Internet Engineering Task Force
INTERNET-DRAFT

K. K. Ramakrishnan,
Gísli Hjálmtýsson,
Kobus Van der Merwe
AT&T Labs. Research
Flavio Bonomi,
Sateesh Kumar,
Michael Wong
CSI ZeitNet/Cabletron

August, 98

UNITE: An Architecture for Lightweight Signaling

<draft-ramakrishnan-mpls-unite-01.ps,.pdf >

Status of this memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

To view the entire list of current Internet-Drafts, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), ftp.nordu.net (Northern Europe), ftp.nis.garr.it (Southern Europe), munnari.oz.au (Pacific Rim), ftp.ietf.org US East Coast), or ftp.isi.edu (US West Coast).

Abstract

Communication networks need to support a wide range of applications with diverse service quality requirements. The current widespread use of best-effort communication also suggests that the overhead for establishing communication both in processing and latency needs to be kept at a minimum. With ATM signaling, every flow, including a best-effort flow, suffers the overhead of end-to-end connection establishment. ATM signaling complexity is further exacerbated by having variable length messages with a large number of information elements using very flexible encoding, sent on a single control channel. The inclusion of QoS processing and connectivity in the initial setup of a connection requires sequential hop-by-hop processing. Variable length messages involves both a single point of resequencing as well as relatively slow, software based processing. In recognition of these shortcomings, the MPLS working group has opted to use topology driven label distribution as its default label distribution mechanism, while at the same time acknowledging the possible need for on-demand label distribution. We see these different approaches as points on a range of solutions and we do not wish to open a debate concerning the relative merits of each approach. However, we believe that if there is a need for on-demand label distribution, then there is a need to do this very efficiently. In this light we have decided to bring to the MPLS working group our architecture for lightweight signaling. While in its current form it is applicable to an ATM environment, we believe that it represents a step forward in the evolution of signaling for high speed networks. It holds the promise of processing signaling in hardware, thereby enabling substantial speed up of connection setup, so as to meet the needs of contemporary applications.

Our proposed lightweight architecture for ATM signaling is called UNITE. The fundamental philosophy of UNITE is the separation of connectivity from QoS control. This has the potential to eliminate the round-trip connection setup delay, before initiating data transmission. Using a single cell with proper encoding, we avoid the overhead of reassembly and segmentation on the signaling channel. With fixed formats, we believe that a hardware implementation is feasible. Performing QoS negotiation in-band allows switches in the path to process QoS-requests in parallel, facilitates connection specific control policies, supports both sender and receiver initiated QoS, and allows for uniform treatment of unicast and multicast connections.

Note on Applicability

This Internet Draft is based on an ATM Forum contribution and as such is written within an ATM context. However, we believe that the UNITE approach to signaling might also be of value within the context of MPLS and have therefore decided to present it to the MPLS working group to solicit feedback. We hope to extend and modify this Internet Draft to be applicable for on-demand label distribution in MPLS based on the feedback received.

1 Introduction

The goal of lightweight signaling is to reduce the penalty of connection setup, while supporting service guarantees. A lightweight signaling protocol should ideally support and enhance *both* connectionless and connection-oriented services. Because of a desire to foresee the signaling needs of any and all applications that are likely to use the network, current ATM signaling is complex and slow, multiple messages are required to set up a connection, and considerable processing is required to parse the complex signaling messages.

In this internet draft, we describe UNITE, a lightweight signaling protocol for ATM networks. We are motivated by the need to more efficiently support data applications that typify current Internet traffic while providing facilities to support applications that require stringent quality-of-service such as telephony. Furthermore, this work is aimed at reducing the complexity of ATM signaling, improving the performance of ATM call processing, and improving ATM as a general purpose transport infrastructure.

The principal idea behind UNITE is a *complete separation of connectivity from quality-of-service*, or more generally, service attributes. The connectivity setup message is reduced to a single ATM cell, with fixed field sizes and positions, avoiding the overhead of reassembly and segmentation on the signaling channel, allowing it to be fully processed in hardware. Exploiting per-VC queueing, data can be forwarded immediately after a one-hop exchange, rather than suffering a full-round-trip latency. However, we recognize that not all switches are likely to have per-VC queues, and switches may initially want to support connection establishment in software. For this reason UNITE accommodates both software processing and FIFO switches using a “marker/marker-acknowledgment” protocol between switches.

UNITE reduces connection setup cost sufficiently, so that establishing connectivity becomes *comparable* to forwarding and populating a cache in a router. A UNITE switch can therefore reasonably be expected to setup new connections at a rate competitive with routing in a connectionless networks. Conversely, IP-type best effort data flows suffer sufficiently small delay penalty for establishing a connection over the ATM infrastructure that it becomes viable to set up a connection even for the shortest of flows. Thus, UNITE is ideally matched to carry Internet traffic (IP) over ATM networks.

UNITE uses in-band messages for QoS establishment. It builds on the extensive work done for QoS in ATM networks, including the specification of classes of service, admission control and related issues such as conformance and policing. Because the QoS messages are sent on the established VC, we can exploit parallelism to improve the throughput and latency for QoS establishment.

In part due to its simplicity, UNITE supports both source and destination initiated QoS, supports multipoint-to-multipoint connections and recognizes the possible need for variable QoS to different participants [i, ii] (variegated multicast trees).

UNITE has been implemented in a software prototype. Early performance measurements confirm our expectations for a higher signaling throughput and lower call setup latency.

In the next section we describe UNITE’s connection setup for best effort connections. Subsequently, in Section 3, we describe UNITE’s support for multicast. In Section 4, we provide details of UNITE’s QoS Management, and then deal with interoperability issues, both with UNI as well as with existing switches. Section 7 summarizes the benefits of UNITE and then we conclude. Finally, in Section 9, we briefly consider the applicability of UNITE in an MPLS environment.

2 UNITE Connection Setup

UNITE uses a separate, initial mechanism for setup of connectivity to enable a fast connection setup. This is shown in Figure 1.

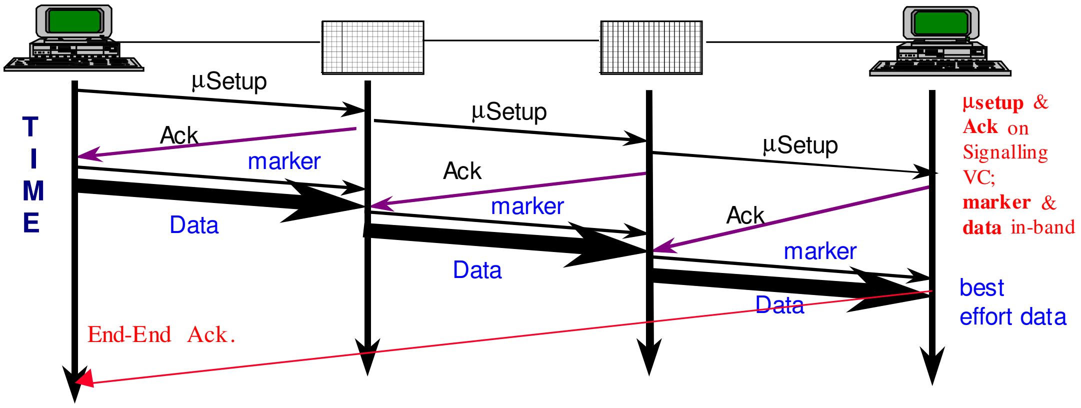

Figure 1: The UNITE Connection Setup

The “calling station” issues a **μ-setup** (micro-setup), which is a single cell, on the signaling channel that includes all the information necessary to establish a best-effort connection with a remote “called station”. The switch that receives the μ-setup determines the route (based on the destination address and a broad QoS class identification) to forward the μ-setup on the correct output port. After allocating VC resources on the upstream link for the connection and forwarding the μ-setup, the switch returns a single cell **μ-acknowledgment** to the upstream node on the signaling channel. If appropriate, the switch may allocate per-VC buffers for the switch at that time also. On receiving the μ-ack, the upstream node transmits a single cell **marker** on the established VC (in-band). This marker serves as the 3rd step of a three-way handshake. Subsequent to transmitting the marker, the upstream node may transmit data on the VC, on a best-effort basis. The above sequence of steps is repeated at each hop. Virtual Circuits established are bi-directional, with VC-ids allocated in the conventional manner by switches. While we believe we can accommodate multiple address formats, we are currently using existing NSAP addresses and address allocation methodologies. We assume the existence of link-layer management, such as ILMI. The μ-setup is routed to the destination on a hop-by-hop basis, using routing tables that are setup based on existing PNNI information dissemination and route-computations. We also use existing cell-formats and currently defined AAL5 framing.

The commitment provided by the connection is that data is transmitted on a best-effort basis. Since the QoS class information is provided in the μ-setup, the path selected even for the best-effort connection may be on a more informed basis than pure best-effort with no a priori knowledge. Data may begin flowing from an upstream node to the downstream node immediately upon completion of the μ-setup on that hop. The **latency** suffered by a best-effort flow to use the connection-oriented nature of ATM is thus **only a single hop round-trip propagation delay, plus the time needed to setup state on that hop**. Data is buffered on a switch (with per-VC buffers) when it arrives on a VCid for which a forwarding table entry has yet to be setup. In a subsequent section in this draft, we describe methods to accommodate FIFO switches, and also when the processing of the signaling messages is performed in software. This is enabled by the use of an optional **marker-**

acknowledge, that allows for a downstream switch (or node) to require the upstream switch (or node) to delay transmitting of data until it is ready to receive data.

To ensure that no persistent loops form, UNITE uses a combination of a unique Flow-ID for the connection and an **end-end acknowledgement**. When the destination receives the μ–setup, it sends an in-band (on the established VC) end-end ack to the source. This indicates to the source that a loop-free path has been established. Only upon receiving the end-end ack does the source issue a RELEASE at any time in the future when it needs it. Issuing a RELEASE prior to receiving the end-end ack may “erase” the Flow-ID maintained at a switch. This is undesirable because it will be unable to recognize the μ–setup that may come back as a result of a loop. The combination of the unique Flow-ID and holding back the RELEASE until the end-end ack is received enables us to avoid loops.

2.1 ESTABLISHING CONNECTIVITY, PHASE 1: THE MICRO-SETUP

The μ-setup and the associated μ-acknowledgment are sent on a well-known signaling VC. The processing of the μ-setup at the switch includes the following functions, in order:

1. A route lookup for the μ-setup, identifying the port on which to forward the micro-setup.
2. Allocation of a VC from the VC address space on the upstream link. We assume that all connections are created bi-directional (to minimize the overhead of both ends establishing connections).
3. Allocation of a reasonable amount of buffering at the switch for that VC, if appropriate.
4. Initiating an ACK-timer for the μ-setup. This timer is for error recovery when the μ-setup is lost or when the downstream switch does not successfully progress the μ-setup.
5. Forwarding the μ-setup downstream on the link that the route-lookup function determined as the “best path” towards the destination end-system.
6. Mark the incoming VC state as *DISCARD*, so that the switch discards all incoming cells on this VC. This enables us to clear previously buffered cells for the upstream link on the newly assigned VC, if there are any. The VC state transitions to *FORWARD* state subsequently, when a MARKER acknowledging the ACK is received.
7. Finally, a VC id is returned to the upstream switch in the μ-ACK. The upstream node may begin transmitting data on receipt of the μ-ACK. The forwarding of the data to the downstream next hop has to await the completion of the processing at the next hop switch and the return of a corresponding VC id for the flow.

We have chosen to provide reliable delivery within the UNITE signaling framework itself, rather than layering it on top of another reliable transport mechanism. Current ATM UNI signaling uses a reliable transport protocol, SSCOP for transporting signaling messages thus re-incorporating some of the overhead for processing a signaling message, and makes it difficult to implement in hardware. The 3-way handshake obviates the need for a reliable transport for carrying signaling messages.

A simple, efficient encoding of the setup is vital: we use a single cell for the μ-setup, with only essential components in it, thus allowing for hardware implementation. In addition, it allows for distributed call setup to be implemented in a switch (especially important when there are a large number of ports). The μ-setup uses a unique end-to-end *Flow-id.* All control exchanges use this Flow-id. Included in the μ-setup is whether the call is unicast or multicast capable. Multicast and unicast connections have nearly identical mechanisms for both connection setup and QoS setup.

UNITE adopts hop-by-hop routing of the micro-setup, in contrast to the traditional source-routing used in ATM's PNNI routing protocols. However, source-routing has been used to avoid loops in connection-oriented networks. Since UNITE's Flow-id is a unique end-to-end call-reference identifier, this may be used to detect loops. When a duplicate μ-setup is received with the same Flow-id, without it being a retransmission (or on a different port than the port the earlier copy was received on) it indicates a routing loop. UNITE suppresses multiple μ-setups (a mechanism we also use for multicast connections for normal operation). A controller might also send a release in the backward direction for the Flow-id (or allow timers to subsequently close the connection). This mechanism along with the rules for issuing a RELEASE after an "end-end acknowledge" is received by the source ensures that a successful connection does not contain a loop. Routing loops are mostly transient inconsistencies in routing tables, which we expect to be corrected by subsequent updates as part of the normal operation of the routing protocols.

The μ-setup being a single cell allows the switch to avoid re-assembly and segmentation. In addition, all of the requirements to keep the cells in sequence may be ignored: a μ-setup cell may be served in any order, relative to the others. Thus, we could choose to process the μ-setup for some classes in hardware, and others in software, if so desired. Furthermore, it allows for a truly distributed implementation of the μ-setup because there is no need for a single point of re-sequencing the cell streams for signaling messages arriving on different ports. A fixed format μ-setup cell also assists hardware implementations.

The fields of the μ-setup cell are as follows, with reference to Figure 2:

1. Flow-id (8 bytes) - A unique (end-to-end) *Flow-id* identifying the micro-setup from source. This comprises two sub-fields:
    - A unique source identifier. For example, this could be the host "Ethernet" address, that is unique through the use of an address ROM (6 bytes).
    - A source unique sequence number (2 bytes).
2. Type (1 byte) - type of signaling cell. Includes a Retransmit bit.
3. QoS Class (1 byte) - for minimal QoS sensitive routing. (Potentially broken up into a nibble for "class definition" and a nibble for specification of the size of the dominant parameter for that class.
4. Reserved (1 byte) - for future use. Anticipating the potential use of a Virtual Private Network Identifier, we could include 3 bytes for a VPN ID by removing the User-User Information byte from the AAL5 trailer. The use of such a VPN ID is for further discussion.
5. Protocol ID (5 bytes) - allows the caller to specify the network layer entity addressed at the called station and eliminates a need for a second exchange to establish this connectivity. SNAP encoding is assumed by default. The 5 bytes includes the OUI and PID fields.
6. Destination Address (20 bytes) - destination NSAP address.
7. A VPI/VCI that is assigned by the upstream node for the connection when it is appropriate. This is determined by which end of a link is supposed to allocate the VPI/VCI value for a new connection, just like the current convention.
8. AAL5 Trailer (8 bytes) - the standard ATM AAL5 trailer including the CRC and length.

In addition, of course, is the 5 byte ATM cell header. The VC id on which the μ-setup is transmitted is a common, well-known signaling VC.

A switch maintains a timer associated with the μ-setup that has been transmitted to the downstream hop. This timer is cleared upon receiving the ACK from the downstream switch. A switch that has timed out after transmission of the μ-setup retransmits the μ-setup request. The re-transmitted μ-setup is identical to the previous except for a retransmit bit in the type field. As a result it can be retransmitted by hardware.

2.2 Establishing connectivity, Phase 2: The ACK for the Micro-setup

The μ-Acknowledgment of the connection setup upon successful processing of the μ-setup is returned upstream to the previous switch or host. The information provided has to be adequate for appropriate processing at the upstream switch or the original host requesting the connection. The downstream switch maintains a timer associated with the μ-ACK transmitted upstream, for re-transmitting μ-ACKs. (This timer is cleared when the MARKER is received in the third phase of the μ-setup and therefore also protects against loss of the MARKER.) The μ-ACK has the following fields:

1. Flow-id (8 bytes): the Flow-id received in the micro-setup, to enable to upstream node to match this ACK to the request.
2. VPI/VCI returned for the request (3 bytes)
3. The NSAP address that was the same as the one received in the μ-setup.
4. There is also a bit to indicate to the upstream whether a Marker-Acknowledge is to be expected or not, before transmitting data. A second bit is used to inform the upstream switch on whether it should delay sending its marker-acknowledge until it has received one from downstream (Refer to Section 5.2).

2.3 Establishing connectivity, Phase 3: The Use of a Marker

The final part of the 3-way handshake on the hop-by-hop μ-setup is an in-band MARKER. The MARKER serves not only to acknowledge the μ-ACK message, but is also essential to mark the beginning of the new data flow. The use of the 3-way handshake ensures that data at both the upstream node and the link related to the newly allocated VC id are flushed of old data at the time the downstream switch receives the MARKER. The 3-way handshake also allows for recovery from loss of the μ-ACK. The MARKER is the first cell sent in-band by the upstream node. Everything that follows this marker is a valid data cell for the new flow. The MARKER includes the Flow ID, the NSAP address of the connection initiator (source), and a bit to indicate if the version of the MARKER is a retransmission or not. The switch controller may, for example, use the source NSAP address for functions, such as VC re-routing or generating an out-of-band RELEASE.

The upstream node, *after* sending the MARKER, sends data on this VC id if the downstream node has not requested that a Marker-Acknowledge is to be expected. If a Marker-Acknowledge is to be expected, then the upstream node transmits data only after receiving the Marker-Acknowledge.

3 Call Setup for Multicast

UNITE incorporates the functionality of having multipoint-to-multipoint communication [iii] as an integral part of the signaling architecture. The simpler cases of point-to-multipoint multicast calls are simple sub-cases of this overall multicast architecture.

The simple difference between a unicast call and a multicast call is that the μ-setup issued indicates that the call is potentially a multicast call. For the purposes of this discussion we assume that the underlying network forwarding mechanism can manage issues such as cell interleaving [iii]. Therefore, we describe procedures that are applicable for core-initiated joins (for core based trees [iv,v]), which are similar for source-initiated join for a source-based tree. We then describe leaf-initiated joins for other participants that join subsequent to the call being setup [vi,vii].

3.1 Core/Source Initiated Joins

Core initiated joins (or source initiated joins) are relevant when the set of participants is known initially. The core issues a µ-setup knowing the NSAP address of each individual participant. Since there is no way to package more than one NSAP address in the micro-setup, an individual micro-setup is issued for each of the participants. We think this is not as important, because, (a) the µ-setup is relatively “cheap” and (b) the number of participants that subsequently join using the leaf-initiated joins may dominate.

The first µ-setup issued to a participant includes a label (in the “Type” field) to indicate that it is a “multicast-capable” call setup. The rest of the µ-setup is similar to that described for a unicast call. The Flow-id is determined by the originator (i.e. the core or sender). The Flow-id acts as a call-reference identifier for the multicast call. The µ-setup issued for joining subsequent participants uses the same Flow-id, again labeled as a multicast. The µ–ACK that comes back from the downstream hop returns a VC id as with unicast calls. The MARKER transmitted by the core (or source) is sent in-band, on the VC id returned in the ACK.

The Flow-id used in the micro-setup is retained at the switch, as a means of identifying the multicast call. During joins, the switch sending the µ-setup maintains state, which includes the Flow-id and the destination NSAP address to which the setup was issued (the new leaf). This way, ACKs that return for the individual setups issued may be matched up by the switch, for purposes of managing their retransmission.

The initiator of the µ-setup (core or source) sends the MARKER when it receives the first µ–ACK. Upon receiving subsequent µ–ACKs, the source/core knows that the VC is already open (operational) and therefore, doesn’t generate a further MARKER. At a new branch point on the multicast tree, however, a MARKER is required to the new destination: this is because that branch of the tree needs to be flushed of any old data that is currently in existence for that VC identifier. The controller is responsible for generating and sending this in-band MARKER. Subsequently, data may be forwarded on that VC id, as a result of a proper 3-way handshake. Figure 3 illustrates this scenario.

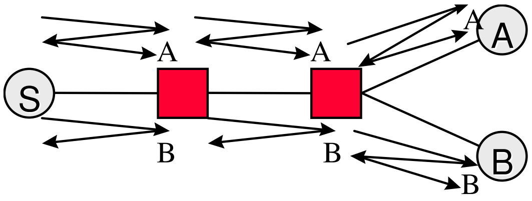

Figure 3: Core initiated join. Observe, that the marker on the last hop to B is generated by the controller at the branch point.

3.2 Leaf Initiated Joins

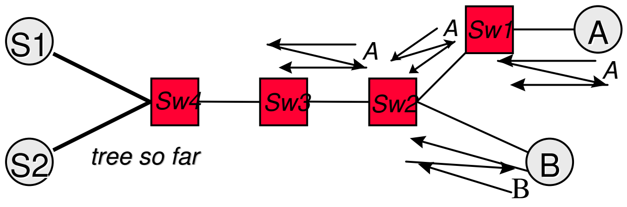

Figure 4 : Leaf Initiated Join: A's LIJ is progressed to switch four. B's LIJ is suppressed at switch two.

The mechanisms for Leaf Initiated Joins (LIJ) are similar to those suggested in the conventional ATM Forum UNI 4.0. However, instead of having a separate "LIJ" and "Add-Party" mechanism, UNITE uses the same mechanisms of the μ-setup for performing a LIJ. Consider Figure 4, where two participants A and B wish to join the multicast tree, that currently ends at Switch 4. The LIJ is a μ-setup (the Type indicator indicates that this is a LIJ for a multicast call) from one of the participants, that is directed towards the core/source, using the NSAP address corresponding to the core/source. The Flow ID used in the micro setup is the multicast call reference identifier, and is stored at the switches as the micro setup is forwarded upstream towards the core. We assume that the underlying call routing mechanisms direct the μ-setup towards the source/core in accordance with the appropriate criterion (e.g., shortest-path or least cost). When a LIJ arrives at a switch from another participant, such as B, the Flow ID is recognized as already existing at the switch, and the forwarding of B's μ-setup is suppressed. This may be done only if the core does not wish to be notified of the address of an individual leaf joining. Note that this happens even though the LIJ of the first participant added on this branch, has not yet reached the tree at Switch 4. When the μ–setup from A is issued, the 3-way handshake results in the marker being forwarded by the switches upstream. This effectively "opens" up the VC from the node A up to the branching point, at Switch 4. Along with the suppression of the μ–setups, subsequent markers are also suppressed at the switches.

4 DETAILS OF QoS MANAGEMENT

The second part of UNITE is a separate means of full QoS specification and negotiation. This allows both a very flexible QoS management process, as well as the ability to incorporate QoS renegotiation with ease. As discussed in the previous sections, the μ-setup includes a QoS byte that can be used in the original connection setup to support "coarse" or aggregate level QoS (e.g., by allowing some differentiated decision for the forwarding of the μ-setup). UNITE supports detailed QoS signaling (or "full" QoS signaling) that is performed in-band on the already established best-effort VC. We anticipate that a large subset of flows will not use the additional phase of a QoS setup for establishing a distinctive quality of service. The QoS class specification that is provided in the initial μ-setup may be adequate for a reasonably large subset of best-effort flows (e.g., a large class of TCP/IP and UDP/IP flows carrying non-real-time data clearly don't need to have a subsequent QoS setup phase). Similarly, well-understood real-time flows such as uncompressed telephony traffic (μ-law, *a*-law) may be adequately specified as being "delay- sensitive."

The assured QoS for the flow begins after the QoS negotiation completes, end-to-end, in a similar fashion to the conventional UNI QoS setup. Also, we believe that most of the more sophisticated QoS management will be handled in software as is the case in the current UNI framework. However, UNITE's framework provides a more flexible and efficient QoS management in the following dimensions:

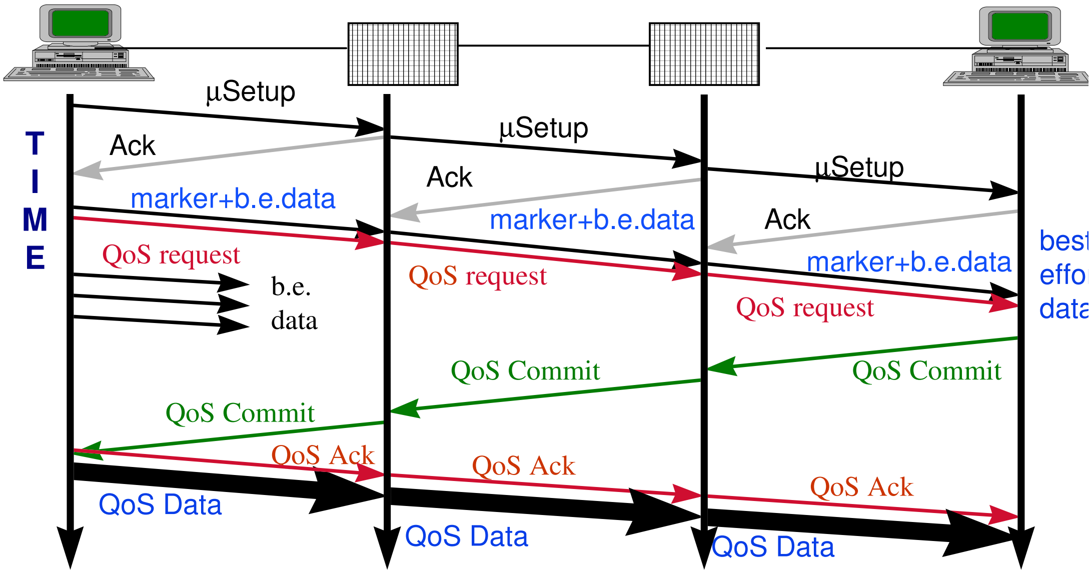

Figure 5 : Protocol for Establishing QoS in UNITE.

1. UNITE QoS requests may be initiated by the source or the destination of the original best effort connection setup. In the more general case of multicast connections, QoS requests may be source/core initiated or leaf initiated.
2. UNITE QoS in-band signaling allows QoS renegotiation originating from any of the connection end points.
3. UNITE QoS in-band signaling enables potentially different QoS negotiation modalities and implementations taking advantage of parallelism in the processing of the QoS setup across multiple switches in the end-to-end path.

For those flows that require a detailed QoS negotiation, we use the process of QoS setup described in Figure 5. The QoS request may immediately follow the marker, as shown in Figure 5, or may be submitted after the call is established. The receiver, after processing the request sends a QoS Commit, that commits the reservation. To adjust over-committed reservations, and to confirm the QoS reservation to the receiver, the originator sends a QoS Ack. The delay until a QoS flow begins on the forward path is an end-to-end round-trip plus the processing at the destination. On the reverse path, a confirmed QoS flow begins one round-trip after the QoS Commit is issued from the destination.

For compatibility with existing ATM, we anticipate that the QoS request, Commit and Ack, would be encoded as in the UNI connection “setup” and “connect” messages, as far as the QoS information is concerned. For our purposes in this section, we treat the end-system that initiates the QoS setup request as the “QoS source.” The end-system that responds to the QoS setup request at the other end is the “QoS destination”. During the QoS negotiation, data may still flow on the connection on a best-effort basis. Cells that belong to the QoS negotiation message are marked with a Payload-Type Indicator (PTI), possibly as RM cells, so that it may flow to the controller on the switch. Thus, in fact, QoS signaling and data cells (or messages) may be interleaved because of the PTI value being distinct from one another.

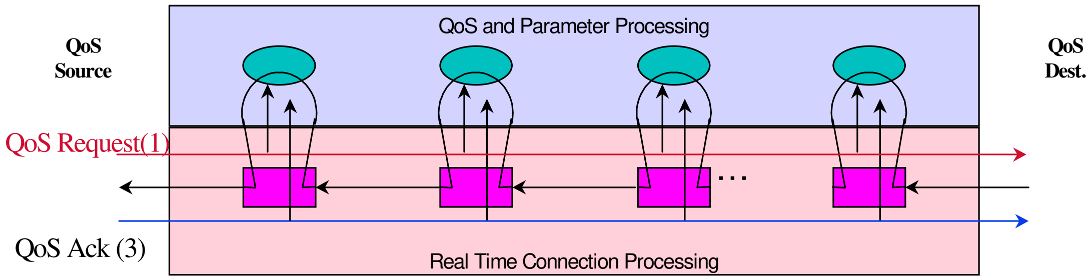
QoS Source
QoS and Parameter Processing
QoS Dest.
QoS Request(1)
. . .
QoS Ack (3)
Real Time Connection Processing

Figure 6: Three Way QoS Setup.

Various alternatives for detailed QoS negotiation can be considered here, including the conventional three way setup described in Figure 6, and one which is consistent with the RSVP-like signaling proposed for IP networks. With reference to Figure 6, the QoS request is multicast to all switch controllers in the path and to the next link at each switch, facilitating parallel processing in the controllers (1). The Commit message traverses the reverse path, "slaloming" to every controller, collecting the commitments (2). The QoS Ack. multicasts the commitment to all controllers (3).

In UNITE a QoS request may be initiated by any participant of a multicast, the core (if present), source or a leaf. Moreover, unless otherwise dictated by higher level policies, core/source and leaf initiated QoS may all be used at different times for the same multicast. As an illustration of the potential of UNITE, we describe the case of Leaf Initiated QoS request by referring to Figure 7.

The leaf initiated QoS requests carry the "demand" from the receivers upstream. When the QoS request arrives at a switch, the "demand" is noted at the switch. The switch conveys upstream, the maximum of all the demands observed from the different branches (a leaf node or a switch may be at the end of the branch). Note that different leaves may issue their QoS requests at different times. The switch examines each QoS request and transmits a request upstream only if the QoS request is higher than the current maximum. When the "demands" arrive at the core/sender, the "permit" returned is the minimum of the offered capacity, the demands received from the leaves and the available link capacity.

Note that each switch needs to maintain state, which is the "demand" and the "permit" returned for each branch for the multicast call. The leaf may be a source or a receiver, requesting a QoS on a shared distribution tree (e.g., CBT).

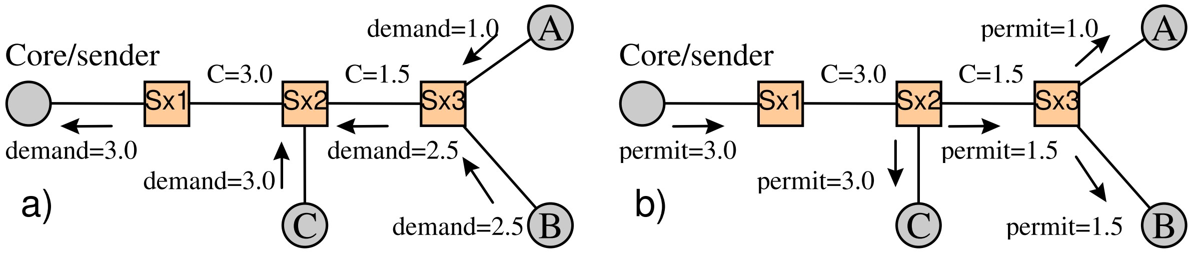
Core/sender
C=3.0
C=1.5
demand=1.0
A
Sx1
Sx2
Sx3
demand=3.0
demand=3.0
demand=2.5
a)
C
demand=2.5
B
Core/sender
C=3.0
C=1.5
permit=1.0
A
Sx1
Sx2
Sx3
permit=3.0
permit=3.0
permit=1.5
b)
C
permit=1.5
B

Figure 7: Multicast QoS. ***Leaf initiated QoS, a) demand phase, b) permit phase.***

5 Interoperability Issues

In this section, we describe how to use UNITE with existing switches – including software based implementations and FIFO switches.

5.1 Interoperability with existing Switches

The proposed UNITE protocol discussed in Section 2 assumes that switches will be able to do per-VC queueing and furthermore will be able to handle the processing of the Marker in-band. Processing of the Marker involves changing the state of the per-VC queue so that arriving cells are buffered rather than dropped. (This ensures that valid data cells, that might follow the marker back-to-back, will be queued, while any invalid cells, e.g. cells in flight from an erroneous connection, will be dropped.) Current ATM switches do not necessarily provide these capabilities, however, and it is crucial that UNITE can still function on such legacy switches. An extra (but optional) Marker-Acknowledge message is introduced to deal with these issues.

If a switch is processing the Marker in software, it cannot guarantee that queue state will change from discard to queueing in time to cater for valid data cells following the Marker. In fact because of different switch architectures and implementations, the amount of time it takes to process the Marker in software will vary greatly. An upstream node therefore has no way of knowing how long to delay before it can start forwarding data cells. By having a downstream node sending the Marker-Acknowledge message only when it is ready to receive data from the upstream node, this problem is solved. An illustration of the optional use of the Marker-Acknowledge is given in Figure 9. A downstream node indicates in the µ-Acknowledge message whether it requires the upstream node to wait for a Marker-Acknowledge or not. The Marker-Acknowledge mechanism can therefore be used on a hop-by-hop basis as dictated by local switch capabilities. When a downstream node has requested the use of the Marker-Acknowledge message, the upstream node starts a timer when it sends the Marker downstream. This timer is cleared when the Marker-Acknowledge message is received from downstream, or, if the timer expires the Marker is retransmitted. Note that the penalty for using the Marker-Acknowledge is two round-hop worth of delay as opposed to one round hop in the ideal case. The hop-by-hop nature of the original protocol is however maintained

The Marker-Acknowledge message is also used to cater for FIFO switches, as illustrated in Figure 10. A FIFO switch will not be able to buffer data cells until it receives an acknowledgment from downstream. (Indeed some FIFO switches might not even be able to accept cells into the switch without having received the outgoing VCI from the downstream switch.) A FIFO switch will then simply delay sending the Marker-Acknowledge until it is capable of forwarding data cells. This in itself is however not enough. If the upstream switch is itself a FIFO switch, then the second FIFO switch has to also indicate to the upstream switch that it should not send a Marker-Acknowledge message upstream until it has received a Marker-Acknowledge message from downstream. (In the non-FIFO case described above, a switch can send a Marker-Acknowledge message upstream, as soon as it is capable of receiving data. If both the upstream and the downstream switches are FIFO switches, however, the upstream switch should wait until the downstream switch is capable of receiving data.) The Acknowledgment message therefore also needs to indicate to the upstream node whether it should wait for a downstream Marker-Acknowledge, before it can send its Marker-Acknowledge upstream. (If the upstream node is not a FIFO switch and is capable of buffering data, it can simply ignore this indication in the Acknowledgment message.)

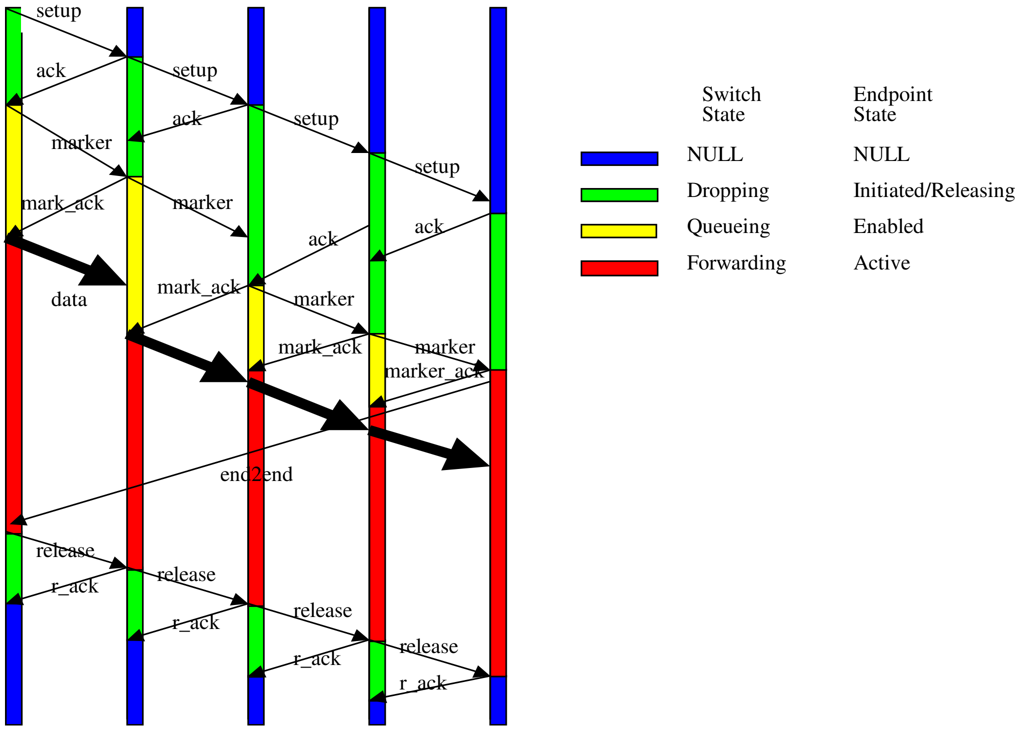
setup
ack
setup
ack
setup
marker
setup
mark_ack
marker
ack
ack
data
mark_ack
marker
mark_ack
marker
marker_ack
end2end
release
r_ack
release
r_ack
release
r_ack
release
r_ack
Switch State
Endpoint State
NULL
NULL
Dropping
Initiated/Releasing
Queueing
Enabled
Forwarding
Active

Figure 9: Use of the Optional Marker Acknowledge to enable downstream switches to control upstream switch transmission of data until it is ready

This approach has the effect of changing the hop-by-hop delay of the UNITE protocol into a partial end-to-end delay across consecutive FIFO switches. (A setup request will proceed, with data following, on a hop-by-hop basis until a FIFO switch or switches are reached. Forwarding of data will then be delayed until the last FIFO switch in the sequence is ready to receive the data.)

6 CONSIDERATIONS ON UNITE IMPLEMENTATION.

The fundamental features of UNITE, namely, the separation of connectivity from full QoS processing, the specification of single cell signaling messages and the simplified reliability support via timers and retransmission of basic messages, enable a broad range of implementation scenarios for UNITE.

At one extreme, UNITE may be implemented completely at the software level. The only functionality required at the hardware level is the ability to recognize in-band control cells used for UNITE signaling arriving at the switch ports, and to route such cells to the switch controller. In the most basic software implementation per VC queuing would not be required, and early data transmission (before end-to-end acknowledgment) may not be supported. We believe that, while the full latency improvement potential of UNITE is not achieved with such an implementation, significant improvement in call processing capacity as well as fairness improvements may indeed be achieved.

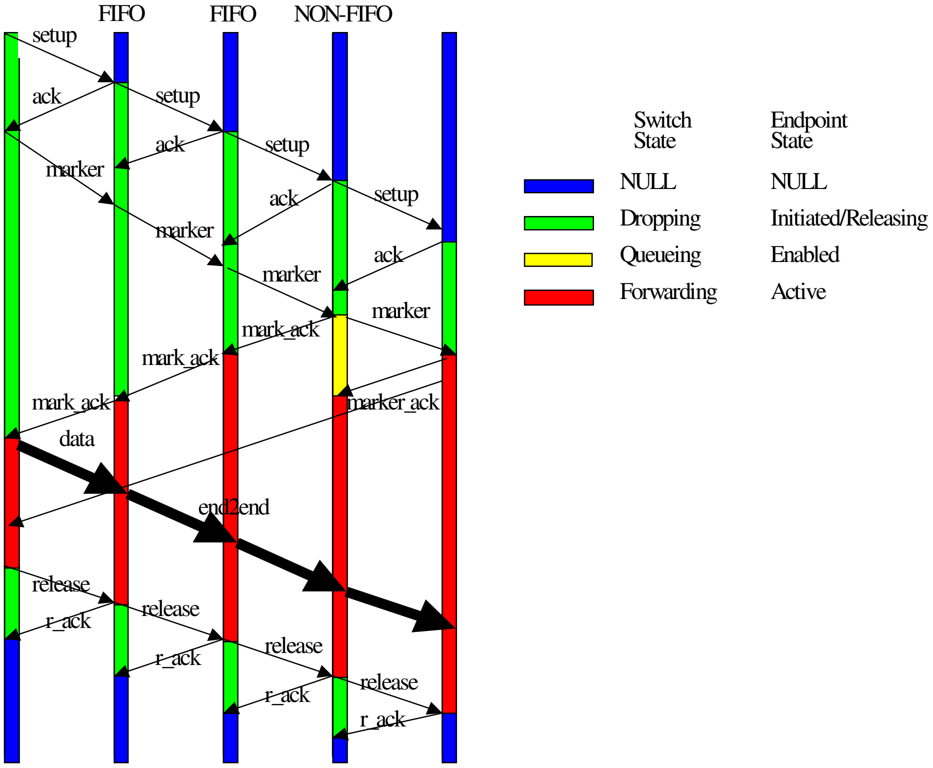
FIFO
FIFO
NON-FIFO
setup
ack
setup
ack
setup
marker
ack
setup
marker
ack
marker
marker
mark_ack
mark_ack
mark_ack
marker_ack
data
end2end
release
r_ack
release
r_ack
release
r_ack
release
r_ack
Switch State
Endpoint State
NULL
NULL
Dropping
Initiated/Releasing
Queueing
Enabled
Forwarding
Active

Figure 10: Use of the Marker Ack with a sequence of FIFO switches.

At the opposite extreme in the range of implementations of UNITE is the scenario where as much call processing functionality is implemented in the hardware, most likely located in the switch line cards and host adapters, and the switch supports advanced queuing and scheduling schemes. In this scenario the full potential of UNITE may be manifested, with close to a single hop round trip latency before the inception of data transmission, and vast call set up capacity increases for best effort or basic QoS calls. Such capacity increases are naturally scaleable with the switch port capacity and the number of ports on the switch, thanks to the distributed nature of the implementation enabled by UNITE. We believe that a call processing capacity of several thousand calls per second per OC-3 port is feasible within a UNITE framework. Note that even in an advanced implementation the full QoS call processing would be handled at the software level.

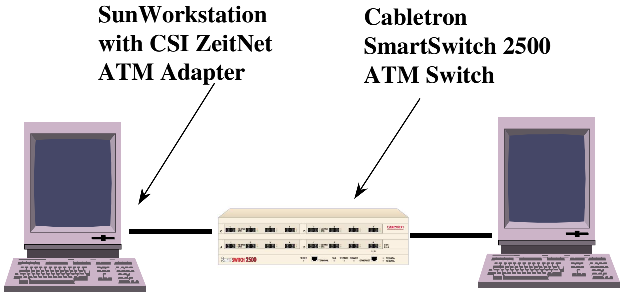

Figure 11: UNITE Prototype Setup.

It is reasonable to conceive a UNITE implementation in which the port processing modules on the port cards support the following functions in hardware:

1. Capture/injection of UNITE signaling cells.
2. Management of timers, retransmissions and state changes in the call processing state machine.
3. Forwarding of micro-setup to the correct outgoing link, based on fast address lookup.
4. Allocation of incoming labels (i.e., incoming/outgoing VPI/VCI and Tags used for routing through the fabric) out of local label pools managed (on longer time scales) by the central switch controller.
5. Basic QoS support. This may imply forwarding and queueing/scheduling based on a the QoS byte in the micro-setup.
6. Control of queue scheduling based on UNITE control messages received (e.g., blocking until a message is received).

A subset of the functionality listed above may also be implemented within the Adapter SAR ASIC, namely, signaling control cell capture/injection and management of timers, retransmissions and state changes in the call processing state machine.

The switch control processor would, in this scenario, be responsible for:

0. Monitoring and management of label pools allocated in real time by the Port Processing Modules.
1. Call accounting and monitoring.
2. Switch level resource management.
3. Full QoS call processing, including Call Admission Control and support of sophisticated bandwidth reservation schemes and management of appropriate scheduling schemes.
4. Initialization, monitoring of error conditions and switch level management.

A range of UNITE implementations falling in between the extremes described above can naturally be conceived, including the case of the current generation of switches supporting per VC queuing in the switch fabric, but still handling UNITE control cells in software.

Large signaling performance advantages could be gained in this case by exploiting the early data transmission feature of UNITE.

In order to explore the implications of a basic UNITE software implementation we developed a UNITE prototype completely in software over a network including two ATM switches and two adapter cards. A picture of the prototype setup is shown in Figure 11. In the prototype in-band UNITE signaling cells are supported by OAM cells.

To evaluate the performance improvement with UNITE, we compared the performance of UNI 3.0 versus the prototype UNITE implementation. The tests used a mature UNI 3.0 implementation, a Radcom test box acting as a source and destination, and a Cabletron ATM Switch. The UNITE tests used two Sun workstations with Cabletron/Zeitnet ATM adapters.

One important issue for connection-oriented communication is the amount of memory used to keep state for each individual connection. At least comparatively, UNITE is efficient in using memory for VC state, using only 128 bytes per best-effort VC in our prototype. In contrast, UNI uses almost 1500 bytes for a best-effort VC. Thus, there is the potential for UNITE to support a much larger number of VCs on switch ports.

Table 1: Setup Latency Comparison (in milliseconds).

	SAR	Q.SAAL	SIG			ROUTING	TOTAL
			CAC	FABRIC	MISC. PROT.		
UNI	2.00	3.40	0.30	0.35	4.45	0.40	10.9
UNITE	0.50	0.00	0.0	0.30	0.50	0.40	1.7

We measured the UNITE connection setup latency and throughput. Our preliminary results, using 100 µsecond clock granularity in our measurements, were as follows: The best effort connection setup latency through an individual switch was 1.7 ms with UNITE. In comparison, a best-effort UNI connection took 10.9 milliseconds. The various components of this service time are shown in Table 1. In terms of throughput, UNITE got approximately 700-800 calls/sec, while with UNI we got approximately 130 calls/sec. We believe that with some simple optimizations, UNITE could easily get over 1000 calls/sec. We expect that even more substantial improvement could be achieved with UNITE with a streamlined/hardware implementation.

7 BENEFITS OF UNITE

In this section we summarize and reorganize, as a quick reference, the benefits of UNITE discussed in this internet draft.

1) Separating connectivity from QoS enables UNITE to:
   a) Achieve high throughput for establishing connections.
   b) Have a very low latency to begin data transmission because we don’t have to wait for an end-end message exchange.
   c) Have throughput and latency for connection establishment be independent of the complexity of the QoS class and other service characteristics. Complex QoS specifications are allowed for those connections that need it.
   d) Support QoS establishment and renegotiation in a similar fashion. This enables simple ways to change parameters for flows.
   e) Allow for communication on a best-effort basis even upon failure of a QoS request.

2) UNITE is ideally matched to carry Internet traffic over ATM networks.
3) UNITE is optimized for distributed hardware implementation of signaling within a switch on a per-port basis.
   a) A single cell, fixed length, fixed format micro-setup allows for high-speed processing of the setup message.
   b) No single point of re-sequencing or SAR is needed, and no software stack such as SSCOP is required for supporting basic connection establishment.
   c) Reliability is achieved via simple timers and retransmissions that are easily implemented in hardware.
   d) State and context information for connectivity requires only a small amount of memory and can be kept in a distributed fashion, even on a line card.
4) Separation of connectivity and QoS and sending QoS related messages in-band allows the network to have QoS setup initiated by sources or destinations.
5) UNITE provides isolation of QoS negotiation to connections that require it
   a) Multiple passes, complex QoS negotiation or other service characteristics may be allowed.
6) Supports a full range of multicast architectures, including multipoint-to-multipoint.
   a) QoS can adapt to the capabilities of the branches of the tree.
7) Builds on the QoS work done for ATM and IP.
   a) Accommodates a wide range of QoS models: UNI, RSVP and future evolution.
9) Builds on substantial amount of the work done for PNNI for QoS-sensitive routing.
10) Allows communication on a path selected based on a coarse class specification. Hence even simple connectivity can be better than “true best-effort”.
11) Inter-works with existing UNI switches.
12) Allows for legacy switches and various levels of hardware implementation complexity.

8 SUMMARY

We have described a protocol for lightweight signaling. The key idea behind the protocol, is the separation of connection establishment and QoS processing. This makes connection setup independent of QoS processing complexity, benefiting most flows and best effort flows in particular, as the channel becomes immediately available on best effort basis. The separation allows all flow specific signaling, i.e., QoS messages to be carried in-band, thus protecting the shared signaling resources. UNITE signaling unifies initial QoS setup and renegotiation, and supports source/core initiated QoS as well as receiver initiated QoS requests.

The connectivity setup message is a single ATM cell (called μ-setup). The μ-setup and μ–acknowledgment are exchanged on a hop-by-hop basis on a “signaling channel”. By incorporating a minimal QoS class identification in the μ-setup request, UNITE has the ability to provide QoS sensitive routing. Data flow on the best-effort VC may begin without waiting for the setup to be completed over the entire end-end path. The use of per-VC queuing permits the source to start sending data on a best-effort basis as soon as the connection has been setup on the next hop.

Subsequently, QoS setup requests and acknowledgments flow in-band on the best-effort VC that is initially setup. The QoS for the flow is assured upon completion of the end-end exchange of the QoS setup and ack. The complexity of QoS messages and their processing is isolated to those VCs requiring it, without impacting other VCs. In addition, it allows for the QoS requester to be either the source or destination of the connection. The architecture recognizes the need for multipoint-to-multipoint connections, and the possible need for variable QoS to different participants in the multicast group.

9. Applicability OF UNITE to MPLS

As we stated earlier, this Internet Draft is based on an ATM Forum contribution and as such is written within an ATM context. However, we believe that UNITE might also be of value within the context of MPLS and have therefore decided to present it to the MPLS working group to solicit feedback.

Clearly, UNITE currently uses ATM addresses, to be applicable to ATM. However, we believe that the protocol could be used with IP addresses, with hop-by-hop forwarding of the micro-setup at the MPLS switches using conventional link-state routing, such as OSPF. Because of the inclusion of the QoS class in the micro-setup, we can take advantage of potential enhancements to IP to accommodate QoS-sensitive routing.

We believe that UNITE might be applicable to the following objectives of the MPLS working group. We reiterate below, these specific objectives:

```
1. Specify standard protocol(s) for maintenance and distribution of
label binding information to support unicast destination-based routing
with forwarding based on label-swapping.

2. Specify standard protocol(s) for maintenance and distribution of
label binding information to support multicast routing with forwarding
based on label-swapping.

4. Specify standard protocol(s) for maintenance and distribution of
label binding information to support explicit paths different from the
one constructed by destination-based forwarding with forwarding based on
label-swapping.

6. Specify a standard way to use the ATM user plane

   a) Allow operation/co-existence with standard (ATM Forum, ITU, etc.)
ATM control plane and/or standard ATM hardware
   b) Specify a ‘label swapping’ control plane
   c) Take advantage of possible mods/improvements in ATM hardware, for
example the ability to merge VCs

7. Discuss support for QOS (e.g. RSVP).
```

UNITE is a framework that is efficient in providing connectivity, with very low latency for a source to begin transmitting data when using on-demand label distribution. An integral part of the framework is providing very flexible support for QoS, accommodating multiple QoS models including sender and receiver initiated QoS. Multicast support fits naturally in UNITE, with common procedures applicable for unicast and multicast (both source and receiver joins). UNITE allows the network to scale to large numbers of nodes because of the ability to support on-demand label distribution efficiently. Further, UNITE achieves scalability in the following dimensions:

1. UNITE can achieve high throughput for label distribution.
2. UNITE enables initiation of packet forwarding with low latency.
3. UNITE minimizes the amount of state needed in the network.

UNITE uses a QoS hint to route the setup. The explicit path established in this manner may therefore be different from "default" destination based forwarding because it uses QoS sensitive destination based routing.

As it is currently defined, UNITE is an ATM control protocol and is therefore directly applicable to objective (6).

UNITE also directly addresses QoS issues without making any assumptions about the specific QoS model that is used. For example, RSVP can be combined with UNITE to perform the actual resource reservations.

If particular MPLS switches do not support native IP forwarding, then the need for UNITE appears even more compelling, especially in an environment where services other than best-effort are provided. (e.g., in a Diffserv type of environment, it would be wasteful to distribute labels for all service classes across the whole network).

9 References

[i] L. Zhang, S. Deering, D. Estrin, S. Shenker, “RSVP: A New Resource ReSerVation Protocol”, IEEE Network Magazine, Sept. 1993.

[ii] Danny J. Mitzel, Deborah Estrin, Scott Shenker, and Lixia Zhang, “An Architectural Comparison of ST-II and RSVP”, Proceedings of Infocom’94, 1994.

[iv] M. Grossglauser & K.K. Ramakrishnan (1997) “SEAM: Scalable and Efficient ATM Multicast,” *Proceedings of IEEE Infocom'97*, April 1997, Kobe, Japan.

[v] T. Ballardie, P. Francis, and J. Crowcroft, “Core Based Trees (CBT),” in Proc. ACM SIGCOMM'93, San Francisco, California, September 1993.

[vi] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei, “An Architecture for Wide-Area Multicast Routing,"” in Proc. ACM SIGCOMM'94, London, August 1994.

[vii] ATM Forum, “ATM User-Network-Interface (UNI) Signalling specification version 4.0,” July 1996.

[viii] S. Deering, “Multicast Routing in Internetworks and Extended LANs,” in Proc. ACM SIGCOMM'88, Stanford, California, August 1988.

Authors’ Adresses

K. K. Ramakrishnan, Gísli Hjálmtýsson, Kobus Van der Merwe
AT&T Labs. Research
180 Park Avenue, Florham Park, N.J. 07932
kkrama@research.att.com,gisli@research.att.com,kobus@research.att.com
Phone:+1 973 360 8766
Fax: +1 973 360 8871

Flavio Bonomi, Sateesh Kumar, Michael Wong
CSI ZeitNet/Cabletron
5150 Great America Parkway, Santa Clara, CA, 95054
fbonomi@ctron.com, skumar@ctron.com, mwong@ctron.com
Phone: +1 408 565 9360
Fax: +1 408 565 6501