Efficient Resource Management for Hard Real-Time Communication over Differentiated Services Architectures

Byung Kyu Choi   Riccardo Bettati
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112, USA
{choib, bettati}@cs.tamu.edu
Tel:1-409-845-5469, Fax:1-409-847-8578

July. 25, 2000

Abstract

We propose an efficient strategy for resource management for scalable QoS guaranteed real-time communication services. This strategy is based on sink trees, and is particularly well suited for differentiated-services based architectures. We first show that finding a set of sink-trees in a given network is NP-complete. Then we propose a heuristic algorithm that always efficiently produces a set of sink-trees for a given network. Sink-tree based resource management integrates routing and resource reservation along the routes, and therefore has a number of advantages over other resource management scheme, in terms of: admission probability, link resource utilization, flow set up latency, signaling overhead, and routing overhead. In this paper we show by simulation experiments that even for simple cases the sink-tree based approach outperforms the uniform resource-assignment based approach by up to 40% in terms of admission probability.

Key Words: resource management, sink-tree, QoS, real-time services, end-to-end delay, differentiated services, admission probability, resource utilization.

1 Introduction

This paper addresses an efficient resource management over computer networks in which scalable QoS guaranteed real-time communication services are provided. In this context, by *real-time* we mean that a packet is delivered from its source to the destination within a predefined end-to-end deadline. Packets delivered beyond this end-to-end deadline are considered useless. Internetworking technology is increasingly being used for applications with this kind of requirements, for example, Voice over IP, military communications, and industrial control systems.

Traditionally, *best-effort* service has been the main type of service available over internetworks. While this type of service has contributed much towards the rapid growth of the Internet, it can not support applications that have real-time requirements.

A systematic approach based on the connection admission control and packet scheduling was proposed within the *Intserv* architecture of the IETF [6] to accommodate the real-time requirements. This approach however necessitates that information about every connection is kept by each node along the path for admission control and packet forwarding purposes. The IS architecture therefore does not provide scalable QoS guaranteed services, and the lack of scalability is due to overhead both at connection establishment and during the connection lifetime.

This lack of scalability is, to a large extent, being addressed within the DS (Differentiated Services) architecture [13, 17, 18]. From the user's point of view, the DS model partitions each user-level *flow*[1] into one of a set of predefined *classes*. Packets of each class are served inside the network according to a class based scheduling policy. The result is that routers are aware only of *aggregations* of flows. Each edge router is supposed to aggregate the individual flows into a small number of such aggregate flows. In this fashion, the DS model makes the network scalable regardless of the number of flows.

In order to provide connections with bandwidth and delay guarantees, an admission control mechanism must be in place. The admission control procedure can be centralized or distributed. However, in order to be effective, it must be aware of every node's current workload and admissible resources [20].

The more such such information is available to the admission control procedure, the higher is it effective in terms of admission probability of new connections. There is an obvious trade-off between the *effectiveness* of such a procedure and its *efficiency*, as collecting and maintaining this information would require significant signaling overhead.

There is a research work reporting that signaling and resource management should be taken into account at the same time to maximize the resource utilization in a network [22]. Any admission control procedure inevitably includes some amount of signaling overhead between the decision maker and the nodes involved. As the network grows, the number and length of signaling messages have no way but to increase, therefore

[1] In the following, we will use the term flow to indicate a stream of data between a source and a destination, and the term connection to indicate the virtual circuit that needs to be established to carry the flow.

affecting efficiency and eventually effectiveness of the admission control.

In this paper, we will illustrate how scalability of admission control can be significantly increased by appropriately moving portions of the resource allocation decisions from runtime to configuration time, when the network or the service-level agreements are being (re-)defined.

We will propose *sink trees* as the underlying resource allocation method, and will show how the use of sink trees provides high admission rates and high resource utilization, low connection establishment latencies and low signaling overhead, and low routing overhead.

We will show that the use of sink trees allows for the admission control decision to be done locally at the entrance to the network, i.e., at the ingress node. In this fashion, the signaling overhead becomes nearly free. In addition, we will show that the overhead of maintaining routes is significantly reduced as well with the use of sink trees.

The rest of the paper is organized as follows. In Section 2, we describe previous work. The sink-tree resource management approach is presented in Section 3. In Section 4 we describe the problem of how to find sink trees for a network and show this problem is NP-complete. The end-to-end delay model used in the sink-tree construction is analyzed in Section 5. A heuristic algorithm for the construction of sink trees is presented in Section 6. Experimental evaluation and comments on the implementation are given in Sections 7 and 8, respectively. Conclusions and future works are described in Section 9.

2 Previous Work

Most currently deployed packet forwarding schemes for internetworks do not guarantee end-to-end delays. The delay on current computer networks purely depends on the dynamic load of network links. A number of schemes have been proposed for network services that provide QoS guarantees. Some of the proposals have been implemented and deployed within IS architecture, for example, NetEx, RSVP, and Tenet. NetEx [11, 21], for example, is based on extentions of Cruz's methodology for delay analysis [4], and provides a connection-oriented real-time communication service by relying on strict admission control [8]. RSVP [5] has been proposed as the signaling and resource reservation in the Integrated Services architecture. Tenet Scheme II [7] uses a two-pass resource allocation scheme, with extensive functionality needed for multi-party communication. One common shortcoming of all these systems is their limited scalability. Because they all are flow aware, their signaling and admission control increase with increasing size of the network or number of connections.

Recently, the Boomerang protocol [23] has been proposed as alternative to RSVP, and a complementary role of RSVP along with DS has been addressed [29]. However, Boomerang still adopts the same concept RSVP relies on, namely the soft state and refresh. This makes Boomerang inappropriate for hard real-time services, because the soft state and refresh can not guarantee the allocated resources over the life time of the connection.

Some work has been done on guarantees of end-to-end delay deadlines within DS architecture [27, 30]. The basic idea in [27] is to move per-flow information from core routers to edge routers by relying on estimation. The key points of the estimation technique are: 1) each core router relies on dynamic packet status information for admission control and packet scheduling, 2) dynamic packet status information is carried with each packet. With this information, the core router estimates each flow's dynamic information such as end-to-end delay. In contrast to this, work in [30], significantly reduces the run-time overhead of admission control by doing some of the computation off-line, that is, during network design or configuration. Also, by using a simple packet forwarding mechanism such as class-based static priority scheduling, its runtime overhead is lower than that of [27]. In addition, [30] shows that the lack of flow awareness has little impact on the admission probability. However, neither of the two approaches takes into account overhead for resource management and signaling, which can severely affect their overall performance and scalability.

For the DS architecture, much research needs to be done to provide maximum resource utilization, minimum run-time overhead including both signaling and routing while guaranteeing end-to-end delay deadlines. Our goal is to improve the overall scalability within DS architecture by combining pre-runtime resource allocation with runtime admission decisions. In this paper we present the use of *sink trees* to allocate resources in order to minimize the signaling overhead at runtime. The following section presents sink-tree resource management and how it can be used to achieve high resource utilization and high admission probability while keeping the the signaling and admission control overhead low.

3 Sink Tree-Based Resource Management

The basic idea of sink-tree resource management is to assign as much resources to the link as signaling overhead can be reduced and resource utilization improved by appropriately configuring the network before runtime. In this paper, we assign routes and resources along routes so as to form *sink trees*. For a given destination node on the edge of the network, the sink tree is a directed tree containing all the nodes in the network so that there is a path from each edge node to the destination node. We will show below that, with appropriate allocation of resources along the paths for that tree, the admission decision for a new flow to the destination can be located in the ingress node for that flow. This makes the signaling procedure very simple.

While the benefits of sink trees for signaling are obvious in terms of signaling overhead, they are far from so in terms of effectiveness of the approach. However, as the simulation study shows later in this paper, it outperforms uniform resource assignment strategy (which we will call "flat-fixed" in the following discussion) even in the special case where all nodes have the same resource requirement and all links have the same capacity. In the following we shortly discuss how typical network operations are performed with sink trees.

Edge Node Information Each edge node has two types of information. One is the mapping table between the destination IP address in the input packet and the corresponding IP address and port number of the egress router. The other is the tree information corresponding to the egress router. Tree information includes available bandwidth in the tree, the parent node in the tree, network delay in the tree, and the number of connections in the tree. The information is illustrated in Figure 1.

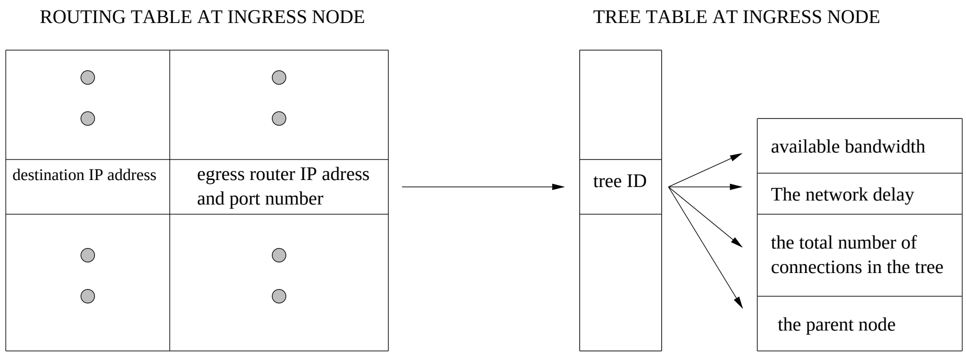

Figure 1: Edge Node Information

Connection Admission During connection establishment, the connection initiator presents the admission controllers with a connection request message, which contains destination IP address, required bandwidth, and required network delay for the connection. Secondly, there should be a connection tear-down message from either initiator or the receiver. Finally, the input traffic is assumed to be regulated by a leaky bucket at the traffic source. Once the ingress node receives a connection request message, it looks up its routing table for the corresponding egress router and port number. From this information it determines which sink tree to use. Then it determines whether to admit or reject the request based on available bandwidth and network delay in that tree. Available bandwidth, network delay in the tree, and total number of connections are updated at each admission or tear-down of a connection.

Routing Once a connection is admitted, a label is given to each input packet according to the sink-tree it belongs to. So, each packet leaving the ingress router for the parent, has a new label in the packet. This label is used in packet forwarding in core (internal) routers in the domain and is deleted when the packet leaves the egress router (root) for a neighboring domain. Consequently the label is effective in a domain only. In other words, the core routers do not see IP addresses, instead, they deal with only the packet label for routing. The label contains the tree ID only. The routing overhead therefore, is minimized since the route is fixed before the first packet is forwarded.

Packet Forwarding The packets with the same label will be multiplexed at the parent node all the way up to the root. Basically, the packet with the same label is forwarded FIFO (First In First Out) bases in its tree. Packets with different labels do not affect each others. At each link, there are FIFO schedulers as many as the number of trees traversing the link. So virtually each link has WRR (Weighted Round Robin) packet scheduler. In that way, the core routers do not need to keep track of each connection. In another view, the core router has no idea about how many connections it supports and which packet belongs to which connection.

4 Model For Finding A Set Of Sink-Trees

Once we have a set of sink-trees for a given network, we can run them as described in the previous section. However, the first question arises naturally if such sink-trees exist for a given network. So we model this problem as a graph design problem.

Problem Formulation With a given graph $G = (V, E)$, we have $\{T_1, \dots T_t\}$, $(T_i \in Z^+)$, $\{C_1, C_2, \dots, C_e, \}$, $(C_j \in Z_0^+)$, $\{R_1, \dots, R_V\}$, $R_i = \{r_{i,1}, r_{i,2}, \dots, r_{i,k}\}$, $(r_{i,j} \in Z_0^+)$, $d_{i,j} = (d_{1,1}, d_{1,2}, \dots, d_{n,n})$, $(d_{i,j} \in Z^+)$ and an integer D. Is there a set of sink-trees which are rooted at each egress router's output port under the three constraints below ?

Where, T_i is the given output bandwidth of a sink-tree. C_l is the link capacity for link l. $r_{i,j}$ is the bandwidth requirement from node n_i to the sink-tree j (i.e., to the egress node's output port j). So R_i is the vector representation of the bandwidth requirements from node n_i to each sink-tree (each output port of egress routers). $d_{i,j}$ denotes the worst-case delay in packet forwarding from node i to node j. It has an infinite value if there is no edge between node i and node j. Finally, Z^+ is the positive integer and Z_0^+ is non-negative integer.

For convenience, in this paper, we differentiate the edge from the link. From now on, we mean an edge a logical association between any two nodes of a sink-tree, while we mean a link a physical association between any two nodes of a given graph. So a link can have many edges on it, each of them belongs to its own sink-tree. Now the three constraints follow.

- *link capacity* : On any link, the sum of bandwidths, allocated on the edges on the link, should not exceed the link capacity c_l. For each link,

$$\sum_{i=1}^{t} B_l^i \leq C_l \tag{1}$$

  Where,
  B_l^i is the bandwidth allocated on the ith edge on link l.
  t is the maximum number of edges on link l. (equal to the number of sink-trees)
  C_l is the capacity of link l.

- *depth-aware bandwidth allocation* : In a sink-tree, the bandwidth allocated to any edge at depth i, should be equal to or greater than the sum of bandwidths of its own children edges at depth $i + 1$ in the tree. We let the root depth 0.

$$B_l^i \geq \sum_{(j=1,m=1)}^{(e_j,l_j)} B_m^j \tag{2}$$

Where,
e_j is the number of children edges (at depth $i+1$) of the parent edge at depth i.
l_j is the number of children links of link l.
Some links in $(l_1, l_2, \cdots, l_j)$ have a child edge of the tree while some of them do not.

- *bounded network delay* : For simplicity, we assume there is the worst-case delay in packet forwarding between any pair of source and destination nodes [4, 8, 12, 30]. In this paper, we use *the network delay* as the sum of delays a packet experiences while it is forwarded throughout the given network and *the node delay* as the delay a packet experiences while it is forwarded from a node to one of its adjacent nodes. So every simple path's delay, which covers from an ingress to an egress node, should be bounded by the given network delay.

$$MAX\{D_{p1}, D_{p2}, \ldots, D_{pb}\} \leq D \tag{3}$$

Where, D_{pi} is the worst-case delay in simple path pi. D is the given network delay bound. Since $D_{pi} = d_{1,2} + d_{2,3} + \cdots + d_{j-1,j}$, (for example, $d_{1,2}$ is the worst-case packet forwarding delay from node 1 to node 2 in the simple path being considered.) D_{pi} is the sum of worst-case local delays in each simple path.

Again, what we are going to do is to answer the question if there is a set of sink-trees for a given network.

THEOREM 1 *Finding a set of sink-trees which meets the three constraints for a given network is an NP-complete problem.*

Proof of THEOREM 1 A naive approach implies the potential complexity of the problem. Let the output ports of the egress nodes be the sinks of sink-trees such that they are numbered from 1 to t. Let n_1 the number of sink-trees meeting the above three constraints for output port 1. So we define n_2 for output port 2, $\ldots$, finally n_t for the last output port t. Since we need to see if there is any set of sink-trees which again meets the above three constraints altogether, we have to try each combination of $(n_1 \times n_2 \times \cdots \times n_t)$ sink-trees. Therefore the complexity depends of the number of spanning trees with a sink at an egress router's port. What it tells is, basically, the complexity comes from a large set of combinations. So we look at the single sink-tree first, instead of considering the whole problem. Because if the single sink-tree problem is polynomialy solved, we can easily guess that the whole problem has roughly t times larger complexity than that of the single sink-tree problem, since each sink-tree is drawn from a given network.

THEOREM 2 *Finding a single sink-tree which meets the three constraints for a given sink and network is an NP-complete problem.*

The proof of theorem 2 is provided in Appendix A.

5 End-to-End Delay Analysis

Network Model In order to calculate the worst-case end-to-end network delay, we consider the network decomposable. That means the network, in terms of delay, consists of two types of packet servers: constant-delay servers and variable-delay servers. We assume the constant delay values are given, then once we get an upper bound on each variable server, we can easily have the end-to-end delay. So, the worst-case end-to-end delay for a flow is obtained by summing up all the worst-case local delays along the route the flow traverses [8].

DEFINITION 1 *The worst-case end-to-end delay for the real-time traffic is formulated as the sum of the worst-case local delays on its route, where:*

$$d = \sum_{k=1}^{S} d_k$$

d is the worst-case end-to-end delay of the real-time traffic flow.
S is the number of nodes which the real-time flow under consideration traverses.
d_k is the worst-case local delay of the real-time traffic at its Server k.

Sink-tree Model In terms of delay, the sink-tree is considered as a fixed resource allocation for a pre-classified traffic. Since we have maximum k number of sink-trees, a link can also have maximum k number of resource allocations. In this paper, we assume that each resource allocation for a sink-tree is exclusively reserved for that sink-tree, such that even though some of allocations are idle a congested allocation can not take advantage of those idle allocations. This can be easily implemented with WRR link scheduling discipline for the sink-trees at each link. In other words, the sink-tree does not share its resource with each other. In a sink-tree, the traffic is treated as FIFO bases.

Traffic Model For simplicity of analysis, we consider a special case in which we have only two classes of traffic: real-time with priority and non real-time best-effort traffic such that the real-time traffic is never delayed by non real-time best-effort traffic. In addition, all flows in the real-time class have the same traffic characterization. Finally the input real-time traffic to the network is constrained by a (β, ρ) function [4], and inside the network, there is no traffic regulator.

THEOREM 3 *Let flows in the real-time class have (β, ρ) input constraint function. Then the output traffic at Server k is bounded a (β, ρ) function with parameters $\bar{\beta}$ and $\bar{\rho}$, where:*

$$\bar{\beta} = \sum(\beta + \rho \sum d_k) \quad (4)$$

$$\bar{\rho} = \sum \rho \quad (5)$$

$\bar{\beta}$ is the burst size at the output of Server k.
β is the burst size of each flow belongs to the real-time class.
d_k is the worst-case local delay of the real-time class traffic at Server k.
$\bar{\rho}$ is the average rate at the output of Server k.

Therefore, once we know the value of d_k, the worst case end-to-end delay is readily obtained by this theorem. Now we derive an upper bound for d_k.

Delay Model The general idea about the delay model is, we derive the worst-case local delay by assuming the maximum possible traffic flows at each server through the network. So at any given time, Server k has $n_{k,j}$ number of traffic flows through its Input Link j. Then, we define $F_{k,j}(I)$ as the input traffic constraint function for the aggregated real-time traffic on Server j's Input Link k. That means $F_{k,j}$ is a function of (β, ρ) based on Theorem 3. On the other hand, since this is an aggregated function, the constraint function is expressed as the sum of individual flows on the same link.

$$F_{k,j}(I) = \sum_{m=1}^{n_{k,j}} H_{k,j,m}(I), \tag{6}$$

where $H_{k,j,m}(I)$ is the constraint function for the m^{th} flow in the real-time class on Input Link j of Server k.

Now, the worst-case local delay for the flow in a sink-tree can be expressed by a general form using the input traffic constraint function in (6).

$$\begin{aligned} d_k &= \frac{1}{C_{l,u}} \max_{I>0}\left(\sum_{j=1}^{N} F_{k,j}(I) - C_{l,u} * I\right). \\ &= \frac{1}{C_{l,u}} \max_{I>0}\left(\sum_{j=1}^{N} \sum_{m=1}^{n_{k,j}} H_{k,j,m}(I) - C_{l,u} * I\right). \\ C_{l,u} &= C_l - (B_l^1 + \cdots + B_l^{u-1} + B_l^{u+1} + \cdots + B_l^t) \\ &= C_l - \sum_{i=1, i \neq u}^{t} B_l^i \end{aligned} \tag{7}$$

Where, $C_{l,u}$ represents the bandwidth assigned to the edge belongs to Sink-Tree u on Link l. Equation (7) says that the worst-case local delay depends on the sum of individual constraint function values which we do not know in advance. However, because we are interested in the bound on the delay, not the exact run-time value which can be known only at run-time, we can introduce a common upper bound $H_k(I)$ for $H_{k,j,m}$. Here, H_k means the traffic constraint function of the flow which experiencing the longest delay from the source to one of the immediate predecessor servers of Server k.

THEOREM 4 *The following inequality holds: for $j = 1, \cdots, N$*

$$F_{k,j}(I) \leq n_{k,j} * H_k(I), \tag{8}$$

where $H_k(I)$ can be formulated as follows based on Theorem 3 and [4]:

$$H_k(I) = \min\{C_{l,u} * I, T + \rho * Y_k + \rho * I\}. \tag{9}$$

where

$$Y_k = \max_{Path \in S_k} \sum_{j \in Path} d_j, \tag{10}$$

T is the burst size of source traffic of the flow in real-time class. S_k is the set of all possible paths upstream from Server k for flows that traverse Server k. Y_k is the maximum delay a flow experienced before arriving at Server k.

A deterministic worst-case end-to-end delay analysis based on network decomposition, inevitably requires the concept *maximum delay, here, named* Y_k which is well presented in [12].

Observing Theorem 4, we can give a bound on d_k in (7) with d_k^* using Inequality (8).

$$d_k \le d_k^* \quad = \quad \frac{1}{C_{l,u}} \max_{I>0} \left(\sum_{j=1}^{N} n_{k,j} * H_k(I) - C_{l,u} * I\right). \tag{11}$$

Since Equation (11) depends on $n_{k,j}$ which can only be known at run-time, we can even replace it with its maximum value on a link. Intuitively, the value of $n_{k,j}$ is bounded by the resource allocated for the class and sink-tree the flow belongs to.

$$\sum_{j=1}^{N} n_{k,j} \le \frac{B_l^u}{\rho}. \tag{12}$$

By substituting Inequality (12) into Equation (11) we have:

$$d_k^* \quad \le \quad \frac{1}{C_{l,u}} \max_{I>0} \left(\frac{B_l^u}{\rho} * H_k(I) - C_{l,u} * I\right). \tag{13}$$

Eventually Inequality (13) has no run-time variable any more. The only one thing not determined yet with it is the maximum function. In fact, it depends on the distribution of $n_{k,j}$ on the input links. Fortunately, a paper addressing resource-utilization based admission control reports that in this case, the worst case happens when the number of input flows are evenly distributed over Server k's input links [31]. So we safely use the following theorem giving the maximum d_k^{**} for Inequality (13).

THEOREM 5 *The delay in (11) is maximized when*

$$\max\{n_{k,1}, n_{k,2}, \cdots, n_{k,N}\} = \lceil \frac{B_l^u}{\rho N} \rceil. \tag{14}$$

Consequently, the final result for the worst-case local delay is given by substituting (14) into Inequality (13):

THEOREM 6 *Finally the delay d_k is bounded as follows:*

$$\begin{aligned} d_k \leq d_k^{**} &= (T + \rho * Y_k)\frac{\alpha}{\rho} + (\alpha - 1)\frac{\alpha(T + \rho Y_k)}{\rho(N - \alpha)}. \\ \alpha &= \frac{B_l^u}{C_{l,u}} \end{aligned} \tag{15}$$

Where, α is the portion of link bandwidth assigned to the real-time priority class traffic in Sink-Tree u on Link l. This is necessary because the sink-tree does not share resource with each other. The proof of Theorem 6 is provided in [31] with a fixed α over the all links of the given network. The difference between this work and [31] is that in the sink-tree case, the worst-case local delay is repeatedly recalculated whenever a new node is added to form a complete sink-tree in the sink-tree construction process as shown in the next section, while in [31] the worst-case local delay is calculated off-line under the assumption that the network is loaded to the extent possible.

Because we are going to simulate the two classes only later in this paper, so far, we derived an upper bound for d_k for the case, where there are only two classes: real-time priority class and non real-time best-effort traffic for simplicity. The generalized equation for Equation (15) is provided in [28] again with a fixed α, where multiple real-time classes are treated with different priorities. Therefore Equation (15) is a special case of [28] for the single real-time priority class traffic divided into non resource-sharing sink-trees. The worst case end-to-end delay obtained here is used in the heuristic algorithm for constructing sink-trees proposed in the next section.

6 A Heuristic Approach For Constructing Sink-Trees

Although finding a set of sink-trees for a given network under the three constraints is NP-Complete, we need a practical algorithm which can always produce a set of sink-trees in polynomial time which may or may not have the reduced requirements in both bandwidth and network delay. The reason is, even though the set of sink-trees produced by a heuristic algorithm may not satisfy all constraints, we can still run such sink-trees as our resource allocation strategy. Therefore, what we try to achieve by a heuristic algorithm is, to always produce a set of sink-trees in polynomial time with the minimum network delay and with or without reduced constraints for a given network. The reason why we give priority to the network delay is that because we construct the sink-tree instead of searching for the best, we do not know whether the result will violate the network delay at the end. So it would be always safe to construct the sink-tree with the locally minimum path delays. With the link capacity, we think that it is usually large enough to accommodate the path bandwidth requirement like in [30]. So it will relatively be unlikely that the capacity constraint is violated prior to the network delay constraint when we construct a sink tree. However, because we do not try to find an optimal value, which might be a minimized reduction in requirement, we do not say either our heuristic algorithm produces a best possible sink-trees or it guarantees performance with a certain percentage of difference from the optimal. The reduction has two cases. One is for the link capacity and the other is for the network delay. In other words, if a sink-tree has bandwidth requirements which exceed some link capacities, we reduce the bandwidth requirement down to the point where no bandwidth requirement exceeds any link capacity in the tree. Likewise, if a path delay exceeds the network delay requirement, we reduce the network delay requirement so that the path delay does not exceed the network delay any more. Consequently, our basic idea of heuristic algorithm is represented in Figure 2 by a flow chart.

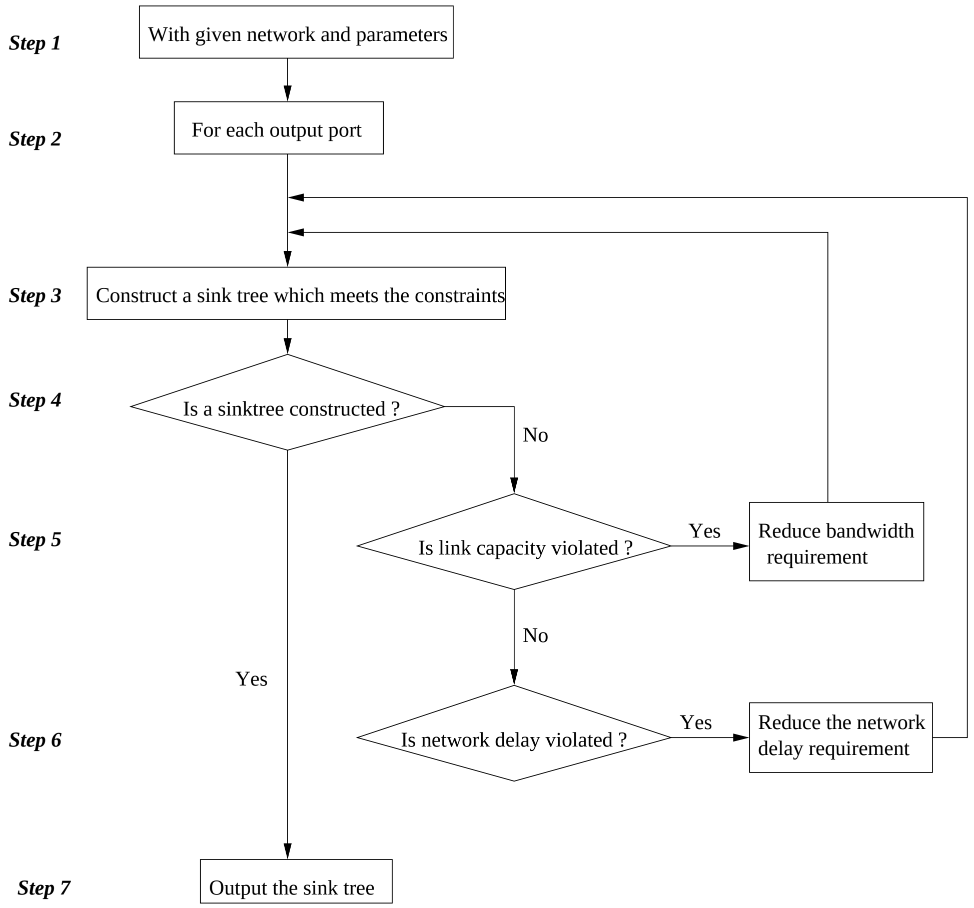

Figure 2: A heuristic algorithm for a set of sink-trees

Algorithm As seen in Figure 2, the algorithm always stops with a sink-tree for each output port. So if the given network has t number of output ports, the algorithm iterates exactly t times. Since each iteration in turn, has two internal loops (Step 5 and 6), the run-time complexity of the algorithm eventually depends on Step 5 and 6. Since we wish to construct the sink-tree in polynomial time, we limit the execution of the loops for just one time for each violation such that the loop executions are bounded by the number of edges $|E|$ for each sink-tree. Now we consider Step 3, which is the core algorithm for each sink-tree. Basically, to construct a sink-tree is to make a spanning tree with given parameters. So we consider it as *capacitated, delay-bounded spanning tree*. As the name implies this is easily implemented with some modifications on MST (Minimum Spanning Tree [3]). The difference between the two is, our algorithm works under the two constraints, capacity and delay, while MST algorithm takes only one thing into account, the weight. Each node delay here, is considered as weight in MST. The algorithm of Step 3 is shown in Figure 3.

```
1.  T = {v}
2   Make all neighbors of v fringers
3   While T is not spanning do
4           For each fringer u in the order of increasing path delay
5                   If it satisfies the path delay
6                           Then if it satisfies the path capacity
7                                   Then go to line 11
8                                   Else stop and return the capacity violation
9                           Else stop and return the delay violation
10          If there is no such fringer, then stop and return the sink tree
11          Include u in T
12          Delete u from fringers
13                  For each neighbor w of u do
14                          If w is a fringer
15                                  Then make w a fringer with shorter path delay
16                          Else if w is not in T
17                                  Make w a fringer
```

Figure 3: A heuristic algorithm for a sink-tree

THEOREM 7 *The heuristic algorithm always produces a set of sink-trees with or without reduced requirements on the capacity and/or the end-to-end delay in polynomial time.*

The proof of theorem 7 is provided in Appendix B.

7 Experimental Evaluation

In order to evaluate the performance of the sink-tree resource management, we ran a suite of simulation experiments to compare it with a flat-fixed resource management [30], where every link has the same link capacity of 155 Mbps and a fixed portion of the link capacity is set aside for the real-time traffic. In this experiment, we compare the two systems in terms of admission probability and link utilization.

To be fair in comparison, we use the same simulation environment and parameters which were used in [30]. First of all, Figure 4 shows the topology of the MCI ISP backbone network, which we use throughout the experiments. In these particular experiments, all routers can act as edge routers and intermediate routers as well. Other simulation parameters for the flat-fixed system remain the same. So, we simulate the admission control behavior in the system by simulating flow requests and establishments at varying rates with a constant average flow lifetime. Requests for flow establishment form a Poisson process with rate λ, while flow lifetimes are exponentially distributed with an average lifetime of 180 seconds for each flow. Source and destination edge routers are chosen randomly however, the flow is set up differently for the two systems. In the flat-fixed system, the path is selected based on SPF (Shortest Path First) routing strategy which most modern routers practice. In the sink-tree system however, since each edge has a path to every other node in the network, the path is pre-defined from any node to any other node.

We consider a system with two classes. One is real-time class. Another is non-real-time class. We assume that all flows in the real-time class have a fixed packet length of 640 bits (RTP, UDP, IP headers and 2 voice frames) [14], and a flow rate of 32 Kbps. The end-to-end delay requirement of all flows is fixed at 100ms.[2] Thus, the input traffic of each flow is constrained by a leaky bucket with parameters $T = 640$ bits and $\rho = 32$ Kbps. This kind of QoS requirements is similar to "Voice-over-IP" [24]. All links in the simulated network have the same capacity of 155 Mbps and 10.541935 % of the link capacity is allocated for the real-time applications. In other words, every link can accommodate 510.62498 flows at the same time. This value comes from the fact follows.

For the sink-tree system we set each node in the given network has the same bandwidth requirement from it to every other node, in this particular case, it is 40 flows (or 1.28 Mbps). After the construction of sink-trees, the system has 19 sink-trees in it, as many as the number of nodes. Because each node has its own sink-tree. The total sum of bandwidth assigned to each link in sink-tree divided by the number of total links gives 510.62498, which corresponds to 10.541935 % of 155 Mbps-capacitated link. Practically speaking, this resource assignment is preferable for the flat-fixed system. Because if we have different bandwidth requirements for the nodes, the sink-tree system which reflects the requirements will definitely accommodate more requests than the flat-fixed system which blindly assigned resource with a flat rate.

Figure 6 shows the admission probabilities for the real-time class in the four cases as a function of arrival rates. The curve named "flat-fixed" represents admission probabilities for the flat-fixed system mentioned above for comparison. Three other curves show the admission probabilities of sink-tree system where, we classified three cases in resource sharing. Firstly the title "no-sharing" means the case in which each node can not use any other node's resource even if some of them are idle. For example in Figure 5, all nodes 1, 2, and 3 have the same bandwidth requirement of 10 to the root (Node 4). Accordingly, after sink-tree construction, each link has allocated capacity of 10, 20, and 30 from left. Now suppose node 2 has 10 flows established already while node 1 has none. In no-sharing strategy, a new flow request from node 2 to root 4 is rejected even though

[2] In a real situation this value is not correct, as we only consider queuing delay for end-to-end delay deadline. There are many other factors like propagation delay, Codec (Coder and Decoder) delay or others we do not know now. More accurate values can be found in [14]. We use this value is just for the reference. With other values, we would see similar results.

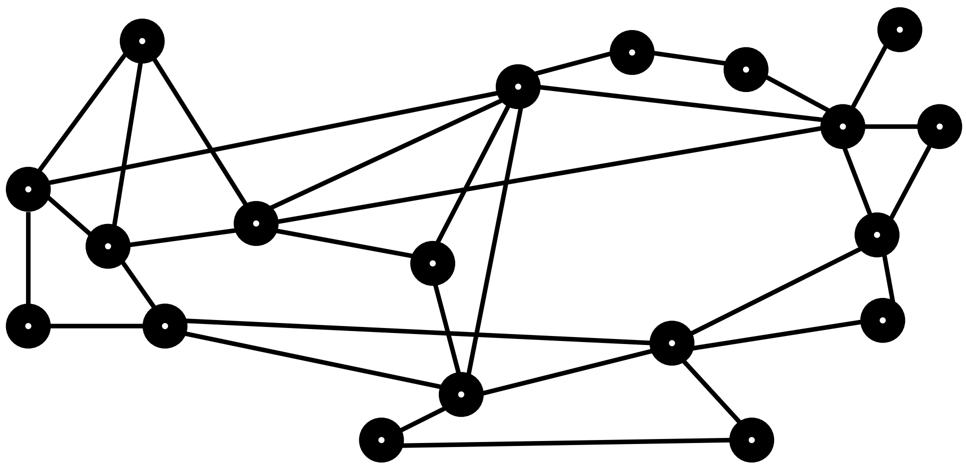

Figure 4: A network example

node 2 still has room which was originally reserved for node 1. Secondly in path-sharing strategy on the other hand, the new request at node 2 to root 4 is admitted as long as node 2 has resource unused. So the resource is shared only in a path in the tree. Lastly in tree-sharing strategy, the new request is accepted as long as there is one tree which has path from node 2 to root 4, and the path has still resource unoccupied.

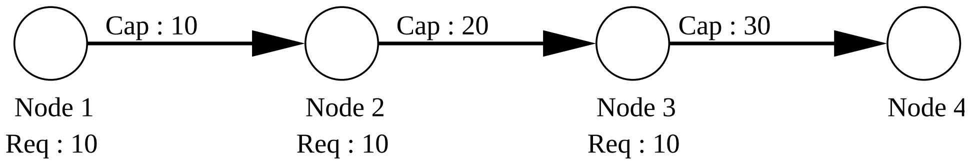

Figure 5: An illustration of resource sharing

As can be seen in Figure 6, the three cases of sink-tree systems perform better than the flat-fixed one. Even no-sharing system outperforms the flat-fixed one. In the case of no-sharing, if we wish to allow 90% chances of flow admission, no-sharing system's performance is approximately 40% larger than that of flat-fixed one. The importance of this result is that even with the same bandwidth requirement at every node and the same link capacity, so it sounds like there will be no or a little difference between sink-tree and flat-fixed strategies, the sink-tree structured resource management brings a lot of benefit. Therefore, even if the bandwidth requirement is not given at the network design time, a simple assumption of the same requirement at every node will promote the revenue up to about 40%.

The differences between the three resource-sharing strategies in the sink-tree system are not very big. At 90% chances of flow admission point, tree-shared strategy outperforms no-sharing by about 7%, which in return requires search time among trees at admission time. The evaluation of searching cost is beyond of the scope of this paper. Furthermore, the admission performance of tree-sharing can be more improved by adopting link-sharing in which every resource is shared link by link not by tree by tree. In that case however, we can not apply the end-to-end delay formula (15) any more since the equation assumes WRR packet scheduler at each link. Moreover, as Figure 7 shows, the improvement is expected only a little since the resource is almost exhausted already. We expect that the difference will shrink more if only border routers are edge routers. So we attribute the difference to the topology of the given network.

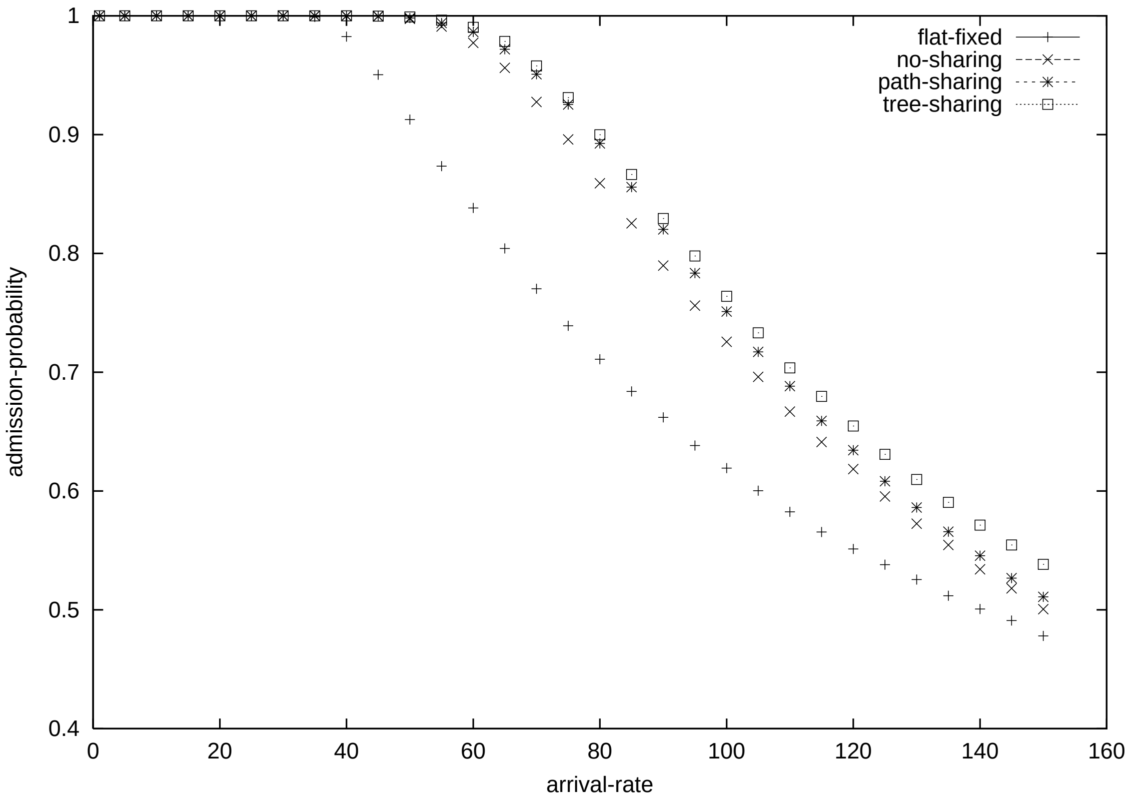

Figure 6: Admission probabilities

Figure 7 shows the utilization ratios for the four cases. To be fair, the link utilization is defined by the number of total flows in the network divided by the total resources alloted. By comparing Figure 6 and Figure 7, we observe that at 90% chances of flow admission point, the flat-fixed strategy utilizes resources only around 63%, while no-sharing does around 86%. It says that even no-sharing strategy outperforms flat-fixed by about 37% in link resource utilization. The figure also shows that in tree-sharing strategy, after arrival rate of 80 flow-requests per second, the utilization slope is getting close to horizontal line. Correspondingly the admission probability in Figure 6 after the arrival rate of 80, goes down almost linearly. So the two figures support each other.

8 Implementation Consideration

The sink-tree resource management concept can be implemented in many ways. Generally speaking however, the implementation is divided into two parts. One is the sink-tree construction and the other is the signaling.

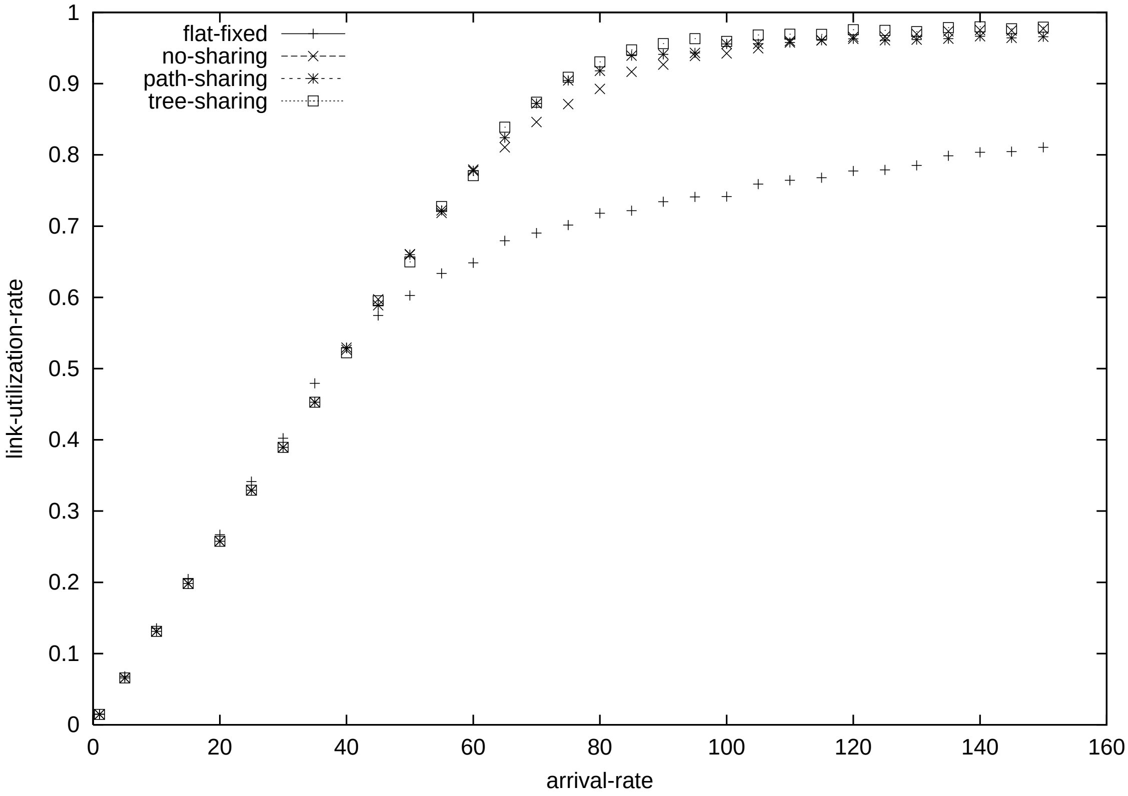

Figure 7: Link utilization

Because the construction is performed off-line at network design time, it is not an issue. The signaling is in turn, divided into two parts. One is a middle-layer signaling immediately below the user-layer signaling like H.323 [9] or SIP [19]. The other is a lower-layer signaling for the maintenance of sink-tree itself. The middle-layer signaling will be in charge of flow request, admission decision, and flow tear-down. In case of H.323, the gatekeeper call control signaling [15] can be combined with the sink-tree middle layer signaling. Anyway the middle-layer signaling will be completely different from RSVP [10], because there will be no more refresh and soft state control. The study on the middle-layer sink-tree protocol is future work.

The lower-layer signaling can be implemented by any protocol which supports label switching concept [16]. However, MPLS (MultiProtocol Label Switching) has been well studied [26], recently for DS architecture [25], we think the sink-tree can be easily supported by MPLS. The details for how to support the sink-tree with MPLS is for further study.

9 Conclusions

We have proposed an efficient resource management named ”sink-tree” strategy for scalable QoS guaranteed real-time communication services, especially in DS architecture. Because the finding a set of sink-trees in a given network is NP-complete problem as proved in Appendix A, we proposed a heuristic algorithm which always produces a set of sink-trees with or without reduced requirements in polynomial time for the given network.

The sink-tree resource management brings a lot of benefit on: admission probability, link resource utilization, flow set up latency, signaling overhead, and routing overhead. However, as the first work on the sink-tree, we showed by simulation experiments that even with uniform bandwidth requirement and uniform link capacity, the sink-tree outperforms flat-fixed system by up to 40% improvement in admission probability. This result strongly implies that the sink-tree system will have a broad positive impact with other benefits which have not been studied yet in full.

Now we are extending this work with link-by-link resource sharing with a new worst case end-to-end delay analysis, in which each link server has only one FIFO server instead of having as many FIFOs as the number of sink trees. The link-by-link resource sharing within sink-tree paradigm will utilize the resource in maximum.

Also, since the sink-tree's benefit on admission probability and resource utilization has been shown as the baseline, we will study the benefit on signaling overhead by comparing with the generic signaling procedure within DS architecture.

In practical point of view, VoIP (Voice over IP) or Internet Telephony system can make use of the sink-tree system. Because the expected QoS by users will be close to that of the current telecommunication system, in order to provide the same or better QoS than the telecommunication system does, a new resource management strategy like sink-tree will be necessary because current industrial resource reservation scheme over the computer network can not support hard reservation and simple signaling.

Lastly, the implementation can be easily achieved with the help of MPLS system combined with ITU-T H.323 standard, especially with gatekeeper call control signaling. Because the gatekeeper has its own operational zone, it can easily act like a bandwidth broker for a given network.

References

[1] Christos H. Papadimitriou, "The Complexity of the Capacitated Tree Problem," *Technical Report No. TR-21-76*, Department of Computer Science, Harvard University, 1976.

[2] M. R. Garey and D. S. Johnson, "Computers and Intractability," 1979, W. H. Freeman and Company.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, "Introduction to Algorithms," *Technical Report No. TR-21-76*, 1990, The MIT Press.

[4] R. L. Cruz, "A Calculus for Network Delay, Part and I Part II," *IEEE Trans. on Information Theory*, Jan. 1991.

[5] L. Zhang, S. Deering, D. Estrin, S. Shenker and D. Zappala, "RSVP: a new resource reservation protocol," *IEEE Networks Magazine*, vol. 31, No. 9, pp. 8-18, September 1993.

[6] R. Braden, D. Clark and S. Shenker, "Integrated Services in the Internet Architecture," *RFC 1633*, Jun. 1994.

[7] R. Bettati, D. Ferrari, A. Gupta, W. Heffner, W. Howe, M. Moran, Q. Nguyen, and R. Yavatkar, "Connection establishment for multiparty real-time communication," *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video*, Durham, New Hampshire, Apr. 1995.

[8] A. Raha, S. Kamat, W. Zhao, "Guaranteeing End-to-End Deadlines in ATM Networks," *The 15th IEEE International Conference on Distributed Computing Systems*, in 1995.

[9] G. Thom, "H.323 : The Multimedia Communications Standard for Local Area Networks," *IEEE Communications Mag.*, Dec. 1996.

[10] P. P. White, "RSVP and Integrated Services in the Internet: A Tutorial," *IEEE Communications Mag.*, May 1997.

[11] B. Devalla, C. Li, A. Sahoo, and W. Zhao, "Connection-oriented real-time communication for mission critical applications-an introduction to NetEx: a portable and efficient tool kit," *Aerospace and Electronics Conference*, NAECON 1997., pp. 698 - 707 vol.2 1997.

[12] C. Li, R. Bettati, W. Zhao, "Static Priority Scheduling for ATM Networks," *IEEE Real-Time Systems Symposium (RTSS'97)*, San Francisco, CA. Dec. 1997.

[13] K. Nicols, V. Jacobson, L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet," *Internet-Draft*, Nov. 1997.

[14] Thomas J. Kostas, Michael S. Borella, Ikhlaq Sidhu, Guido M. Schuster, Jacek Grabiec, and Jerry Mahler, "Real-Time Voice Over Packet-Switched Networks," *IEEE Network Mag*, Jan./Feb. 1998.

[15] "ITU-T Recommendation H.323 : Packet-based multimedia communication systems," *ITU-T H.323*, Feb. 1998.

[16] A. Viswanathan, N. Feldman, Z. Wang, R. Callon, "Evolution of Multiprotocol Label Switching," *IEEE Communications Mag.*, May. 1998.

[17] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, "An Architecture for Differentiated Service," *RFC 2474*, Dec. 1998.

[18] Y. Bernet et al., "A Framework for Differentiated Services," *Internet-Draft*, IETF, Feb. 1999.

[19] M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg, "Session Initiation Protocol," *IETF RFC 2543*, Mar. 1999.

[20] X. Xiao, L. M. Ni, "Internet QoS: A Big Picture," *IEEE Network Mag.*, March/April, 1999.

[21] A. Sahoo, B. Devalla, Y. Guan, R. Bettati, and W. Zhao, "Adaptive connection management for mission critical applications over ATM Networks," *International Journal of Parallel and Distributed Systems and Networks*, Special Issue On Network Architectures for End-to-end Quality-of-Service Support, to appear.

[22] P.Goyal, A. Greenberg, C. R. Kalmanek, W. T. Marshall, P. Mishra, D. Nortz, and K. K. Ramakrishnan, "Integration of Call Signaling and Resource Management for IP Telephony," *IEEE Network Mag.*, May/June 1999.

[23] G. Feher, K. Nemeth, M. Maliosz, "Boomerang : A Simple Protocol for Resource Reservation in IP Networks," *IEEE Real-Time Technology and Applications Symposium*, June 1999.

[24] F. Borgonovo, A. Capone, L. Fratta, C. Petrioli, "End-to-End Call Admission Control for Telephony Services over IP Networks," *IEEE Network and IEEE Internet Computing joint issue*, submitted.

[25] I. Andrikopoulos, G. Pavlou. "Supporting Differentiated Services in MPLS Networks," *IEEE International Workshop on QoS*, May/June 1999.

[26] I. Widjaja, A. Anwar, "Performance Issues in VC-Merge Capable Switches for Multiprotocol Label Switching," *IEEE Journal on Selected Areas in Communications*, Vol. 17, No. 6, June 1999.

[27] I. Stoica, H. Zhang, "Providing Guaranteed Services Without Per Flow Management," *ACM SIGCOMM*, Sep. 1999.

[28] B. Choi, D. Xuan, C. Li, R. Bettati and W. Zhao, "Technical Report TR00-007," Texas A&M University, Department of Computer Science. Jan. 2000.

[29] Y. Bernet, "The Complementary Roles of RSVP and Differentiated Services in Full-Service QoS Network," *IEEE Communications Mag.*, Feb. 2000.

[30] B. Choi, D. Xuan, C. Li, R. Bettati, W. Zhao, "Scalable QoS Guaranteed Communication Services for Real-Time Applications," *The 20th IEEE International Conference on Distributed Computing Systems*, Taipei, Taiwan, pp. 180-187, April 2000.

[31] D. Xuan, C. Li, R. Bettati, J. Chen, W. Zhao, "Utilization-Based Admission Control for Real-Time Applications," To appear in *The IEEE International Conference on Parallel Processing*, Canada, Aug. 2000.

Appendix

Reviewers are welcome to take a look at these proofs and we will be more than happy to receive any comments on these. These proofs are placed here to improve the overall readability of this paper. However, if this paper is accepted and these proofs are too long for the number of pages allowed, we will make these available as a technical report of TAMU with any comments reflected.

A Proof of THEOREM 2

Following the traditional method of proving for NP-completeness, we reduce a well-known NP-complete problem, "Satisfiability Problem in Conjunctive Normal Form [2]" to the single sink-tree problem. The reduction technique we use here is basically the same as in [1] with a different point of view, where the capacitated spanning tree problem is proved to be NP-complete. The reason for using the same technique is, even though the capacitated spanning tree problem looks similar to the single sink-tree problem, the constraints are different in that the former has an integer D as a bound for the sum of costs of all the edges and the latter has an integer D as a bound for the cost of a simple path. While the proving technique applies for the both cases.

Let SAT be an instance of the satisfiability problem with variables $\{X_1, X_2, \ldots, X_n\}$ and clauses $\{C_1, C_2, \ldots, C_m\}$. Now we construct an instance $SINKTREE$ of the sink-tree problem with $2n + 2m + 1$ nodes, $n + m$ capacity for each link, unit bandwidth requirement (1) for each node, unit or two units node delay (1 or 2) for each edge, and lastly the network delay bound of $n + 2$, such that this instance has a feasible solution (sink-tree) if and only if SAT is satisfiable.

For each variable X_i, of SAT, $SINKTREE$ has two nodes X_i and $\bar{X}_i$, where each pair of $(X_i, \bar{X}_i)$ are connected to the next pair of $(X_{i+1}, \bar{X}_{i+1})$. The node delays from nodes X_i, $\bar{X}_i$ to nodes X_{i+1}, $\bar{X}_{i+1}$ for $(i = 1, 2, \ldots, n-1)$ are all units (1). The node delays from the sink to $(X_1, \bar{X}_1)$ are units too. For each clauses C_j of SAT, there are two nodes c_j and d_j in $SINKTREE$. The nodes $d_2, \ldots, d_m$ are linked to d_1, which in trun linked to $(X_n, \bar{X}_n)$. Again, all node delays from $(X_n, \bar{X}_n)$ to d_1, and from d_1 to $(d_2, \ldots, d_m)$ are units too. Finally, we join c_j and X_i or $\bar{X}_i$ if the literal X_i or $\bar{X}_i$ is in the clause C_j. Note here that the node delays from X_i or $\bar{X}_i$ to c_j are all two units. This completes the construction of $SINKTREE$. An illustration for the Boolean formula $B = (X_1 + X_2)(\bar{X}_1 + X_2 + X_3)(\bar{X}_2 + \bar{X}_3)$ is shown in Figure 8. We will now prove that SAT is satisfiable if and only if $SINKTREE$ has a sink-tree with the network delay bound of $n + 2$.

First, if $SINKTREE$ has a sink-tree, that means the sinktree must have met the three constraints. With link capacity constraint, since each node has a unit bandwidth requirement, the sink-tree must have $2n + 2m$ capacity in total, because the sink node can not be counted for source. Moreover, since each link has capacity of $n + m$, and there are only two links connected to the sink, the sink-tree should have two branches with the equal capacity of $n + m$. In this case, since the bandwidth requirement of each node is just 1 and each

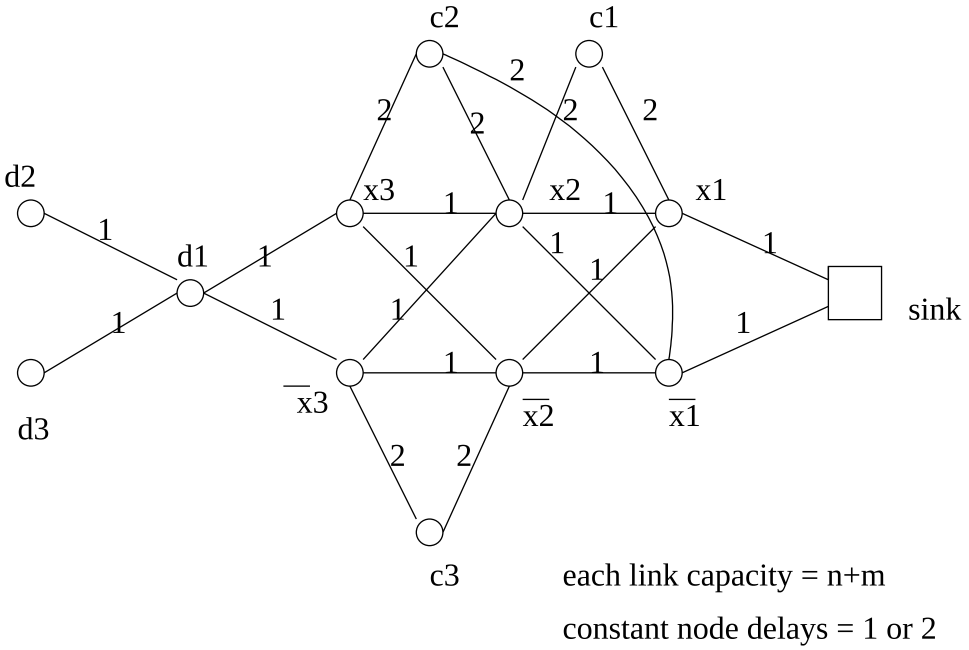

Figure 8: An illustration of the reduction

link capacity is equally $n+m$, the second constraint "depth-aware bandwidth allocation" is satisfied trivially. To meet the third constraint, the network delay should be less than or equal to $n+2$. Now we need to consider the node d_1. Because it connected to $(X_n, \bar{X}_n)$ and all nodes from d_2 to d_m are connected to d_1, all nodes from d_1 to d_m should belong to only one branch of the tree. In addition, since they have exactly two units of node delays (2), the branch having those nodes should have exactly one node from each pair of $(X_i, \bar{X}_i)$ nodes. Because if it has any node c_j, it will violate the network delay bound $n+2$. Consequently, the other branch must have the other node from each pair of $(X_i, \bar{X}_i)$ nodes plus all c_i nodes $(c_1, c_2, \cdots, c_m)$, such that the other branch has at most $n+2$ network delay for the worst case, which means it has to have the nodes from c_1, to c_m as its leaves. Otherwise it will violate the network delay bound either. Now, we consider that the first branch is the false assignment for the set of variables $(X_1, \bar{X}_1, \ldots, \bar{X}_n)$ of SAT. Then since the second branch has nodes $(c_1, c_2, \ldots, c_m)$ as its leaves, we consider the nodes from each pair of $(X_i, \bar{X}_i)$ connecting to $(c_1, c_2, \ldots, c_m)$ as true assignment for SAT. Because each clause C_i has at least one edge connected to $(X_i, \bar{X}_i)$, which is true assignment.

Second, if SAT has a solution, there must be a set of true assignments to the set of variables $(X_1, X_2, \ldots, X_n)$. That means it takes exactly one node from each pair of $(X_i, \bar{X}_i)$. For those c_j which have more than one edges connected to the $(X_i, \bar{X}_i)$, we remove the edges leaving only one to make them leaves of the branch. So the branch has the network delay of $n+2$ at most. Consequently, the rest of nodes in the pairs of $(X_i, \bar{X}_i)$ plus d_i nodes $(d_1, d_2, \ldots, d_m)$ construct the other branch which meets the network delay of $n+2$. So $SINKTREE$ has a solution.

In this reduction, we used the unit bandwidth requirement, the equal link capacity of $n+m$, and unit or two units of node delays (1 or 2). However, with unit node delay only for all edges, the reduction does not hold the corrrectness any more. In other words, $SINKTREE$ can have a solution even if SAT does not, under the condition that all edges have the constant unit delay in packet forwarding. This case may or may not be proved NP-complete with another reduction technique. However, since this case is neither general nor practical, we do not pursue its complexity here any more. Exactly speaking, the proof says that the single sink-tree problem is NP-complete in general where the node delay has two or more values. Therefore, in the strong sense, $SINKTREE$ has a solution if and only if SAT has a solution, so is NP-complete.

B Proof of THEOREM 7

Here we prove that 1) the algorithm always produces a set of sink-trees in polynomial time and 2) the sink-trees have the minimum network delays. The polynomial time proof follows first. In line 1 in Figure 7, the vertex v is the specific egress router as the root. So, it is safe that we consider it the initial spanning tree T. In line 2, all neighbors of v become fringers. The fringer is defined that if a node is a fringer, it is out of the spanning tree but connected with it by an edge. In other words, if an edge is a fringer, it has one end in the spanning tree and the other end in the out of the spanning tree. In line 3, all the neighbors of v have become fringers. In line 4, the fringer u of minimum path delay is chosen to be tested. If there are more than one fringers which have the same path delay, we choose the one which has the minimum path capacity requirement. Through line 5 and 6, if node u has been verified appropriate, then it is added to T in line 11 and deleted from the fringers in line 12. This procedure guarantees that the new T is still the spanning tree we are trying to achieve, because it satisfies the path delay and path capacity requirements. By the path capacity requirement, we mean that the simple path which contains the new selection (node u), has no problem in capacity in accomodating the bandwidth requirement of node u. In other words, all ancestor edges have enough capacity to accomodate the additional bandwidth requirement of u. In line 10, the algorithm normally stops and returns with the constructed sink-tree. In line 13, we consider the neighbors of u, which may or may not the fringers. If they are in the fringers already, we make w the fringer which has the shorter path delay. Otherwise, if they are not in T, we add them in the fringer in line 17. Again, all nodes are considered in line 3. In terms of time complexity, we visit every edge in line 3. Every edge, in turn, visits every node in line 4. So the time complexity of Step 3 in Figure 2 is $|V| \cdot |E|$. Again, in Figure 2, Step 5 may cause another visit up to the sink from each node being visited by an edge in Figure 7, which multiplies the time complexity of Fig. 7 by $|E|^2$. Step 6 in Figure 2 does not require additional time complexity since the network delay can be reduced immediately without referencing the sink-tree being constructed. So, the total time complexity of the whole heuristic algorithm, with a straightforward data structure is given by $k \cdot |V| \cdot |E|^3$. Therefore the whole algorithm in Figure 2 always produces a set of sink-trees in ploynomial time. Second, the proof for the minimum network delay is obvious with line 4 in Figure 7.