Software Switch Extensions for Portable Deployment of Traffic Control Algorithms

G. Kormentzas[1], and K. Kontovasilis[1]

[1] National Center for Scientific Research "DEMOKRITOS",
Institute for Informatics & Telecommunications,
GR-15310 AG. PARASKEVI, POB 60228, GREECE
e-mail: {gkorm, kkont}@iit.demokritos.gr
Tel: +301 650 3167
Fax: +301 653 2175

Abstract. ATM networks employ traffic control as an important mechanism for assuring QoS levels, while also achieving economical resource usage. Current ATM networking equipment incorporates rather elementary Traffic Control Algorithms (TCAs), hardwired into the software controlling the devices. This arrangement does not allow upgrading to more advanced control schemes, as these become available. Considering this problem, the paper proposes a software infrastructure for portably and transparently embedding advanced traffic control functionality in ATM switches. In the proposed infrastructure, there are abstract software entities and programmable interfaces between entities, both complied with the emerged P1520 reference model. Given that current signaling standards do not support externally defined traffic control functionality, the paper discusses in detail an appropriate for the presented infrastructure signaling protocol, called Virtual Signaling Protocol (VSP). The examination of VSP messages that run through the L interface (of the P1520 model) constitutes one of the main objectives of this paper. The functionality of the presented infrastructure (as well as VSP) is tested by means of an implemented prototype system.

1. Introduction

ATM, a particularly pervasive BISDN technology provides explicitly and by design the foundations for deploying Traffic Control Algorithms (TCAs). These algorithms are necessary for balancing the tradeoff between efficient utilization of network resources and assurance of particular QoS (Quality of Service) levels required by the users.

Given the significance of traffic control, current ATM switches come equipped with a suite of TCAs (such as routing, Call Admission Control (CAC), resource configuration/allocation, etc.), specified and implemented by the manufacturer. In most cases, internal details of the algorithms are not released and access to the switch control software is not provided. Even in those cases where some access is allowed through an API (Application Programming Interface), it is usually limited to configuration tasks and does not allow modifications to the core of TCAs.

S. Rao and K.I. Sletta (Eds.): INTERWORKING 2000, LNCS 1938, pp. 264-277, 2000.

As long as ATM switches are closed boxes that execute a restricted set of manufacturer TCAs, it is difficult for network operators and/or value-added service providers to: (a) dynamically reprogram ATM switches in order to, e.g., extend their functionality and (b) implement customized traffic control policies and/or protocols. Addressing the inflexible architecture of current ATM switches, the paper proposes an open software infrastructure in which specific TCAs can be portably installed and transparently instantiated on demand. With this architecture, the network operators/service providers of a shared ATM network infrastructure:

a) are freed from manufacturer-dependent traffic control mechanisms,
b) can develop signaling protocols and control programs richer than standard ones,
c) can quickly introduce new (advanced) TCAs to support QoS and
d) can implement different traffic control policies according to their particular needs and/or purposes.

The software infrastructure presented herein includes abstract software entities and generic interfaces between entities, both complying with the P1520 reference model (see Figure 1) [1]. The said model has been developed in the context of the IEEE P1520 standards development project [2], with the aim of standardizing software abstractions of network resources and APIs for manipulation of these resources. The P1520 reference model follows a layered view of a programmable network[1] that is similar to the abstraction of an operating system in a personal computer [4]. An operating system provides programming interfaces for applications to make use of the physical resources of the computer. With an analogous software layer over a programmable network, different TCAs (including customized signaling protocols) can operate on the same network.

The rest of the paper is organized as follows: Section 2 presents the proposed flexible and extensible software architecture through which drastic changes in the traffic control functionality of an ATM switching system can be achieved. The next Section

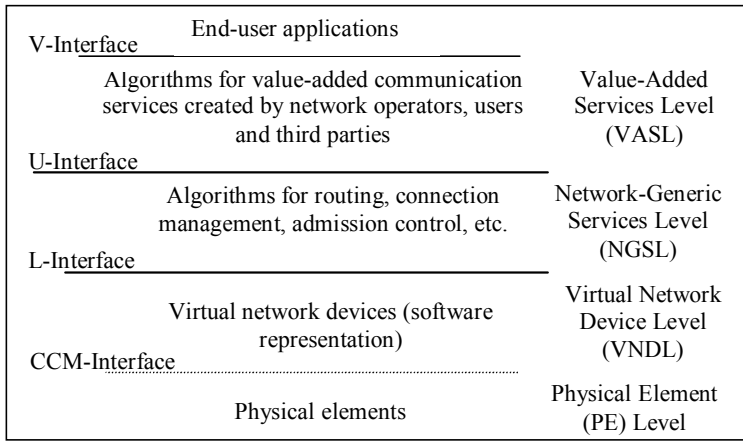

Figure 1. The P1520 reference model

[1] According to [3], programmable calls a network of which the functionality can be extended by dynamically installing software extensions on the networking nodes.

3 discusses, in the form of a set of messages, a Virtual Signaling Protocol (VSP) suitable for the proposed architecture. A prototype system that implements the abovementioned software infrastructure is presented in Section 4. Lastly, Section 5 concludes the paper, outlining also some plans for future exploitation.

2. A Software Infrastructure for Portably and Transparently Embedding Traffic Control Functionality in ATM Switches

In this section, we propose a software infrastructure through which the control plane functionality of an ATM switch can be extended.

Figure 2 presents the proposed open infrastructure for portably and transparently embedding TCAs in an ATM switch. As it may be seen in this figure, the heart of the proposed software infrastructure constitutes an *abstract information model* (called Switch-Independent Management Information Base (SI-MIB)) that is placed on top of the target ATM switching system. The SI-MIB provides a virtual environment for portably deploying TCAs by separating the functioning of these algorithms from the details of the switch hardware and low-level management/control software. This device independence is achieved through generic traffic control constructs (in other words, logical switch-independent objects appropriate for traffic control schemes) that constitute software abstractions of the resources and the traffic load conditions within the corresponding ATM switch. (An in-depth description of the SI-MIB can be found elsewhere [5].)

Generally speaking, an abstract information model: (a) has the ability to semantically represent the different MIB trees of various controlled/managed with a small yet comprehensive number of constructs, and (b) allows for high-level, device- and manufacturer-independent control/management actions [6].

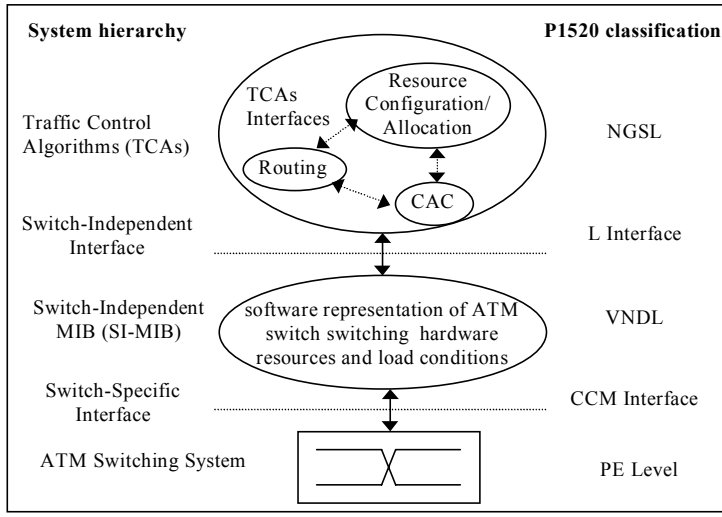

Figure 2. Software switch extensions for portable deployment of TCAS

Returning to the proposed open control architecture, in terms of the P1520 initiative, the TCAs belong to the Network Generic Service Level (NGSL), the switching hardware and low-level software at the Physical Element (PE) level, while the SI-MIB standing in between, corresponds to the Virtual Network Device Level (VNDL).

The software entities implementing a TCA communicate with the SI-MIB at a virtual level through a switch-independent interface, called an L-Interface, in terms of P1520. The L interface is programmable and allows a TCA to access the abstract software version of the switch representation within the SI-MIB. Since this interface deals only with the generic traffic control constructs of the SI-MIB, different TCAs can be applied, without requiring any additional effort for customizing the L interface to the functioning of the particular algorithm. Moreover, as the L interface is uncoupled from the equipment details, the abstract software implementing a particular TCA can be used as is on switches of different manufacturers.

Matching the real status of the switch to the abstract counterpart maintained within the SI-MIB is achieved through a switch-specific interface (a CCM-Interface, in terms of P1520). The CCM interface is *not* programmable, but a collection of switch-specific protocols or software drivers that enable exchange of state (and management/control information in general) at a very low level between a switch and the respective SI-MIB.

Given the proposed open architecture, the exchange of control information between switches can not take place through control paths/channels established by fixed standards signaling protocols, as these protocols do not support externally defined traffic control functionality. Therefore, it is necessary an open signaling protocol, which will ensure that the enhanced control functionality supported by the presented software infrastructure, can be communicated between different switches. Addressing this issue, the next section presents a signaling protocol appropriate for the proposed architecture.

3. A Virtual Signaling Protocol

As mentioned in the previous Section 2, the proposed software architecture overriding the standards-based signaling protocols requires the definition of a customized one. Towards this direction, this section discusses in detail how the software entities of the proposed infrastructure handle a call request/call release in the context of a Virtual Signaling Protocol (VSP) suitable for this infrastructure.

3.1 Call Set-up

When a user requests a new call, one of the following things may happen:

1. All CAC modules involved to the new connection(s) of the call, as well as the destination accept the call request, thus the new connection(s) is(are) set-up successfully;

2. One of the involved CAC modules rejects the new call, then an alternative (if it exists) route is examined and only if all routes fail, the new call is rejected;
3. The destination rejects the call request and a posteriori the new call is rejected.

For the first case (i.e., an accepted call request), the interactions between the end terminals (Source and Destination respectively) and the routing modules, CAC modules and SI-MIBs that are placed on top of the switches are depicted in Figure 3. (The number associated with each message in Figure 3 indicates the message's relevant position in a chronological sequence.)

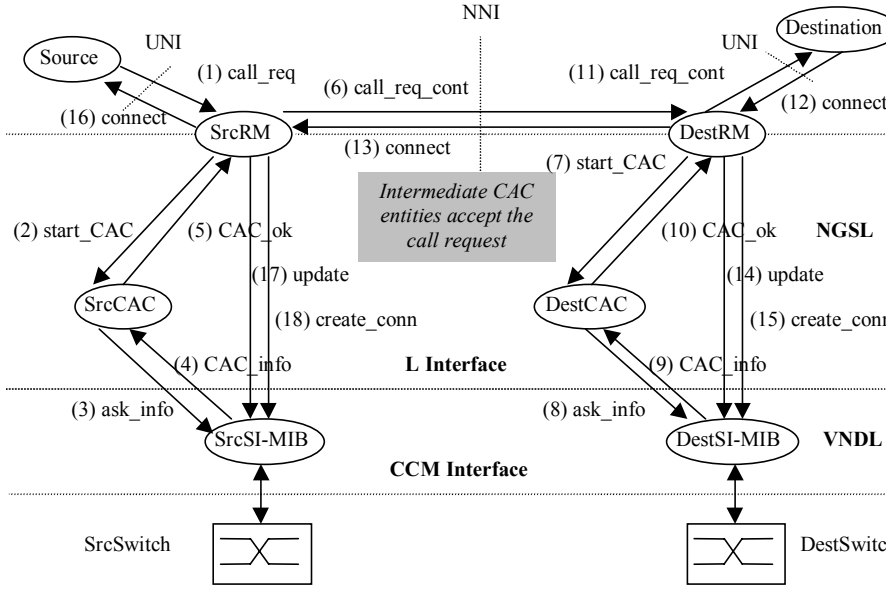

Figure 3. Interactions between the proposed software entities in case of an accepted call request

Elaborating on Figure 3, a user (Source) sends a message `call_req` to the source routing module (SrcRM). This message includes as parameters the addresses of the Source-Destination pair (parameter *S-D*), an indication of the desired QoS level (parameter *QoS*), in terms of delay or cell loss, that the new call may tolerate, a set of parameters describing the traffic profile for this call (parameter *TrProf*) and a unique call reference identifier (parameter *RefNum*).

Based on the parameters *S-D* and *QoS*, the SrcRM finds an appropriate route for the new call. Specifically, the SrcRM defines the relevant output port (parameter *portOut*) of the source switch (SrcSwitch) and the next switch (parameter *NextSw*). The routing in the rest of switches is based on a unique route number (parameter *routeNum*) identifying the selected route. Additionally, the SrcRM reserves an input and an output VCI (parameters *VCIn* and *VCIout* respectively) in order to be used for the establishment of the appropriate VC crossconnection in case of call acceptance.

Subsequently, the SrcRM sends a message `start_CAC` to the source CAC module (SrcCAC) in order to initialize a CAC process. This message contains the parameters *portOut*, *QoS*, *TrProf* and *RefNum*. The SrcRM uses the first two of these parameters in a message `ask_info` in order to retrieve from the source SI-MIB

(SrcSI-MIB), through a message `CAC_info`, appropriate information for the application of a CAC scheme. This information consists in available resources (parameter *res*) and the load conditions (parameter *bgTraffic*) within the (specified by the parameters *portOut* and *QoS*) Virtual Multiplexing Unit (VMU)[2] of the source switch (SrcSwitch).

Using the information of the parameters *QoS*, *TrProf*, *res* and *bgTraffic*, the SrcCAC performs a CAC scheme in order to accept or reject the new call request. Given that the SrcCAC accepts the call (see Figure 3), it proceeds to compute the new value *updatedbgTraffic* of the parameter *bgTraffic*. Then, it informs the SrcRM for the acceptance of the new call request, through a message `CAC_ok` with parameters *RefNum* and *updatedbgTraffic*.

Receiving the message `CAC_ok`, the SrcRM continues the per hop CAC process in each one of the ATM switches of the selected route. In this context, it sends a message `call_req_cont` to the routing module of the next ATM switch. The said module operates as described above. The parameters of the message `call_req_cont` are the same as the message `call_req`, plus the parameters *routeNum* and *VCIout*. As mentioned, the parameter *VCIout* will be used in case of an accepted call for the appropriate call set-up.

According to the scenario depicted in Figure 3, the message `call_req_cont` (following the abovementioned per hop process) reaches at the destination routing module (DestRM). Again, the described CAC process is repeated and finally, given that the destination CAC module (DestCAC) accepts the new call request, a message `CAC_ok` returns from the DestCAC to the DestRM. Then, the DestRM (through the message `call_req_cont`) informs the destination for the new call request.

As the destination accepts the call request, it sends to the DestRM a message `connect` with the parameter *RefNum*. The DestRM transfers this message to the previous routing module, which passes it to its own previous routing module, etc. Eventually, the message `connect` reaches at the SrcRM and from there it goes to the source.

Besides passing a message `connect` to the previous routing module, each routing module initializes a message `update` (with parameters *portOut*, *Qos* and *updatedbgTraffic*) and a message `create_conn` (with parameters *PortIn*, *VPIin*, *VCIin*, *PortOut*, *VPIout* and *VCIout*). The first message updates the traffic load of the VMU (defined by the parameters *portOut* and *Qos*) in order to reflect the newly established call. The second message incurs the set-up of the crossconnection defined by the parameters of the message. At the final stage, the successful set-up of a new call is the result of the application of messages `update` from all the involved to the new call ATM switches.

We proceed now to examine the two cases where the new call request is rejected, either by the network, or by the destination.

At the fist case (see Figure 4a), the call request is rejected as there is no route where all the involved CAC modules accept the call. By comparing Figures 3 and 4(a), it is apparent their concurrence as long as the CAC modules of a selected (by the

[2] VMU is a virtual notion that may correspond to only a part of an actual ATM multiplexer (e.g., a VP equipped with part of the buffer and link capacity of some output port of an ATM switch).

SrcRM) route accept the new call request. But, when a CAC module (the DestCAC according to the scenario of Figure 4a) rejects the call request, there is a slight variation. Instead of message `CAC_ok`, the DestCAC sends to DestRM a message `CAC_reject` with parameter *RefNum*. Subsequently, the DestRM passes a message `call_reject` with parameter *RefNum* to the previous routing module, which transfers it to its own previous routing module, etc. Eventually, the message `call_reject` reaches at the SrcRM.

Receiving the message `call_reject` (or a message `CAC_reject` in case where the SrcCAC rejects the new call request), the SrcRM examines alternative routing paths. If it finds a route where all the involved CAC modules accept the call request (Figure 3), the call is accepted, otherwise (Figure 4a) it is rejected. The SrcRM informs the source for the rejection of the call request through the message `call_reject`.

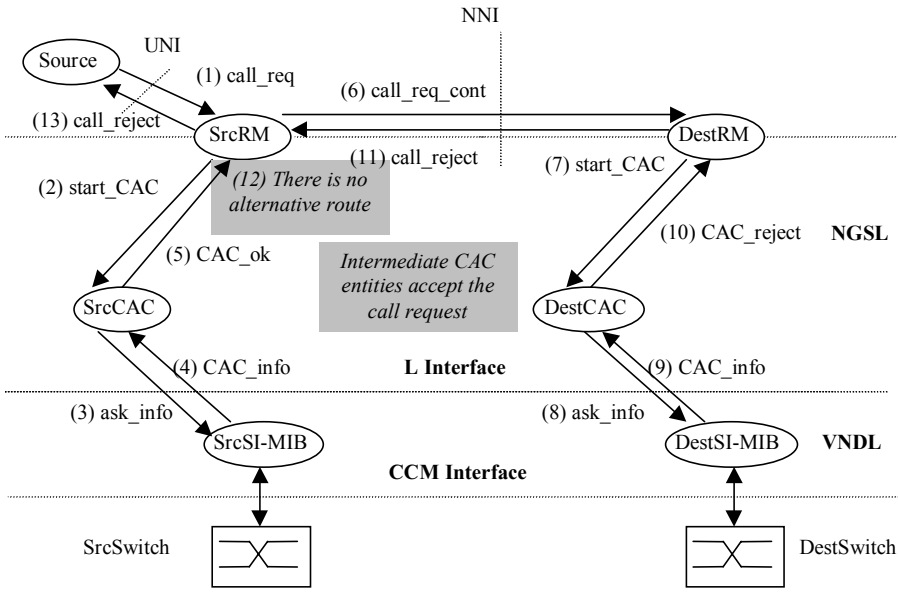

Figure 4a. Call rejection by the network

A different scenario for the rejection of a new call request is depicted in Figure 4b. According to this scenario, the call request for a selected (by the SrcRM) route is accepted from all the involved CAC modules. Thus, the DestRM sends to the destination a message `call_req_cont`. But, the destination, instead of a message `connect`, answers to DestRM with a message `call_dest_reject`. The parameter of this message is the *RefNum*. Receiving the message `call_dest_reject`, the DestRM passes it to its previous routing module, which transfers it to its own previous routing module, etc. Eventually, the message `call_dest_reject` reaches at the SrcRM, and from there it goes to the source of the new call request.

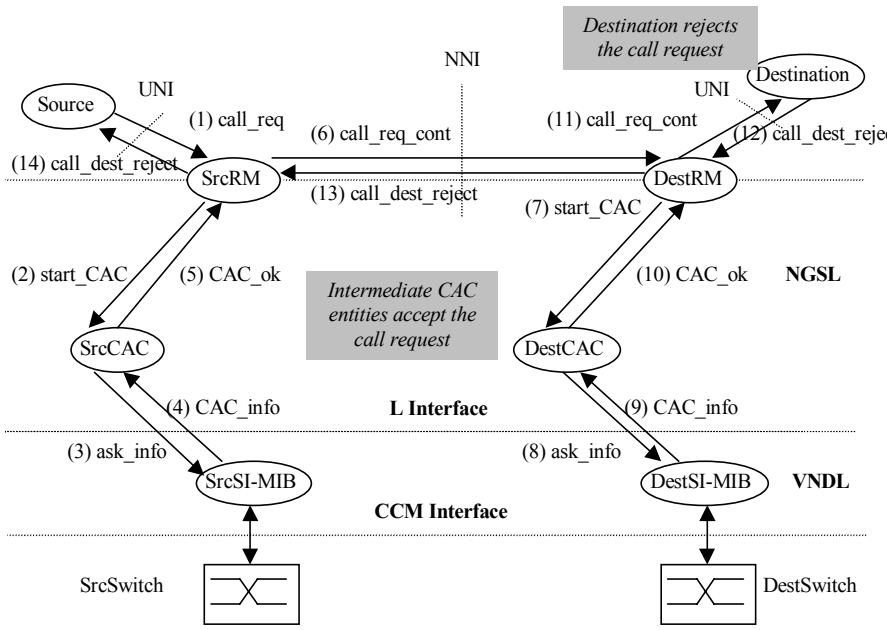

Figure 4b. Call rejection by the destination

3.2 Call Release

According to the call release scenario depicted in Figure 5, the call release is decided by the source. In this case, the source sends a message `call_release` (with parameter *RefNum*) to the SrcRM. The SrcRM passes this message (via the SrcCAC) to the SrcSI-MIB, in order to receive through a message `CAC_info_release` (with parameters *QoS*, *TrProf*, *res* and *bgTraffic*) the required information for the computation of the new value *updatedbgTraffic* of the parameter *bgTraffic*. After this computation, the SrcCAC sends a message `CAC_release` (with parameters *RefNum* and *updatedbgTraffic*) to the SrcRM.

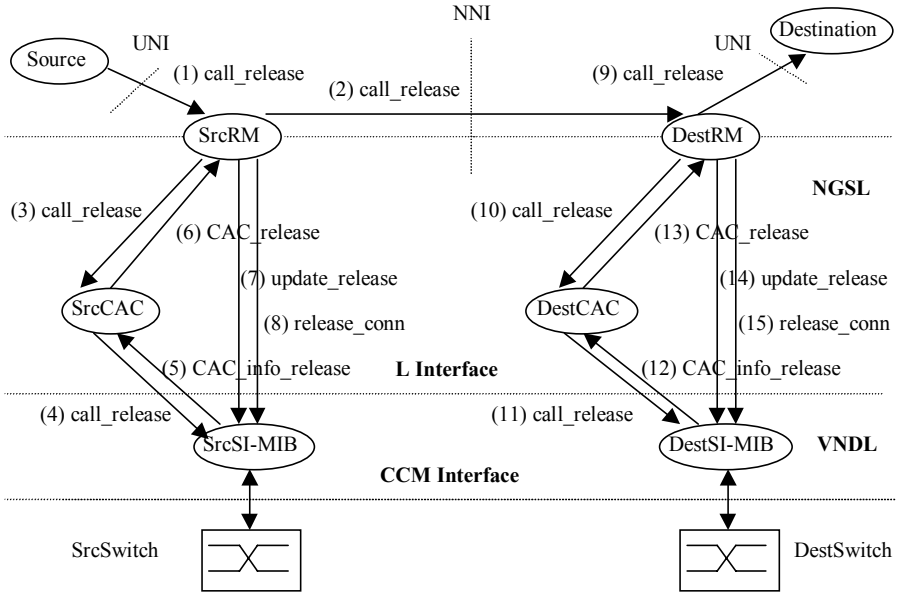

Figure 5. Call release

Receiving the message `CAC_release`, the SrcRM forwards a message `update_release` (with parameters *portOut*, *QoS* and *updatedbgTraffic*) to the SrcSI-MIB, in order to update the traffic load of the VMU defined by the parameters *portOut* and *Qos*. Additionally, the SrcRM sends a message `release_conn` (with parameters *PortIn*, *VPIin*, *VCIin*, *PortOut*, *VPIout* and *VCIout*) to the SrcSI-MIB, in order to incur the release of the crossconnection defined by the parameters of the message. As this process is repeated for all the switches that are involved to the route of the discussed call, the DestSI-MIB releases the last relevant VC crossconnection of the outgoing call.

By examining Figures 3, 4 and 5, it is apparent that some of the messages (e.g., ask_info, CAC_info, update, etc.) run through the L interface. Table 1 includes all messages that are relevant to the L interface. The set of these messages constitute (to the best knowledge of the authors) the first effort in the literature to employ the L interface, as specified in P1520, in a standardised and systematic way for the purposes of hardware independent traffic control.

Table 1. A proposal for applying the L interface structure to portable provision of traffic control

Message	Parameters	Short description
ask_info	portOut, Qos	Through this message NGSL asks from VNDL the traffic control information that corresponds to the VMU defined by the parameters of the message.
CAC_info	res, bgTraffic	By this message, VNDL returns the information requested from the message ask_info (network resources and traffic load conditions within the particular VMU).
update (update_release)	portOut, Qos, updatedbgTraffic	Through the update message NGSL updates the traffic load of the VMU defined by the parameters portOut and Qos.
create_conn	PortIn, VPIin, VCIin,PortOut, VPIout, VCIout	By the message create_conn, NGSL requests the set-up of the cross-connection defined by the parameters of the message.
call_release	RefNum	By the message call_release, NGSL asks from VNDL traffic control information related to the connection defined by the parameter RefNum.
CAC_info_release	Qos, TrProf, res, bgTraffic	Through this message VNDL returns the information requested from the message call_release (network resources, traffic load conditions and QoS of the VMU within traverse the connection defined by the parameter RefNum, as well as the traffic characteristics of the flow that produces the said connection).
Release_conn	PortIn,VPIin, VCIin,PortOut, VPIout, VCIout	By the message release_conn, NGSL requests the release of the cross-connection defined by the parameters of the message.

4. A Prototype Implementation

The section discusses the implementation of a prototype system that follows the general architectural framework of Sections 2 and 3 and makes use of intelligent software agents.

Although not mandated by the design, and for the purpose of facilitating the implementation process, the software entities of the prototype infrastructure have been built on top of an agent platform [7] that supports communication primitives according to the ontology specified by the FIPA organization [8]; therefore communication between the various distributed software entities in the prototype is achieved by means of an Agent Communication Language (ACL). The underlying agent platform used in the implementation was based on background work, conducted in the scope of the IMPACT[3] research project [9]. Again, this choice was made for achieving a faster development cycle, without being forced by design or other constraints.

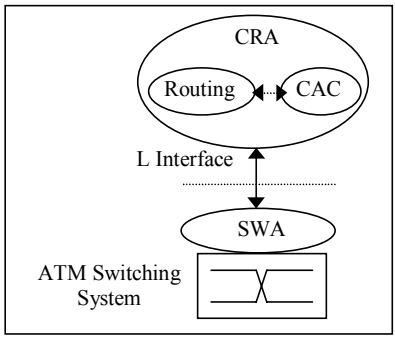

Figure 6. Structure of the implemented software infrastructure

Figure 6 outlines the structure of the implemented software infrastructure. As shown in this figure, the implemented infrastructure consists of distributed software entities that communicate through the facilities of the agent platform just mentioned. Two different types of agents may be identified: the *Switch Wrapper Agent* (SWA) that furnishes an abstract software environment for performing management and control operations on an ATM switch; and the *CAC & Routing Agent* (CRA), which is the software entity expressing in abstract and portable terms the actual traffic control algorithms. Comparing with Figure 2, the SWA corresponds to the VNDL of an ATM switching system, while the CRA plays the role of NGSL. The remaining of this section elaborates on implementation details for each of the two agent-types. Each topic is treated in an individual subsection. (An in depth description of the said agents can be found elsewhere [10].)

4.1 CAC & Routing Agent

The CRA encapsulates the core of traffic control functionality available to the implemented prototype. This type of agent should normally has been called “Traffic Control Agent”, but the name CRA was chosen to reflect the fact that this prototype implementation emphasizes CAC procedures and, necessarily, routing actions, which are intimately interwoven with CAC decisions.

The main function of the current CRA implementation is to receive call requests, to determine if they can be accepted through the corresponding switch (without violating the QoS constraint of both the incoming call and all other established connections) and, if yes, forward the call to the next node along the route being checked.

A CRA is implemented as multithreaded software. This is because CRA should be capable of handling/processing simultaneous incoming calls. Call requests arrive at

[3] The IMPACT project (Implementation of Agents for CAC on an ATM Testbed AC324) was funded by the European Commission under the Advanced Communications Technologies and Services (ACTS) Research Programme.

the CRA through an appropriate message of the agent communication language supported by the underlying agent platform. Upon arrival, the request is placed in a queue of incoming calls, which is served according to a FIFO policy from a thread named `CRAWorkerThread`. For each item removed from the queue, a new instance of a thread named `CallProcessThread` is spawned for processing the call, simultaneously with other pending requests.

The `CallProcessThread` invokes a routing module that selects an appropriate route for the call, this choice depending upon the source-destination pair and the desired degree of QoS. The current implementation supports a simple form of static routing, using fixed universal routing tables. Route selection occurs only at the first network node (the one at the UNI boundary); in subsequent nodes, identification of the selected route is through a corresponding 'label'.

Returning to the operation of CRA, once the call process request is picked by `CallProcessThread` for consideration, the appropriate output port of the corresponding switch is determined, and a software object, called `PortInfo`, is invoked for interrogating the port. (Each output port of the switch corresponds to a separate instance of this object.) `PortInfo` determines the appropriate VMU to check, by consulting the QoS specification of the call request. Information about the current status of this VMU (in terms of the traffic load conditions and the resources available) is maintained (in portable, abstract form) within the SI-MIB of the switch and `PortInfo` retrieves this information by communicating with the SWA. Then, on the basis of the VMU status, and following the rules of the CAC scheme, `PortInfo` decides to either accept or reject the call and relays this decision back to the `CallProcessThread`. If the call was accepted, the CRA signals the next switch along the selected route for the incoming call and the CAC procedure is continued in that next hop.

If, during this process, all switches along the route accept the call, then each involved CRA sends appropriate commands to the SWA attached to the corresponding switch, so that the SI-MIB will update its information to reflect the newly established connection. If, on the other hand, the call is rejected at some node, the fact is back-propagated to the originating node, where an alternate route is selected (if available) and the whole procedure is started over. If no alternate route may be found, the call is rejected by the network.

The above description makes clear that in the implemented prototype, a `CallProcessThread` corresponds to a routing module of Figures 3, 4 and 5, while a `PortInfo` plays the role of a CAC module of the said figures.

4.2 Switch Wrapper Agent

A Switch Wrapper Agent (SWA) is the software component that provides a virtual environment for the portable deployment of control and management operations on the underlying ATM switch. As noted in the previous subsection, during the course of a CAC procedure the SWA attached to an involved switch interacts with the CRA controlling this switch. Specifically, the CRA retrieves from the SWA information about the current status of the involved output port (actually, the appropriate VMU

within that port). Based on this information, the CRA performs the CAC check, and, in case of acceptance, informs the SWA, so that the latter will update the status of the switch to include the new connection. Furthermore, the SWA issues the appropriate low-level management commands to the switch for establishing the physical connection between the input and output ports involved.

The core components of a SWA are:

- SI-MIB, i.e., the switch-independent MIB providing an abstract virtual representation of the resources and traffic load conditions within the switch; and
- a library, consisting of a set of software drivers, for accessing the low-level management facilities of the ATM switch.

In the prototype, the low-level library has been implemented using the Java Management API (JMAPI). Currently there are vendor-specific versions for three models of ATM switches, specifically: FORE ASX-200, CISCO LS1010 and Flextel WSS 1200. Porting the library to other devices is easy, since the task involves only the mapping of the abstract objects in SI-MIB to the counterparts in the specific MIBs of the target switch.

5. Conclusions

Following recent proposals for programmable network infrastructures, the paper proposes a flexible and extensible architecture for portably and transparently embedding traffic control functionality in ATM switches. The proposed architecture includes abstract software entities and generic interfaces between entities, both complying with the P1520 reference model.

The paper focuses on the functionality of the L interface. In this context, it discusses how the presented open architecture (consisting of a software infrastructure and a virtual signaling protocol) provides (to the best knowledge of the authors) a first effort in the literature for applying the L interface structure to portable provision of traffic control.

It should be noted that, although the software infrastructure presented in this paper is tailored to ATM networking equipment, it is fairly general and can be extended for covering network nodes of a different technology (provided that this technology supports the potential for QoS concept and at least some notion of “connection” or, more generally, “traffic flow”). This extensibility is highly desirable, since ATM is not regarded anymore as the sole QoS-aware networking technology. (For recent efforts towards injecting QoS capabilities over IP-based networks, see e.g., [11].)

We plan to extend our work in the directions of applying the presented architecture to non-ATM networking technologies (for a programmable IP router architecture supporting control plane extensibility, see e.g., [12]), performance evaluation and security issues.

References

1. J. Biswas, A. Lazar, S. Mahjoub, L.-F. Pau, M. Suzuki, S. Torstensson, W. Wang and S. Weinstein, The IEEE P1520 Standards Initiative for Programmable Network Interfaces , *IEEE Communications Magazine*, pp. 64-70, October 1998.
2. Information electronically available from `http://www.ieee-pin.org/` and `http://stdsbbs.ieee.org/groups/index.html`.
3. D. Tennenhouse and D. Wetherall,, Towards an Active Network , *Computer Communications Review*, Vol. 26, No.2, pp. 5-18, April 1996.
4. T. Chen, Evolution to the Programmable Internet , *IEEE Communications Magazine*, pp. 124-128, March 2000.
5. G. Kormentzas, J. Soldatos, E. Vayias, K. Kontovasilis, and N. Mitrou, An ATM Switch-Independent MIB for Portable Deployment of Traffic Control Algorithms , In Proc. *7th COMCON Conference*, July 1999, Athens, Greece.
6. K. Kontovasilis, G. Kormentzas, N. Mitrou, J. Soldatos, and E. Vayias, A Framework for Designing ATM Management Systems by way of Abstract Information Models and Distributed Object Architectures , to appear in *Computer Communications*, 2000.
7. J. Bigham, L.G. Cuthbert, A.L.G. Hayzelden, Z. Luo and H. Almiladi, "Agent Interaction for Network Resource Management". In Proc. *Intelligence in Services and Networks '99 (IS&N99) Conference*, Barcelona, April 1999.
8. Foundation for Intelligent Physical Agents, *FIPA 97 Specification, Part 2: Agent Communication Language (ACL)*, 1997. Electronically available from `http://www.fipa.org/` or `http://drogo.cselt.stet.it/fipa`.
9. MPACT AC324, Technical Annex, March 1998. See also `http://www.acts-impact.org/`.
10. J.Soldatos, G. Kormentzas, E.Vayias, K.Kontovassilis, N.Mitrou, "An Intelligent Agents-Based Prototype Implementation of an Open Platform Supporting Portable Deployment of Traffic Control Algorithms in ATM networks", In Proc. of the *7th COMCON Conference*, July '99, Athens, Greece.
11. X. Xiao and L.M. Ni, Internet QoS: A Big Picture , *IEEE Network Magazine,* Vol.13, No 2, March/April 1999.
12. J. Gao, P. Steenkiste, E. Takahashi, and A. Fisher, A Programmable Router Architecture Supporting Control Plane Extensibility , *IEEE Communications Magazine*, pp. 152-159, March 2000.