An ATM Switch Control Interface for Quality of Service and Reliability

Rahul Garg
Dept. of Computer Science
Indian Institute of Technology
Hauz Khas, New Delhi, India
rahul@niyati.iitd.ernet.in

Raphael Rom
Dept. of Electrical Engineering
Technion, Haifa, Israel
and
Sun Microsystems, Palo Alto CA,
rom@ee.technion.ac.il

***Abstract* -- Traditional communication switches include an embedded processor that implements both the switch control and network signalling. Such an architecture is failure-prone due to the complexity of the control software of modern network. Recently, a new approach of controlling switches through an external controller is gaining momentum, due the flexibility and reliability it affords. In addition, open control protocols and interfaces for controlling and managing networks, are now emerging as an alternative to specifications and standards. For instance, in the OpeNet project Sun Labs has designed and implemented an open, high performance ATM network control platform.**

In this paper we describe the OpeNet Switch Control Interface (ONSCI)--an open local switch control protocol, for controlling an ATM switch. An ATM switch controller uses this protocol to setup or tear down virtual circuits and perform other control and management functions in an ATM switch. Important and distinguished features of this protocol are primitives for Quality of Service (QoS) management in the switch and support for fault-tolerant operation in case of failure of a switch controller. The design of these primitives is based on conceptual modeling of the switch architecture. The model is generic enough to cover a wide range of ATM switches.

The other primitives for switch management and control are borrowed heavily from Ipsilon's GSMP protocol. They include primitives for switch configuration, port and switch management, VP management and performance monitoring.

The protocol was implemented and integrated with the OpeNet platform. Without going into specifics of the protocol, we describe its design principles and show how it has affected our protocol.

Keywords: ATM, GSMP, QoS, Fault Tolerance, Availability, Reliability, IP-Switching, Open Interfaces, Admission Control, GCAC, PNNI.

I. Introduction

Communication switches are built from two major components: a hardware component where the data switching takes place, and a software component which provides the control mechanisms and the integration of the individual switch into a complete network. In traditional switches the control software is implemented on a dedicated processor embedded into the switch's hardware. This integration is typically tightly coupled and is tailor-made for the specific hardware and network application. Such an architecture suffers from many disadvantages such as inability to cope with the progress of processor and software technology and the inability to re-use the software.

A new trend has started recently and is gaining momentum: to devise open network architectures based on distributed systems principles. Several research groups as well as network equipment manufacturers are engaged in active pursuit of the problem. Moreover, standardization efforts have recently started by the IEEE (e.g., the P1520 working group on application programming interfaces for networks [10] and its ATM sub-working group) with a goal of defining APIs for future multimedia networks. With such an interface in place, the controller is typically detached from the rest of the hardware, need not be tightly coupled with it, the entire design allows for easy upgrading of software and lends itself to better fault tolerance.

As part of its activities, Sun Microsystems Laboratories designed and implemented OpeNet [12][11], an open, non-proprietary, high performance, switch independent ATM network control platform. To achieve the goal of switch independence, none of the mechanisms deployed by OpeNet rely on any particular switch. However, to be deployed for operation, the OpeNet must be interfaced with an actual ATM switch. Thus the need for a generic interface to control ATM switches was felt. This lead to the development of the OpeNet Switch Control Interface (ONSCI) [13] which is the subject of this document.

Our work has two basic distinguishing features from other works on open ATM switch control interfaces. Firstly, we provide support for managing Quality of Service (QoS) in the switch in a manner compatible with the ATM Forum specifications. Secondly, primitives for fault tolerant operation are provided to assist recovery of a switch controller after its failure. In the event of failure of a switch controller, a backup controller (or the recovering failing controller) may assume the control of the switch and after some recovery operations, starts functioning as its primary controller. Other operations supported by ONSCI are setting up and tearing down VCs, configure and manage its ports, manage its VPs and get other statistics and information related to performance monitoring. The interface uses a message exchange protocol which is client/server in nature; the switch controller sends requests to the switch which responds after processing the request.

Our design also assumes that an ATM switch has very limited computational resources and only a simple software component. This is expected since all of the heavy control and management functionality which used to be present in a traditional ATM switch, has been moved to the external switch controller, leaving mainly the functionality of cell forwarding in the ATM switch. Our preliminary experiments with a GSMP capable ATM switch supplement this assumption. A Sun Sparc Station 20 was easily able to saturate the switch with GSMP requests, indicating that the switch CPU had limited capacity.

There has been similar work on this subject such as, Generalized Switch Management Protocol version 1 [14], and version 2 [15] developed by Nokia (formerly Ipsilon Networks) on which our work is based; work being done in the Center for Telecommunication Research at Columbia University (CTR) [16] and work done at Information and Telecommunication Technology Center (ITTC) at University of Kansas [17].

The Generalized Switch Management Protocol (GSMP) [14] was the first protocol developed to control ATM switches. The protocol was intended to be a part of the newly developed IP Switching technology [18] and was optimized for it. GSMP supports only basic operation like setting up and tearing down virtual circuits, monitoring performance, etc. In terms of Quality of service, GSMP supports only fixed priorities and lacks the more advanced QoS needed in an ATM network. The second version of GSMP [15] addressed the problem of QoS in the framework of Class Based Queuing (CBQ) [19]. This model is appropriate for the Internet suite of protocols like RSVP [20], but is still inadequate (and was not intended) for ATM networks.

QoS extensions to GSMP have also been proposed by others. In the context and framework of Xbind [21] an extension was proposed [16] that uses the model of schedulable region [22] to carry out admission control which is not compatible with the specifications of the ATM Forum. Moreover, this approach requires more than basic computational resources at the ATM switch, which we believe should be limited. In, fact the switch may have to carry out compute intense operations to respond to some of the requests. The work by Evans et al.[17] define a reasonable model of managing an ATM switch that provides QoS guarantees, but proposes only an approach and not a complete specification. None of these addresses fault tolerance issues.

The rest of the paper is organized as follows. Section II contains a discussion of design objectives, key assumptions and goals behind ONSCI. To provide QoS support, the model of switch's resources and how the controller manages these resources is very important. This is discussed in Section III, which is the major contribution of this paper. Section IV details our assumptions and approach to support fault tolerance. Various messages exchanged between the controller and switch are briefly introduced in Section V and the paper concludes in Section VI.

II. Design Objectives

A traditional *switch* of an ATM network is shown in Figure 1. It has two parts: A switching component, sometimes called the fabric or the cross-connect, a set of tables containing the forwarding information and configuration data, and a processing unit that embodies the control functions.

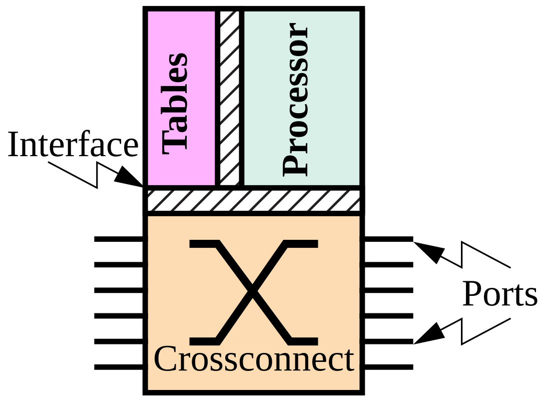


Figure 1: A traditional ATM switch

The ATM switch forwards its incoming cells to different output ports. This forwarding is done at very high speed using the crossconnect. The entries in the forwarding table dictate which cell should be forwarded to which port. Depending upon the switch design, the forwarding table may the stored explicitly, or it may be translated into the state of the crossconnect. Most of the flow control, monitoring, statistics collection is jointly done by the hardware and software components. Other operations, mainly network-wide ones (e.g., network routing), are done solely by the processor. In a typical network, decisions are made by the software running in the processor and then stored in the tables to be used by the cross-connect. Such manipulation of the data structures is typically done by a tight integration between the processor, the cross-connect, and the specific manner in which the data is stored in the tables.

As we indicated earlier, such an architecture is deficient in many ways. To overcome these deficiencies a new architecture is proposed as depicted in Figure 2. The architecture uses the notion of an ATM *node* which is divided into two major parts: The node controller (or switch controller) and the nodal switching subsystem (also referred to as "the switch"). The switching subsystem is essentially a very simple traditional switch where the processor need have extremely limited computational power and a small interconnect-protocol (software) module with which it communicates with the controller.

The switch controller sends messages to the ATM switch through a duplex interconnect. The ATM switch processes the message and sends back the response to the controller. The switch is expected to have a limited capability processor, and a minimal operating system. Its performance characteristics may be insufficient to handle a heavy weight switch control, and is there to perform only simple tasks such as translate the controlling commands given by the controller into the specific manner in which they are stored in the tables. A design decision of our interface is influenced by this approach: the switch control protocol must be lightweight with minimal compute intensive operations.

The switch controller is a generic high performance comput-

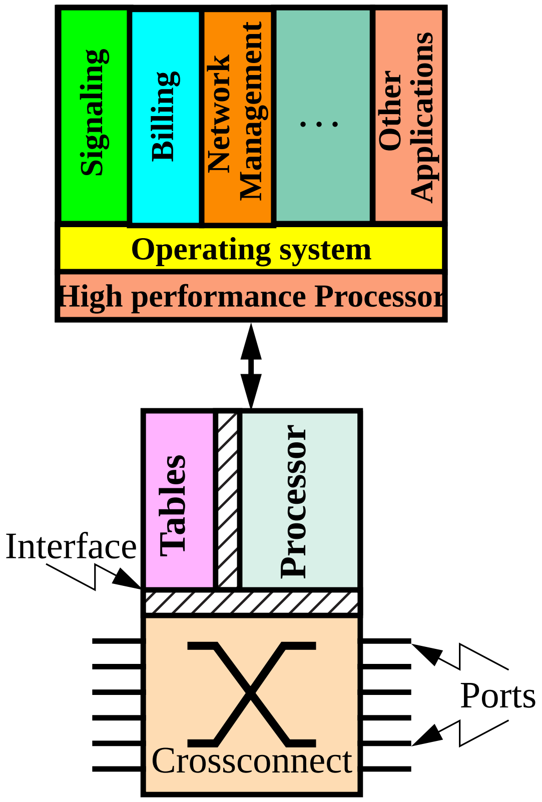


Figure 2: An ATM node

ing platform. No restrictions are imposed on its architecture (it may be a high end workstation for instance). It may run a generic operating system, on top of which various network control applications may be executing. For instance the signalling application may exchange messages with its peers and set up or teardown connections in its ATM switch.

The interconnection between the ATM switch and its controller can either be through one of the regular port of the ATM switch, or through a special port like Ethernet or any other interface. ONSCI uses a regular ATM port for the interconnection. This makes the ATM switch uniform and enables a simpler and richer fault recovery mechanism (Section IV).

There are some requirement from the switch to enable transparent operation of an ATM node. Firstly, the switch must divert all the handling of the special VCs to the attached controller transparently. Special VCs are those designated to handle signalling and control (according to the ATM Forum Specification these carry VCI between 0 and 31 on VPI 0). Because special VC exist on every link, the handling of the special VCs must be done in such a way that allows the controller to determine the port they belong to. In the reverse direction, the controller must be able to send information on any port carrying these VCIs. If a regular ATM port is used to connect controller to ATM switch, as is done in ONSCI, all the special incoming VCs can be diverted to the controller just like a regular ATM connection.

Secondly, the controller should be able to manipulate the switch's tables. To that end it must be provided with the ability to cause the tables and registers of the switch to reflect the reservation of resources.

Thirdly, an interface should be given to the management function of the switch, i.e., those functions that are used for operations such as monitoring, management, configuration control, link failure indications, and processing of OAM cells.

Next, since the switch with its controller are used as a node in an ATM network it is essential to provide support for strict QoS guarantees (as defined by the ATM Forum) to the connections passing through the node. It should also be possible to use more than one signalling protocol. For instance, in one configuration of the network, all the nodes may use PNNI signalling, whereas in another configuration OpeNet signalling may be used.

Fault tolerance is another desirable feature of the design. The fabric is typically far more reliable than the rest of the system because it is logically simpler, may be equipped with redundant hardware and power supplies and traditionally is better tested. To maintain the reliability, the processor that is implemented in the switch is a very simple one, not running any complex operation but just responding to requests from the controller. It is clear that the controller is the susceptible point with respect to reliability. With proper design, as we subsequently show, one can achieve a higher reliability with this architecture compared to the traditional architecture (where every software failure causes a complete nodal crash). Our design also allows to take the controller down in a graceful manner so as to allow upgrading or fixing the control software.

Our main design philosophy is that the switch should be able to continue operating, as much as possible (albeit with limited capability), even with the failed controller and providing a mechanism with which a standby controller (if present) can gain the control of the switch. When a standby controller takes over the control of switch, it needs to recover the state of primary controller just before its crash. Primitives of these operations are included in ONSCI.

III. Models and Structure

The switch control interface is defined in terms of a local protocol between the switch and its controller. The switch interface design is based on conceptual modeling of the switch behavior as well as the way it supports different QoS requirements. These models cover most of the switch implementations regardless of the way the basic switching function is implemented (e.g., space division or shared memory switching), as well as the ways buffers are managed across different connections (e.g., input, output, internal or mixed buffering), and the way QoS is provided (e.g., by using priorities, fair queueing, etc.). It also captures a broad range of the switch core control architecture. The next subsections dwell on the individual models involved; the detailed description of the message and the behavior of the switch and the controller are given in [13].

A. *The Setting*

ONSCI is defined in terms of a message exchange between the switch and a controller, and is not dependent on the physical implementation of this interconnection. This having been said, the preferred implementation is through one of the switch's regular ATM ports. This implementation is much simpler in terms of message exchange and versatility. In such a setting the controller exchanges AAL5 encapsulated messages with the switch over a (pre-configured) VC.

Another benefit of this setting is the ability to have a single controller control multiple switches as depicted inFigure 3.

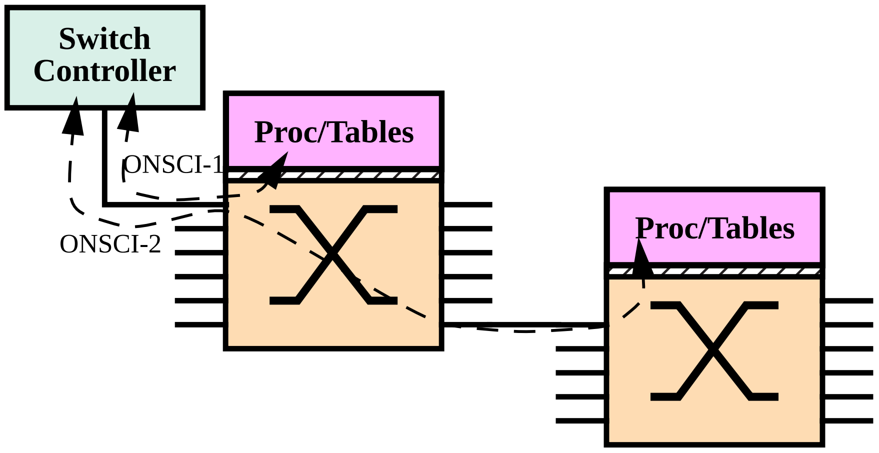


Figure 3: Control of multiple switches

The switch controller controls switch-1 directly (i.e., the control VC indicated by the dot-dashed line) and controls switch-2 indirectly. This is implemented by having the second control VC be configured such that it starts at the switch controller and terminates at the processor of switch-2 (the dashed line in the figure). Such a setting is beneficial if the controller has enough power to control more than a single switch and, more important, this can be a temporary setting when each switch has its own controller but that of switch-2 failed and is now recovering.

B. Protocol Nature}

The interaction between the switch controller and the switch is master-slave. The controller issues a command to the switch by sending a request packet which is encapsulated in an AAL5 frame. The switch performs the request and send a response message with a similar structure. Every packet contains a 32-bit transaction identifier which must appear also in the response message. This identifier is used by the controller to match a response to a request and results in asynchronous operation namely that the controller dictates neither the speed nor the order of execution in the switch. The protocol is essentially a non-reliable transaction protocol that is, if a request (or the response) got lost it is the responsibility of the controller to reissue the request. To simplify the processing, most of the request messages are idempotent, hence the retransmission of a request message doesn't create any inconsistencies and can be retransmitted either with a new or the old transaction ID.

As will be discussed later (Section V), resource management messages are not idempotent and therefore need to be treated a bit differently. In fact, these message carry another identifier and the state of one packet has to be remembered by the switch to allow for retransmissions. This ensures steady operation as long as the controller doesn't send more than one unacknowledged resource management message.

Other than these messages, the switch asynchronously sends event messages to the controller to report certain conditions like a link going down, in a manner similar to that described in [14].

C. Connections

The most basic construct of an ATM switch is a connection which defines the handling of ATM cells. ATM cells arrive at a port carrying in their header the VP/VC identifiers that dictate the port to which these cells should be switched and the quality of service these cells should be afforded. The switch and its controller refer to individual connections by means of an abstract data structure which is referred to as generalized forwarding table.

An entry in the generalized forwarding table consists of a set of input designators, a set of output designators, a traffic descriptor and some additional parameters as depicted in Figure 4. An I/O designator is a triplet of port number, VP and VC identifiers and is the means by which the switch identifies individual data flows.

This representation means that every cell arriving at any input port carrying the VP/VC identifiers (i.e., belonging to a certain input designator) will be switched to *all* the specified output designators i.e., to every port of each output designator - each with appropriate VP/VC identifiers. A unicast connection has a single input and single output designator. A one to many multicast connection has one input designator and multiple output designators. A many to many multicast connection has multiple input and multiple output designators. The traffic descriptor follows the ATM Forum's specification and includes traffic type and other parameters (see [13]). An entry in the generalized forwarding table defines a flow in one direction only. A bidirectional connection includes two entries, one in

Input Designators			Output Designators			Traffic Descriptor	Misc Params
ID_1	....	ID_k	OD_1	....	OD_m		

Port #	VPI	VCI

Input/Output Designator

Figure 4: The structure of generalized VC table

each direction. The port to which the controller is attached treated as other ports and cells can be forwarded to and from it just like other ports.

D. Switch Parameters and QoS Model

The switch can support a set of predefined QoS classes. The set can vary from the minimum set defined by the ATM Forum (CBR, VBR-RT, VBR-NRT, ABR, UBR) [23] to a larger set which is proprietary to an individual switch. The term class identifies the traffic type, parameters relevant to the class and defines the traffic descriptor. Within each class, different calls may have different traffic behaviors like peak bit-rate and average bit-rate which is specified using the traffic descriptor of that class. Other QoS parameters of calls (like delay, cell loss rate) in a class may also vary and are also specified in the descriptor. The QoS in this context means the service offered locally by the switch (for example, additional delay introduced by the switch). The end to end QoS of a call is obtained by composition of local QoS of all its intermediate switch.

The switch has its own proprietary way to translate the class identity and call rate into its internal switch tables. This translation is only known to the switch and may be hidden from the controller. Sometimes this translation is easy as mapping the given QoS class into a internal priority class and including rate parameters for switch policing functions. Other more sophisticated switches need to translate the rate parameters and insert these translation into switch internal buffer management table. The translation to the internal format is part of the tasks of supporting ONSCI on the switch.

E. Switch Accommodation Control Model

In addition to the ability of the switch to carry calls of certain QoS and rates, we assume that an accommodation test mechanism also exists by which the ability of the switch to accommodate an additional connection can be checked. In other words, given some traffic state such as a set of calls which are already established, can a new call of a given rate and given QoS be accommodated? This definition of an accommodation mechanism is by necessity specific to the switch and to the QoS classes supported.

The decentralized signaling and control of the network calls for two types of CAC procedures: local and remote. This results from the manner in which connections are being set up. At the source of a connection (where the user requested it) a routing decision must be made, meaning that the source node must be able to estimate the ability of every switch along the computed route to accommodate the requested connection (we refer to this as the *remote* CAC test). Then, during set-up time, every node along the path must verify that indeed these resources are available (we refer to this as the *local* CAC test). Furthermore, there are two phases during the set-up time: first the node checks for resource availability and sets the resource aside tentatively, and then, when all nodes along the path have confirmed the availability of the resources a commitment is made.

The local CAC test described above, is done by the local controller in cooperation with the local switch. There are several assumptions that must be made with respect to this test. The accommodation test must be conservative in the sense that whenever it indicates that a call can be accommodated, it will be able to obey the QoS contract in the physical switch. On the other hand, the accommodation control test should be fairly accurate and not overly conservative to allow for efficient utilization of all the physical resources available.

The requirements of the remote CAC are quite different. While the local CAC assumes detailed knowledge of the switch’s state, it is unlikely that a controller would know the specifics of any remote switch, nor is it likely to know the exact set of existing calls at that node. To that end a generic call admission control (GCAC) procedure is defined which allows the remote node to make such an approximate calculation. In the ATM Forum’s standard, this procedure takes the new call rate parameters (SCR, PCR, MBS) along with three parameters (ACR, CRM, VF) which are advertised by the remote node for each of its links, and results in a positive or negative test decision.

There are several immediate observations regarding this approach. It is clear that the results of the GCAC procedure should be conservative in comparison to, the more exact and probably more detailed and switch specific, local accommodation procedure. This is to avoid connection setup failure because the remote node overestimated the ability of local node to accommodate the call (based on its local accommodation procedure). Also, while the specifics of remote node are unknown locally, the advertised parameters must be switch dependent (in PNNI GCAC, the VF parameter is the only switch dependent parameter). Therefore it is necessary for the local controller to calculate this parameter so that it can be advertised. This requires that the local controller be able to interrogate the local switch regarding availability of resources; we term this request for switch specific GCAC parameter as *availability check*.

The above observations have led to a CAC model whose

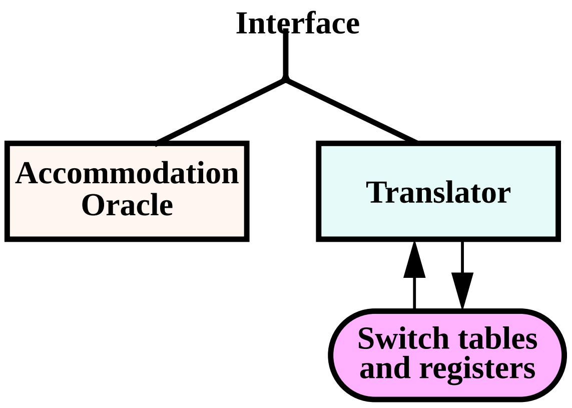


Figure 5: Components of accommodation mechanism

major components are shown in Figure 5. The translator block converts commands which are expressed in terms of a generalized VC table entry to the switch internal formats and parameters. In the reverse direction, the translator will make some of its internal settings available to the controller. When written to the switch tables and registers, switch resources are *committed*. The accommodation test and availability checks are performed with the accommodation oracle whose components are depicted in Figure 6.

At the core of the accommodation oracle is a function or a performance model which can estimate quite accurately whether a new call of given QoS and rate parameters can be accommodated. While such a procedure is an essential block in any ATM system, it is the most switch dependent one. Even if we use known models (such as one of the equivalent capacity models of [23], [24] or [25]) there are necessarily some parameters in these models which are switch specific and capture, for example, the amount and structure of buffering in the switch. It is quite possible that the implementation of the oracle might contain a proprietary switch model. In addition the oracle maintains a current traffic base, whose structure is undefined but which embodies the oracle's view of existing traffic base. To make an accommodation decision the oracle combines its notion of the current traffic with the requested new connection.

With no further assumptions, the traffic base will consist of the list of connections with their traffic descriptors and QoS parameters. This will render the test and commit operations very complex and time consuming, adversely impacting the connection set-up procedure as we described earlier. To facilitate these operations we further assume that the accommodation control function is additive (or cumulative). This means that if A is the set of existing calls over a link of the switch and c_1 and c_2 are two given calls, and if c_1 is feasible under link state $A \cup c_2$, then c_2 is feasible under the link state $A \cup c_1$. This implies that the list of individual connections need not be maintained but that some form of aggregation would suffice and that allocating and de-allocating resources can be easily incorporated into the base. We also assume that the accommodation control function is monotonic which means that if c_1 is feasible under state A and c_2 has a rate description (and QoS) that is smaller than that of c_1, then c_2 is also feasible under state A. With this assumption one availability check would suffice to quickly determine the availability of resources for several (small) connections.

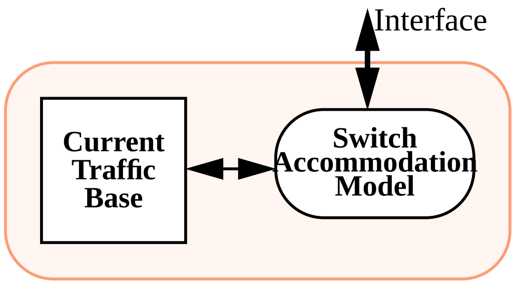
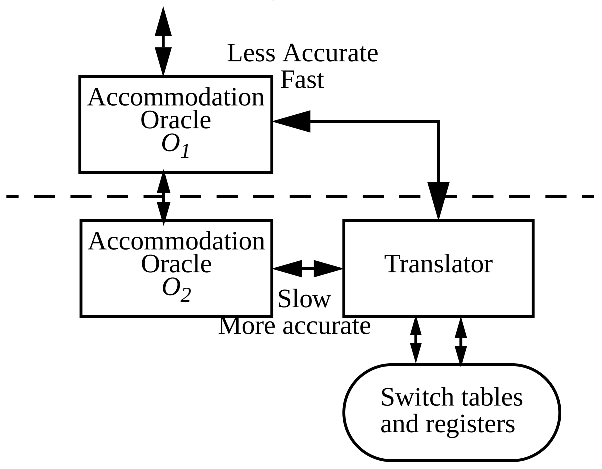


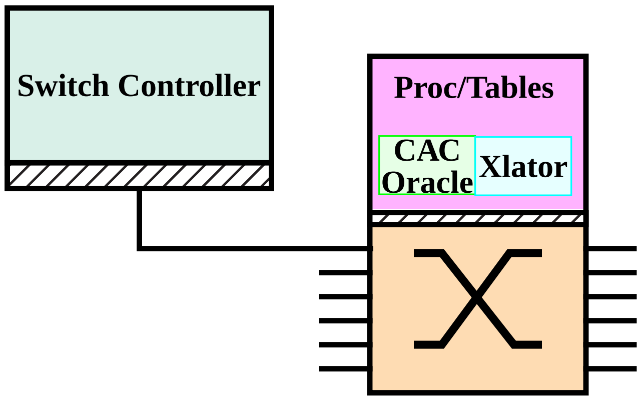
Figure 6: Structure of accommodation oracle

As we indicated earlier, for each call, CAC test must be performed on each of the nodes it traverses which means conducting a computation and interacting with the traffic base. The efficiency of this operation is highly dependent on the manner in which it is implemented. We therefore devise a two tier architecture as shown in Figure 7.

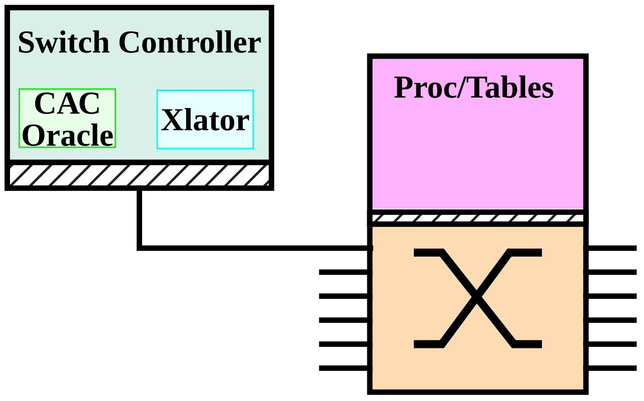

Less Accurate
Fast
Accommodation Oracle O_1
Accommodation Oracle O_2
Translator
Slow
More accurate
Switch tables and registers
Figure 7: Structure of the two tier architecture

The first module of the architecture is a more conservative accommodation oracle o_1 which is based on a simpler switch model (such as GCAC), is present on the controller. The second module is a switch specific and accurate oracle o_2. Any CAC request in the controller is first presented to o_1. Acceptance of the call by o_1 implies that o_2 will also accept the call, so the expensive CAC on o_2 need not be performed. Only the traffic base need be updated. This can be done asynchronously (while the call may proceed). If o_1 rejects a call, the controller can determine that CAC needs to be performed by o_2 and if the result of this test is positive, the call is accepted and the traffic base must be updated.

Finally it is left to determine where these various functions are implemented. Since the depiction in Figure 5 is logical, not physical, all possible combinations are possible but only few make sense. o_1 is clearly implemented in the controller. As we indicated, this helps reduce the call set-up latency. With respect to o_2 there are several options. If a good and accurate model of the switch is available it would make sense to implement o_2 in the controller as well (as shown in Figure 8(b)). If an accurate



(a) Switch internal implementation



(b) External implementation on controller

Figure 8: Accommodation oracle implementation options

model of the resource computation is not available (e.g., the manufacturer will not disclose it), the entire o_2 must be implemented in the switch.

Two configurations are shown in Figure 8. In the implementation shown in Figure 8(a), the oracle and the translator are in the switch. This configuration allows the oracle to directly access all the internal registers of the switch and thus make the most accurate calculations. The configuration depicted in Figure 8(b) is advantageous in that the test and check functions in which the oracle is involved are efficient since they do not require any communication with the switch and can make the full use of the computational resources at the controller. In this case the oracle may have to retrieve switch-specific information from the switch, for which the *private* primitive is provided in the interface. We believe the latter is the preferred implementation.

IV. Switch Reliability Support

The controller is very complex and has different software components for network control, network management and other functions. The switch processor, on the other hand, performs very simple functions: mostly responding to the requests from the controller. The switching fabric is traditionally better tested, may have redundant hardware and redundant power supplies specifically to increase its reliability. Therefore, the ATM switch controller is more prone to failures. It may crash because of a bug in any of its components, or because of a power failure, or it may be intentionally brought down for upgrade or maintenance purposes. ONSCI therefore includes several primitives to enhance the controller's availability.

The controller manipulates the state of the ATM switch in order to perform various network level functions like call setup or teardown but is not involved in the normal data transfer operation. This means that the switch continues switching cells according to its forwarding table entries, even after its controller fails and hence connections that were successfully established before a controller's failure, are not affected by its failure. Only network level control operations are affected. In particular, data on normal VCIs are switched as usual and data on control VCIs which is typically forwarded to the controller, is discarded if the controller is not operational.

Each switch is connected to a primary controller which controls and manages the switch. It may also be connected to one or more secondary controllers which act as backup to the primary controller. When the primary controller fails, one of the secondary controllers assumes the control of the ATM switch and becomes its primary controller. At the system design level there exist an issue of how many secondary controllers exist (per node and per network) and where to locate them. Upon the primary controller's failure the most important issue is locating the secondary controller: where is a secondary controllers located, how is it connected to the switch and how does it assume the control of the ATM switch? Once a secondary controller has assumed control, the question is how does it assemble the necessary state information required to work as a substitute of the failed controller? Once a failed controller recovers, does it make its state consistent with the state of the network? After the crash of a controller while secondary controller was recovering, events like connection tear-down may have happened as a result of which some connections (passing through the switch of the crashed controller) which should have been terminated, may remain partially established, and the question is how are these identified? In the next subsections we delineate ONSCI's answer to these questions.

A. *Multiple Controllers of a Switch*

In the simplest configuration, there exists no secondary controller. thus, when a controller fails it is rebooted and reassumes control of the switch. Such an operation is typically not fast enough for most operational network environment. The simplest configuration that does include secondary controllers is a naive approach where a secondary controller is connected to the switch via another dedicated ATM (control) port. When the secondary controller detects the failure of the primary controller, it assumes control of the ATM switch. This is made possible by having the switch process requests from several ports (although only one would actually be active). In this approach, a switch must set aside at least two dedicated ports for its controller (more if more secondary controllers are present). In addition, two (or more) dedicated high performance machines are needed for primary and secondary switch controllers. The maximum number of secondary controllers would be a static parameter limited by the number of ports available in the ATM switch for controllers. Therefore this naive approach is inadequate mainly because the backup controllers are assigned statically to the ATM switches.

We propose a more dynamic approach, in which (redundant) secondary controllers may be located anywhere in the network and the binding between an ATM switch and its controller is dynamic. This would allow sharing redundant controllers thereby providing any desired degree of reliability.

We use a notion of a *control point* which is a handle using which the switch can be controlled. A control point at a switch is defined by an I/O designator, i.e., the tuple (input port, input VCI, output port, output VCI). A request message sent to an input designator of a control point is interpreted by the ATM switch as an ONSCI message. The response is sent to the corresponding output designator. Every switch has at least one default control point, which is used by the switch to locate its primary controller when the switch is powered up.

Control points may be created (or deleted) dynamically by a controller whenever needed (subject to a maximum, which is a configuration parameter). After the creation of a control point, a controller sets up a bidirectional VC from the switch's control point to a redundant controller somewhere in the network, which then becomes a secondary controller of the switch. If a switch has *N* control points, then potentially *N* controllers can have full control of the switch. In practice, only the primary controller is active at a time. All other controllers are secondary and remain inactive till the primary controller fails. Note that with this approach, only a virtual circuit (as opposed to a dedicated port) is needed to connect a secondary controller to a switch. The controller can be present anywhere in the network, as long as a connection between the switch and controller can be established.

Such a setting allows even more sharing. For example, it is possible for a controller to control more than one switch as indicated earlier in Figure 3. This means that an active controller can act as a secondary for another node, and assume control as necessary. If this secondary controller does not have enough computational power this might not be a good long term solution but might be useful if this is only temporary, e.g., until the original failed controller reboots.

After a secondary a controller assumes control of an ATM switch, and becomes its primary controller, it may set up another secondary controller. This involves locating a free secondary controller, setting up a logical path from the controller to the ATM switch and creating a control point in its ATM switch. The former two operations need support from higher level protocols, whereas the latter needs just an interaction between the controller and its ATM switch, which is provided by ONSCI.

B. Rebuilding State Information

When a secondary controller takes over the control of an ATM switch, it must first rebuild the state of the primary controller, then it should create an additional secondary controller (if possible) and finally advertise its presence in the network and start working as the primary controller of the switch. Of these, the first one is the hardest. The relevant state of the primary controller includes list of connections (in the generalized VC table), traffic descriptor and QoS of individual connections, and other information like link status of individual link, the load on ATM switches etc. The secondary controller gathers the relevant information by three methods: from periodic broadcasts, by querying neighbors, and by querying switch.

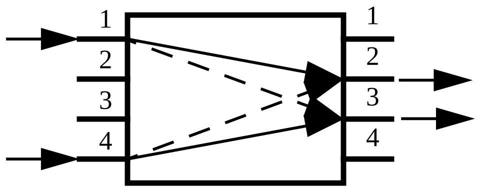


Figure 9: Problem in reconstructing forwarding table

The generalized VC table is most difficult to reconstruct. It can be reconstructed neither from broadcast information nor from neighbors. This is illustrated by the following example. Consider a four port ATM switch as shown in Figure 9. Assume that connection A arriving at input port 1 is switched to the output port 3. Let another connection B at input port 4 be switched to the output port 2. Also assume that both the connections have identical QoS parameters and traffic descriptors. By querying neighbors, a recovering controller, would only have the information that there are two connections, one arriving at port 1 and other at port 4 and one departing at port 3 and another departing at port 2. With this information it cannot be determined whether incoming connection on port 1 is being switched to port 3 or to port 2. This information can only be obtained from the state of switch's interconnect.

Information about the state of the switch's crossconnect along with the information from neighbors and broadcast information is sufficient to reconstruct generalized forwarding table and data structures needed for other control purposes like signalling. State of the traffic base of accommodation control oracle is constructed by accumulating the traffic descriptor and QoS of each connection to the traffic base. Recall that in Section III.E we assumed that the accommodation control procedure is cumulative. Therefore the connections can be reaccumulated in any order to reconstruct the traffic base.

The state information thus constructed reflects the state of the controller at the time it crashed. Some events (like connection teardown) may have occurred between the crash and subsequent recovery. Thus, the recovering controller needs to carry out corresponding update in its state and possibly some additional operations, described next.

C. Partial and Zombie Connections

It is possible that due to unforeseen transient circumstances, the state of the current node is incompatible with its neighbors'. For example, if a controller crashes during the operation associated with a connection teardown. In such cases a connection teardown message will not be able to traverse the entire connection resulting in a partial connection which should ideally have been completely torn down. These partial connections may unicast as well as multicast. Formally, a partial connection is a connection such that there is one neighboring controller which believes that the connection exists and there is another which believes it has been closed. A zombie connection is one where none of the adjacent controllers believes that the connection exists.

The controller must teardown partial connections for which either all the input branches have been deleted, or all the output branches have been deleted. In addition, it must also update its current traffic base. If only some of the input and output branches have been deleted, the forwarding table must be modified appropriately and the traffic base should be updated. For zombie connections, the forwarding table entries must be removed, and the current traffic base must be updated.

The deleted input and output branches of a partial connection can be figured out from information supplied by the neighboring controllers. Thus the controller can initiate connection teardown if needed. In order to update the current traffic base, the controller needs traffic descriptor and QoS parameters of connections. This information may also be obtained from the neighbors.

Zombie connections are more difficult to deal with. Their existence can be identified but their traffic descriptors and QoS parameters may not be recoverable. Thus it is not easy to reclaim the resources allocated to zombie connections and update the current traffic base. There are several possible approaches to solve this problem.

The most accurate one is rebuilding the traffic base from scratch. Once the controller has all the information regarding the active connection a new traffic base can be built which, of course, would not include the resources previously allocated to the zombie connections. This operation is costly in terms of computation and time and may not be practical in certain circumstances since it delays the recovery process.

In another approach, a higher layer protocol is used. Such a protocol maintains the traffic descriptor and QoS information connections even after a connection terminates. The duration for which this information is maintained is slightly more than the time taken by a secondary controller to recover. This information is later supplied to the recovering controller on request.

Another, more practical alternative, is to record parameters of very large connections separately, in a recoverable fashion. In this way we guarantee that zombie connections would not tie up too many resources, and the recovery can be done fast. Thereafter, when the time and computation resources are available, a complete traffic base reconstruction can take place. To implement this capability we use a small storage in the switch in which the controller can store certain information (see the General command described in Section V).

The primitives in ONSCI protocol provide support needed by the controller from the switch in order to make the network tolerant to faults in the controller. With proper design of protocols at higher layer, and support from ONSCI it is possible to build a fault tolerant network.

V. Basic Operations

The ONSCI switch interface is modeled as an extension to the GSMP [14]. As is described in the previous sections the extensions made are in particular at the call model, QoS support and fault tolerance. We leverage as much as possible the GSMP formats.

In terms of call model, we consider the most general types of connections as described in Section III.C. In terms of QoS, we support a very general model in which the switch and the accommodation control mechanism may support calls with large number of QoS values. Each connection may request any traffic descriptor and QoS value. This is different from [14], in which only priorities (but no admission control) are supported and [26], where each class also identifies the (single) rate at which a connection may operate. It differs from [27] in that only a single control platform is considered. Our generic admission control model is derived from the PNNI standard [23], as OpeNet advertises the same parameters. However, the availability check can be easily modified to include additional parameters or different GCAC policies.

We divide the menu of all commands to several families, which we briefly describe in the following. Interested readers are referred to [13].

Resource Management

This group of commands deals with the management of the resources in the switch regarding their allocation and assignment to the individual connections. As such they are typically associated with specific ATM connections that specify certain QoS and traffic descriptors as per the definition of Section III.C. This is the only group that contains commands that are not idempotent (as mentioned in Section III.B). Therefore care must be taken to ensure that there is no inconsistency due to loss of a resource management message.

In order to achieve consistency, the controller queues a resource management request if response to an earlier resource management request has not arrived. The new request is sent only after response to all the earlier requests are obtained. If response is not received within stipulated time, the controller retransmits the request. This is essentially a stop-and-wait protocol. Thus, at any time, the controller has at most one outstanding unacknowledged resource management message.

The controller maintains a resource sequence number which is carried by every resource management request message and is incremented after a request is sent. The response to a request must have the same resource sequence number. The resource sequence number is used by the switch to identify duplicate requests due to retransmission. The switch maintains an expected resource sequence number, which is one greater than the resource sequence number of last successfully processed request. The switch also keeps a complete copy of the last response sent. If the switch receives a request with sequence number one less than the expected, it is a retransmission of the previous request. In this case the switch re-sends the last response which it has stored. In case the resource sequence number of a request is equal to the expected one, the switch processes the request as a new one and sends the response. The switch then stores the new response in place of the old one and increments the expected resource sequence number. If none of the above conditions are true, then the request message is ignored.

This group of commands includes the *new connection* command which is used to set up a new connection. The request includes a representation of a generalize VC table entry, delineating the ports, the VCIs and the associated traffic descriptors. The group also includes the *drop connection* command which is used to completely erase all traces of an existing connection and the *change* command which is used to alter the QoS parameters associated with the connection. Commands also exist for adding or dropping input or output branches to an existing connections thereby providing full support for a dynamic multicast environment.

The command *get CAC parameters* is used to get the switch dependent parameters such as VF, ACR and CRM. These parameters are used in the GCAC model and are used for estimating if a remote node can accommodate a call and are periodically advertised by a switch as described in Section III.E.

Configuration

This family of commands allows to receive configuration reports from of the switch or its ports. The *switch configuration command* reports the various global capabilities of the switch. This command would typically be invoked only during start-up of switch or controller, for example after recovery from crash. The *port configuration command* similarly reports the static configuration information of the specified port and is used when only a single port (rather than the entire switch) recover.

VP Management

VP management is an important part of ATM nodes, especially when used for telecommunications purposes. VPs have traditionally much longer lifetimes so it is not important to be able to manage every detail of the VP separately. consequently, we chose to manage the VP as a whole and provide primitives to establish and terminate VPs. The ONSCI supports a variety of VP types (unicast, multicast, etc.) each with its own allocated resources.

Port and Switch Management

These commands allow to manage dynamic behavior of the switch and its ports. The *port management* command allows a port to be brought up, taken down, brought in loopback mode etc. The *switch management* command is used to manage dynamic behavior of the switch and is similar to that of [14].

Events

This family of commands allows the handling of events that happen at the switch and have to be reported asynchronously by the switch to the controller such as link failure and recovery. While in general the ONSCI uses a master-slave approach in which the controller issues a request and the switch responds, here the switch initiates the report. This is the only family of commands operating in this direction. We have adopted here directly the model used in GSMP[14].

Fault Tolerance

After failure of a switch controller, either a standby controller gains the control of the switch and performs recovery, or the primary controller restarts from scratch and attempts to recover as discussed in Section IV. In either case, the fault tolerance commands provide the minimal state information at the switch, necessary for recovery.

An important observation to be is the amount of state information which is typically substantial. The controller must maintain state for each port and for each connection of which many thousands might be active at any time. There are several concerns here. First, the switch may not have enough available memory to assemble a message containing the entire information it must pass to the controller. Second, the amount of information might exceed the size of an AAL5 packet which means that the information cannot be transported in a single exchange. All this must be coupled with the approach that the switch must remain simple and the operations it performs should minimize the amount of computation.

Our approach is that of modularization. The controller first acquires configuration information: such as number of ports and list of active ports. Then the controller requests, for each port, the list of input VPs and output VPs (which is done with the *get input VP list* and *get output VP list* commands). Thereafter, for each port and each VP, the list of VCs is requested via the *get input VC* and *get output VC* commands. Lastly, given an input VCI. a VPI, and port number, the *get forwarding information* command reports a list of output VCs and port numbers to which cells from the specified input VC and port, are forwarded.

Modularizing the request reduces the amount of information that is reported for every request, but does not guaranteed that the response to every request can fit into a single message. A continuation mechanism is devised for that purpose. When a list is requested, such as a VC list, we assume that these are ordered by the switch in a certain, unspecified way (e.g., some random order in memory). When the switch responds with a partial response it provides the controller with an index (or handle) which, if included in a subsequent request of the controller, will cause a continuation response to be generated.

Beyond the state recovery commands mentioned above, ONSCI provides management of the control points. The *add control* command creates a new control port specified by input port, input VCI, output port and output VCI for the switch. The control port allows a standby controller to control the switch. In addition to responding to its earlier controller, the switch also responds to any request on the new control port. In other words, the switch processes all requests sent to the input port and VCI of a control port and sends the response on the output port and VCI of the control port. This command is typically issued during start-up when the primary controller sets up a backup controller for the switch. During the recovery phase the backup controller may use this command to add a new backup controller since it has become primary controller from backup. The *delete control* command removes a control port. This is typically done during recovery, to reclaim control port of the failed controller.

The *add command log* command redirects all the subsequent requests along with their responses to the specified output port and VCI. This output is monitored by a standby controller which constructs the state of the primary controller using the log of commands and their responses. This provides an alternative method for recovery after the failure of primary controller. The *delete control log* command is used when the redirection of requests and responses to a particular port and VCI is not needed. Like other commands for fault tolerance, both these commands are typically used either during start-up or during recovery phase.

Performance Monitoring

Upon request the switch will convey to its controller a variety of information regarding the dynamics of switch and the individual ports. These statistics can be obtained on a per port basis or on per VP basis or per VC basis.

General

A small number of commands are provided for other, miscellaneous operation. We assume the switch has a very small amount of memory which it makes available to the controller. The controller can store and retrieve information from this storage without the switch interpreting it. For example, the controller may store there certain time stamps that might be useful for a secondary controller after the primary failed or certain GCAC parameters.

It is well understood that an interface such as ONSCI cannot possibly be comprehensive, given the large variety of switches. It is also recognized that some switches can perform certain operation for which there are no ONSCI command available or which require several commands. For that purposes ONSCI defines the *private* command which uses the interface and message exchange mechanism to invoke and report the results of a privately defined operation.

VI. Discussion and Conclusion

We have proposed the OpeNet Switch Control Interface (ONSCI), a new open ATM switch control interface based upon principles of distributed systems. ONSCI is switch independent and supports a variety of primitives to allow many network control platforms to be implemented on top of it. We chose the ATM Forum’s traffic management model and the OpeNet signalling as an example.

Two innovative contributions are included in ONSCI: (1) support for a resource management scheme for the provision of general QoS that is both switch and signaling platform independent (2) support for a fault tolerant operation in the vein of increasing availability. Our design assumes limited switch capability in terms of the amount of memory and particularly CPU power. The idea behind this design is to allow simple (read: stable and reliable) processor as part of the switch.

One of the major problem in designing this interface was how to identify resources of the switch allocated to a particular connection. Imagine a situation when a branch is to be added to an existing multicast connection. In order to compute the amount of additional resources needed for this purpose, the resources already allocated (buffers, bandwidth, etc.) must be known. The CPU of the switch performs only very simple operations and is incapable to storing this resource mapping. The controller cannot store this information either as the amount and structure of switch resources is switch specific which cannot be generalized.

Resource identification has other benefits also. If the switch resources allocated to a particular connection were identifiable, all the commands in the resource management group could have been made idempotent, simplifying the protocol. The problem of reclaiming resources from zombie connections would have become much easier.

Explicit resource identification requires enormous bookkeeping in the switch (and a more complicated switch model). It also needs significant software on the switch. This conflicts with our original goal of decoupling software and hardware components of a traditional ATM switch, distributed implementation of these two components, and open switch control interface. Despite the advantages of explicit resource identification, we decided not to opt for it. The trade-off in favor of simple switch software is desirable.

We solved the resource identification issues in a variety of ways. In terms of the resource management commands, we chose to implement those in a more reliable manner which includes specific ACKs for each request. In terms of resource searching, we require that all messages that refer to resources should carry enough information to allow the switch to infer, rather than compute or search, the resources involved (e.g., including all I/O designators in identifying resources of a multicast connection). This in addition to the assumption that resource accommodation is monotonic and cumulative.

The inability to identify resources impacts the recovery process after failures. To that end we provided primitives that will allow to reconstruct the state of a failed controller, considering the fact that a large amount of information can be acquired from neighboring controllers.

ONSCI was implemented and integrated with the OpeNet control platform. Pure ONSCI implementation on an ATM switch was not available, but we had access to GSMP-capable ATM switch. We wrote a thin layer of software which converts ONSCI message to GSMP messages and GSMP response to ONSCI response. Using this basic functionality of ONSCI, like connection management, statistics etc., were successfully tested.

VII. Acknowledgments

The authors thank Israel Cidon and Moshe Sidi for their valuable comments and suggestions while designing the protocol. Thanks to Amit Gupta who provided much needed help in design as well as implementation phase.

VIII. References

[10] IEEE, *Proposed IEEE Standard for Application Programming Interfaces for Networks*. http://www.iss.nus.sg/IEEEPIN/

[11] I. Cidon, T. Hsiao, A. Khamisy, A. Parekh, R. Rom, and M Sidi, *An Open and Efficient Control Platform for ATM Networks*, (January, 1996). Sun Microsystems Laboratories

[12] I. Cidon, T. Hsiao, P. Jujjavarapu, A. Khamisy, A. Parekh, R. Rom, and M Sidi, *The OpeNet Architecture,* Sun Microsystems Laboratories, Mountain View, CA, USA (December 1995). http://www.sunlabs.com/technical-reports/1995/smli_tr-95-37.ps

[13] R. Greg and R. Rom, *The OpeNet Switch Control Interface Specification*, Sun Microsystems Laboratories, Mountain View, CA, USA (November 1997).

[14] P. Newman, W. Edwards, R. Hinden, E. Hoffman, F. Ching Liaw, T. Lyon, and G. Minshall, “Ipsilon’s General Switch Management Protocol,” RFC 1987 (August 1996).

[15] P. Newman, W. Edwards, R. Hinden, E. Hoffman, F. Ching Liaw, T. Lyon, and G. Minshall, “Ipsilon’s General Switch Management Protocol Specification version 2.0,” RFC 2297 (March 1998).

[16] C. M. Adam, A. A. Lazar, and M. Nandikesan, *QOS Extensions to GSMP,* Center for Telecommunications Research, Columbia University, New York (April 1997).

[17] Joe Evans, “ATMF Extensions to GSMP,” in *Proceedings of OPENSIG Fall’97 Workshop: Open Signalling for ATM, Internet and Mobile Networks,* Columbia University, New York, NY (October 1997).

[18] Newman, P., Minshall, G., and Lyon, T.L., “IP switching-ATM under IP,” *IEEE/ACM Transactions on Networking,* **6**(2) pp. 117-29 (April 1998).

[19] S. Floyd and V. Jacobson, “Link-sharing and Resource Management Models for Packet Networks,” *IEEE/ACM Transactions on Networking,* **3**(4) pp. 365-386 (August 1995).

[20] Zhang, L., Deering, S., Estrin, D., and Shenker, S. et. al., “RSVP: a new resource ReSerVation Protocol,” *IEEE Network*, **7**(5) pp. 8-18 (September 1993).

[21] A. A. Lazar, Bhonsle, S., and Lim, K.S., “A Binding Architecture for Multimedia Networks,” *Journal of Parallel and Distributed Systems,* **30**(2) pp. 204-216 (November 1995).

[22] J. Hyman, A. Lazar, and C. Pacifici, “Real-Time Scheduling with Quality of Service Constraints,” *IEEE Journal on Selected Areas of Communications*, pp. 1052-1063 (September 1991).

[23] The ATM Forum, *Private Network-Network Specification Interface v1.0 (PNNI 1.0),* The ATM Forum (March 1996).

[24] L. Georgiadis, R. Guerin, V. Peris, and K. Sivarajan, “Efficient QoS Provisioning based on Per Node Traffic Shaping,” *IEEE/ACM Transactions on Networking,* **4**(4) pp. 482-501 (August 1996).

[25] A. Elwalid and D. Mitra, “Effective Bandwidth of General Markovian Traffic and Admission Control of High Speed Networks,” *IEEE/ACM Transactions on Networking,* **1** pp. 329-343 (1993).

[26] A. A. Lazar and Franco Marconici, *Towards an API for ATM Switch Control,* Center for Telecommunications Research, Columbia University (April 1996). http://www.ctr.columbia.edu/comet/xbind/xbind.html

[27] K. Van Der Merwe and Ian Leslie, *Switches and Dynamic Virtual ATM Networks,* (July 1996). Computer Laboratory, University of Cambridge