An Efficient Rerouting Mechanism for Dynamic Network Resource Management

Karthikeyan Mahesh
Dept. of Computer Science and Engineering
Indian Institute of Technology-Madras
Chennai 600036
karthik@cs.iitm.ernet.in

S.Lakshminarayanan
Dept. of Computer Science and Engineering
Indian Institute of Technology-Madras
Chennai 600036
lakme@meena.iitm.ernet.in

S.V.Raghavan
Dept. of Computer Science and Engineering
Indian Institute of Technology-Madras
Chennai 600036
svr@cs.iitm.ernet.in

Abstract

Modern multimedia applications require new services from communication systems. Quality of Service(QoS)[6] is one of the core issues in multimedia systems. QoS is established through negotiation between users and service providers. The negotiation typically involves allocation and management of resources in order to attain a desired level of quality. Unfortunately, in present day systems, the network utilization in terms of the number of requests serviced is far from the optimum. In this paper we propose an efficient network resource management mechanism for dynamically handling the requests over a network. We exhibit a method of rerouting existing connections and accommodating new connections through this route in order to maximize the number of connections serviced and the net data transfer through the network. The algorithm also includes renegotiation of QoS parameters for existing connections with the end system to increase the utilization of the network. Simulation experiments indicate that the network utilization, in terms of number of connections served and the amount of data transferred in the network, increases substantially in comparison to existing procedures.

1. Introduction

Modern high speed networks can transmit data at hundreds of megabits per second. This has given rise to a whole spectrum of applications. For the first time, it is possible to use a computer for multimedia applications like teleconferencing — applications once the domain of specialized equipment. In such applications, many audio and video streams have to be transmitted across the network. Traffic of such kind makes certain demands over and above those of normal data transfer that is possible through the network. In particular, they require guarantees on the bandwidth and delay in the communication channel. These guarantees cannot be provided by a network that practices best effort delivery. The concept of **QoS** was evolved to deal with such requirements[4].

QoS is normally specified using three parameters namely bandwidth, delay and jitter[3]. Before data transmission, an application indicates to the network the bandwidth required for the channel, and upper bounds on end to end delay and delay variation. In traditional approaches the QoS requests are granted only if there exists a path with the available resources to service the request. They do not address the issue of rerouting existing connections in order to include new ones. The resources of the network are thereby not effectively managed by the existing methods. The network utilization is also far from optimum.

The main bottleneck in existing methods is the lack of efficient network resource management. In this paper we propose a novel strategy to maximize the network utilization dynamically while providing QoS guarantees. We achieve this by dynamically altering the routes of existing connections in the network if necessary to accommodate new connection requests. The main idea of the paper is to exhibit a method of maximizing the number of connections through the network with the minimum amount of overhead. Thereby the net data transfer through the network is also maximized at any point of time.

In section 2, we describe the design and implementation of this protocol. Both source and switch level rerouting are

0-7695-0253-9/99 $10.00

explained and their differences are studied. In section 3, we study the performance of both rerouting mechanisms and compare them with existing mechanisms. We found that both techniques gave substantial improvements in terms of throughput and response time.

2. Design and Implementation

The goal is to develop a signaling protocol that allows existing connections to be rerouted, if necessary, when a new connection request arrives. The network supports QoS specified by three parameters— bandwidth, delay and delay variation. In traditional signaling mechanisms, a path is chosen between the endpoints and a connection request is sent along that path. The connection is made when the request is accepted by all switches up to the remote endpoint. In case an intermediate switch is unable to provide the required QoS, the connection is refused. We modify this protocol so that the connection is not immediately rejected if an intermediate switch is overloaded. Instead, we keep the request pending, and attempt to reroute an existing connection. If the new state of the network is able to support the required QoS, the connection can be made.

2.1. Connection Setup

The network is considered to be a network of switches. Nodes that are end systems connect to one or more of these switches. Suppose a particular node A wants to make a connection to a node B with QoS parameters Q. The connection setup algorithm proceeds as follows. We find n paths between the nodes A and B, of which one is the shortest. We check whether these paths support the required QoS. For a path to support a given QoS Q, the spare bandwidth at each switch and link must be greater than or equal to the bandwidth requested, and the sum of the processing delays at the switch and the propagation delay over each link should be less than the maximum delay. If such paths are found, we can set up a connection. Initially, we choose the shortest path of these for the route. The other $n-1$ paths are saved, however, in order to achieve fast rerouting if that is required. If paths satisfying the QoS guarantees cannot be found, we proceed to the next stage, described below.

Let us take the n paths identified in the previous stage of the algorithm. At present, none of these paths support the required QoS. Let us consider the first switch in each path that fails the QoS requirements (S in the figure). If any of these switches can offload some connections to others, it may be possible to satisfy the incoming request. For this, a connection C passing through the switch is chosen and is rerouted by giving it the next available path of the n paths chosen for it when C was set up. After this rerouting, the switch S has enough capacity to honor the connection request, which is allowed to pass through the switch. If the request is blocked by another switch downstream, the same procedure is repeated. After the successful completion of this procedure, we get a set of paths. The shortest among them is chosen for the connection, and the rest of them are saved for future use if the connection needs to be rerouted.

We use the following procedure to choose the connection C to be rerouted. First, we observe that after C is rerouted, the switch must have enough capacity to honor the new connection request, i.e.,

$$QoS_C + QoS_{available} \geq QoS_{new\_connection} \quad (1)$$

We choose the connection C with the minimum QoS satisfying the above equation in order to minimize the impact of this rerouting on other switches and links. After the connection C is selected, the switch sends a *Reroute* message to the previous switch in the route of C. This *Reroute* message can be handled in two ways — giving rise to two rerouting strategies: source level rerouting and switch level rerouting.

2.2. Source level Rerouting

Source level rerouting is the simplest form of rerouting. In this technique, the source node maintains a table of paths for each active connection. These are the paths that were found during connection setup. At any given time, the source is using one of these paths for this connection. The paths are ordered from the shortest to the longest. In source level rerouting, we choose paths to be as disjoint as possible.

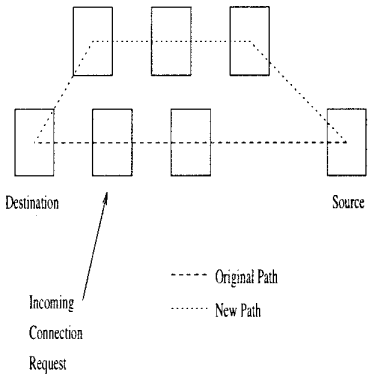

Figure 1. Source Level Rerouting

When a reroute message arrives, the source must attempt to use a different path. It examines each of the paths and verifies whether it meet the QoS requirements (this must be done now because the state of the switches and links along these unused paths may be different from that when

the connection was set up). The best path (shortest path) not passing through the switch S that satisfies the QoS requirements is chosen. The source reserves resources along this path, and updates the connection table, setting this as the present path. The resources along the previous path are released. Now, the overloaded switch S has enough resources to satisfy the incoming connection request. It propagates this connection request to the next switch on its route, continuing the connection setup procedure.

If the source is unable to reroute the connection C (this happens when all other paths in the connection table for C do not meet its QoS requirements), it sends a message to the switch S. S the has to reject the incoming request. This connection will presumably be satisfied by another switch on another route.

The key attraction of source level rerouting in that it is simple to implement. The changes made to a switch are minimal. This keeps the switch implementation simple and fast. On the other hand, every reroute has to be handled by the source. In a large network where the average number of hops between two nodes is high, it takes considerable time to effect a reroute. The connection setup overhead is thus substantial when the network is overloaded. The other strategy we are going to discuss — switch level rerouting — fixes some of these problems.

2.3. Switch level Rerouting

In switch level rerouting, the table of paths is kept in the switches. During connection setup, each switch calculates n shortest paths from itself to the destination. The n paths are stored in a table associated with each active connection. Note that n is relatively a small number in switch rerouting.

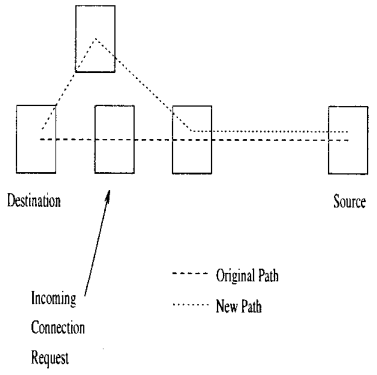

Figure 2. Switch Level Rerouting

As in source level rerouting, a switch that is overloaded sends a *Reroute* message to its predecessor along the path of a connection it has chosen to reroute. When an upstream switch receives a *Reroute*, it uses an algorithm similar to that of the source in source level routing, namely, it finds out which of the paths satisfy its QoS requirements, and chooses the shortest path among them. Resources are then reserved along the new path and the old one is released. This is done without the knowledge of further upstream switches.

When a switch is unable to satisfy the *Reroute*, however, instead of rejecting it outright, it passes it to its predecessor along the path of the connection being rerouted, who follows a similar procedure. This policy is called *crank back*. The crank back is performed as long as switches can satisfy the delay requirements of the old connection. When all switches fail to reroute the connection, the *Reroute* is rejected—which means that the new connection request is refused by the switch. Another variation is the *probabilistic crankback* approach where a switch delivers a *Reroute* message to its predecessor based on a probability measure calculated based on the present load of the switch.

On a lightly loaded network with many interconnections between switches, it is very likely that the reroute succeeds with the first switch itself. It usually does not propagate upstream. This implies that switch level rerouting is much faster than source level rerouting under these conditions, as the new connection request can be satisfied faster. However, when the load on the network increases, many switches may fail to reroute, forcing the performance of switch level rerouting to approach that of source level rerouting. Thus switch level rerouting is a more attractive technique under most conditions. The only drawback of switch level rerouting is that the complexity of switches increases, making implementation of high speed switches more difficult and expensive.

2.4. Communication between Switches

There are various control packets that are exchanged between the switches to keep learning about the state of the network. Each switch keeps monitoring the status of its local neighbourhood with the help of these control packets. Assume that the hop H is identified for a set of connections C and that S_1 and S_2 are the two ends of the hop. A single *HopCont* packet containing information about the QoS status of all the connections for which H is identified as a hop is sent from S_2 to S_1 at regular intervals of time. This back propagation of control packets ensures that all the strategic switches associated with a particular connection have the information regarding QoS status from that switch to the destination at regular intervals of time.

The *Reroute* control packet exchange occurs when connections from a particular switch need to be rerouted and an *OK* or *Fail* message is the corresponding reply packet. The control packets *SetQoS* and *CheckQoS* are responsible for establishing and checking QoS requirements for a con-

nection along a particular path. The control Packet *Close* is set to close a connection. *ReqLowQos* and *AckLowQos* packets are used for lowering the QoS requirements. Two additional control packets *NewSwt* and *AckSwt* are used for inserting new switches in the network dynamically.

3. Performance Analysis

Experiments were performed to test the power of the rerouting mechanisms discussed above. Various simulations were performed and the network utilization achieved by the above models were compared with that of the existing models.

3.1. Setup

In this section, we describe the network that is simulated. The network is represented by a graph with 100 nodes, with the edges in the graph corresponding to communication links. The state of the network is kept in a 100×100 matrix. Each element of the matrix is an ordered triple (B, D, B_o) where B is the bandwidth of the link between the nodes (B is 0 if there is no link between the nodes), D the delay through the link, and B_o is the bandwidth available in the link at any given time. Both D and B_o are variable and depend on the state of the network at that point in time.

The graph consists of several clusters connected to each other. The links within a cluster have large values for B and small values for D while those between clusters have smaller values for B and larger values for D. This resembles the structure of the Internet—with a large number of high speed LANs connected to each other by lower speed lines.

Nodes in the graph are either end-hosts or switches. The network is simulated by generating connection requests between end-hosts. The generated requests are services using both source and switch level rerouting mechanisms.

3.2. Results

The rerouting mechanisms are compared with the existing models based on the number of connections serviced by the network, the net data transfer rate at various time instants and the response time to a reroute request.

The number of connections serviced as a function of time for the 3 models and the is indicated in Figure 3 and is compared with with the maximum possible at that instant.

The graph in Figure 3 clearly shows that the two rerouting mechanisms are much superior to existing methods, with source rerouting performing slightly better.

The graph in Figure 4 indicates the net data transfer through the network with the two rerouting policies and with existing methods.

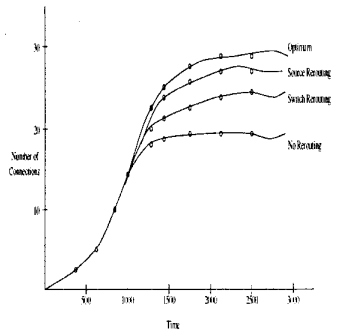

Figure 3. Connections Serviced vs Time

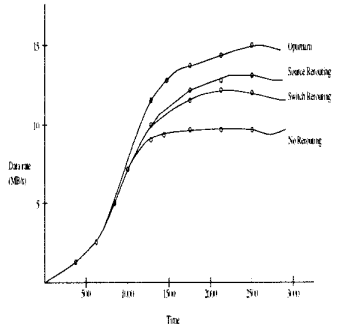

Figure 4. Net Data Transfer Rate vs Time

The graph shows that both rerouting approaches perform much better than existing methods. Source rerouting has a higher data transfer rate than Switch rerouting because it has more paths to choose from.

In the next experiment, the variation of the response times of both rerouting algorithms with network load was studied. Figure 5 summarizes the results.

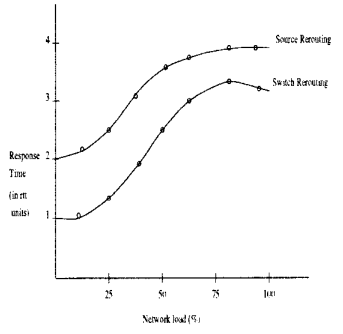

Figure 5. Response Time vs Network Load

We see that at low loads switch rerouting is much faster than source rerouting, but as loads increase the performance of both mechanisms converge towards each other.

The response time of both rerouting algorithms is also measured as a function of simulation time. The results of this experiment are shown in Figure 6.

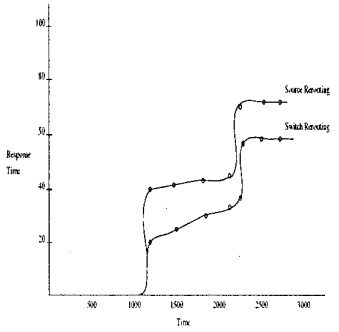

Figure 6. Response Time vs Time

Figure 7 shows the effect of rerouting on an existing connection. In the figure, we plot the end to end delay of a connection that is rerouted to support a new connection.

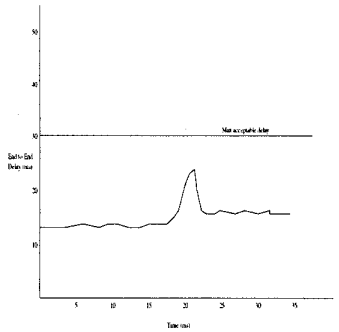

Figure 7. End to End Delay vs Time

4. Concluding Remarks

This paper presents a QoS negotiation and management mechanism through efficient rerouting techniques. We presented two types of rerouting, namely, Source and Switch level rerouting and discussed the mechanisms of implementing them. Results of the simulation experiments show that the performance of the two rerouting mechanisms is phenomenally better than existing methods.

In Switch level rerouting extra functionality is added to the switch and the design of such a switch might be complex. In such cases, one can adopt the Source rerouting mechanism. The performance of the Source rerouting mechanism in terms of number of connections serviced is slightly higher than that of Switch level rerouting, but response time for switch level rerouting is better.

The above mechanism involves parallel bookkeeping about the status of the network, to achieve the optimal alternative route for a connection. The handshake protocol between the switches is simple and straightforward and does not involve much of an overhead. We also ensure that the bandwidth used by the control packets is a meagre percentage of actual bandwidth used for data transfer.

References

[1] M.Ott, G.Michelitsch and D.Reininger and G.Welling, An Architecture for Adaptive QoS and its Application to Multimedia Systems Design, Special Issue of Computer Communications on Building Quality of Service into Distributed Systems.

[2] T.Abdelzaher and Kang G.Shin, End-host Architecture for QoS Adaptive Communication, IEEE Real-Time Technology and Applications Symposium, Denver, Colorado, June 3-5, 1998.

[3] A.Campbell, C.Aurrecoechea, L.Hauw, A review of QoS Architectures, ACM Multimedia Systems Journal, 1996.

[4] A.Campbell, G.Coulson and D.Hutchison, A Quality of Service Architecture, ACM Computer Communications Review, April 1994.

[5] A.Campbell, R.Liao, Y.Shobatake, Delivering Scalable Flows using QoS Controlled Handoff, ATM Forum/97-0341

[6] A.Vogel et al., Distributed Multimedia and QoS: A Survey, IEEE Multimedia, Summer 1995, pp 10-18.

[7] A.Hafid and G.V.Bochmann, An Approach to Quality of Service Management for Distributed Multimedia Systems, International Conference on Open Distributed Processing (ICODP-95), Australia, Feb 1995, pp 319-340.