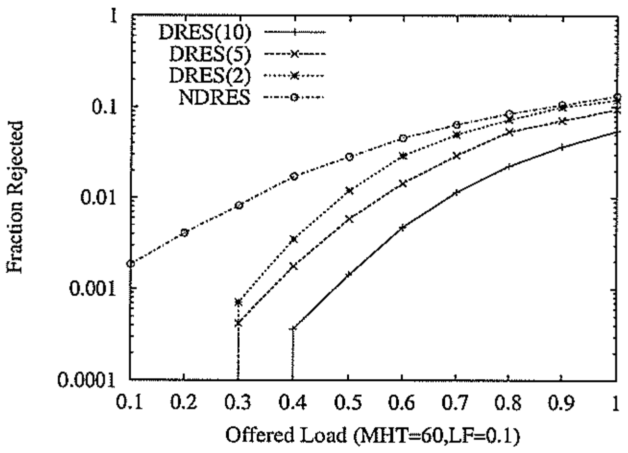
(a) FR vs. OL



(b) FR vs. OL



(c) FR vs. OL



(d) FR vs. OL

Figure 9: Performance Results: Parking Lot Topology

5.2 Experimental Results: Parking Lot Topology

Fraction Rejected vs Load: Figures 9(a)-9(d) show the impact on the rejection fraction for the *parking lot* topology. We compare DRES, NDRES,NDRES+1 and NDRES+2 models. In Figure 9(a), for an maximum bandwidth request fraction of 0.05 and an MHT of 60 time units, DRES (8 $\times 10^{-4}$) has a factor of 12 improvement over NDRES (100 $\times 10^{-4}$), a factor of 3 improvement on NDRES+2 (25 $\times 10^{-4}$) and a factor of 4 (30 $\times 10^{-4}$) improvement on NDRES+1 at a load of 0.8. DRES does not reject any flow upto a load of 0.8, while the NDRES protocols start rejecting requests from loads of 0.4 for NDRES and 0.5 for the Retry variants. From a rejection fraction perspective in Figure 9(a), at a value of $2 \cdot 10^{-4}$, NDRES supports a load of 0.45, NDRES+1 supports a load of 0.53, NDRES+2 a load of 0.6 and DRES supports a load of 0.8 leading to an 80% gain over NDRES, a 50% gain over NDRES+1, and a 33% gain over NDRES+2.

Figure 9(b) shows the rejection fraction for a maximum requested bandwidth fraction of 0.1. With larger requests, the performance of all schemes degrades compared to Figure 9(a). However, DRES (15 $\times 10^{-3}$) still has a factor of 15 improvement over NDRES (250 $\times 10^{-3}$), a factor of 6 improvement over NDRES+2 and a factor of 7 improvement over NDRES+1 at an load of 0.5. As the load is increased, the benefits are reduced. At an load of 0.8, DRES offers a factor of 2 improvement over the Retry protocols and a factor of 5 improvement over NDRES. From Figure 9(b), at a value of $2 \cdot 10^{-3}$, NDRES supports a load of 0.1, NDRES+1 supports a load of 0.4, NDRES+2 a load of 0.45 and DRES a load of 0.55, with a factor of 5.5 gain over NDRES, and gains of 40% and 20% over NDRES+1 and NDRES+2 respectively.

In Figure 9(c), the holding time of the *main-stream* requests is increased from 1 to 2 minutes. Since resources are now held for longer times and the defer time is not adjusted accordingly, the performance benefits of DRES decreases. The DH ratio is now 8% as opposed to 16% in the previous plots (i.e., the defer time bound is 10 with duration of 120). At a load of 0.5, DRES (8×10^{-3}) offers a factor of 7 improvement over NDRES(60×10^{-3}), a factor of 2 improvement over NDRES+2 and a factor of 3 improvement over NDRES+1. Again, from the rejection fraction perspective, at a value of $1 \cdot 10^{-2}$, NDRES supports a load of 0.1, NDRES+1 supports a load of 0.35, NDRES+2 a load of 0.4 and DRES a load of 0.55, with a factor of 5.5 gain over NDRES, and gains of 80% and 40% over NDRES+1 and NDRES+2 respectively.

Figure 9(d) shows the impact of different DH ratios on the performance for an MHT of 60 units. With a DH ratio of 3% ($D_{end} = 2$), DRES offers a factor of 5 improvement over NDRES. As the load is increased, the benefits of DRES(2) decrease and at a load of 0.8, DRES(2) offers negligible improvement over NDRES. DRES(5) is in between the two extremes of DRES(2) and DRES(10) offering a factor of 6 improvement at a load of 0.4 and about 30% improvement at a load of 0.8. figure 9(d) shows that at a value of $2 \cdot 10^{-3}$, NDRES supports a load of 0.1, DRES(2) supports a load of 0.35, DRES(5) a load of 0.4 and DRES(10) a load of 0.55, where DRES(10) has gains of 60% and 40% over DRES(2) and DRES(5) respectively.

5.3 Simulation Model: IP Backbone Topology

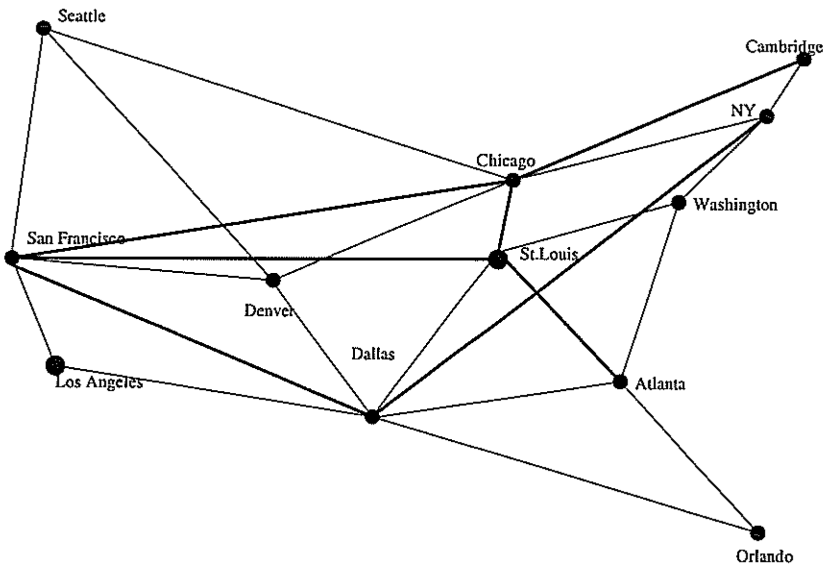
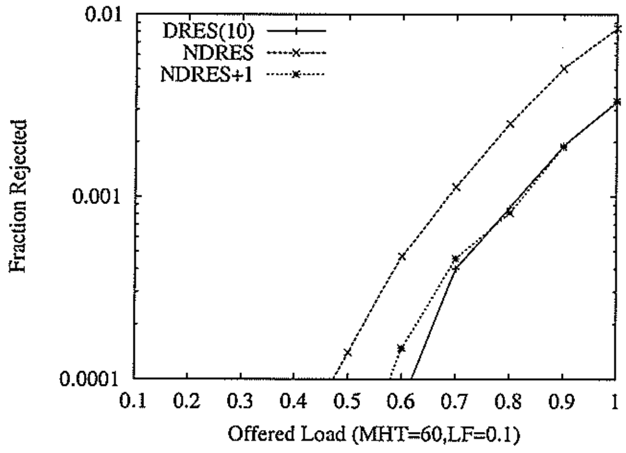


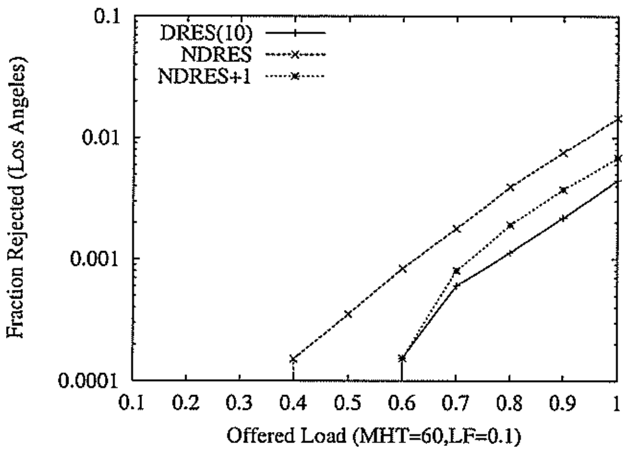
Figure 10: IP Backbone Topology

In this section, we will discuss results for the IP backbone topology shown in Figure 10 which is an approximation of AT&T Worldnet's IP Backbone [30] comprising of 12 core routers. The mean holding time of requests is chosen to be 60 time units. Each node is a potential source. Destinations are randomly chosen using a uniform distribution. Due to the use of shortest path routing (minimum hop count), the load generated is not uniformly distributed in the network which works to our advantage since we want to highlight the benefits of DRES in an unbalanced network. Note that we are not using a QoS routing approach. There are two kinds of links that are shown as thin and thick lines in Figure 10. The thin *access* links represent OC-48 links, while the thick *core* links represent OC-192 links. The manner of assignment of the link capacities emulates a core backbone with OC-192 links that is surrounded by access links of OC-48 capacity. In addition to the performance statistics for the overall network, we will also focus on two nodes that are highlighted in Figure 10 (St.Louis,Los Angeles). Los Angeles is an access node with OC-48 links and will be typically be more saturated than the core node representing St.Louis. Thus, we would expect to obtain better performance from the core node.

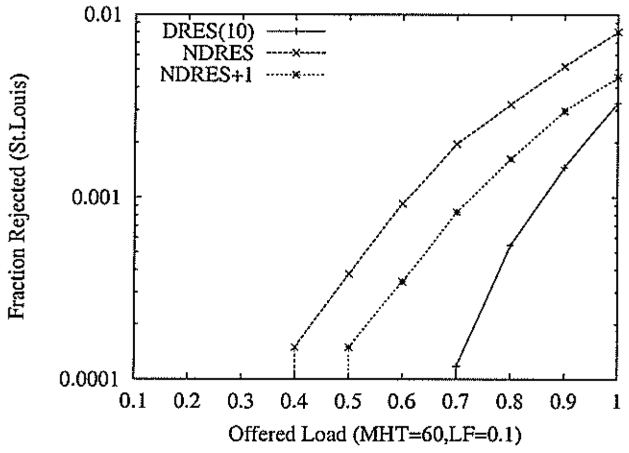
5.4 Experimental Results: IP Backbone Topology



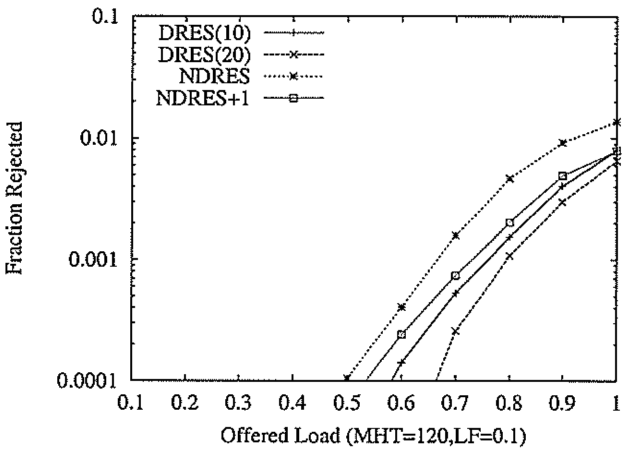
(a) FR vs. OL



(b) FR vs. OL



(c) FR vs. OL



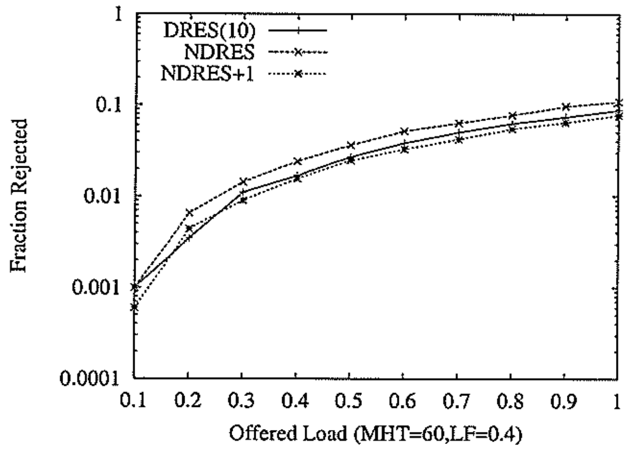
(d) RT vs. OL

Figure 11: Performance Results: IP Backbone (LF upto 10% of OC-48 link)

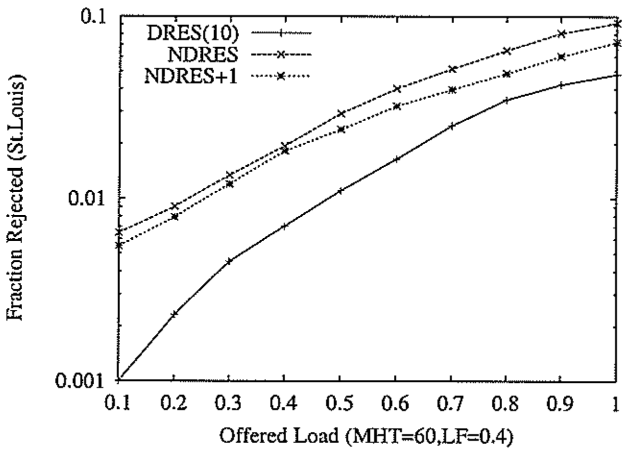
Fraction Rejected vs Load: Figures 11(a)-11(d) show the performance for the IP backbone topology where the maximum requested bandwidth is 10% of the OC-48 link or 2.5% of the OC-192 link, and call durations are for 60 time units. At the outset, it should be noted that the performance gains obtained in the parking lot topology are not comparable to the performance in this topology. The overall FR values include links which are lightly loaded due to the shortest path routing scenarios. Thus, the gains when averaged over all links will be reduced. From Figure 11(a), we see the rejection fraction results, where DRES has a factor of 10 improvement over NDRES at a load of 0.6 and a factor of 1.5 improvement over NDRES+1. At a rejection fraction of $1 \cdot 10^{-4}$, NDRES supports a load of 0.48, NDRES+1 a load of 0.58 and DRES a load of 0.64, resulting in a 30% improvement over NDRES.

Specifically, Figure 11(b) which focusses on the *access node (Los Angeles)*, we see that at a rejection fraction of $2 \cdot 10^{-4}$, NDRES supports a load of 0.44, while DRES and NDRES+1 are similar in supporting a load of 0.65 leading to a 50% gain over NDRES. At higher loads, DRES performs better than NDRES+1 as well. For a rejection fraction of $2 \cdot 10^{-3}$, NDRES supports a load of 0.7, NDRES+1 a load of 0.8 and DRES a load of 0.9 leading to gains of 30% and 13% over NDRES and NDRES+1 respectively. Figure 11(c) shows the rejection fraction for *core node (St.Louis)*. Even though, the core node has much larger link capacities, any path that uses an access link is subject to a lower available link capacity. Hence, we do not see a significant improvement in the absolute rejection values. However, deferring still significantly benefits requests that are passing through the core. At a rejection fraction of $4 \cdot 10^{-4}$, DRES supports a load of 0.8, while NDRES and NDRES+1 support loads of 0.6 and 0.65 respectively, leading to gains of 33% and 30% respectively. The gains of DRES over NDRES+1 is much more significant than in Figure 11(b), since there is potentially more available bandwidth (a request occupies atmost 2.5% of the core links as opposed to 10% of the access links).

Figure 11(d) shows the impact of increasing the call duration from 60 time units to 120 time units. If the defer bound is not scaled accordingly, the performance benefits are reduced. At a defer to hold time ratio of 1/12 or 8% for a rejection fraction of $1 \cdot 10^{-4}$, DRES(10) supports a load of 0.6, while NDRES and NDRES+1 support loads of 0.5 and 0.55 respectively leading to gains of 20% and 10%. We also show the effect of scaling the defer bound along with the hold time using the DRES(20) results. DRES(20) can support a load of 0.7 at the above rejection fraction leading to a 40% gain over NDRES, a 27% gain over NDRES+1 and a 16% gain over DRES(10). We note that these gains of DRES(20) over NDRES and NDRES+1 are superior compared to the gains of DRES(10) in Figure 11(a), even though both have the same DH ratio.



(a) FR vs. OL



(b) RT vs. OL

Figure 12: Performance Results: IP Backbone (LF upto 40% of OC-48)

Figures 12(a)-12(b) show the effect of increasing the maximum requested bandwidth to 40% of the OC-48 link. Thus, the probability of an access link being in a saturated state is high compared to the previous case. As expected, the overall results in Figure 12(a) do not show any significant improvement for DRES, as the access links can access only two requests, before deferring. This leads to a large number of requests being dropped due to the expiry of defer timers. However in Figure 12(b), we once again see a significant impact of deferring in core routers. At a rejection fraction of 0.01 (due to large requests, a larger fraction of requests are dropped), NDRES and NDRES+1 support loads of 0.2 and 0.25 while DRES(10) supports a load of nearly 0.5, which is a factor of 5 improvement.

Summary: From the results, we see that deferring has a larger impact on core links with large capacities compared to small access links. Since core links can accommodate more requests, deferring prevents the rejection of a larger number of requests leading to higher gains. Thus, we could envisage a network with deferring used in the core and a simple RETRY model used at the access links.

6 Issues

As we have seen from the simulation results, DRES can provide significant gains over the traditional resource reservation approaches. However, while DRES has its potential benefits, there are several aspects that could be improved upon, leading to potentially even more performance gains. Specifically, we will discuss *partial reservations, deadlock and fairness* issues and then describe our approaches in enhancing DRES to provide additional improvement in performance.

- *Deadlock* is a potential hazard with any distributed protocol. However, since DRES uses timeouts, any cyclic holding of resources lasts only until one of the request times out and is rejected. Now, if this process is repeated, then there is potential for *livelock* as well. However, we minimize this possibility by rerouting the rejected request on an alternate path, using routing mechanisms outlined subsequently. *Partial Reservations* are an inevitable result of the *livelock* problem. When a particular reservation is blocked at a router, the reservation request holds resources at all the previous hop routers on its path until the reservation either succeeds or times out. If the reservation succeeded, then the blocking of resources were justified at the previous hops. However, if the request failed, then the blocked resources would have been better utilized by other requests. Thus, we define this scenario as blocking due to a *partial reservation.*

- *Fairness* is another standard requirement when dealing with a distributed queuing scenario. The current DRES processes requests in the order of the defer timer values. One problem with this approach is that a critical request that is large could be accepted over several smaller requests that are less critical. Performance (Fraction of requests admitted) could be potentially improved by rejecting the large request in favour of the smaller requests. Thus, there are alternate queuing strategies that could improve performance:

  Class based queuing: Requests can be classified based on the request bandwidth into different classes, with each class having its own queue. We plan to evaluate priority queuing mechanisms where each queue (class) has an associated priority. One possible approach is to allow smaller requests to be processed at a higher priority than large requests. Thus, we define an arbitration round where requests are processed from each class. To prevent starvation of the large request class, we can put a bound on how many requests can be admitted from a given class in a given round.

- *Alternate Routing:* The baseline DRES approach assumed the existence of a path that was chosen using a standard routing metric like minimum hop count (OSPF) or the *widest shortest path* [1]. Routing is an orthogonal problem to resource reservation. However, if we were to choose the route that had the maximum probability of admitting a particular request, then deferring on that route would be worthwhile since the request has an increased chance of acceptance. Hence, the integration of routing with DRES and the impact on performance needs to be evaluated.

7 Predicting Defer Times for Early Rejection

In this section, we describe an enhancement to basic DRES that minimizes the amount of time that a request is deferred. The approach here is to estimate the probability tht a given request is likely to be satisfied before the defer time bound. If it is unlikely that request will meet the specified defer bound, then the request is rejected *earlier* than the basic DRES approach which only rejects a request when the defer timer times out. The benefits of *Early rejection* are given below:

- Reduces blocking due to *Partial Reservations*: By minimizing the defer time of a "long shot" request, we minimize the blocking due to partial reservations. Thus, when a flow is discarded, resources blocked by this flow can be released to satisfy other requests.

- Facilitates Rerouting: *Early rejection* need not entail rejecting the request outright. If the decision to early reject is made well within the user-specified defer bound, then the network can retry the request transparently on an alternate route, increasing the chances of admission.

We propose three different approaches based on the information available to each router. All three approaches involve routers that know the average defer time experienced by ongoing requests. Recall that our network model assumes a "Bandwidth Broker" that each router informs when a deferred request is admitted. The statistics are used to obtain the mean and standard deviation of the defer time (μ, σ). The three schemes that we propose differ in the amount of information that is transmitted in addition to the overall $[\mu, \sigma]$, which we present below.

7.1 Network and Resource Reservation Model

The network model essentially consists of a typical cloud of routers, with a "Bandwidth Broker" (BB) [17] to act as a resource manager for the cloud. While DRES is a distributed protocol, routers send defer time estimates to the BB which calculates mean defer times that are used in Section 7. Users send call or flow requests specifying the QoS needed by an application. We use *calls,flows*, and *requests* synonymously for the rest of the discussion. Each call could be associated with a peak bandwidth requirement (B_i), and an end-to-end delay requirement D_i. Each call also has a duration or holding time after which the resources allocated to that call are released. A small portion of the link capacity is reserved for sending control messages such as reservation requests. Non-reservation oriented traffic (best-effort traffic) will be able to access whatever remaining link capacity that is not allocated to the reservation oriented traffic.

7.2 Minimum Information Case

This approach assumes that for a given request, every router knows the number of hops to the destination. Now, this information can be easily distributed in the link state updates and are accurate assuming that the paths between pairs of routers do not change frequently. The goal here is to predict the defer time that the request will experience on the path. If there is only a small chance of the request being admitted, then it is rejected. Thus, every router will perform the following admission check:

$$T_{phop} + \alpha \times \{\mu + \beta \cdot \sigma\} \cdot n_{hop} \leq T_D \tag{11}$$

where T_{phop} is the time spent so far, n_{hop} is the number of hops remaining in the path from the current hop, T_D is the end-to-end defer bound, and α, β are tunable parameters.

The second term in the above control check is the time that the request is estimated to spend at the remaining hops. α is the parameter that decides how much weight is to be associated with the defer estimate. It is feasible to assume $[\mu, \sigma]$ follow a particular probability distribution such as the normal distribution. A *gaussian* predictor assumes that given a normal distribution, β is a multiplier that provides a probability handle in that with a probability of 1 - F(β), we expect the defer time to be $\mu + \beta \times \sigma$, where F($\beta$) is the cumulative distribution function of the normal distribution.

7.3 Partial Information Case

This approach assumes that in addition to the minimum information that is available, each router also knows the available bandwidth on the remaining hops. Thus, given a request $r(b, T_D)$ with bandwidth b, the term n_{hop} in Equation 11 is defined as follows:

```
INIT: $n_{hop} = 0$
FOR EACH remaining link L in path
IF $L_{free}$ < b
    $n_{hop}$++
```

Figure 13: Calculation of n_{hop}

The admission check is then applied as in Equation 11. The key difference with the previous scheme is that the number of hops remaining are only counted if they have insufficient bandwidth. In the *Minimum Info* scheme, the hops were counted irrespective of whether the links had sufficient bandwidth leading to a pessimistic n_{hop} computation. Also, the $[\mu, \sigma]$ values used in this admission test will be different from those in Equation 11 since they are the estimates at hops where requests are likely to be deferred.

7.4 Complete Information Case

This approach assumes that we have defer time estimates for every hop. These estimates are distributed using link state updates, and are accurate provided the network is relatively stable. The admission control for a request $r(b, T_D)$ that is routed on path P at router R_X then becomes:

$$T_{phop} + \sum_{L \in \{P-S\}} \alpha \cdot (L_\mu + \beta \cdot L_\sigma) \leq T_D \tag{12}$$

where $[L_\mu, L_\sigma]$ are the mean and variance of the defer time distribution at link L, and the set S is the set of links that are previous hops to the router R_X.

7.5 Scalability Issues

The above approaches all apply to a particular traffic class. Thus, in a queuing model where requests are classified based on the bandwidth size into different classes, each class will have its own associated $[\mu, \sigma]$. We assume the use of a link state protocol such as OSPF which periodically will send updates containing new estimates. In order for these approaches to scale well, we use threshold based triggers to trigger the link state updates. For a particular class, a link state update is sent only if the new value or estimate crosses a threshold defined for the existing estimate. This helps minimize the number of updates sent, at the expense of working at a coarser granularity level.

7.6 Prediction Methodology

Prediction of defer times is done by mesuring defer times of requests that have been handled recently and using the measured values as predictors of future defer times (See [11]). As shown in Figure 14, we compute the predicted defer time over window $i+1$ as a function (F) of the defer times $\{d(i)\_1, ..., d(i)\_n\}$ measured for requests in window i. Thus, even if we make an inaccurate prediction, it will only affect a window T of flows until it is corrected. Obviously, the smaller the window, the more sensitive it is to fluctuations in the defer time while any inaccurate prediction only affects a small period, while a larger window will be more stable to fluctuations, misprediction would affect a much larger window of arrivals.

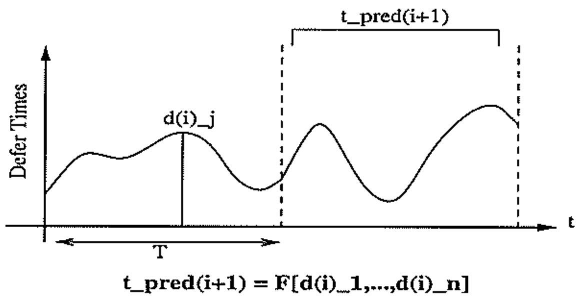


Figure 14: Time Window Measurement of Defer Times

7.7 Discussion

All the aforementioned schemes attempt to perform *Early Rejection*, and reject an infeasible flow early enough so as to avoid holding resources that could be better utilized by other flows. This allows the freeing of resources that would have otherwise been unnecessarily blocked by the infeasible flow. While this potentially improves performance, the infeasible flow is rejected on the chosen path. Assuming that the flow was rejected early enough so that there is still a substantial portion of the time left on the end-to-end defer bound, there is now a chance to retry or reroute the flow on an alternate path. However, it is possible that the route selection process is not optimal in choosing an feasible path for the flow. Thus, there could be another path which would have admitted the request and prevented early rejection. This leads us to the orthogonal problems of QoS Routing and Resource reservation. In all the baseline DRES approaches, we assume the existence of a path which could either be a minimum hop count path, or a minimum weight QoS path. We now attempt to optimize on the selection of suitable routes that maximizes the admission probabilities of deferred requests.

8 Routing and DRES

The aforementioned DRES schemes all assume the existence of a particular route that is chosen using a standard routing mechanism. We now take a closer look at the route selection process and analyze the impact on the benefits of DRES. We assume the use of a simple QoS routing algorithm which we call the Widest Shortest Distance Algorithm (WSDA). We also propose an alternate QoS algorithm called *Parallel Probe (PP)* which uses the defer time estimates as a QoS constraint. We plan to evaluate the performance of DRES using each approach.

8.1 Widest Shortest Distance Algorithm (WSD)

This algorithm uses two simple metrics to evaluate the *Quality* of a route. Essentially, from a backbone provider's perspective, the longer the distance between two routers, the longer the delays to the user. Thus, hop count is no longer the primary criterion. Instead, we use a two phse approach to finding the "shortest" path. In the first phase, we use the distance between the routers to pick a set of "shortest paths" from a set of feasible paths. The second phase then picks the "widest" path (having the largest bottleneck capacity) from this set. The algorithm is extremely simple and is easy to deploy over an existing link state protocol such as OSPF. The link state updates are used to inform routers about the current available capacities, so as to allow accurate computation of bottleneck capacities. We define S_d^s as the set of shortest paths (distance metric) from source s to d, and B_d^s as the path with the largest bottleneck capacity. The number of paths that are maintained is a tunable parameter, and the paths are computed such that all paths have a cumulative cost that is bounded by $\theta \cdot H$, where H is the cost of the minimum distance path. The WSD algorithm is shown in Table 3.

```
WSD Algorithm
INIT: Precompute shortest paths $S_d^s$
FOR Request $r(b)$ from $s$ to $d$
FOR P $\in S_d^s$
    Find widest path $B_d^s$; Route $r(b)$.
```

Table 3: WSD Algorithm

8.1.1 Scalability

With any link state protocol, scalability is a big concern due to routers exchanging frequent link state updates leading to increased processing overheads. There are two enhancements of using *threshold based updates, and path precomputation.* Link state updates are not triggered every time the available link capacity changes. Instead, if the available capacity crosses a bandwidth threshold, only then is the update triggered. A simple update policy is to quantify the link capacity into exponential bandwidth classes where class sizes grow geometrically and are not equal. There are alternate mechanisms that use periodic updates to minimize the information exchanged as well as the processing. The other element is path computation where computing new routes for every update is a wasteful process since the topology information does not change as frequently. In effect, the above enhancements trade accuracy for scalability, since there is a possibility of using stale information due to threshold updates.

In addition, there is another element of link state routing that was previously explored in [31] that deals with *routing instability.* In [31], the problem of *route flapping* is presented where routes may oscillate at a high frequency due to the inaccuracies of link state information. This problem is compounded by the fact that most link state updates are pathological and do not provide meaningful information. This creates a processing overload at the router which could stop responding to meaningful updates. As a result, some routers could be unnecessarily removed from routes leading to invalid routes. In the next section, we present a different approach that does not rely on link state updates, and is scalable enough to be implemented in hardware.

8.2 Parallel Probe Algorithm (PP)

The key idea here is to *send parallel probes on alternate routes when making reservations.* We assume the use of a centralized route server that maintains alternate paths between all sources and destinations. These paths could be precomputed using schemes such as *Widest Shortest Path* (WSP) and maintained. Recomputation of these paths happens periodically when the bandwidth available on links on a given path cross a threshold. In practice, the WSP algorithm precomputes shortest path trees for a given source to all possible destinations. Assuming, we have a set of k alternate paths, where k is a tunable parameter, we initiate RESV_PROBE packets on all these paths, as shown in Figure 15.

The probe packets obtain information as to the current load, as well as predicted defer times at various routers on the different paths. In addition, each probe packet will install state about the predicted defer time for that request in the router. The probes also carry the latest estimate of the defer time so far. Thus, if the existing defer time predicted over the previous hops was T_d^i, the new value carried by the probe will be $T_d^i + t_{pred}$ where t_{pred} is the defer time predicted at that hop. The idea of installing state is to prevent another low priority request from overriding the current request. Pre-emption happens only if the new request has a higher probability of acceptance or a lower predicted cumulative defer time. In such a case, a new request could *override* the existing request and install its state at the router.

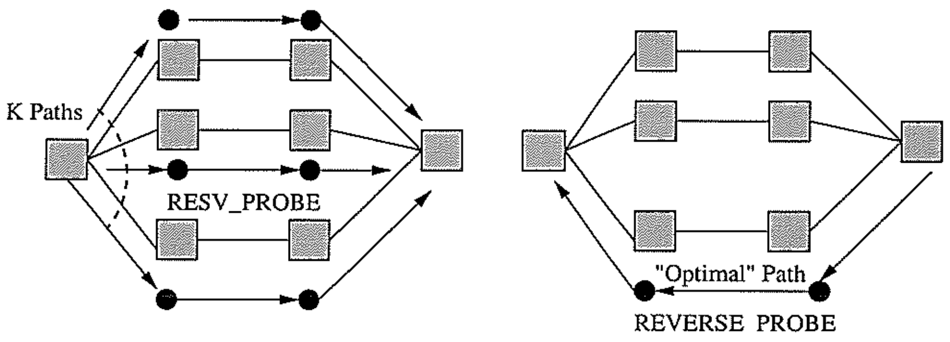


Figure 15: PPA working

As soon as the probe packets reach the last hop router, the data collected is assimilated and the last hop router decides which path is most feasible for the flow. If there are simultaneous requests, then the router prioritizes the requests in the order of their delays or waiting times. The key point is that reservations are now confirmed on the way back to the source only on the most feasible path. The last hop router now sends a REVERSE_PROBE to confirm the reservation. In the reverse path, the probe sent will attempt to reserve resources on the chosen path. It is possible that the path chosen may no longer be feasible due to the overriding of this request by some other request. In such a case, the probe packet will reach the source with a failed message. The reservation state in the forward pass will time out if there is no confirmation of the reservation.

The dissemination of data with the last hop router can be explained using Figure 16. There are two possible cases, where the last hop router is non-unique or unique. The first case with non-unique last hop routers, (Ex: multiple routers connect to the user on a LAN), would result in the probe being sent on the LAN. The probe would reach the other routers, and each last hop router would verify its probe with those that it receives, after which the router that has the best probe statistics would automatically initiate the reservation on the reverse path to the source. In the unique last hop router case, the last hop router would receive the probe packet from two or more routers on different paths, and decide on one of the paths. It would then initiate reservation as before on the chosen reverse path. In order to prevent any deadlocks, each last hop router would have a timeout period in which to receive other probes from other last hop routers. If it does not receive any other probes, it uses its own probe to initiate reservation. The algorithm is shown in Figure 8.2:

Discussion: The main contribution of PP is to achieve parallelism in choosing the best possible path for a given flow request. As a consequence, it would maximize the probability of acceptance for a given flow request by choosing the best possible path. It still takes effectively one round trip time as in the non-parallelized mode of operation. Furthermore these probe packets are of the order of a few bytes, and control traffic have a reserved fraction of every link.

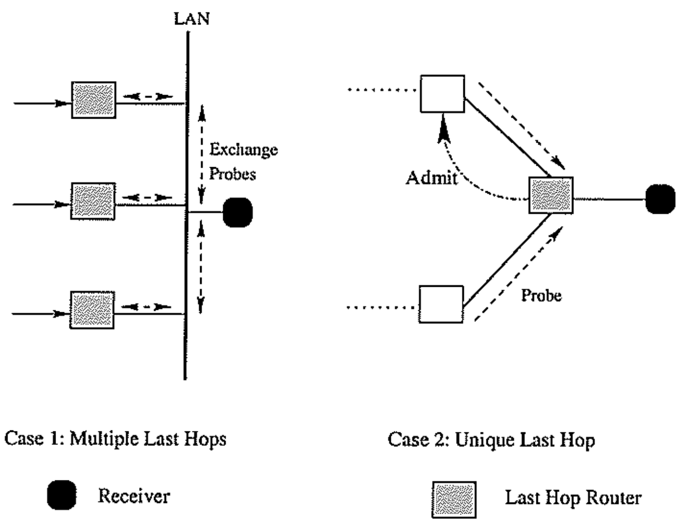


Figure 16: Different Topological Scenarios for PP

```
A. Arrival of $r$ at Source: $(req_i(B_i, D_i, T_s^i))$
        Obtain set of k paths $\{S^k\}$
        Initiate RESV_PROBE packets on paths in $S^k$
B. Arrival of RESV_PROBE(r) at Intermediate hop $(t_{pred}^i)$
        IF no existing state $(t_{pred}^{old})$
            THEN Install predicted defer timer state $(t_{pred})$
            ELSE IF $t_{pred}^i > t_{pred}^{old}$ OVERRIDE state
            ELSE Stop forwarding on this path
C. Arrival of RESV_PROBE(r) at Last Hop
        Compare and find minimum defer time path
        Initiate REVERSE_PROBE on chosen path
D. Arrival of REVERSE_PROBE(r) at Intermediate hop
        IF existing state matches $r$ confirm reservation
        ELSE forward REJECTION_PROBE to source
```

Figure 17: PP Working

Probing would not increase the overall setup delay since alternate routes would be picked such that they would be comparable in terms of end-to-end latency. Even if that is not the case, the use of time-outs to cancel probes that take too much time to reach the last hop ensure that only the best possible routes are picked both in terms of bandwidth as well as end-to-end latency. Minimizing the latter resource is inherently built into the above mechanism. In addition, the PP approach achieves load balancing as well as attempting to match a flows requirements to the links in an optimal manner.

Pre-emption: Due to pre-emption, the RESV_PROBE sent on the chosen "optimal" path could be rejected due to another request that has higher success probability. In such a case, it may be worthwhile to re-initiate the PP algorithm to find an alternate optimal path. Basically, the approach here is to tradeoff a larger initial delay for path selection so that any deferring that occurs on the chosen path is minimized and the request is admitted. Note that in the *prediction* approaches, the deferring on a path was minimized by early rejection. Thus, we see two different angles to minimizing the defer time for a request r. The PP algorithm minimizes the deferring in order to admit the request r. The *Prediction* approaches minimizes the deferring to do *early rejection* in order to benefit *other* requests.

Re-routing: An additional aspect is the rerouting of a request that was rejected early on a path P using the prediction mechanisms described in Section 7. If the request was rejected early enough so that there is still a significant amount of time left in the end-to-end user specified defer bound, then alternate routing mechanisms can be used in order to admit the request. One simple approach is to use the above algorithms to find an alternate route by first deleting the links of the path P from the graph and running the QoS routing algorithm on the subgraph to find an optimal route to retry the rejected request.

Scalability: The WSP link state approach traded accuracy for scalability by using thresholds for trigerring updates. Thus, updates were made only if thresholds were crossed (coarser grain) as opposed to an update every time the available link bandwidth changed. With the probing approach, we do not have to sacrifice accuracy for scalability. Requests are made on-demand and probes directly query routers on the paths avoiding the need for routers to exchange information in the form of link updates. This also makes the information as accurate as possible since the probes query the routers just before routing the request, instead of using stale link information. More importantly, this approach does not lead to *route flapping* since we do not rely on inaccurate information. Since we assume that probes are source routed, the next hop information is self-contained and assumes minimal processing complexity at the router.

Furthermore, the probing can be done in hardware by allowing each router to store its characteristics in the port processors. Every probe would directly get the information from the processor and move on to the next hop avoiding the need for the router software to interact with several hundred probe packets, further reducing the processing overhead, and allowing a scalable implementation. There is one caveat that a source-routing approach may be more complicated when spanning multiple autonomous systems or ISP's since routing information at each ISP is proprietary. One solution could be to allow the ingress edge routers (routers at the ISP boundaries) to expand the source routes, allowing the probe to collect information within the domain, and then use the egress edge routers to remove any proprietary information from the probe when it leaves the domain.

To summarize, the *parallel probing* approach is a dynamic, scalable and efficient routing approach that uses up-to-date routing information with minimal processing complexity to find an "optimal" route while avoiding the need for link state updates. We envisage the use of the probes not just for deferred reservations, but for reservations that require QoS routing. We thus plan to implement the *parallel probing* approach and compare it to the *WSP* algorithm that uses aggregated state information.

9 Topology Experiments

The previous sections described enhancements of *Prediction* that could minimize blocking due to *partial reservations*, as well as QoS routing mechanisms that allowed the optimal path to be chosen for deferring. The ideal case is to evaluate the above enhancements on a realistic topology that could be used by a real-world ISP. DRES provides a way to minimize the cost to an ISP since it minimizes the additional capacity that needs to be added when links are saturated. In this section, we will describe the topology model and experiments that we plan to perform to derive the cost-performance tradeoff to an ISP.

The approach here is to develop a constraint based design of a network as previously described in [27]. The key idea here is to design a network and provision links to be able to handle traffic that are typical of real networks. The key benefit of deferring is that it allows the network to use lower capacity links and use deferring to account for the excess traffic, while maintaining the required QoS. The network operator can flexibly tune the deferring levels so that the delay due to deferring is still acceptable to the user, while ensuring that requests are not dropped. The use of lower capacity links translates to monetary savings for the provider.

Prior work on network design involves specification of a traffic matrix that specifies traffic between every source and destination. With respect to the internet, such a matrix can not be specified due to scalability constraints. As in [27], we represent a network as a graph, with edges for links and vertices for access nodes. In a realistic backbone network, the routers are co-located with cities. In the following discussion, we will describe the traffic constraints that we impose on the nodes, the topologies that the nodes form, the various DRES variants that will be evaluated and their impact on a particular topology with a specific traffic constraint.

9.1 Traffic constraints

In this section, we describe the constraints under which traffic flows in our simulations. The goal is to appropriately dimension the links based on realistic properties such as the distance between routers/cities, as well as using the

Parameter	Explanation
α_u	Max. traffic rate going out from switch u
ω_u	Max. traffic rate coming into switch u
$\gamma(u,v,x)$	Cost of constructing link from u to v with capacity x

Figure 18: Base DRES Algorithm

population of a city to determine the traffic that can be generated at the co-located router. We first define the following terms in Table 9.1.

Pairwise percentage constraints: A typical solution is to maintain source-sink constraints. This resembles the customer-pipe approach in traditional Virtual Private Networks (VPN) where a customer specifies the bandwidth required between source and destination. We augment this by weighting the source sing constraints over all possible routers. For any two switches, U and V, let $\omega(U), \alpha(U)$ be the total incoming traffic and outgoing traffic for switch U, and f(U,V) = $\frac{\omega_V}{\Sigma_{w \neq U}\omega_w}$, and g(U,V) = $\frac{\alpha_U}{\Sigma_{w \neq V}\omega_w}$, then we restrict the traffic from U to V to be μ(U,V) = r_f * min{f(U,V)α_U, g(U,V)ω_V}, where r_f is called the relaxation factor. If $r_f \leq 1$, the complete graph is cheapest. If r_f is large enough such that μ(U,V) $\geq \alpha_U, \omega_V$, then the pairwise constraints can be neglected.

9.2 Topology Design Problem

In this section, we will describe the topology that we assume for our simulation purposes. The choice of the topology is crucial since it influences several parameters that affect the performance. We will use these parameters when discussing the potential impact of DRES on the performance of a given network. Our final goal is to evaluate DRES on this "realistic" topology and obtain the cost-performance tradeoffs of deploying DRES.

Problem Formulation: Specifically, we start out with a standard network topology with appropriately dimensioned links. We also fix a target rejection probability (FR) that we expect for the entire network, as well as a target probability for some of the more heavily loaded "bottleneck" links (FR_l for link l). Our goal is to find the least cost network that can provide a target rejection probability that is $\leq FR$ as well as minimize the rejection probability at the bottleneck link l to be $\leq FR_l$ using DRES. The objective here is to show how using DRES allows the ISP to provide acceptable QoS with a cheaper network.

In addition, we also plan to evaluate the impact of the following:

- Aggregated Links: Depending on the topology, some links may be used more than others and can be called as "bottle-neck" links. Due to these links supporting larger traffic aggregates than other links, any variability in the traffic is smoothed out due to aggregation. The variability that can be exploited will be mainly at non-bottleneck links. This has a direct correlation with *DiffServ* networks which use aggregated chunks.

- Relaxation Factor: As explained above, an $r_f \leq 1$ will imply that the complete graph is the cheapest cost graph. Varying the relaxation factor using the *pairwise percentage* traffic constraint has an impact on the number of links in the graph.

- Alternate paths and QoS Routing: We first plan to evaluate the *Prediction* approaches that allow early rejection of flows so as to minimize resources blocked by *partial reservations.* These rejected flows can then be rerouted to improve the rejection probability. A network topology that has redundant paths between a given source and destination will be benefited by the parallel probing PP algorithms, as well as traditional QoS routing approaches such as *WSP*, which can be used for the dual functions of finding an optimal path for deferring, and for rerouting flows that were rejected by the *prediction* approaches.

- Bursty request arrival: It is also possible to envisage bursty request arrivals. Note that bursty data is assumed to be smoothed out. As shown earlier in Section 3, DRES is ideally suited for bursty request arrivals under certain assumptions. It would be interesting to evaluate the performance gains that can be obtained under such circumstances.

- Heterogeneous traffic classes: The presence of multiple traffic classes will affect the utilization. For example, requests with long holding times will block the links for a longer time. Thus, it will be advantageous if these requests can tolerate a longer defer time as compared to short duration requests. Hence, the prioritization of the traffic classes will have an impact on the performance. Thus, the queuing mechanisms presented in Section 6 could have an impact on performance.

We assume a baseline network of twenty nodes that represent the largest metropolitan cities in the U.S. We arrange the nodes based on their geographical locations. We allocate a monetary budget for each of the nodes which is utilized to connect the city to other cities, as well as account for the infrastructure of routers required at cities or nodes.

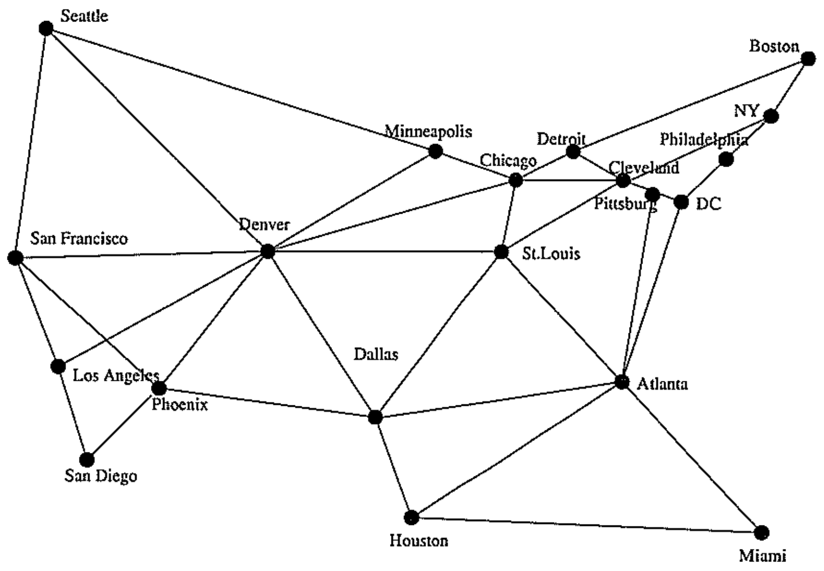


Figure 19: Sample Network Topology for the 20 Largest Metros

10 Related Work

There have been numerous proposals to perform resource reservation in Integrated Service networks. In this section, we highlight some of the prominent ones that are relevant to our approach.

RSVP has two basic reservation messages: *Path and Resv.* The *Path* messages are sent by the sender to establish information about the sender's services in routers. The *Resv* messages attempt to reserve resources and are receiver initiated. However, if resources are unavailable, an error message propagates back to the receiver. One difference is that the state is refreshed periodically, otherwise it times out and the reservation is terminated. In this sense, it is similar to a all-or-nothing approach, except that soft state is used. The control message overhead is significant when compared to DRES, especially for refreshes and is not scalable. Even though this problem is alleviated by merging reservation requests, this creates more difficulties due to heterogeneity. RSVP generates *blockade* states when two or more requests arrive simultaneously allowing a lower reservation request to go through. However, there is no concept of deferring if a flow does not get sufficient bandwidth. Also, it is highly complex to manage and maintain the blockade state. Contrastingly, ATM reservations are true end-to-end reservations, and this follows the all-or-nothing reservation model.

YESSIR [2], a resource reservation protocol for RTP traffic has a concept of *partial reservations* which are made when there is insufficient resources at routers. However, in such cases, YESSIR simply informs the source about the insufficient bandwidth and pushes the problem of deciding whether to reduce the requested bandwidth or to drop the flow altogether to the end-host. YESSIR does not offer end-to-end guarantees with its notion of *partial reservations.* More importantly, having a combination of best-effort reservations along with actual reservations (as results from partial reservations), results in a large number of flows of poor quality as mentioned by the authors, which is unacceptable for providing QoS guarantees.

Recent work on advance reservations [9, 6, 7, 8, 14] propose mechanisms which differentiate between an *immediate* reservation where the call duration is not known, and an *advance* reservation where the duration is known, and the reservation process is initiated far ahead of time. The schemes [9, 8] allow the preemption of *immediate* flows, to allow *advance* reservations to access the link. In such cases, the QoS is downgraded for *immediate* reservations. Also, *immediate* reservations could be dropped so as to accommodate reservations in the future leading to no guarantees for *immediate* reservations. The Tenet real-time protocol suite [10] partitions resources into advance reservations and non-preemptible normal reservations, which can lead to fragmentation. Also, there are no network assisted mechanisms that would allow reservation extension. In [14], the impact of advance reservations on routing algorithms is presented. The computational complexity of different scenarios and their impact on path selection are described.

[24] describes preliminary mechanisms for evaluating the cost of integrating QoS routing with advance reservations. The paper describes three different scenarios which vary depending on whether the duration is known, and the starting and ending times for the requests are known, while investigating how support for advance reservations affects the computational complexity of path selection. The paper also describes intractability results for situations where the amount of bandwidth requested is not the issue, but the amount of bandwidth that is transferred in a given time is the main QoS requirement.

Our focus is to provide QoS for *immediate* reservations that can tolerate some delay before the reservation is completed. DRES does not require the knowledge of the holding time of the flow in advance. DRES actually provides a mechanism to do both *immediate* and *deferred* reservations without any assumptions about the duration. Further, the DRES schemes do not downgrade the QoS provided to any flow due to preemption. This allows the co-existence of both *deferred* and *immediate* reservations. *Advance* reservations force the user to specify the exact time in the future when resources are required. Typically, the user must specify the request well ahead of the actual starting time [8]. Co-existence with *immediate* reservations is only possible when *immediate* reservations are preempted or downgraded in QoS.

Another issue with *advance* reservations is that extending a reservation also involves additional preemptions. Since requests are made far ahead in time, a request for extending an ongoing reservation will typically have to be retried several times before succeeding, assuming resources are currently unavailable, with the same drawbacks of the all-or-nothing approach. With DRES, the chances of successfully extending the reservation are much higher with substantially lower overhead, since the ongoing request could spawn another defer request (for extending the reservation) which waits until either resources become available or till the end of the ongoing reservation. By keeping the request around, there is a higher probability that other reservations will terminate and free up resources.

The tenet real-time protocol suite [10] partitions resources into advance reservations and non-preemptible normal reservations, which can lead to fragmentation since the dynamics of partitioning the resources are hard.

Prior work in route selection algorithms are typically of two categories *flooding* and *preferred-neighbour*. The former approach involves flooding all routers to find a route. The latter approach selects a next hop based on a metric such as shortest path first. One approach that is a hybrid variation of the two is the scheme proposed in [25] for real-time traffic. In this scheme, probes are sent on k alternate paths that simultaneously use k metrics such as hop count to find the best path, and reserve resources on each of the k paths. The probes are sent between Intermediate Destinations (ID), which are a subset of routers along the least cost path. From a given ID, the first probe to reach the next ID wins. The resources reserved on the other k-1 paths are released between the ID's. The process is then repeated till the destination is reached. This protocol has substantial complexity as it attempts to evaluate k alternate metrics between ID's, and on a path with a large number of ID's, this requires significant processor complexity at each ID for processing large number of requests, raising concerns about scalability. Also, the k probes each reserve resources on the path effectively blocking other requests on each of those paths. Furthermore, consider a network where there is signficant sharing of links between the alternate paths. In such case, resources may be reserved at common links for each of the k metrics. Thus, some links could easily become bottleneck links with minimal available capacity due to one or two requests that each may reserve upto k times the requested bandwidth at that hop. If the distance between ID's is large, then the probes could block a significant portion of the network resources.

In summary, DRES with its variants provides a robust model to not only improve on the traditional "all-or-nothing" mode of reservation, but also provide mechanisms to recover from scenarios like failures and renegotiation. Further, DRES integrates QoS routing along with the defer process that allows a comprehensive solution to providing acceptable QoS at minimal cost to the ISP. This is one of the key differences with the other protocols described above that concentrate on particular areas rather than providing a generalized mechanism. DRES does not require the knowledge of the holding time of a flow in advance, making it useful for a wider range of applications. Also, most of the other schemes tend to preempt or drop "immediate reservations" when the advance reservation is activated. DRES does not preempt requests and attempts to accommodate all possible requests. Additionally, some of the other schemes ([8, 9]) also assume that the actual time that the advance reservations begin are far into the future compared to the actual time that the reservation request is generated. DRES is flexible and has no constraints on the "book-ahead" time.

11 Characterization of the Solution

The focal points of the proposed research are listed below.

- Designing the topology framework and evaluating the cost to performance tradeoff for the different DRES variants,
- Integrating QoS routing with DRES, and studying the combined impact of path selection and deferring on performance,
- Adding prediction of defer times to enable early rejection, allowing rerouting of requests,
- Implementation of the DRES variants as Router plug-in modules [3] and evaluation on a real test-bed

The approximate timeline for this proposal is as shown in Figure 20. Note that the timeline has a tolerance of atleast a month.

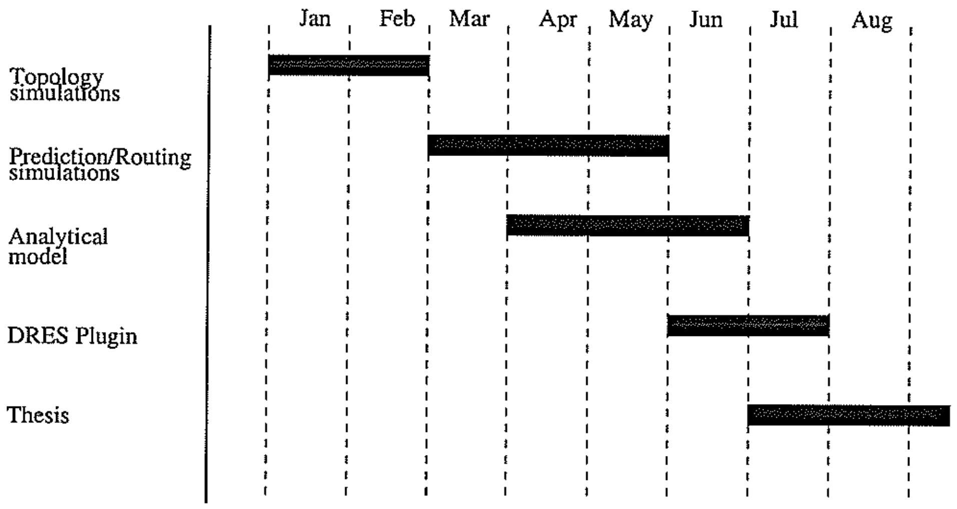


Figure 20: Timeline

12 Conclusions

In this proposal, we highlighted the problems with traditional approaches of RSVP/ATM-style reservations under high network load. When resources are momentarily unavailable, the session initiator needs to manually retry, leading to higher network and signalling processing overhead. To resolve these problems, we introduced DRES, a way of intelligently delaying RSVP/ATM-style resource reservation requests for a short defer time. We highlighted mechanisms that allowed the network to *predict* this defer time. Using simulation, we showed that this significantly improves the chance of admissibility. Depending on the load and situation, these improvements can be quite significant (e.g., upto 10-20 times). We believe that introducing such a mechanism into a resource reservation system would enhance network quality as perceived by the users, while reducing processing load at the network nodes due to moving from a polling-based system to an "interrupt-driven" paradigm. We also presented issues of *Deadlock, Fairness, Partial Reservations and QoS Routing.* We proposed schemes for *predicting* the defer time to allow the *early rejection* of requsts, that helps to minimize blocking due to *Partial Reservations* and helps motivate alternate routing approaches. We also outlined *QoS Routing* mechanisms for finding the optimal path for deferring as well as to reroute requests that were rejected early. We then described the experimental topological framework which we believe will realistically capture the dynamics of a network and allow the investigation of the cost-performance benefits of DRES.

References

[1] Guerin R., et. al. "QoS Routing Mechanisms and OSPF Extensions", *Proceedings of INFOCOM'97*, March 1997.

[2] Pan P., and Schulzrinne H. "YESSIR: A simple reservation mechanism for the internet", *Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Cambridge, UK, July, 1998.

[3] Decasper D., Dittia Z., Parulkar G., and Plattner B. "Router-plugins: a software architecture for next generation routers", *Proceedings of SIGCOMM'98*, September 1998.

[4] Braden B., Zhang L., Berson S., Herzog S., and Jamin S. "Resource Reservation Protocol (RSVP) - Version 1 Functional Specification", *In RFC 2205*, September 1997.

[5] Clark D., and Wroclawski J. "An approach to service allocation in the internet", *Internet draft*, July 1997.

[6] Schill A., Kuhn S., and Breiter F. "Resource reservation in advance in heterogeneous networks with partial ATM infrastructures", *Proceedings of IEEE INFOCOM'97*, March 1997.

[7] Wolf C. L., *et. al.* "Issues of reserving resources in advance", *Proceedings of NOSSDAV'95*, April 1995.

[8] Greenberg A. G., Srikant R., and Whitt W. "Resource sharing for book-ahead and instantaneous-request calls", *IEEE/ACM Transactions on Networking*, vol. 7, no. 1, February 1999.

[9] Schelen O., and Pink S. "Sharing resources through advance reservation agents",*Proceedings of IWQoS'97*, May 1997.

[10] Banerjea A., *et. al.* "The tenet real-time protocol suite: Design, implementation, and experiences", *Technical Report TR-94-059*, Department of Computer Science, University of California at Berkeley, 1994.

[11] Jamin S., *et. al.* "A measurement based admission control algorithm for integrated services packet networks", *IEEE/ACM Transactions on Networking*, December 1996.

[12] Kelly F. P. "Notes on effective bandwidths", *Stochastic Networks*, Kelly F. P., Zachary S., and Ziedins I. (Eds.), Oxford, U.K.:Clarendon, pp. 141-168, 1996.

[13] Kleinrock L. "Queuing systems, Volume 1: Theory", *John Wiley & Sons*, 1975.

[14] Guerin R., and Orda A. "Networks with advance reservations: the routing perspective", *Proceedings of INFO-COM'2000*, March 2000.

[15] Norden S., and Wong K. "ANMAC: An Architectural Framework for Network Management and Control using Active Networks", *Proceedings of the First International Working Conference on Active Networks, (IWAN'99)*, Lecture Notes in Computer Science, vol. 1653, pp. 212-220, July 1999.

[16] Norden S., and Wong K. "ANMAC: An Architectural Framework for Network Management and Control using Active Networks", *Technical Report, Department of Computer Science, Washington University in St.Louis, http://www.arl.wustl.edu/~samphel/iwan.ps.*

[17] Kantawala A., Norden S., Wong K., and Parulkar G. "DiSp: An Architecture for Supporting Differentiated Services in the Internet", *Proceedings of INET'99, The Internet Global Summit*, June 1999.

[18] Ford P. S., and Bernet Y. "Integrated Services Over Differentiated Services", *Internet Draft, draft-ford-issll-diff-svc-00.txt*, March, 1998

[19] ATM Forum. "ATM User-Network Interface Signaling Specification v4.0", 1995.

[20] Hafid A. "Providing a scalable video-on-demand system using future reservation of resources and multicast communications", *Proceedings of IWQoS'97*, May 1997.

[21] Nichols K., Jacobson V., and Zhang L. "An apprach to service allocation in the internet", *Internet Draft*, November 1997.

[22] Clark D., and Wroclawski J. "An approach to service allocation in the internet", *Internet Draft*, July 1997.

[23] Grossglauser M., and Tse D. "A framework for robust measurement admission control", *Proceedings of SIG-COMM'97*, September 1997.

[24] Guerin R., and Orda A. "Networks with advance reservations: the routing perspective", *Proceedings of INFO-COM'2000*, March 2000.

[25] Manimaran G., Rahul H. S., and Murthy C. S. R. "A new distributed route selection approach for channel establishment in real-time networks", *IEEE/ACM Transactions on Networking*, vol. 7, no. 5., October 1999.

[26] Schwartz B., Zhou W., Jackson A. W., Strayer W. T., Rockwell D., and Partridge C. "Smart packets for active networks", *2nd Active Nets Workshop*, March 1997.

[27] Ma H., Singh I., and Turner J.S. "Constraint based design of ATM networks, an experimental study",*Technical Report, WUCS-97-15*, Department of Computer Science, Washington University, 1997.

[28] O'Rourke J. "Computational Geometry in C", Cambridge University Press, 1994.

[29] Apostolopoulos G., Guerin R., Kamat S., and Tripathi S. K. "Quality of service based routing: A performance perspective", *Proceedings of SIGCOMM'98*, August, 1998.

[30] Duffield N.G., Goyal P., Greenberg A., Mishra P., Ramakrishnan K. K., and Van der Merwe J. E. " A flexible model for resource management in virtual private networks", *Proceedings of SIGCOMM'99*, August, 1999.

[31] Labovitz C., et. al. "Internet routing instability", *Proceedings of SIGCOMM'97*, August 1997.