Telecommun Syst (2006) 32:11–30
DOI 10.1007/s11235-006-8200-4

Dynamic resource management in QoS controlled networks

**Venkatesan Iyengar Prasanna · Armin R. Mikler ·
Ram Dantu · Kaja Abbas**

Accepted: 12 December 2005

Abstract This paper addresses the problem of resource fragmentation (RF) in QoS controlled networks. Resources are said to be fragmented when they are available in non-contiguous blocks and hence cannot be utilized by incoming calls with high resource demands. This paper shows the effect of resource fragmentation on QoS controlled networks and presents the Dynamic Resource Redistribution (DRR) algorithm to counteract RF. The DRR algorithm reduces the effects of RF by attempting to redistribute resources in different paths to make resources to incoming calls. A variety of simulation experiments were conducted to study the performance of the DRR algorithm on different network topologies with varying traffic characteristics. The DRR algorithm, when used, increased the number of calls accommodated in the network as well as the overall resource allocation in the network.

Keywords QoS · RSVP · Resource fragmentation · Dynamic resource redistribution

1. Introduction

Many network architectures [3] have been designed to incorporate quality of service (Qos) [1,2,8, 12,17] approaches into networks. The two most popular architectures are Differentiated Service Architecture (DiffServ) [4] and Integrated Service Architecture (IntServ) [5]. DiffServ is intended to facilitate scalable service discrimination in the network without the need for per-flow state and

V. I. Prasanna (✉) · A. R. Mikler · R. Dantu · K. Abbas
Department of Computer Science and Engineering, University of North Texas
e-mail: iyengar@cs.unt.edu

A. R. Mikler
e-mail: mikler@cs.unt.edu

R. Dantu
e-mail: rdantu@unt.edu

K. Abbas
e-mail: kaja@unt.edu

signalling in the node. Services are constructed by setting up appropriate bits in the IP header fields and these bits are used to determine how a packet is forwarded in the network. Since this architecture does not reserve any resources in the network, it is not preferred for most real time applications. On the other hand, IntServ assumes that resources must be regulated in the network in order to deliver QoS thereby making it more suitable for real time applications [5]. Regulation of resources may involve the allocation, maintenance and/or release of resources in the network, which is usually accomplished using a signaling protocol. As varying resource requests arrive and leave the network at arbitrary times, resources may become fragmented as a consequence of being allocated to incoming calls. Resource fragmentation (RF) occurs in the network when resources are available but cannot be utilized because they are available in smaller fragments than the required amount along the path requested. Resource fragmentation cannot be eliminated completely in spite of being undesirable. It occurs due to the random arrival and departure of connections leading to allocation of resources as and when available, usually on a first-come first-serve basis regardless of RF consequences. Moreover, signaling mechanisms and routing protocols are independent of each other, which may lead to RF. For instance, popular routing protocols like Routing Information Protocol (RIP) use hop count as the metric. Hence, the path with the minimal hop count to the destination is selected. This may result in many calls attempting to use the limited resources along the links of the shortest path. Some incoming calls may therefore be dropped as resources may not be available along the shortest path. This is a direct consequence of resource fragmentation because contiguous blocks of resources on the selected path are not available, but there may be sufficient resources distributed along other sub-optimal paths to the destination. Combining routing protocols and resource reservation [11,13] signaling mechanisms to avoid RF is not a viable solution because signaling mechanisms cannot be changed without affecting the routing algorithm and vice versa. One can argue that constraint based routing [9] combines signaling and routing, however, adding multiple constraints into routing, impedes the speed of the routing algorithm considerably [10]. Resource fragmentation is a common problem in networks. This problem along with a possible viable solution is illustrated here using Resource Reservation Protocol (RSVP) [6,7,15,16] controlled networks.

1.1. Need for dynamic resource management

The most important component of resource management is the admission control. The main purpose of admission control module is to regulate the use of resources in the network. Admission control module cannot prevent RF because it does not occur only due to the use of sub-optimal admission control algorithm. RF may occur due to optimal resource allocation in the past or present but that optimal decision might be viewed as sub-optimal by incoming resource demand in future. Lacking clairvoyance, the admission control module in the network is only capable to optimize resource allocation with respect to status quo. That is, future resource demands are unknown and cannot be considered in the current resource allocation decisions. As a consequence, a particular resource allocation made at time t may turn out to be a sub-optimal upon arrival of a new connection at time $t + \Delta t$ (from the new connection's point of view). Dynamic resource redistribution process proposed in this paper can be used to rectify previous decisions and reduce the effect of RF.

1.2. Illustration of RF in RSVP-controlled networks

The Resource Reservation Protocol (RSVP) [6] is an IntServ signaling protocol used to allocate, maintain and release resources in the network. RSVP is an Internet control protocol similar to the

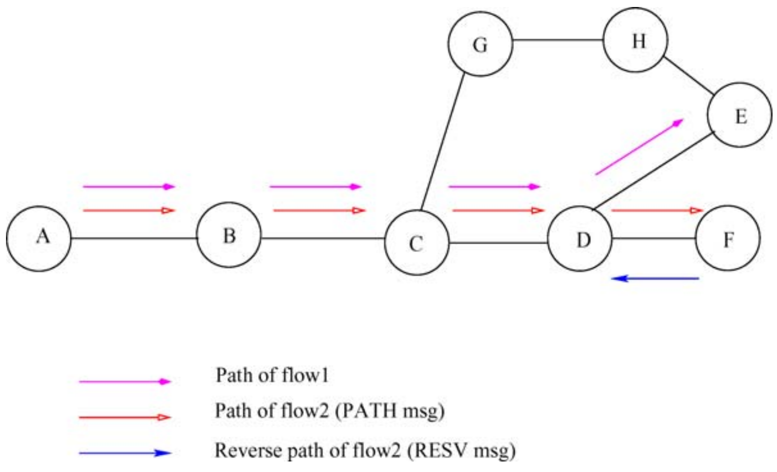


Fig. 1 RSVP

Internet Control Management Protocol (ICMP) and the Internet Group Management Protocol (IGMP), and cannot be used to send application data. RSVP operates above the IP layer and is designed to work with the routing protocol implemented in the IP layer. Application data is carried by transport protocols, such as UDP, and transport protocols interact with RSVP to obtain a QoS guarantee. Resource allocation in RSVP involves two phases. In the first phase, a path is established between the sender and the receiver. In the second phase, resources are allocated along the path established in the first phase. Path establishment is accomplished by sending a *PATH* message to the destination. The *PATH* message consults the local routing table in the intermediate nodes to establish a path between the sender and the receiver. When the *PATH* message reaches the destination, a *RESV* message is sent by the destination to allocate the resources in the path established by the *PATH* message.

An RSVP source starts sending data only after its resource request has been granted by the network. If sufficient resources are not available in the pre-determined path during the reservation (*RESV*) phase, the call is dropped. To illustrate the flow of messages and the corresponding fragmentation of resources, we consider a small section of a network as shown in Fig. 1. Assume that a call ($flow_1$) is active in the path ABCDE. In addition, consider a call request ($flow_2$) that originates at node A for destination F. An RSVP *PATH* message traverses downstream whereas an RSVP *RESV* message traverses upstream towards the sender. Further assume that when the *RESV* message reaches node D, the QoS request cannot be satisfied due to the lack of resources along the path A–F.

If, instead of dropping the current call, node D could release a certain amount of resources and redistribute it on other paths, it might be able to grant the requested QoS for $flow_2$. Specifically, if node D could redistribute the resources of $flow_1$ through node G , it can release the resources associated with $flow_1$ and allocate them to $flow_2$. It is imperative that the entire process of redistributing resources should not affect the data flow of $flow_1$ or any other flow in the network. This can be accomplished by exchanging messages between the nodes concerned. The resource redistribution is said to be dynamic because resource redistribution is attempted only when a resource request fails, i.e., on demand.

This paper describes the resource redistribution process when a reservation request fails during the reservation (*RESV*) phase. The algorithms developed and the messages used in dynamic resource redistribution are presented in Section 2. An analytical model of the resource redistribution process is presented in Section 3. The simulation results for the proposed algorithm are discussed in Section 4. The paper is summarized in Section 5, along with recommendations for future work.

2. Dynamic resource redistribution process

The Dynamic Resource Redistribution Process broadly involves two steps: selecting the flows whose resources will be redistributed (Flow Select Algorithm), and redistributing the resources of the selected flows without affecting existing flows in the network (Dynamic Resource Redistribution (DRR) Algorithm). The DRR algorithm uses the Flow Select Algorithm to select flows for redistribution. This section describes the Flow Select Algorithm and the DRR algorithm. The terms redistribution of resources and redistribution of flows are used interchangeably in this paper.

Let f_b^a be any flow that is passing through node N_a and has b as a flow identifier. Let $f_b^a.qos$ be the resource demand of flow f_b^a. Let f_y^j be the flow that is experiencing resource deficiency at node N_j and flow identifier y. Assume that the network is represented by a graph $G(V, E)$ where $|V|$ denotes the number of vertices/nodes and $|E|$ denotes the number of edges/links in the network. Let F be the set of all flows in the network. Let F^j be the set of all flows passing through node N_j such that $F^j \subseteq F$ and $f_x^j \in F^j$ where $1 \le x \le |F^j|$. In other words, F^j is the set of flows for which resources have been allocated at N_j. Any of the flows in F^j are potential candidates for resource redistribution. That is, these flows may be rerouted so as to circumvent N_j, thereby freeing resources in N_j. Let $F^{j'}$ be the set of flows $\langle f_1^j, \ldots, f_n^j \rangle$ whose resources will be attempted for redistribution. Let f_t^j denote a flow in $F^{j'}$ $(1 \le t \le n)$. Let the transmission delay (d_{ap}) over all links in the network be uniform. Let P be the fixed processing time associated with the resource redistribution process at each node. An alternate path needs to be explored for all the flows in $F^{j'}$ starting from node N_{j-1}^k $(1 \le k \le n)$ to their respective destinations N_d^m $(1 \le m \le n)$. Let T be the set of possible resource/QoS requests. We have considered four classes of resource requests in terms of buffer space, i.e., $T = \{1, 2, 4, 8\}$. Node N_j attempts to accommodate f_y^j using the DRR algorithm.

2.1. Flow select algorithm

When the resource request for a flow fails at a node, a certain number of flows from this node are selected for redistribution, so as to free resources for the failed flow. This is done by implementing either of two different algorithms in the following order

(1) Best Fit Multiple Flow Select Algorithm (MFS)
(2) Best Fit Single Flow Select Algorithm (SFS)

The MFS algorithm is called first in an attempt to redistribute resources of multiple flows, each with a smaller resource request than that of the failed flow. For example, if a flow with a resource request of 4 units of buffer space fails, we try to redistribute flows that have resource requests less than four units each, totalling 4 units. That is, two flows having resources of 2 units, or four flows having resources of 1 unit each are selected. The approach of executing MFS before SFS appears prudent, as the redistribution of flows with smaller resource requirements is more likely to succeed than that of flows with larger resource requests.

If the MFS algorithm fails, we call the SFS algorithm to find a single flow having at least the same resources as that of the failed flow, and redistribute its resources.

The possible classes of resource requests (buffer space requested) considered in this algorithm are 1, 2, 4, and 8 units. The flows selected for redistribution depend on the resource request (*qos*) of the failed flow as follows:

For *qos* demand of 8, MFS will select 2 flows with resources of 4 units each, or 4 flows with resources of 2 units each, or 8 flows with resources of 1 unit each. For the same demand SFS will select 1 flow with resources of 8 units. For *qos* demand of 4, MFS will select 2 flows with resources of 2 units each, or 4 flows with resources of 1 unit each. SFS will select 1 flow with resources of 4 units, or 1 flow with resources of 8 units. For *qos* demand of 2, MFS will select 2 flows with resources of 1 unit each. SFS will select 1 flow with resources of 2 units, or 1 flow with resources of 4 units, or 1 flow with resources of 8 units. For *qos* demand of 1, SFS will select 1 flow with resources of 1 unit, or 1 flow with resources of 2 units, or 1 flow with resources of 4 units, or 1 flow with resources of 8 units.

While choosing flows for redistribution, we do not block partially available resources until the resource deficit is resolved. Instead, the resource request is partitioned into uniformed blocks as explained in this section. Consider, a node gets a resource request of 4 units and 3 units of resources are available in that node. In this case, we try to redistribute 4 units of resources from the current node even though 3 units of resources are available. This is to ensure that there is no blockage of resources (in this case 3 units) until the resource deficit (in this case 1 unit of resource) is resolved.

Algorithm 1. Flow Select Algorithm

```
SET FLOWS to empty set.
IF $qos$ = 1 THEN
    CALL Single_Flow_Select with $qos$ RETURNING FLOWS
ELSE
    CALL Multiple_Flow_Select with $qos$ RETURNING FLOWS
    IF |FLOWS| < 1 THEN
        CALL Single_Flow_Select with $qos$ RETURNING FLOWS
    ENDIF
ENDIF
RETURN FLOWS
```

2.2. Best fit multiple flow select algorithm (Multiple_Flow_Select)

In this algorithm, we first try to minimize the number of flows being redistributed at a node. For example, if a resource request of the failed flow is 4 units, we can redistribute either two flows with resource requests of 2 units each, or four flows with resource requests of 1 unit each. We try to select two flows with resources of 2 units each for redistribution before attempting to select the latter. Thus the MFS algorithm attempts to select fewer flows to minimize the network overhead because of the resource redistribution attempt. We try MFS before SFS because it is easy to redistribute multiple flows with small resource demands than a single flow with a large resource demand. Max is a function that takes a set as a parameter and returns the largest element from the set (refer Algorithm 2.2).

Algorithm 2. Best Fit Multiple Flow Select Algorithm

```
SET W to $\{x | x \in T \bigwedge x < qos\}$
COMPUTE $f(n)$ as $qos \div n$
SET flag to false
SET MFLOWS to empty set
WHILE |W| > 0 AND flag = false
    COMPUTE a as Max(W)
    COMPUTE b as $f$(Max(W))
    Let (a,b) be a two tuple element selected to search $F^j$ for 'b' number of flows
        with a resource allocation of 'a' units.
    SET search_success to false
    SET search_count to 0
    SET i to 1
    WHILE i $\leq |F^j|$ AND search_success = false
        IF $f_i^j.qos$ = a AND $f_i^j.rd\_flag = 0$ THEN
            INCREMENT search_count by 1
            IF search_count = b THEN
                SET search_success to true
            ENDIF
        ENDIF
        INCREMENT i by 1
    ENDWHILE
    IF search_success = true THEN
        SET search_complete to false
        SET search_count to 0
        SET i to 1
        WHILE i $\leq |F^j|$ AND search_complete = false
            IF $f_i^j.qos$ = a AND $f_i^j.rd\_flag = 0$ THEN
                SET $f_i^j.rd\_flag$ to 1
                COMPUTE MFLOWS as MFLOWS $\cup f_i^j$
                INCREMENT search_count by 1
                IF search_count = b THEN
                    SET search_complete to true
                    SET flag to true
                ENDIF
            ENDIF
            INCREMENT i by 1
        ENDWHILE
    ENDIF
    DECREMENT $W$ by $\{a\}$
ENDWHILE
RETURN MFLOWS
```

2.3. Best fit single flow select algorithm (Single_Flow_Select)

A single flow with at least the same resource requirement as that of the failed flow is searched first. For every failed flow, the node tries to find a single flow of the same or higher resource requirement for redistribution. For example, if a flow with a resource request of four units fails, and the MFS algorithm is also unsuccessful, an attempt is made to find a single flow with resources of 4 units for redistribution. If this also fails, an attempt is made to find a single flow with resources of 8 units. Min is a function that takes a set as a parameter and returns the smallest element from the set (refer Algorithm 2.3).

Algorithm 3. Best Fit Single Flow Select Algorithm

```
SET W to $\{x | x \in T \bigwedge x \geq qos\}$
SET flag to false
SET SFLOW to empty set
WHILE |W| $>$ 0 and flag = false
    COMPUTE a as Min(W)
    Let a be an element selected to search $F^j$ for a single flow with a resource allocation of
    'a' units.
    SET search_success to false
    SET i to 1
    WHILE i $\leq |F^j|$ AND search_success = false
        IF $f_i^j.qos$ = a AND $f_i^j.rd\_flag = 0$ THEN
            SET $f_i^j.rd\_flag$ to 1
            COMPUTE SFLOW as SFLOW $\cup f_i^j$
            SET flag to true
        ENDIF
        INCREMENT i by 1
    ENDWHILE
    DECREMENT $W$ by $\{a\}$
ENDWHILE
RETURN SFLOW
```

2.4. Special messages used in the DRR algorithm

In order to facilitate the signaling necessary to dynamically redistribute resources, extra messages need to be incorporated into RSVP. There are a total of eight extra messages that are used, henceforth referred to as *special messages*. Special messages are used to establish new routes and to allocate resources along the new routes. To demonstrate the working of these messages, Let $P_k\langle N_s^k, N_1^k, N_2^k, \ldots, N_i^k, \ldots, N_d^k\rangle$ denote the sequence of nodes (explored by the *PATH* message) along a path from sender N_s^k to receiver N_d^k for any flow k. Assume that a resource request fails at a node N_j due to the lack of resources. Let N_{j-1}^k be the node along the path P_k that is visited just before N_j. The special messages are described below.

S_TRIGGER Message: When a resource request fails at a node, the *S_TRIGGER* message is sent to N_{j-1}^k to trigger the generation of a *S_PATH* message.

S_PATH Message: The *S_PATH* message explores new routes to the destination. The *S_PATH* message must ensure that the new route does not form a loop and does not contain node N_j. If the above two criteria cannot be satisfied, the *S_PATH* message fails and does not proceed further. No attempt is made to find another alternate path.

S_RESV Message: The *S_RESV* message is sent by the destination node along the reverse path of the *S_PATH* message to N^k_{j-1}. This message is sent to allocate resources in each node in the above reverse path. However, no resources are allocated in those nodes (common nodes) that are common to old and new routes. This is due to the fact that they already have the required resources. Nevertheless, in the common nodes, a flag (rd_flag) is set to indicate that the resources of this flow cannot be redistributed. When resources are not available, *S_RESV* fails, and an *S_RTEAR* message is sent to the destination to release the resources. No further attempt is made to allocate resources on any other alternate path. When *S_RESV* reaches N^k_{j-1}, an *S_CONFIRM* message is sent to the destination along the new route.

S_CONFIRM Message: The *S_CONFIRM* message is sent to confirm the reservation of resources along the new path to the destination and to change the path states in the common nodes to reflect the new previous hop and next hop entries along the new path. Until *S_CONFIRM* reaches all nodes common to the old and new paths, they retain their old path state. The old path state is maintained to ensure that in the event of the failure of the *S_RESV* message, the old path is followed. However, if the *S_RESV* message is successful, the new path needs to be followed and the path state information needs to be updated in these nodes. When the *S_CONFIRM* message reaches the destination, an *S_TEAR* message is sent to release resources along the old path in the reverse direction.

S_REFRESH Message: In RSVP, refresh messages are sent periodically to maintain the reservation state. If the nodes do not receive a periodic refresh message, they release their resources. The *S_REFRESH* message is sent by N^k_{j-1} along the old path to ensure that resources are not released prematurely before being allocated along the new path. If the new path is longer than the old path, messages might take longer to reach the destination along the new path. The destination or common node may release the resources due to late arrival of refresh messages. This would result in loss of path to the destination. To keep the resource reservation alive in common nodes, the *S_REFRESH* message must be sent.

NOTIFY_SUCCESS Message: The *NOTIFY_SUCCESS* message is sent by N^k_{j-1} to notify N_j of the successful redistribution of resources. There may be more than one *NOTIFY_SUCCESS* message, depending on the *qos* of the call that failed at N_j. Upon receiving all the *NOTIFY_SUCCESS* messages, N_j allocates the freed resources to the failed call.

S_TEAR Message: The destination node sends the *S_TEAR* message along the reverse old path to release the resources. This is done only after the destination receives the *S_CONFIRM* message. Resources are not released in those nodes that are common to the old and new paths.

S_RTEAR Message: The *S_RTEAR* message originates from the node where the *S_RESV* message fails and is sent to the destination node. The *S_RTEAR* message releases resources in each node it traverses on its way to the destination node. Resources are not freed in those nodes that are common to the old and new paths.

2.5. Dynamic resource redistribution algorithm

This subsection describes the algorithm that determines how the resources are redistributed without affecting the data flow in the network.

Algorithm 4. Dynamic Resource Redistribution Algorithm

1. Let S be a subset of F^j consisting of flows satisfying the Flow Select algorithm. Let i = $|S|$, and $S = \{f_1^j ... f_i^j\}$. If i is zero, exit the algorithm and drop the flow f_y^j, otherwise goto 2.
2. A timer T_{NS} is started at N_j for the successful notification of redistribution of resources (*NOTIFY_SUCCESS*) by each of the flows in set S. A counter C is initialized to i.
3. For each flow in set S, f_k^j, where $1 \leq k \leq i$, the following steps are carried out:
   (a) Let $PHOP_k$ be the node on a path P_k that is visited just before N_j. An *S_TRIGGER* message is sent to $PHOP_k$ from N_j, to trigger the *S_PATH* message.
   (b) From $PHOP_k$ an *S_PATH* message is sent to the destination D_k of flow f_k^j, along a new path $P_{k'}$. This new path is determined in such a way that it does not contain N_j, nor any loops.
   (c) After receiving the *S_PATH* message, node D_k sends an *S_RESV* message to $PHOP_k$ along the reverse path of $P_{k'}$. If the *S_RESV* message fails before reaching the node $PHOP_k$, an *S_RTEAR* message is sent to node D_k along the forward path $P_{k'}$.
   (d) When the *S_RESV* message reaches $PHOP_k$, $PHOP_k$ does the following:
      (i) It sends an *S_CONFIRM* message to the node D_k along the new path $P_{k'}$.
      (ii) It sends an *S_REFRESH* message along the old path of flow f_k^j to node D_k.
      (iii) It changes its path state to reflect the new path along $P_{k'}$ to node D_k.
      (iv) Node $PHOP_k$ finally sends a *NOTIFY_SUCCESS* message after a specified period of time, to N_j to indicate successful redistribution of resources for flow f_k^j. This delay is introduced to allow the *S_CONFIRM* message to reach the destination and confirm the new path.
   (e) Upon receiving the *S_CONFIRM* message from $PHOP_k$, node D_k sends an *S_TEAR* message to N_j along the old path of flow f_k^j to release the resources. D_k also updates its state to reflect the new path.
   (f) On receipt of a *NOTIFY_SUCCESS* message from $PHOP_k$,
      (i) If the timer has not timed out, N_j decreases the counter C by 1. If C becomes 0, N_j releases the resources it had allocated to flows in set S and re-allocates them to flow f_y^j.
      (ii) If the timer has already timed out when N_j receives the *NOTIFY_SUCCESS* message for a flow, N_j removes the state information for the flow that was redistributed and the same flow now maintains its new path.
4. If the timer T_{NS} at N_j times out before it receives the *NOTIFY_SUCCESS* message from all the flows in set S, then flow f_y^j is dropped.

2.5.1. Illustration of the DRR algorithm

Consider a part of the network as shown in Fig. 2. Let a flow f_y^j with a resource request of 4 units of buffer space fail at node j. The subscript y identifies a unique flow in node j. Assume that the

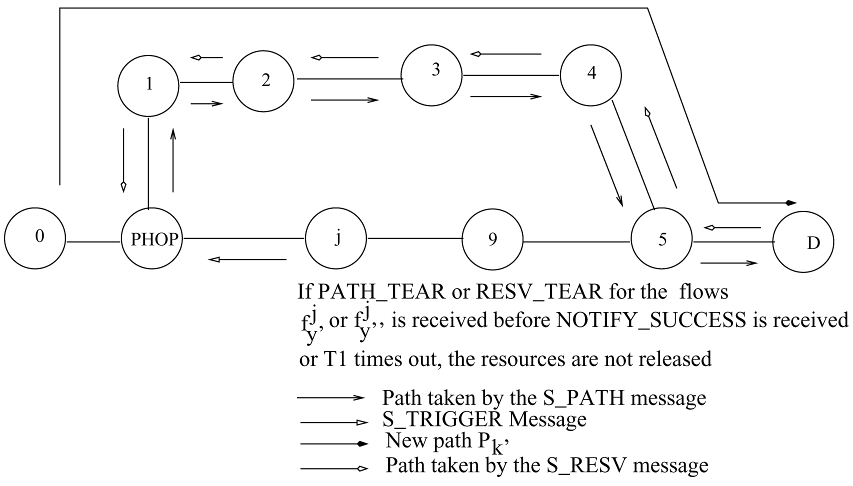

1
2
3
4
0
PHOP
j
9
5
D
If PATH_TEAR or RESV_TEAR for the flows
f$^j_{y'}$, or f$^j_{y''}$, is received before NOTIFY_SUCCESS is received
or T1 times out, the resources are not released
Path taken by the S_PATH message
S_TRIGGER Message
New path P$_k$'
Path taken by the S_RESV message

Fig. 2 Example of resource redistribution

Flow Select algorithm selects two flows $f^j_{y'}$ and $f^j_{y''}$ with resource allocation of 2 units of buffer space to redistribute. Algorithm 4 is used to redistribute resources of $f^j_{y'}$ and $f^j_{y''}$. Figure 2 show paths taken by special messages.

2.5.2. Maintaining a path at all times

When an attempt is made to redistribute the resources of a flow on a different path (new path), the old path is not affected until its resources are successfully redistributed. In other words, at least one path remains active between the sender and the receiver when the DRR algorithm is applied to a flow unless and until it is explicitly torn down by the sender or its reservation state times out. When the *S_RESV* message reaches those nodes that are common to the old and new path, it does not delete the old path information. Thus it is guaranteed that if the *S_RESV* message fails before reaching its intended receiver node, the old path is preserved.

When the *S_RESV* message reaches its intended receiver, it sends an *S_REFRESH* message along the old path to the destination, to keep the old path and reservation states alive. If the new path is longer than the old path, reservation state in the destination node may timeout due to delayed arrival of refresh messages. Therefore an explicit refresh message is sent to keep the path and reservation states alive in the destination node.

When the *S_RESV* message fails due to lack of resources, the *S_RTEAR* message is sent to the destination to release the resources allocated so far. Resources are not released in the nodes that are common to the old and new path, as this would mean loss of resources, not only for the new path, but for the old path as well. Thus, the old path is maintained. The *NOTIFY_SUCCESS* message is sent to node j after a specific delay of time. This delay is introduced to allow the *S_CONFIRM* message to reach the destination and confirm a new path to the destination. If there is no delay in sending the *NOTIFY_SUCCESS* message and the destination is not yet aware of the new path, it sends a *RESV* message to maintain the reservation state along the old path. Upon reaching node j, the *RESV* message may fail if the node has already received all the *NOTIFY_SUCCESS* messages and removed its reserve state.

2.5.3. Alternate path with no loops

When an alternate path exploration is started at N_j, a timer is started for the response of this exploration. If the response is not received within this timeout interval , the call is dropped.While exploring alternate path, the *S_PATH* message carries the identity of N_j. If this message comes to N_j again before reaching destination D, it will not proceed further and would eventually lead to timeout at N_j. This ensures that the alternate path does not contain loops. The *S_PATH* message also will not go on forever as it will be dropped when the Time to Live (TTL) of its Internet Protocol (IP) header becomes zero. The alternate path is selected at random and depends on the routing protocol used. The effect of alternate path discovery and resource allocation on that path on the call setup time needs to be explored.

2.5.4. Special cases

If a *PATH_TEAR* or *RESV_TEAR* message for one of the flows being redistributed is received at node j, the resources reserved for that flow at node j are not released, as they are being held for the new flow. Node j waits until it receives a *NOTIFY_SUCCESS* message for all the flows that are being redistributed or until it times out before releasing the resources to the new flow.

After sending the *S_PATH* message, the reserve state in $PHOP_k$ may be explicitly deleted by the sender or it may time out. When the *S_RESV* message reaches its intended destination, it will not find a reservation in that node. In this case, an *S_RTEAR* message is sent to the destination to release all the resources for the flow, as they are not required any more.

3. Analytical model of the resource redistribution process

An analytical model for the DRR algorithm is presented in this section. The resource redistribution process at N_j and the actual dynamic resource redistribution along alternate paths are illustrated using state diagrams. The notations used in the proofs herein are defined in the previous section.

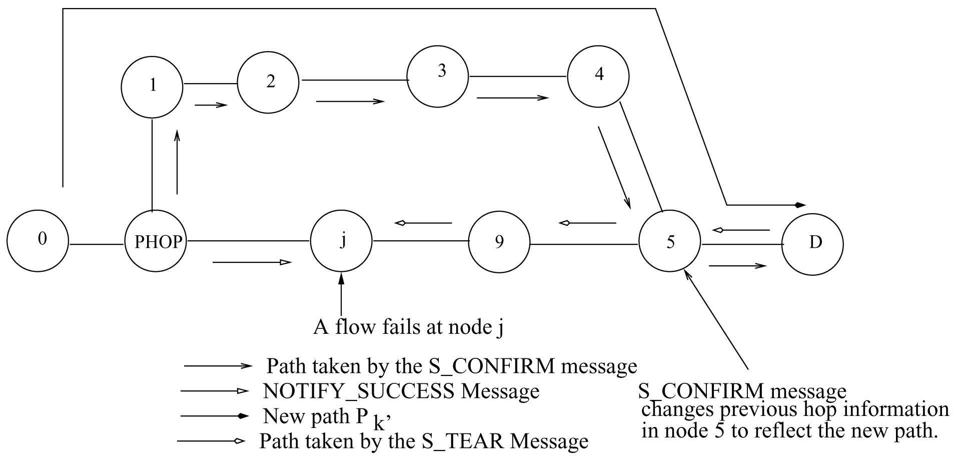

0
PHOP
1
2
3
4
j
9
5
D
A flow fails at node j
Path taken by the S_CONFIRM message
NOTIFY_SUCCESS Message
New path P k'
Path taken by the S_TEAR Message
S_CONFIRM message changes previous hop information in node 5 to reflect the new path.

Fig. 3 Example of resource redistribution

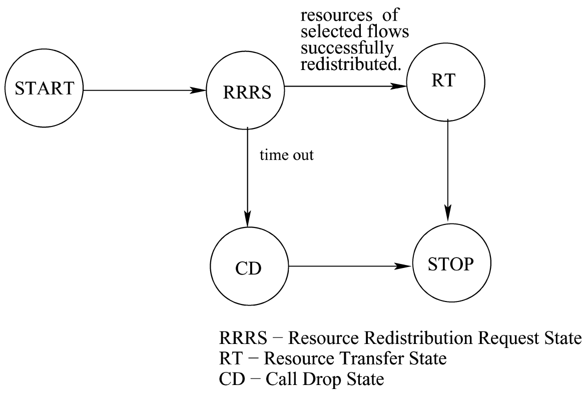

START
RRRS
RT
CD
STOP
resources of selected flows successfully redistributed.
time out
RRRS – Resource Redistribution Request State
RT – Resource Transfer State
CD – Call Drop State

Fig. 4 Resource redistribution process at N_j

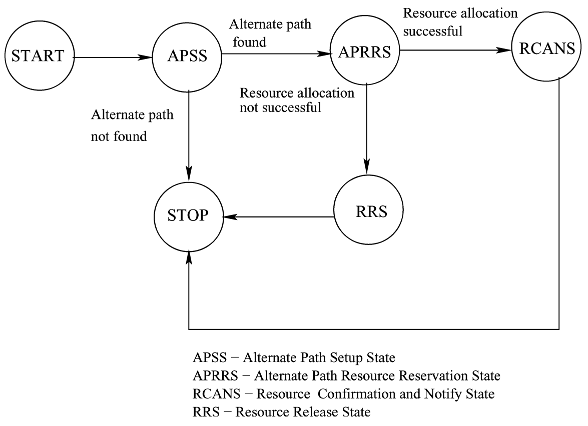

START
APSS
APRRS
RCANS
STOP
RRS
Alternate path found
Resource allocation successful
Resource allocation not successful
Alternate path not found
APSS – Alternate Path Setup State
APRRS – Alternate Path Resource Reservation State
RCANS – Resource Confirmation and Notify State
RRS – Resource Release State

Fig. 5 Expansion of RRRS

The state diagram in Fig. 4 illustrates the states of N_j. In RRRS, resource redistribution request is sent for all flows in $F^{j'}$. If the resource redistribution for all flows in $F^{j'}$ is successful, then their resources will be transferred to f_y^j. Otherwise, f_y^j will be dropped.

Figure 5 is an illustration of the RRRS state in Fig. 4. The figure depicts the state diagram for each flow f_t^j in $F^{j'}$ while an attempt is made to redistribute their resources. In APSS, an alternate path for a flow in $F^{j'}$ is explored. If an alternate path cannot be found, then the algorithm does not proceed further. The time taken to discover that an alternate path is not found is finite as the number of nodes visited is atmost $|V| - 1$ assuming that the network does not contain loops. Even if the network contain loops, this discovery process comes to an end when TTL field in the IP header reaches its maximum limit. Unavailability of alternate paths leads to timeout at N_j. If an alternate path exists, then resources are reserved in this path. Let P_t^j denote the alternate path for f_t^j (a flow in $F^{j'}$) that circumvents N_j. Successful allocation of resources on P_t^j leads to RCANS. In this state, P_t^j is confirmed and a notification message is sent to N_j to indicate successful redistribution of resources for f_t^j. However, shortage of resources along P_t^j leads to RRS where resources allocated so far in that path are released. After the resources are released, the algorithm does not proceed further. Shortage of resources along the alternate path also leads to timeout in N_j.

Corollary 3.1. *The resource redistribution process converges in finite time.*
After resource redistribution requests for all flows in $F^{j'}$ have been sent, the algorithm waits

for a fixed time (T_{NS}) to receive the notification for successful redistribution of resources. In the absence of such a notification, f is dropped and the STOP state is entered. Otherwise, required resources are transferred to f and the STOP state is entered. Since T_{NS} and time taken to transfer the resources is finite, the STOP state is entered after finite time.

4. Results

The main focus of this paper is to evaluate the overall performance of RSVP (unicast communications) when Dynamic Resource Redistribution (DRR) is incorporated. A variety of simulation experiments has been conducted to measure the performance of RSVP_DRR. The simulator, traffic model, network model and the experiments conducted are described in this section.

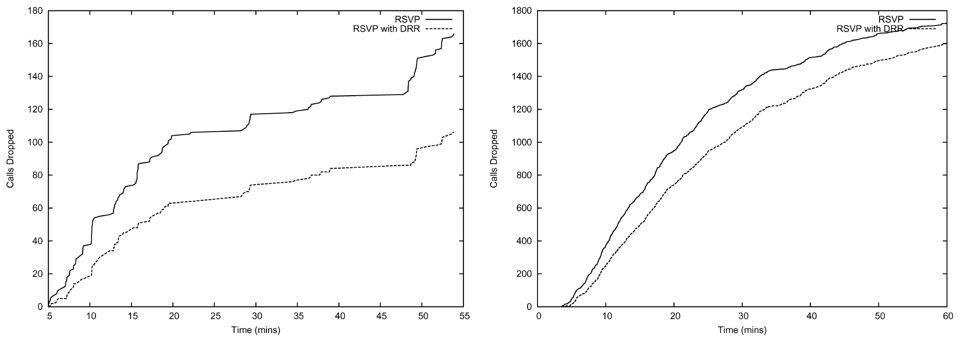
An object oriented event based simulator (NRLSIM) has been developed to measure the efficacy of the DRR algorithm. The NRLSIM is a simulation engine that can be used to simulate any event driven system. The type and the nature of the event is transparent to the simulator and can be custom defined by the model. The simulator acts as a black box and can be used to simulate any defined event.

Traffic in the network is modeled in terms of flow requests with specific buffer requirements. A request is characterized by a source node, destination node, buffer requirements and the duration of this flow. The generation of requests is Poisson distributed with an average arrival rate of λ calls per minute. The duration of each flow is exponentially distributed with an average duration of D minutes. Buffer space requests are uniformly and non-uniformly distributed over the set $T = \{1, 2, 4, 8\}$.

The DRR algorithm has been tested on different networks modeled by graphs of different sizes and average node degrees ($\bar{d}$). Every node in the network acts both as a router and end host. The state of a node refers to the collective state values of data structures associated with the node, such as path state, reserve state, buffer space, etc. These data structures may grow or shrink depending on events that occur in the node. An event may or may not change the state of the node. For example, upon receiving a *PATHTEAR* message for a specific flow, the node may have to remove its path and reserve states if they already exist for that flow. Otherwise, no action is taken. Links in all topologies have a fixed transmission delay of 3 and 2 ms for RSVP messages and special messages respectively. Each node in the network is assumed to be RSVP aware, i.e., it can process and forward RSVP and special messages. Messages sent over any link in the network are also assumed never to be lost or damaged. The network remains static throughout the duration of the simulator. Each node maintains a routing table with multiple routes for all destinations, if possible. Dijkstra's shortest path algorithm is used to find the shortest path between nodes to complete the routing tables in all the nodes present in the network before starting the simulation of the algorithm.

4.1. Experimental analysis

In all the experiments, the performance of RSVP with DRR (RSVP_DRR) is compared to conventional RSVP. There are three broad categories for comparisons—number of drops (request failure due to resource unavailability), resource utilization in the network, and the time complexity. When RSVP is simulated, all the traffic characteristics (source node, destination node, resource request, duration of flow and time of request) are stored and the same traffic pattern is used while executing RSVP_DRR. Experiments have been conducted on a 30 node network with varying average node degrees of 3, 5 and 7. These experiments were conducted for two different distributions of QoS requests. In the first set of experiments, calls with QoS requests of



(a) Hold Time = 5 min (b) Hold Time = 20 min

Fig. 6 Cumulative drop graph for a network size of 30 nodes with an average node degree 5 (non-uniform QoS request distribution)

1, 2, 4 and 8 units of buffer space are generated with a probability of 50%, 25%, 15% and 10% respectively. In the second set of experiments, calls with QoS requests of 1, 2, 4 and 8 units of buffer space are each generated with a probability of 25%. Different average call holding times (t_{avg}) have been used for the experiments. The comparisons between RSVP_DRR and RSVP are discussed for different t_{avg}, in terms of total number of calls dropped and resource utilization in the network.

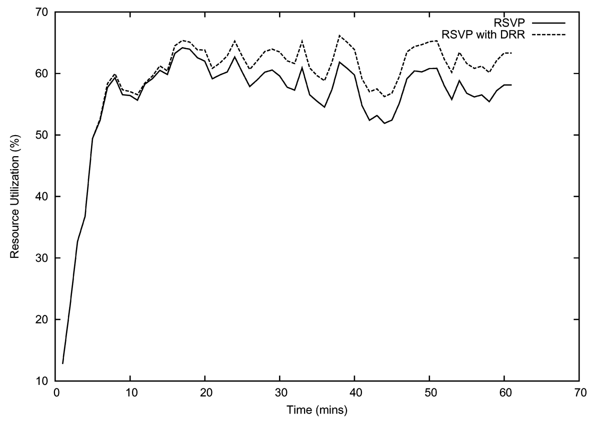
4.2. Non-uniform QoS requests

In these experiments, calls with QoS requests of 1, 2, 4 and 8 units of buffer space are generated with a probability of 50%, 25%, 15% and 10% respectively. The number of calls dropped in RSVP_DRR and RSVP is measured and compared, in order to evaluate the performance of RSVP_DRR. The results confirm that fragmented resources in the network increase the number of calls dropped in conventional RSVP, and that the redistribution of resources reduces the degree of fragmentation in the network.

When average node degree ($\bar{d}$) of the network is 5, RSVP_DRR compares favorably to RSVP in terms of the number of calls dropped. Initially, there are no call drops in the network, as shown in Fig. 6. After a significant number of calls have been established, resources may become unavailable for new incoming calls. Incoming calls are dropped when their resource requests cannot be granted by the network. At this point, a steep increase in cumulative call drops is observed due to resource unavailability. However, the slope of the curve that characterizes the number of calls dropped decreases after a period of time. This decrease is a direct consequence of resources becoming available in the network after the initial calls have been serviced.

The number of calls dropped increases as the average holding time of a call increases as can be seen in Fig. 6. There is a marked difference in the number of calls dropped when t_{avg} is 5 minutes and when t_{avg} is 20 minutes. This is due to the fact that calls request and hold resources for a shorter duration, and thereafter release the resources within a short period of time in the former. This greatly reduces the number of drops. The longer a call holds resources, the less available resources there are in the network for that duration, and consequently higher are the number of calls dropped.

RSVP_DRR closely follows the performance of RSVP, but is nevertheless an improvement on it in terms of the number of calls dropped (Fig. 6). When t_{avg} is small, RSVP_DRR is more



(a) Hold Time = 5 min

Fig. 7 Resource utilization for a network size of 30 nodes with an average node degree of 5 (non-uniform QoS request distribution)

effective than when t_{avg} is large, as resources are available sooner for redistribution. There is a decrease of 36% in the number of calls dropped in RSVP_DRR over RSVP, when the average call holding time is 5 minutes. When t_{avg} is increased, this percentage decreases (16% for 10 minutes, 9% for 15 minutes and 7% for 20 minutes) as resources are held longer by the calls before being released.

Resource utilization is a measure of the ratio of allocated resources over total available resources in the network. Resource fragmentation leads to under-utilization of resources in the network. This experiment has been conducted to investigate the effect of redistribution of resources on resource utilization in the network.

Resource utilization in the network is higher with RSVP_DRR than with RSVP, as seen in Fig. 7. When the network succeeds in redistributing resources, more resources in the network are allocated. Further, when t_{avg} is large, resources of more calls will be redistributed as compared to when t_{avg} is small. It can be concluded that redistribution of resources reduces resource fragmentation and thereby improves resource utilization in the network over conventional RSVP.

Time complexity of an algorithm is the total time taken to execute the algorithm under given conditions. It is likely that a call will re-attempt to avail resources even after initially being denied. This holds true for both RSVP_DRR and conventional RSVP. In this experiment, successive attempts are made to accommodate a call until its resource request can be granted by the network. Usually, unavailability of resources leads to a call being dropped in both RSVP_DRR and RSVP. The interpretation of unavailability of resources in RSVP_DRR is that the attempt(s) to redistribute the resources was also unsuccessful. When a resource request fails, an attempt is made to allocate resources for the failed flow by re-attempting the request until it is successful. The delay between each attempt to allocate resources is exponentially distributed with an average delay of 5 minutes. This experiment compares the time complexities of RSVP_DRR and RSVP and projects the total time required to accommodate all calls in the network. The resource request for a denied call is repeated until its request is granted by the network. The likelihood of a call succeeding after having failed initially in RSVP_DRR is higher than in RSVP (see Fig. 8). This is due to the fact that resource redistribution is successively attempted in RSVP_DRR to grant resources to the denied call, which increases the chances that the call is successful sooner. The results verify that all calls are indeed, more likely to succeed with RSVP_DRR, within a shorter period of time than in RSVP. Thus, the total time taken to execute RSVP_DRR is less than RSVP.

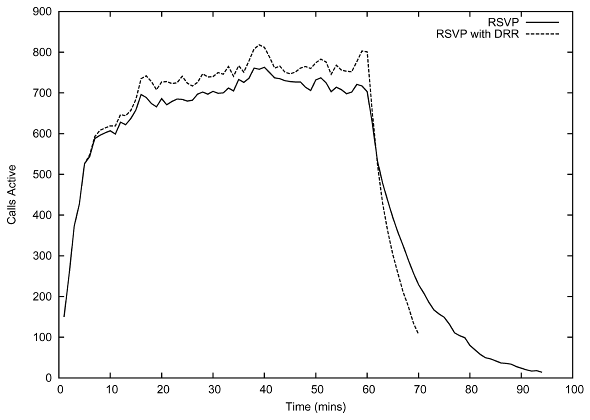


Fig. 8 Time complexity experiment for a network size of 30 nodes with an average node degree of 5 (non-uniform QoS request distribution)

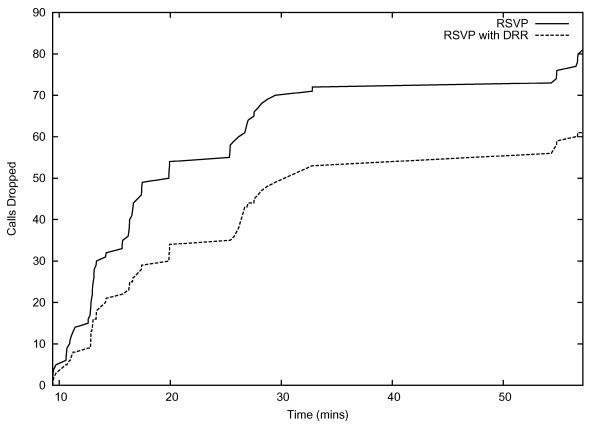


Fig. 9 Cumulative drop graph for a network size of 30 nodes with an average node degree of 7 (non-uniform QoS request distribution)

Figure 8 illustrates the time complexity of RSVP and RSVP_DRR. Initially, the same number of calls are active in the network for both RSVP and RSVP_DRR. This is because there are no call drops and all incoming calls can be accommodated as resources are available in the network. For subsequent calls, when resources may become scarce in the requested path, and RSVP_DRR tries to redistribute resources to accommodate more calls. Thus, there is an increase in the number of active calls in the network for RSVP_DRR than for RSVP. Consequently RSVP_DRR takes less time to accommodate all calls as compared to RSVP, as it attempts to redistribute resources if necessary. RSVP does not redistribute resources, and therefore has to wait until resources are available in contiguous blocks. The last recorded time in the time complexity graph is the time at which all calls have been successfully granted resources by the network. RSVP_DRR takes approximately 70 minutes versus 94 minutes taken by RSVP to successfully accommodate all calls in the network (see Fig. 8). Therefore, the redistribution of resources implemented in RSVP_DRR reduces the total time taken to accommodate all calls in the network.

When $\bar{d}$ is increased to 7, the number of calls dropped decreases in RSVP. This is due to the fact that there are more paths between any two nodes in the network. The RSVP_DRR algorithm also performs well as shown in Fig. 9. However, the percentage difference in number of calls dropped between RSVP_DRR and RSVP is not as high as might be expected. This is due to the fact that the size of the routing table is finite. That is, a routing table can store only finite number of routes (next hop entries) for a particular destination. Even though $\bar{d}$ is 7, the number

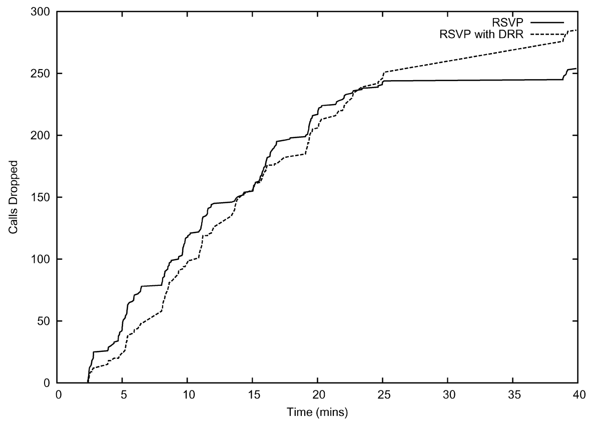


Fig. 10 Cumulative drop graph for a network size of 30 nodes with an average node degree of 3 (non-uniform QoS request distribution)

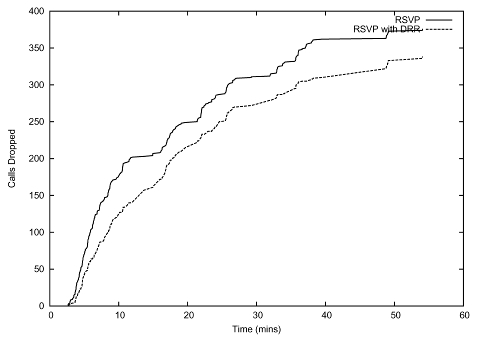
of alternate path choices has been limited, thereby curtailing the performance of RSVP_DRR. The results for resource utilization and time complexity mimic those that are observed in Figs. 7 and 8 respectively.

When $\bar{d}$ is reduced to 3 (Fig. 10), the total number of calls dropped using RSVP_DRR is higher than with RSVP. One plausible explanation is that in a sparse network, the number of nodes directly connected to other nodes is small. Hence there are few alternate paths that can be used for redistribution of resources. Even if a path is found to redistribute resources, another call originating at one of the nodes along the selected alternate path may have to be dropped because resources were redistributed on this path.

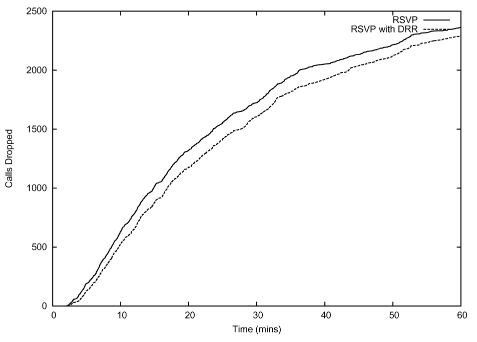
4.3. Uniform QoS requests

In these experiments, calls with QoS requests of 1, 2, 4 and 8 units of buffer space are generated with a probability of 25% each. These experiments measure the performance of RSVP_DRR and RSVP in the scenario where the traffic is burst with an equal probability as that of normal traffic. The experiments reveal that there are more calls dropped here than in the non-uniform distribution of QoS requests. Lower class QoS requests are easier to accommodate in the network, than higher class QoS requests. Here, the distribution of calls with a QoS request of 8 units is the same as that of calls with a QoS request of 1 unit. Hence, the number of calls dropped in this distribution increases.

When $\bar{d}$ is 5 (see Fig. 11), RSVP_DRR compares favorably to RSVP in terms of the number of calls dropped. However, the total number of calls dropped increases from what has been observed for the non-uniform QoS request distribution. This is due to the fact that more calls with resource requests of 8 and 4 units are generated and not all of them can be accommodated. The difference between the total calls dropped for RSVP_DRR and RSVP is less than that in the non-uniform QoS request distribution. Redistribution of resources of calls with a high QoS demand is more difficult than of calls with a low QoS demand. Since calls with a high QoS request are generated with the same probability as calls with a low QoS, it is more difficult to accommodate the former in the network. This contributes to the decrease in the difference between the total number of calls dropped for RSVP_DRR and RSVP. Irrespective of the call holding times, it is observed that the change in QoS request distributions affects the difference in the number of calls dropped between RSVP_DRR and RSVP. The average call holding times however, contribute to the increase in the total number of calls dropped i.e., we observe an increase in calls dropped for large t_{avg}.



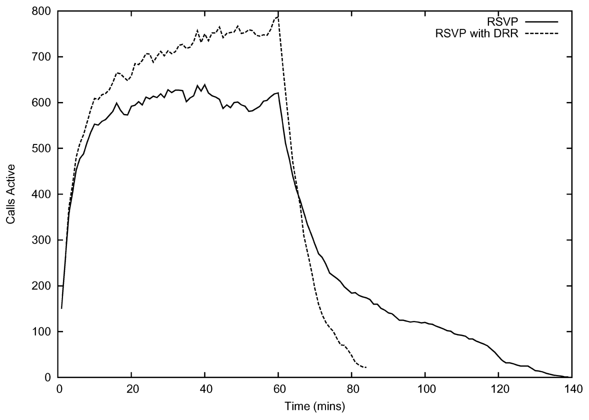
(a) Hold Time = 5 min



(b) Hold Time = 20 min

Fig. 11 Cumulative drop graph for a network size of 30 nodes with an node degree of 5 (uniform QoS request distribution)

Fig. 12 Time complexity experiment for a network size of 30 nodes with an average node degree of 5 (uniform QoS request distribution)



The experiment for time complexity (Fig. 12) in a uniform traffic distribution shows that the time taken to successfully accommodate all calls in the network is greater than the time taken in a non-uniform QoS request distribution (Fig. 8), in both RSVP_DRR and RSVP. This difference is due to the generation of increased calls with QoS requests of 4 and 8 units. The time taken to allocate and/or redistribute resources for these calls is longer, contributing to the increased time taken to accommodate all calls in the network.

RSVP_DRR compares favorably to RSVP for both uniform and non-uniform distributions of resource demands. All the experiments discussed in this section were run on a 30 node graph with varying topologies (i.e., varying $\bar{d}$). However, our results show that RSVP_DRR scales well when the network size increases.

5. Summary

The Dynamic Resource Redistribution algorithm implemented with RSVP (RSVP_DRR) reduces the effects of resource fragmentation in the network. The results for both non-uniform and uniform distribution of QoS requests indicate that RSVP_DRR is an improvement on RSVP in terms of the number of calls dropped. There are fewer calls dropped with RSVP_DRR than

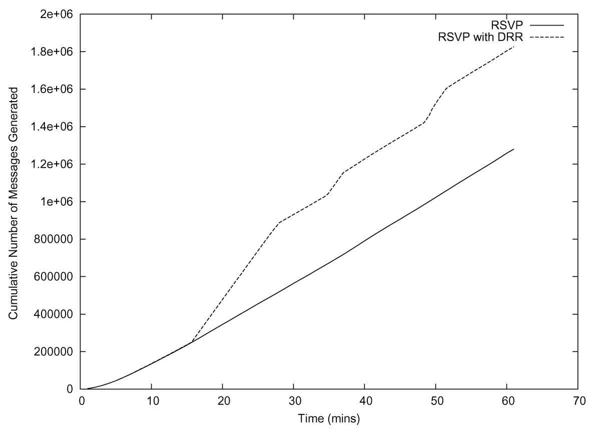


Fig. 13 Message complexity for a network size of 30 nodes with an average node degree of 5

with RSVP, and resource utilization in the network increases. This is a direct consequence of reduced resource fragmentation when RSVP_DRR is used. The magnitude of QoS demand is a factor in the performance of RSVP_DRR. When the QoS request is higher, calls are more likely to be dropped, as it is more difficult to accommodate such calls, and redistribute resources for large QoS requests. The average node degree of the network, and availability of multiple routes to the same destinations also affect the performance of RSVP_DRR. This is due to the fact that resources are redistributed on alternate paths and availability of multiple paths obviously improves the performance of RSVP_DRR.

RSVP_DRR, as explained in the previous sections, uses the Flow Select algorithm to select the flows whose resources may be redistributed in the network. In this specific algorithm, only an even number of flows is selected for resource redistribution. This algorithm could be extended to select an odd number of flows for resource redistribution.

It should be noted that RSVP_DRR is implemented only for unicast communications in this paper. However, RSVP itself was developed keeping multicast communications in mind, and it is currently being supported in many networks that facilitate multicasting. Therefore RSVP_DRR needs to be extended for multicast communications.

The performance of RSVP_DRR is better than RSVP because resources are redistributed on alternate paths. As mentioned earlier, the selection of the flows for redistribution depends on the Flow Select algorithm. However, alternate paths to the destinations of these selected flows are selected at random. Intelligent route selection algorithms can be incorporated for better performance.

Message overhead is a measure of the number of extra messages exchanged by nodes in the network due to the dynamic redistribution of resources. As seen in Fig 13, RSVP_DRR has a higher message overhead (44% more) than that of conventional RSVP. The performance of RSVP_DRR, nonetheless offsets this increase in message count.

Implementing timers in the intermediate nodes may eliminate the exchange of some special messages thereby reducing the message overhead of RSVP_DRR. Optimizing the message overhead is left as future work. For example, sending an explicit S_RTEAR message could be avoided, if timers were implemented in the nodes to release resources after a period of time, if the S_CONFIRM message were not received. This is an alternative to be explored in the future.

Guaranteed Load Service (GLS) [14] of Integrated Service Architecture (IntServ) controls the queuing delay experienced by a datagram and guarantees that this delay does not exceed a maximum limit. This guarantee is valid only as long as the flow conforms to its traffic specification advertised before the data transmission. RSVP is an IntServ signaling protocol used to provide GLS. This factor (delay guarantee) is however not considered by RSVP_DRR while redistributing resources and exploring new paths for the flows of redistributed resources, as the number of intermediate nodes on the new path is unknown. The new path may be longer than the old path, thus possibly increasing the queuing delay guaranteed by GLS. The repercussions of such an increase in queuing delay are yet to be investigated. This issue was not addressed in this work, since the experiments were conducted on the assumption that the maximum allowed queuing delay would not be exceeded. It is possible that in spite of being longer, the alternate path for the flows to be redistributed may not have a longer queuing delay. Nevertheless, it is possible that this assumption may not hold true in all cases, and consequently resource redistribution may affect the guaranteed QoS.

The algorithm presented in this paper presents an extension to RSVP, to facilitate dynamic redistribution of resources. Ongoing research is focusing on the above discussed issues, and their results will be made available.

References

1. Introduction: Quality of Service Overview, CISCO Systems. [Online]. Available: www.cisco.com/univercd/cc/td/doc/product/software/ios120/12cgcr/qos_c/qcintro.htm
2. White Paper—The Need for QoS, July 1999. stardust.com Inc. [Online]. Available: citeseer.ist.psu.edu/275228.html,
3. C. Aurrecoechea, A. Campbell, and L. Hauw, A survey of QoS architectures, Multimedia Systems 6(3) (1998) 138–151.
4. S. Blake, D. Black, M. Carlson, et al., An architecture for differentiated services, (1998) RFC 2475. [Online]. Available: www.faqs.org/rfcs/rfc2475.html.
5. R. Braden, D. Clark, and S. Shenker, Integrated services in the internet architecture: An overview, (June 1994) RFC 1633. [Online]. Available: www.faqs.org/rfcs/rfc1633.html.
6. R. Braden, L. Zhang, S. Berson, et al., Resource reservation protocol (RSVP)—Version 1 functional specification, (September 1997) RFC 2205. [Online]. Available: www.faqs.org/rfcs/rfc2205.html.
7. R. Braden, D. Estrin, S. Berson, et al., The design of RSVP protocol, ISI, Tech. Rep., (July 1996).
8. P. Ferguson and G. Huston, *Quality of Service—Delivering QoS on the Internet and in Corporate Networks*. John Wiley and Sons (1998).
9. B. Jamoussi, R. Callon, R. Dantu, et al. , Constraint-based LSP setup using LDP," (Jan 2002) RFC 3212. [Online]. Available: www.faqs.org/rfcs/rfc3212.html.
10. F. Kuipers, T. Korkmaz, M. Krunz, and P. Mieghem. A review of constraint-based routing algorithms, Technical University Delft, The Netherlands, Tech. Rep., (June 2002).
11. D. Mitzel, D. Estrin, S. Shenker, and L. Zhang, A study of reservation dynamics in integrated services packet networks, in: *INFOCOM* (2) (1996) pp. 871–879.
12. Microsoft Corporation, QoS Technical White Paper, (September 1999) [Online]. Available: www.microsoft.com/windows2000/techinfo/howitworks/communications/trafficmgmt/qosover.asp.
13. S. Shenker and L. Breslau, Two issues in reservation establishment, in: *Proceedings of ACM SIGCOMM'95*, (Cambridge, MA, 1995).
14. S. Shenker, C. Partridge, and R. Guerin, Specification of guaranteed quality of service, (September 1997)RFC 2212. [Online]. Available: www.faqs.org/rfcs/rfc2212.html.
15. W. Stallings, *High-Speed Networks : TCPIP and ATM Design Principles* (Prentice Hall 1997), ISBN: 0135259657.
16. J. Wroclawski The use of RSVP with IETF integrated services, (September 1997) RFC 2210. [Online]. Available: www.faqs.org/rfcs/rfc2210.html.
17. W. Zhao, D. Olshefski, and H. Schulzrinne, Internet quality of service: An overview, Columbia University, New York, Tech. Rep., (2000) [Online]. Available: citeseer.ist.psu.edu/zhao00internet.html