An Efficient Signaling Structure for ATM Networks

Dakang Wu
Department of Computer Science
Washington University
St. Louis, Missouri 63130-4899

Abstract

As ATM becomes widely accepted as the communication standard for high speed networks, the signaling system structure and protocols that support ATM become more and more important. To support existing, future and unknown applications, the signaling system has to be very flexible and efficient. In this paper we define the signaling problem, present several possible signaling system structures, compare the advantages and disadvantages of these systems, and then we propose a new signaling system structure. The fundamental idea of the new signaling system is the logical separation of the signaling system structure from the underlying communication network, even though they may be built on the same physical network. The proposed signaling system structure shows very promising performance in terms of signaling latency and scalability.

1 ATM and signaling problem

ATM networks are basically connection-oriented, before the data communication can take place, the tables at switches [12] have to be set up. After a connection has been accepted by the system, some *quality of services* (QOS) should be guaranteed. To guarantee the *QOS*, resource reservation is necessary. A signaling system for an ATM network is responsible to maintain all the connection and switch status and managing the resources, such as bandwidth and table entries. In our model, one or more interconnected switches are grouped to form a *node* which is under the control of a single *control processor (CP)*. The CP manages the resources of the underlying switches. To the outside world, a node is a virtual switch with signaling information storing and processing capabilities. A multipoint connection involves a group of users. Everyone in the group receives a copy of data anyone in the group sent. Nodes have to communicate in order to establish and maintain connections. To design a signaling system for an ATM network, we have to address the following issues: how to organize the nodes; how to distribute the information among the nodes; how the nodes pass information and what are the protocols the nodes must follow; how efficient are the protocols; how to deal with failures.

In this paper, we consider these issues. In the rest of this section, we will formally define the multipoint signaling problem and we will give some measures to evaluate a signaling system. Section 2 briefly introduces several signaling system designs and compares their performance in terms of the measures given in section 1. Section 3 proposes a Virtually Hierarchical Signaling System(VHSS) which is very simple and performs well in terms of almost all the important measurements. Section 4 concludes the paper and raises some issues for further research.

1.1 Signaling problem

A communication network that supports multipoint connections can be modeled as an undirected graph [1] $G = (V, E)$. Each link (u, v) has an integral capacity $\gamma(u, v)$. We assume that each vertex of the graph has an identifier that uniquely distinguishes itself from other vertices. There are two types of vertices: a terminal vertex represents the user which is the source of connection operation requests; a switching vertex represents a node as we have defined. A *connection request* is a tuple $[c, T, r, w]$, where c is a *connection identifier* that uniquely identifies the connection in the network, $T \subseteq V$ is a set of *terminals*, r is a distinguished switching node called the root of the connection and w is the resource requirement. The root is the first node that joins the connection and stays in the connection until the whole connection is destroyed. If q is a connection request, we use $T(q)$ to denote the terminal set of q. For simplicity, in this paper, we consider bandwidth as the only resource managed so that we use the word bandwidth and resource interchangeably.

[1] More generally, we can model the network as a directed graph. To use a directed graph model, we have to distinguish sources from sinks. For simplicity, in this paper, we use the undirected model.

Let $H = (W, F)$ be a subtree of G. We say that H is a *connection graph* that implements a connection request $[c, T, r, w]$ iff for any u, $v \in T$, there exists, in H, a path from u to v, and no subgraph of H has this property.

A *connection descriptor* is a pair (q, H) where q is a connection request and H is a connection graph that implements q. For a set of connection descriptors C and any link $(u, v) \in E$, define the relative load on edge (u, v) imposed by C to be

$$\lambda_C(u, v) = \sum_{\substack{([c,T,r,w], H=(W,F)) \in C \\ \text{Such that } (u,v) \in F}} w \;/\; \gamma(u, v)$$

We say that C is feasible if for all edges (u, v), $\lambda_C(u, v) \leq 1$.

A *Signaling system* is said to be *incremental* if terminals can be added or removed at any time and no rerouting of existing connections is allowed. Since most multipoint applications need to maintain connections dynamically, we only consider algorithms for an incremental system.

A *Signaling System* dynamically maintains a feasible set C of connection descriptors for a network G under the following operations.

$create(r, w)$ adds a connection descriptor $([c, \{r\}, r, w], H = (\{r\}, \{\})]$ to C where c is a new connection identifier.

$destroy(c)$ removes the connection with identifier c from C and releases all the resources taken by the connection c.

$join(c, u, w)$ adds a new terminal u to a connection identified by c with bandwidth request w. When successful, this operation replaces the old connection descriptor $D = (q, H)$ with a new connection descriptor $D' = (q', H')$ where $T(q') = T(q) + \{u\}$, H' implements q', and H is a subgraph of H'. An unsuccessful *join* operation returns nil and no resources will be allocated.

$drop(c, u)$ removes the terminal u from the connection indicated by c and releases the resources related to u. This operation replaces the old connection descriptor $D = (q, H)$ with a new connection descriptor $D' = (q', H')$ where $T(q') = T(q) - \{u\}$, H' implements q', and H' is a subgraph of H.

One constraint of the algorithm is that it must be on-line, in the sense that the decision about routing or rejecting a *join* request has to be made without any knowledge of future requests.

The goal of a signaling system design is to design both the architecture and algorithms of the signaling system so that it maintains a feasible set of connection descriptors efficiently.

1.2 Correctness and performance

1.2.1 Correctness

The signaling system has to solve the signaling problem correctly. For scalability, reliability, and flexibility reason, we favor a distributed solution. As any distributed system, the correctness can be divided into two parts: safety and liveness. Safety states that the result of all operations maintains or implies a global feasible set of connection descriptors as defined. The liveness condition can be divided into two parts. First we define the *low level liveness* to be that all operations should finish in finite time. The high level liveness will guarantee that the signaling system set up connections successfully when some conditions are satisfied. We say that a signaling system that satisfies both the safety and the low level liveness conditions is a correct system. A correct system provides one solution to the signaling problem, but it does not guarantee to be useful. Turner [13] defines *responsiveness* as a measure to evaluate the usefulness of a signaling system. This separation of correctness and responsiveness helps to narrow down the problem. Correctness is required for any signaling system, while the responsiveness is a matter of degree.

1.2.2 Efficiency

As for most distributed systems, time complexity and message complexity are the measures used to evaluate the efficiency of a system. In the theoretical analysis, we assume message transmission can occur instantly. Processing a message at a CP takes one unit of time.

To simplify the analysis of time complexity, we define a *single request response time* to be the number of time units that elapse from sending of an operation request to receiving the response with no other message processing in the whole system.

1.2.3 Scalability

Since telecommunication systems can become very large, scalability is a big concern. We say a system is scalable if the worst case response time increases at most logarithmically with increasing network size. Since each signaling connection at a signaling point, defined in next section, imposes some computational load on the signaling point, we require that the number of signaling connections at any signaling point should not grow with the network size. Routing is one of the biggest computational jobs in the signaling system. We say a signaling system is not scalable if the routing algorithm has to keep track of the network status for all the nodes and links in the network.

2 Signaling system design

We define a *Signaling Point* (SP) to be any entity in the network that can generate and process signaling messages. A CP of a node is an SP. A terminal is an SP. Since we are concerned about the network signaling system, we do not model the terminal's activity. The only thing a terminal does is to initiate an operation request and receive the response. Later in this paper we use the word SP to indicate a non-terminal SP unless specifically stated. All the nodes controlled by one SP is called the *signaling domain* of the SP.

A signaling system consists of two parts: the signaling network and the signaling protocol. A signaling network defines the topological connection of SPs, which can be different from the topology of the network it controls. With ATM, people can setup signaling connections freely. Two SPs that are connected by a signaling connection are considered signaling neighbors even though they may not be connected by a physical link. In this way, the signaling network can be configured with great flexibility.

To decide to accept and route a *join* request or to reject it, two types of information are necessary: network status information which includes network topological information and network load information; and connection information which includes the resource requirement for each connection and the layout of the connection graphs. If these two pieces of information are available, a signaling system can make the admission and routing decision easily. Unfortunately, these two pieces of information are not always available. One of the signaling system's job is to get this information or to use incomplete information together with heuristics.

We assume that a connection identifier contains the address of the root of the connection. The root address information can be retrieved through a mapping $root(cid)$ where cid is the connection identifier. This root address information can be used in the routing algorithm to find the next hop towards the root. This routing towards root method guarantees to construct an acyclic connection graph at any time. We assume that signaling links preserve the message order. We assume that, at each SP, there exists a resource manager which records the link capacity and usage. The resource manager provides several SP-wide accessible functions that are used to manage the resources.

- $findPath(bandwidth, s, ds)$ finds a path from the signaling domain s to any element in a set of signaling domains ds with the required bandwidth. Some shortest path algorithm and cost metric can be built into this algorithm. If such a path exists, the path, a sequence of SPs that control the connected segments of the network, is returned. Otherwise nil will be the return value.

- $reserve(bandwidth, path)$ reserves the *bandwidth* on all the links of the given *path*.

- $allocate(bandwidth, path)$ modifies switch tables to setup a connection with the reserved *bandwidth* on all the links of the *path*.

- $release(bandwidth, graph)$ releases the bandwidth on all the links specified in *graph*.

In this section, we give several signaling system designs and list the advantages and disadvantages of each of the designs. At the end, we give a performance comparison of the systems discussed.

2.1 Centralized signaling system

A centralized signaling system is the simplest to design. Many existing private ATM networks take this approach. In such a system, every user has a signaling connection to a central control processor. Every connection request is forwarded to the central *Control Processor (CP)*. Since the CP manages all the resources and knows all the connection status, it can make routing decisions easily.

The advantages of a centrally controlled signaling system are: it is easy to design the protocol and make it correct; there is little communication overhead.

Disadvantages of the centralized design are obvious: it is not scalable since the central controller has to maintain all connections for the whole network; the requests are processed one by one which eliminates any possibility to process requests concurrently; if the CP fails, so does the whole system.

2.2 Distributed signaling system

Both network information and connection information can be distributed in the network [5]. In a completely distributed signaling system, every node (a virtual switch) manages the bandwidth of the links that are adjacent or internal to that node. CPs do not have global knowledge about the network status, nor do they have global knowledge about connections. They retrieve the necessary knowledge by message-passing.

The advantages of such a system are: requests can be processed concurrently; there is no computational bottle neck as in the centrally controlled case; and a single node or link failure only affects the connections routed through the node or link.

The disadvantages are: messages go hop by hop so that the *single request response time* is $\theta(d)$ where d [1] is the network diameter. Efficiency of the signaling system depends greatly on the routing algorithm. If the routing algorithm does not have enough global network status knowledge, it could be very inefficient.

2.3 Hierarchical signaling

In a traditional hierarchical network like the traditional telephone network [11], user addresses are organized hierarchically. Each SP has its management domain. The domain of a higher level SP is the union of the domains of its *signaling children*'s, the next lower level SPs. When a connection request comes with the user address out of its domain, the SP point passes the request to its *signaling parent*, the next higher level SP. When the connection has been set up, all the data traffic goes through the same hierarchy as the control signals go.

The advantages of this design are: the number of signaling connections at a node equal the number of its signaling children plus one, which counts the connection to the signaling parent; if the branching factor is a constant B, then the single response time is $O(min\{log_B n, d\})$ where n is the number of SPs in the network; it is easy to design the signaling algorithm for such a network since there exists exactly one path for each pair of nodes.

The disadvantages are: it is too restrictive for the communication network so that the resources will be wasted on the way up and down the hierarchy; and some higher level SPs may become bottlenecks for signaling processing as well as bottlenecks for resources.

2.4 Performance comparison

Figure 1 compares the performance of the signaling system designs discussed where n is the number of terminals, N is the network size of non-terminal nodes, d is the diameter of the network, and B is the minimum branch factor in a hierarchical network. From the table we can see that the centralized system is not scalable since the number of signaling connections is n, as well as that its routing algorithm has to collect network status information for the whole network. The distributed approach also suffers from the scalability problem because the single request response time is $O(d)$. The traditional hierarchical signaling system scales well, but it mixes the signaling system and the data communication system. All the data traffic has to go up through the hierarchy and then go down creating unnecessary traffic and cause possible high level bottlenecks.

System Type	Single Request Response Time	Number of Signaling Connections
Centralized	$O(1)$	n
Distributed	$O(d)$	$deg(u)$
Hierarchical	$O(log_B(N))$	$O(B)$

Figure 1: **Signaling System Performance Comparison**

3 Virtually hierarchical signaling

3.1 Virtually hierarchical system

The main disadvantage of a traditional hierarchical network is that it restricts the network topology to be hierarchical which is not desirable in general networks. On the other hand, hierarchies are natural in the communication world. General communication patterns show great locality geographically and organizationally. Accordingly, the user address usually reflects this hierarchy. The administration domain also suggests some natural hierarchies.

In this paper, we propose a *Virtual Hierarchical Signaling System*(VHSS) in which a hierarchically organized signaling network controls a communication network with arbitrary topology. Like Signaling System No 7 (SS7) [3], a signaling network is independent of the communication network so that all the advantages of common channel signaling apply here. Unlike SS7, where a complete physical network is built just for signaling, in our proposal, the signaling system is built on the same ATM network so that the cost of the signaling system can be greatly reduced.

Figure 2 shows a hierarchical signaling network with three levels of hierarchy that controls an underlying general communication network. The hierarchical signaling network is composed of SPs and signaling links. Each level i SP, except the top level SP, has a signaling connection, the dotted line in the figure, with its parent SP. An SP is a logical concept. An SP comprises software running on a computer together with signaling links to its signaling parent, signaling children, and, in some cases, signaling neighbors. Multiple SP processes can run on the same physical computer so that a powerful computer can be efficiently used.

Each SP knows its *management domain*, the set of nodes under its control. Each SP manages the re-

[1] Assume that the routing algorithm at each node can find a neighbor which is at least one hop closer to the root of the connection than the current node is

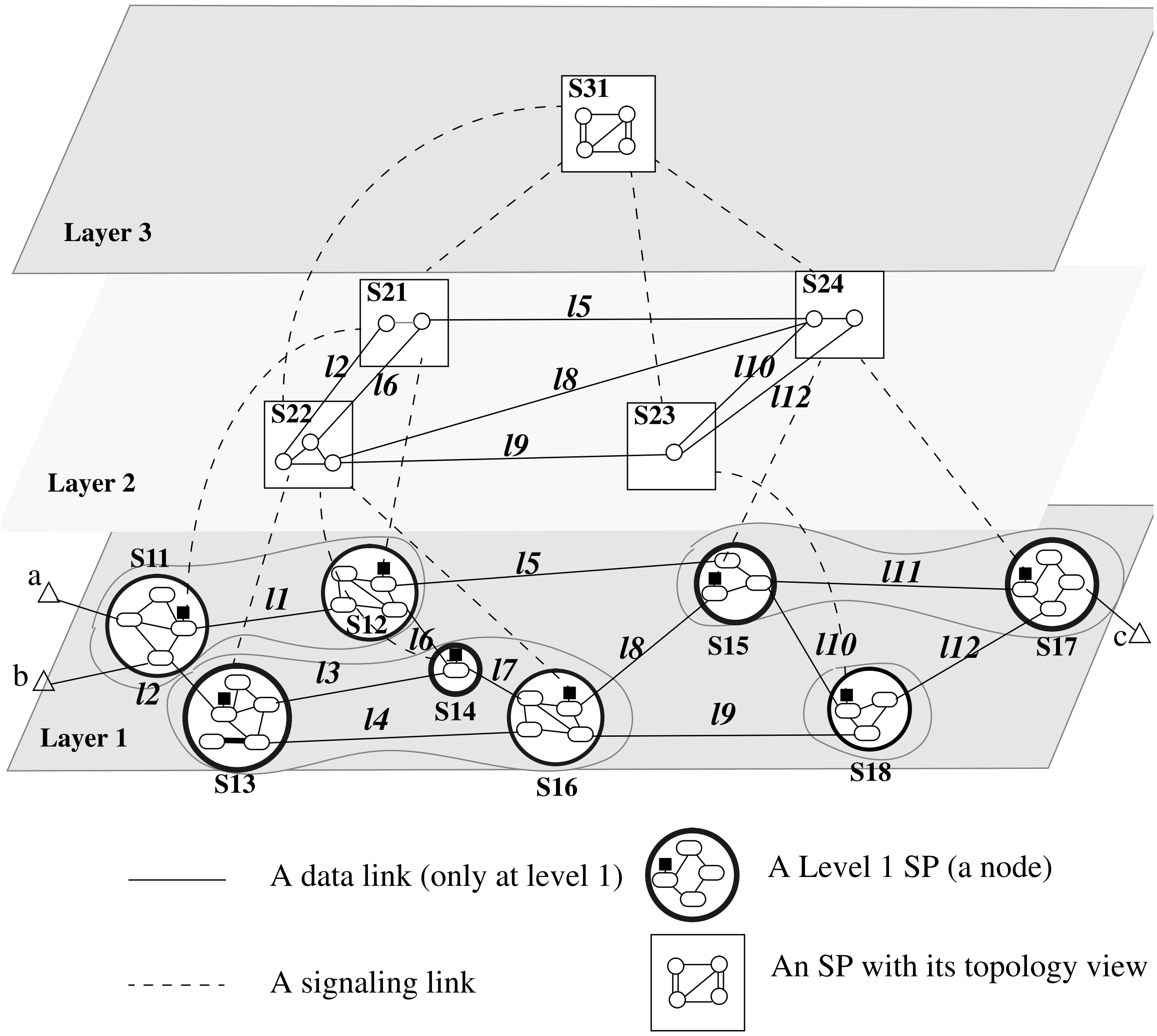

Figure 2: **A Virtually Hierarchical Signaling System**

sources for all the links that connect two of its child signaling domains. A link that connects two level $i-1$ domains is called a *level i link*. Only a level 1 SP really sets up switch tables that physically allocates bandwidth to a connection. The higher level SPs manage resources while making routing decisions. One important feature of VHSS is that links are partitioned so that each link is solely managed by one SP. When an SP makes routing decisions, it knows the exact link status of all the links managed by the SP.

The signaling algorithm is quite simple. Each SP maintains a connection data structure, as shown in figure 3, for every connection that has registered at that SP. A connection object contains a graph data structure that stores the connection graph. An access function $updateGraph(cid, updateCommand, path)$ is called to update the connection graph. If the value of *updateCommand* is ADD, a path specified by the *path* parameter is added into the connection graph. If the value of *updateCommand* is REMOVE, the path is removed from the connection graph. A path is a data structure that contains a list of child level of SPs. The vertex set of the graph can be accessed through con.graph(V).

When one SP can not resolve a *join* request, (the connection data structure does not exist at the SP and the root of the connection is not in the SP's management domain), the request is passed to its signaling

```
connection {
   connectionId:         cid;
   bandwidth:            bw;
   connectionGraph:      graph;
   void                  updateGraph(ConnectionId,
                         {ADD | REMOVE}, path);
   Path                  findPathToBranch(sp);
   /* returns a path from the sp to the first branching point */
}
```

Figure 3: **Connection Data Structure**

parent. This upward message passing continues until either a connection data object for the connection is found or the root of the connection is in the SP's management domain. At that time a shortest path from the requesting SP to the connection is calculated for the request. This shortest path only works at the current level. A *setup* request will be sent to all the signaling children along the path to request that they set up the connection. This *setup* message propagates downwards until it reaches the level 1 SPs. If all the level 1 SPs successfully allocate the bandwidth for the request, an ACK will propagate back to the higher level SP who has sent the *setup* message. Otherwise a NACK will be passed to that SP. The result will pass back to the user down the hierarchy along the reverse path of the *join* request messages.

3.2 VHSS algorithms

Each operation is assigned a unique identifier when it starts. At each SP that participates in the operation there exists an object called an *operation handler*. An operation handler is created when a message related to that operation is delivered at the SP for the first time. It is the operation handler's responsibility to process all the messages related to that operation. An operation handler is destroyed when the operation completes. For any connection, at most one operation handler is assigned as the *current handler*. Only the current handler can process an incoming message. The operation corresponding to the current handler is said to be the *current operation*. A handler has a priority attribute which can be either HIGH or LOW. A handler created as the consequence of receiving a message from its signaling parent, such as *setup*, is assigned a HIGH priority. Otherwise it has LOW priority. A High priority handler will never change its priority. A LOW priority handler may change its priority to HIGH when a message from its signaling parent is processed or it has sent messages to its signaling children. A HIGH priority handler prevents the creation of other handlers for the same connection. Generally speaking, top-down operations have higher priority than the bottom-up operations.

We assume that there exists a *message delivery system*. The message delivery system guarantees the reliable delivery of messages to their proper operation handlers. When a message is received, the message delivery system will deliver the message to the current handler if this message is part of the current operation. If the handler for the message does not exist, the message delivery system may either create a new operation handler and deliver the message to it or block the message depending on the priority of the current operation handler and the incoming direction of the message. The decision is made based on the algorithm shown in Figure 6. When the current handler finishes, the message delivery system will scan through the blocked messages as if they were just received. Some of the message delivery system's functions defined here could be modeled in our signaling algorithm. We put them in the message delivery system only to simplify the presentation of the signaling algorithms. The priority assignment and FIFO property of the message delivery system are crucial to guarantee the correct behavior of the protocol. Figure 4 and Figure 5 show the signaling algorithms for each of the operation handlers.

We define some functions and mappings used in the algorithms. A function $domain()$ at an SP gives a set of node addresses that are under the control of the current SP. An SP-wide accessible routine $send(dest, message)$ sends the $message$ to the destination $dest$. The function $getConnection(cid)$ searches the system to find the connection with identifier cid. A connection object is returned if the search is successful. Otherwise *nil* will be the return value. At each SP, a mapping $SP(u)$ or $SP(lk)$ maps a node u or a link lk to a child SP where node u resides or the link lk incident to. A level i path is a list of interleaved level $i-1$ SPs and level i links. A macro $L(p, r)$ returns the link which is on the left of SP r in the path p. It returns nil if r is the leftmost in the path. Similarly $R(p, r)$ gives the right link of r. A macro $numberOfSP(p)$ gives the number of SPs in the path p. A macro *lastLink(p)* gives the last link in path p. A function *rollback()* eliminates all the effect of the current operation.

A *join* message is processed in four phases. When a *join* operation is initiated by a user, a *joinReq* message is generated and this *joinReq* message is passed up the signaling hierarchy until either a connection data object is found or the root of the connection is

Join Handler

```
/* Handler-wide accessible variables
and initial values */
  Link          srcLK := nil;
  Path          p := nil;
  integer       receivedResp := 0;
  OpResult      opResult := ACK;
  SP            opParent := nil;
  integer       i := Current SP's level;

• joinReq(cid,bw,lk)
/* comes from a child SP */
  srcLK := lk;
  con := getConnection(cid);
  if ((con ≠ nil) OR (root(cid) ∈ domain()))
    if (con = nil)
      con := new connection(cid, bw);
    p := findPath(bw,SP(lk),con.graph(V));
    if (p = nil)
      send(SP(lk),joinResp(cid,NACK));
      rollback();
    else
      reserve(bw,p);
      if (i = 1)
      /* level 1 SP really allocate bandwidth */
        allocate(bw,p);
        send(SP(lk),joinResp(cid,ACK));
        delete this;
      else
        for (r ∈ p)
          send(r,setup(cid,bw,L(p,r),R(p,r)));
        end for
      endif
    endif
  else
    send(parentSP, joinReq(cid,bw,lk));
  endif

• joinResp(cid,result)
  send(SP(srcLK), joinResp(cid,result));

• joinCommit(cid,command)
  if (i ≠ 1)
    for (r ∈ p)
      send(r,joinCommit(cid,command));
    end for;
  endif;
  if (command = ACK)
    if (i = 1)
      allocate(bw,p);
    endif
    con.updateGraph(ADD, p);
  else
    rollback();
  endif
  delete myself;

• setup(cid,bw,lk1,lk2)
/* come from the signaling parent */
  opParent := parentSP;
  if (lk1 = nil)
    lk1 = srcLK;
  endif
  if (lk2 ≠ nil)
    con := new connection(cid, bw);
    p := findPath(bw,SP(lk1),{SP(lk2)});
  else
    con := getConnection(cid);
    if (con = nil)
      con := new connection(cid, bw);
    endif
    p := findPath(bw,SP(lk1),con.graph(V));
  endif
  if (p = nil)
    send(parentSP,setupResp(cid,NACK));
  else
    reserve(bw,p);
    if (i = 1)
      send(parentSP,setupResp(cid,ACK));
    else
      for (r ∈ p)
        send(r,setup(cid,bw,L(p,r),R(p,r)));
      end for
    endif
  endif

• setupResp(cid,result)
  receivedResp := receivedResp + 1;
  if (result = NACK)
    opResult := NACK;
  endif
  if (receivedResp = numberOfSP(p))
    if (opParent ≠ nil)
      send(parentSP, setupResp(cid,opResult));
    else /* start committing */
      send(SP(srcLK), joinResp(cid, result));
      for (r ∈ p)
        send(r,joinCommit(cid,opResult));
      end for
      if (opResult = ACK)
        con.updateGraph(ADD,p);
      else
        rollback();
      endif
      delete myself;
    endif
  endif
```

Figure 4: **Signaling Algorithms for VHSS (*join* Operation Handler)**

Create Handler

```
• create(cid,bw,sp)
  con := new connection(cid,bw);
  send(sp,createResp(cid,ACK));
  delete myself;
```

Drop Handler

```
  SignalingPoint srcSP := nil;
  Path p := nil;
  Link l;
  integer i := Current SP's level;

• dropReq(cid,sp,l)
/* a dropReq may come from a signaling child
  or a level 1 signaling neighbor */
  srcSP := sp;
  con := getConnection(cid);
  if (i = 1)
    p := con.findPathToBranch(sp);
    l := lastLink(p);
    if (l is a higher level link)
      release(bw,p-l);
      con.updateGraph(con.bw,REMOVE,p-l);
      send(parent,dropReq(cid,currentSP,l));
    else
      release(bw,p);
      con.updateGraph(con.bw,REMOVE,p);
      delete myself;
    endif
  else
    if (l is a higher level link)
      send(parent,dropReq(cid,currentSP,l));
    else
      if (l is the only level i link in con for sp)
        release(con.bw, {l});
        con.updateGraph(cid,REMOVE,{l});
        send(sp,dropCommit(cid,ACK));
      else
        send(sp,dropCommit(cid,NACK));
      endif
      delete myself;
    endif
  endif

• dropCommit(cid,command)
  con := getConnection(cid);
  if (i = 1)
    if (command = ACK)
      release(bw,l);
      con.updateGraph(cid,REMOVE,{l});
      send(SP(l),dropReq(cid,currentSP,l));
    endif
  else
    if (command = ACK)
      delete con;
    endif
    send(srcSP,dropCommit(cid,command));
  endif
  delete myself;
```

Destroy Handler

```
• destroyReq(cid)
  con := getConnection(cid);
  if (exist higher level links in con)
    send(parent,destroyReq(cid));
  else
    release(con.bw, con.graph);
    for (r ∈ con.graph(V))
      send(r,destroyCommit(cid));
    end for
    delete myself;
  endif
  delete con;

• destroyCommit(cid)
  con := getConnection(cid);
  release(con.bw,con.graph);
  if (i ≠ 1)
    for (r ∈ con.graph(V))
      send(r,destroyCommit(cid));
    end for
  endif
  delete con;
  delete myself;
```

Figure 5: **Algorithms for VHSS (*create*, *drop*, *destroy* Handlers)**

if (the message is for the current handler)
deliver the message to the current handler
else
if (the current handler has LOW priority **and** *the message comes from the parent)*
create a new operation handler for the message, then deliver the message to that handler and assign it as the current handler
else
block the message
endif
endif

Figure 6: **Algorithm for the Message Delivery System**

in the SP's domain. A multicast phase begins when the higher level SP chooses a path to route the connection request and multicast a *setup* message to all the child SPs along the path. A converge-cast phase followes in which every SP reports the operation status after it gets a response from all its children that are involved in the operation. In the fourth phase, a *joinCommit* message propagates to all the SPs participated in the operation to inform the SPs to commit or abort the operation. At the same time a *joinResp* message is passed to the user to report the result of the operation.

A *create* operation handler is the simplest. A *create* handler only appears at the lowest level. When a *createReq* is received, the level 1 SP creates a connection data object and sends an ACK back. A *createReq* message carries the connection identifier, the bandwidth and the user sp identifier.

A *drop* operation is a little bit tricky. Since the higher level SP does not know the connection graph internal to a lower level signaling domain, the *drop* operation has to proceed hop by hop at the lowest level. Whenever a level i link is involved, a *dropReq* message is passed to the level i SP. The SP makes a *commit/abort* decision based on the connection graph at that level. Two message types are involved in a *drop* operation. A *dropReq* message carries a connection identifier, the sender SP, and the link identifier indicating the link where the *dropReq* came from. A *dropReq* message serves to report to the signaling parent to start a *drop* operation as well as to be passed at the lowest level to request release of physical resources.

A *destroy* operation is started by the owner of the connection by sending a *destroyReq*. This *destroyReq* is passed up the hierarchy until the highest level SP that is involved in the connection is reached. Then a *destroyCommit* message is multicast to all the SPs that are involved in the connection. The connection data object is deleted when the *destroyCommit* is received. At the bottom level, all the resources allocated for the connection are released.

3.3 Correctness and performance

In this section, We give all the correctness lemmas and theorems without proving. Details of proofs can be found in [16].

In a distributed signaling system, the network status and the connection graph information is distributed in the network. To prove the correctness of the distributed algorithms for a signaling system, we need to extend our problem definition in a distributed way.

A *local connection descriptor* at an SP s is defined as a tuple $[s, c_s, r_s, w_s, H_s = (W_s, F_s)]$ where s is the SP's identifier, c_s is a connection identifier, r_s is the root of the connection, w_s is the bandwidth requirement and: if s is a level 1 SP, H_s is a subgraph of G whose vertices are restricted to s and/or its neighbors in G; if s is a higher level SP, H_s is a graph whose vertices are its child SPs and its signaling neighbor SPs, and whose edges are the links of G that connect two signaling domains of its vertices.

Let GD denote a set of (global) connection descriptors. Let GD(c) be a connection descriptor with connection identifier c. Let LD be a set of local connection descriptors. Let LD(c) be the subset of LD with connection identifier c, and $\mathrm{LD}_s(c) = [s, c, r_s, w_s, H_s = (W_s, F_s)]$ denote the elements of LD(c) whose first element is s.

We say that LD(c) implements $\mathrm{GD}(c) = ([c, T, r, w], H = (W, F))$ if for all $\mathrm{LD}_s(c) \in \mathrm{LD}(c)$, $r = r_s$, $w = w_s$, and

(1) if s is a level 1 SP
$W = \bigcup_{LD_s(c) \in LD(c)} W_s$, and $F = \bigcup_{LD_s(c) \in LD(c)} F_s$
and for all $u, v \in V$, $(u, v) \in F_u \Leftrightarrow (u, v) \in F_v$
(2) if s is not a level 1 SP
for all $(u, v) \in F_s \Leftrightarrow (u, v) \in F_{SP(u)}, (u, v) \in F_{SP(v)}$
and $(u, v) \in F_s \Rightarrow SP(u), SP(v) \in W_s$

We say that LD implements GD if LD implements every descriptor in GD and no others. This defines the safety conditions for the VHSS in a distributed way. The condition (1) is general enough to give the correctness conditions for any distributed signaling system. The condition (2) only applies to VHSS and is included to ensure that VHSS fulfills condition (1).

The local connection manager algorithm, running at each SP of a signaling network, maintains a set of local connection descriptors. Together, they implement a set of global connection descriptors. Notice that our connection data structure is consistent with a local connection descriptor. We define a *good state* to be a state where both condition (1) and (2) are satisfied.

We define an event $e(s, c, op, typ, [status])$ to be one call of any handler's event processing routine, where s is an SP's identifier indicating the place where the event occurs, c is the connection identifier, op is the operation identifier, typ is the message type, and the optional $status \in \{ ack, \ nack \}$ gives the result of the event. We say $\mathcal{E}$ is an execution of the system iff $\mathcal{E}$ is a sequence of events that satisfies the natural order of events. Let $\mathcal{E}(c)$ be the subsequence of $\mathcal{E}$ with all events related to connection c. A *stable state* is a state in which no event will occur if there are no new user requests. We define a *complete execution* to be an execution with all the operations are completed. We say that a system is correct if starting from any good stable state, it is guaranteed to reach a good stable state after any complete execution.

Lemma 1 *If the system starts in any good stable state and starts a complete execution $\mathcal{E}$ in which all events relate to a specific operation of connection c, then the system will stop at a good stable state where the new LD(c) implements the new GD(c).*

A *drop* operation propagates and gets committed hop by hop. So we can think an external *drop* operation as composed of multiple internal *drop* operations, each of which starts when a level 1 SP receives a *dropReq* and ends when a *dropCommit* has been processed at the same SP. When we refer to an operation in the following lemmas, we refer to this kind of internal operation.

Lemma 2 *If the system starts in any good stable state and starts a complete execution $\mathcal{E}$ in which all operations relate to a single connection c, then there exists another execution $\mathcal{E}$' which is a permutation of $\mathcal{E}$ such that in $\mathcal{E}$' all the events relating to a single operation are consecutive and both $\mathcal{E}$ and $\mathcal{E}$' leave the system in the same final state.*

Corollary 3.1 *If the system starts in any good stable state and starts a complete execution $\mathcal{E}$ in which all operations relate to a single connection c, it will end in a good stable state.*

The operations for different connections only interact through the resource manager. We call the points when an SP executes a *reserve*, an *allocate*, or a *release* the *resource access point*. If we can rearrange the execution sequence in such a way that the operations for each connection are serialized and the order of the resource access points are unchanged at each SP, then the new execution will lead to the same state as the original execution. This argument leads to the following lemma.

Lemma 3 *If the system starts in any good stable state and starts a complete execution $\mathcal{E}$, then there exists another execution $\mathcal{E}$' which is a permutation of $\mathcal{E}$ such that $\forall c \in C$, $\mathcal{E}$'(c) is serialized and for all SPs, the order of the resource access points are the same as the order in $\mathcal{E}$.*

Lemma 4 *The capacity constraint is always strictly observed.*

Theorem 3.1 (Safety) *The VHSS is a safe system.*

Now we will give the low level liveness of our system.

Theorem 3.2 (Low Level Liveness) *If at some point in an execution, no new operation requests are made, after a sufficiently long time period all operations finish.*

Lemma 5 *For every node u, the CP of u has deg(u)+1+(number of local users) signaling connections. For every other SP, there are O(B) signaling connections where B is the maximum branching factor of the SP. The single request process times for different requests are given in Figure 7 where n is the network size, b is the minimal branching factor, and d is the diameter of the network.*

3.4 Live-lock problem

In this section we give one responsiveness lemma of VHSS named *live-lock* free.

Live lock is an interesting phenomenon in a distributed system. In our context, the live-lock problem can be stated as follows: There is a set of *join* operation requests, each of which can be successfully routed by the $findPath$ functions at different levels if the operations come individually. Since the events of the operations are interleaved in some order, none of them can be satisfied. In the following lemma, we prove that VHSS is live-lock free.

Lemma 6 *Live-lock will never occur in VHSS.*

System Type	Single Request Processing Time	Single Request Message Complexity	Busiest CP Message Complexity
create	$O(1)$	$O(1)$	$O(1)$
join	$O(B * log_b(n))$	$O(d)$	$O(B)$
drop	$O(d)$	$O(d)$	$O(B)$
destroy	$O(log_b(n))$	$O(d)$	$O(B)$

Figure 7: **Single Request Processing Complexity**

4 Conclusion and future work

VHSS takes advantage of the flexibility that an ATM network provides. The same ATM network works both for data communications and signaling processing in an integrated way. The signaling protocol is simple to implement. It scales well which is extremely important for a public telecommunication network.

In this paper, we assume all the system components are reliable which is not the case in a real system. By combining VHSS with the layered approach [16], especially at the higher level, it can be made reliable and cost efficient.

In this paper we omit one important measure for a signaling system: how efficient is the signaling system in terms of managing the resources. Some measures such as the throughput, profit, or competitive ratio can be used to evaluate the efficiency of using resources [1, 6]. To evaluate these measures, we may have to expand our model to include link cost, per call profit or similar attributes for networks and connections. Besides theoretical analysis, we are planning to do some simulations to compare the performance of a VHSS with other signaling systems.

5 Acknowledgements

In completing this work, I have had the benefit of many fruitful (and often spirited) discussions with professors and colleagues at Washington University. Dr. Jonathan Turner spent a lot of his time with me to discuss the ideas and detailed system design. Andy Fingerhut, John DeHart, and Ken Goldman all provided very useful input.

References

[1] Baruch Awerbuch, Yossi Azar, and Serge Plotkin. Throughput-competitive on-line routing. In *Proceedings of 34th FOCS93*, 1993.

[2] Baruch Awerbuch and Yossi Azar et al. Competitive routing of virtual circuits with unknown duration. In *Proceedings of SODA93*, 1994.

[3] CCITT. Specification of signaling system no. 7. *CCITT Blue Book*, 6, 1988.

[4] Israel Cidon, Inder Gopal, and Adrian Segall. Fast connection establishment in high speed networks. *IEEE Trans. on Networking*, pages 469–481, 8 1993.

[5] John DeHart and Dakang Wu. Connection management network protocol (cmnp) specification. Technical Report ARL Working Node, Washington University Computer Science Department, 1993.

[6] Anja Feldman et al. Competitive analysis of call admission algorithms that allow delay. Technical Report CMU-CS-95-102, Carnegie Mellon University School of Computer Science, 1995.

[7] J. J. Garcia-Lune-Aceves. Loop-free routing using diffusing computations. *IEEE Trans. on Networking*, pages 130–141, 2 1993.

[8] M. Imase and B. M. Waxman. Dynamic steiner tree problem. *SIAM J. on Disc. Math.*, pages 369–384, 3 1991.

[9] R. Rey. *Engineering and Operations in the Bell System.* Marray Hill, 1983.

[10] Adrian Segall and Jeffrey M. Jaffe. Route setup with local identifiers. *IEEE Trans. Commun.*, COM-34:45–53, 1 1986.

[11] W. Stallings. *ISDN.* MaCMillan, 1989.

[12] J. S. Turner. An optimal non-blocking multicast virtual circuit switch. *IEEE INFOCOM*, pages 298–305, 1994.

[13] Jonathan S. Turner. Issues in distributed control for atm networks. Technical Report WUCS-95-12, Washington University Computer Science Department, 1995.

[14] B. M. Waxman. Routing of multipoint connections. *IEEE J. Select. Areas Comm.*, pages 1617–1622, 6 1988.

[15] Dakang Wu. An efficient signaling structure for atm networks. Technical Report WUCS-95-09, Washington University Computer Science Department, 1995.

[16] Dakang Wu. A survey of network signaling. Technical Report WUCS-95-08, Washington University Computer Science Department, 1995.