Network Control as a Distributed Object Application

Huw Oliver, Søren Brandt, Andrew Thomas, Nathalie Charton*
Intelligent Network Platform Department
HP Laboratories Bristol
HPL-97-17
January, 1997

E-Mail: [heo,sbb,dat]@hplb.hpl.hp.com
           Nathalie.Perdigues@aar.alcatel-alsthom.fr

broadband network control, ATM, connection management, distributed processing environment, CORBA

This investigation into Network Control as a Distributed Object Application was undertaken as part of the ACTS project ReTINA (AC048).

The goal of ReTINA is to develop and demonstrate an industrial strength open distributed processing environment (DPE). This DPE will support distributed real-time multimedia applications over emerging broadband networks.

In terms of technology, the project will deliver a CORBA-compliant distributed object-oriented DPE with real-time and quality of service management capabilities.

We describe our experiences and performance figures from broadband (ATM) connection management using a distributed (CORBA based) control system. The control system described is the lowest layer of a general connection management architecture such as TINA or Xbind.

Internal Accession Date Only

*Alcatel Aslthom Recherche, Paris, France

Network Control as a Distributed Object Application.

Huw Oliver, Søren Brandt Andrew Thomas and Nathalie Charton
(heo@hplb.hpl.hp.com, sbb@hplb.hpl.hp.com, dat@hplb.hpl.hp.com and
Nathalie.Perdigues@aar.alcatel-alsthom.fr)

Hewlett-Packard Laboratories, Filton Road, Stoke Gifford, Bristol. BS12 6QZ.
Alcatel Alsthom Recherche, Route de Nozay, 91460 Marcoussis, Paris, France.

1. Introduction

The following work was undertaken as part of the ACTS Project ReTINA (AC048).

The goal of ReTINA is to develop and demonstrate an industrial strength open distributed processing environment (DPE). This DPE will support distributed real-time multimedia applications over emerging broadband networks.

In terms of technology, the project will deliver a CORBA-compliant distributed object-based DPE with real-time and quality of service management capabilities.

We describe our experiences and performance figures from broadband (ATM) connection management using a distributed (CORBA based) control system. The control system described is the lowest layer of a general connection management architecture such as TINA [5] or Xbind [4].

1.1 ATM/ISDN Forum style signaling

The TINA or Xbind architectures are an attempt to improve upon the current approaches to Connectivity Services in Broadband ISDN/ATM.

Existing work on Connectivity Services in Broadband ISDN/ATM standards bodies has concentrated on traditional signalling techniques. These come from the telephony world. For example, the ATM standard for signalling (SVC) is now Q.2931 which is based on the narrowband ISDN (digital telephony) Q.931 standard

———— NNI Signalling
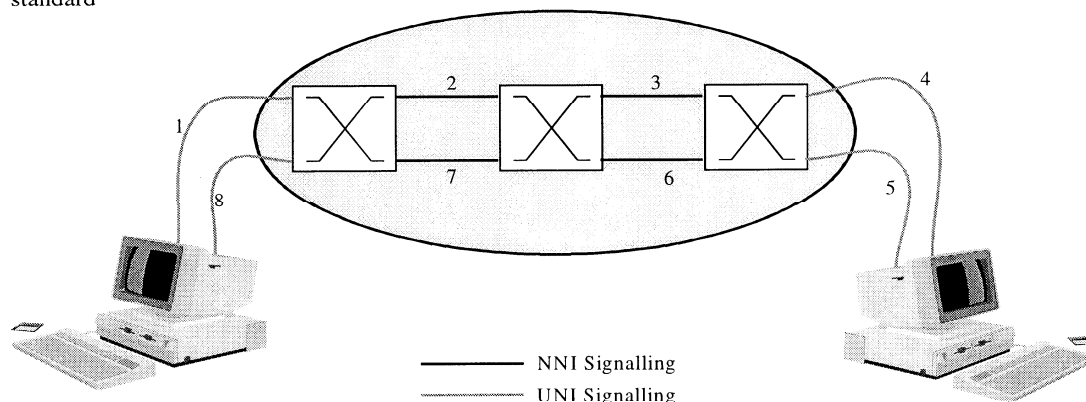
⋯⋯⋯⋯ UNI Signalling

While there are suggested enhancements to Q.2931 still under study, there are several outstanding problems or, more accurately, limitations to this signalling paradigm that will greatly compromise the potential success of the new public networks.

The limitations with this model can be summarized as

1. They are geared to running a single connection within a single call. We need much more flexibility. We need complex topologies (and we need some notion of a session to encompass these multi-party connections).
2. They assume calls last for minutes and that set-up latencies are correspondingly non-critical. Now we want near instant set-up times. An example is a user switching between video channels.
3. The implementations of UNI signalling have too large a footprint. All code needed to run the protocol has to be built into the terminal. The switch also needs to run the entire protocol leading to a huge amount of code running on the switch. The code is also all mixed up with session level logic about what the end parties can do.
4. More than the code, the specification of the ATM-Forum UNI 3.0 requires 143 text book pages to define the connection management signaling. This is because it must define the delivery semantics and encoding at the bit level. An equivalent CORBA based interface for slightly more functionality requires 3 pages and is mainly self documenting. The expression "CORBA based" automatically defines the delivery semantics and encoding.

We have been motivated, therefore, to look at Connection Management approaches that overcome the above problems and that meet some new requirements. We can summarize the requirements as:

- complex topologies: we need the ability to establish connections of more complex topologies beyond simple point-to-point connections. We will want multipoint-to-multipoint connections. We will want to provide for connectivity of multiple types of media flow. We will also need higher level abstractions for constructs to describe these operations. Note that this is nowadays possible as the Customer Premises Equipment becomes at least as intelligent as the switching equipment.
- We want to separate the connectivity software from the switching hardware. The intention is to allow the access to connectivity services by a third party. The hope is to achieve service creation, deployment and management as easily as it currently is in the Personal Computer and Internet worlds.
  This separation should be achieved via an open Connection Management interface.
  Connection Management needs access to interfaces that can provide information about the state of the network elements and their current utilisation. Information about the network topology and current link utilisation facilitates the set-up process. To this end, each connection controller should maintain a view of the network to enable it to calculate a complete provisional route that then must be verified.
  This need for access to resource information also applies to call admission algorithms. They need knowledge of the state of the links in the net (i.e. the traffic load) in order to perform.
- Connection Management should provide for connectivity of multiple types of media 'flow'. Of these, certain traffic flows will need special handling. Connection Management needs to be able to respond to sender requests for special handling of real-time services. This is important to support applications that require some degree of consistent throughput, delay and/or jitter.
  Following from the network's need to deliver to a particular flow a quantitatively specified Quality of Service, it is usually necessary for the network to set aside certain resources such as a number of buffers for that flow. Thus we need to be able to create and maintain resource reservations on each network element along the target path.
- Fast Set-up: For many services the speed of call set-up and modification is critical. Examples are very demanding scenarios (including those, for example, when integrated televisions and computers appear). For these, the call set-up processing will be at least as demanding as the current call processing demands on telecommunications Switch Control Points. Other services, for example Video On Demand will have undemanding set-up latencies (10 seconds for a set-up will be acceptable).
- Management: Connection Management should integrate seamlessly with multimedia general network management covering configuration, performance, fault management, accounting and security.

There are many other requirements not detailed here such as scalability, interoperability, simplicity, monitoring, flow synchronization, ownership and joining mechanisms.

1.2 Connection Management Architectures

Connection management architectures such as TINA [5] and Xbind [4,6] separate connection management into two layers:

1. Link by link setup (bottom layer).
2. Routing (top layer).

Link by link setup is concerned with the creation of connection through single network elements such as ATM switches. Routing is concerned with the determination of a particular path through the network to effect a connection with certain characteristics (i.e. quality of service guarantees). Sections 1.2.1 and 1.2.2 provide short introductions to routing in TINA and Xbind.

1.2.1 Routing in TINA

In TINA, routing is achieved by the decomposition of trails into smaller requests to Connection Performers down a CP hierarchy as illustrated in Figure 1. The pattern is repeated until the switch agents in the fabric are directed to set up their part of the trail. The routing, then, is done as part of the recursive edge creation done by successive of Network Management Layer Connection Performers (NML-CP) hierarchy calls. No part of the network has a complete knowledge of the internal configuration of its subnetworks.
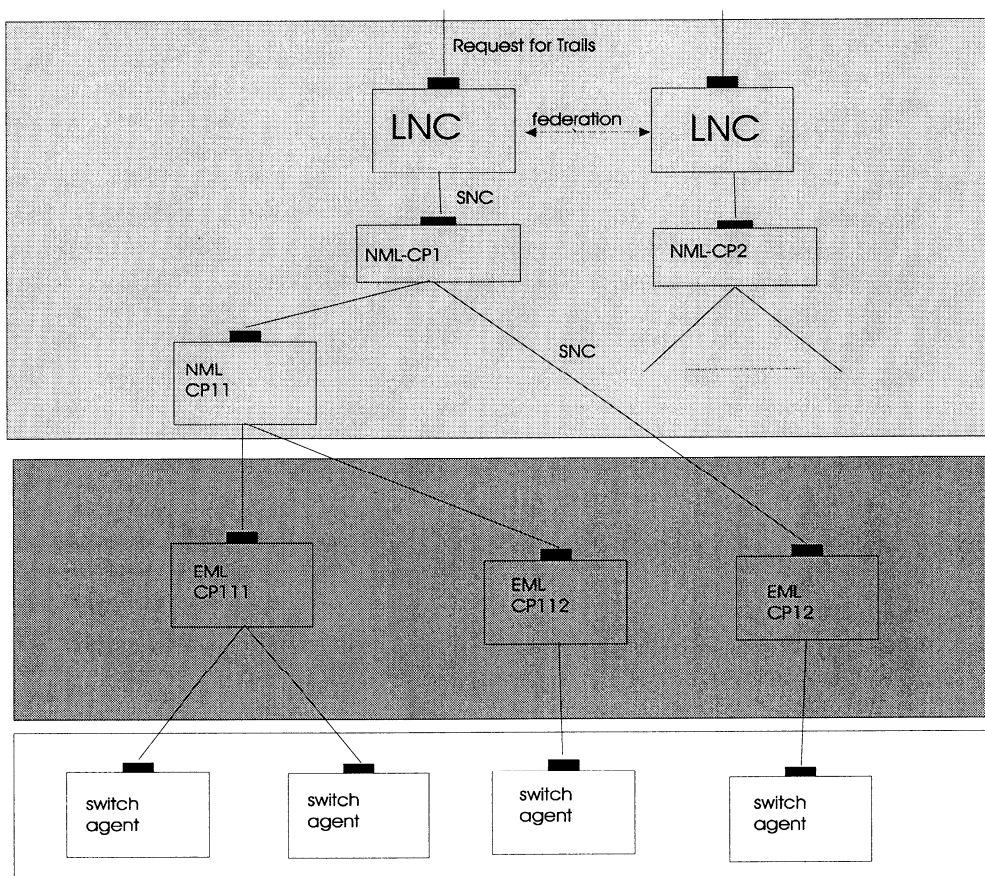

Figure 1:TINA Network Control Structure

3

The Layer Network[1] Co-ordinator (LNC) receives requests for connectivity in a layer network. It has federation capabilities with other LNCs of other domains in the network. Because these capabilities are immature and are the subject of further study in TINA, we have only looked at single domain layer networks.

Underneath the LNC there is a hierarchy of Network Management Layer Connection Performers (NML-CPs). Each NML-CP attempts to satisfy requests to create, modify and release connections within its own subnetwork. The NML-CP may pass these requests on to other NML-CPs in charge of various partitions of the subnetwork and this may continue recursively.

Ultimately the request will be passed to an object that has access to an agent on a real switch. These Element Management Layer Connection Performers (EML-CPs) attempt to satisfy requests to create, modify and release real connections between termination points of the subnetwork.

The critical interface, for this discussion, is the Subnetwork Connection (SNC) interface to the NML-CP. This is based on an abstract model of the network resources called the NRIM [12]. Essentially it is concerned with the creation, deletion and modification of edges. It is the specialization (in object-oriented terms) of this interface that is supported by the EML-CP.

1.2.2 Routing in Xbind

In Xbind, the goal is to make the network programmable by exposing the network state. The various binding algorithms (Xbind does not specify these) operate upon a set of interfaces known as the Binding Information Base (BIB). Xbind seeks to populate the BIB so that interesting algorithms can be built on top of the information that the BIB holds. Connection Management is then an algorithm which runs on top of the BIB.

The BIB offers a wide range of objects types: some deal with terminal capabilities, which might be used in constructing the Terminal Connection Session Manager (TCSM) of TINA; others deal with switching hardware and there are objects representing resources that are used to determine whether QoS constraints can be met.

With respect to network control, then, the Xbind BIB is currently populated with objects for controlling ATM switches. These objects allow the construction of QoS constrained physical links, algorithms external to the switch construct end-to-end connectivity out of the primitive services offered by the switch objects. This flexibility for example allows 3rd party connection set-up, leaf initiated joins and so on. Routing algorithms in Xbind are again applications running over the BIB.

In summary, with Xbind the programmer is free to implement any algorithm over the BIB. The connection management framework in TINA implies hierarchical routing. However this approach is not prescriptive for implementing connectivity in TINA. The programmer can implement any algorithm (hierarchical or not) assuming he/she respects the upper level interface of the connection management which is specified as TINA connectivity provider Reference Point (ConS-RP).

1.3 Open Switching

The key ingredient to support architectures such as TINA and Xbind is that each switch advertises its resources, making them accessible to algorithms running outside the switch. Third party providers can then develop hardware independent connectivity services where their software utilizes low-level open interfaces for switch control and configuration.

Figure 2 shows two terminals making and receiving requests to/from an off switch connection controller. The connection controller coordinates the set-up requests to the switches involved in the path.

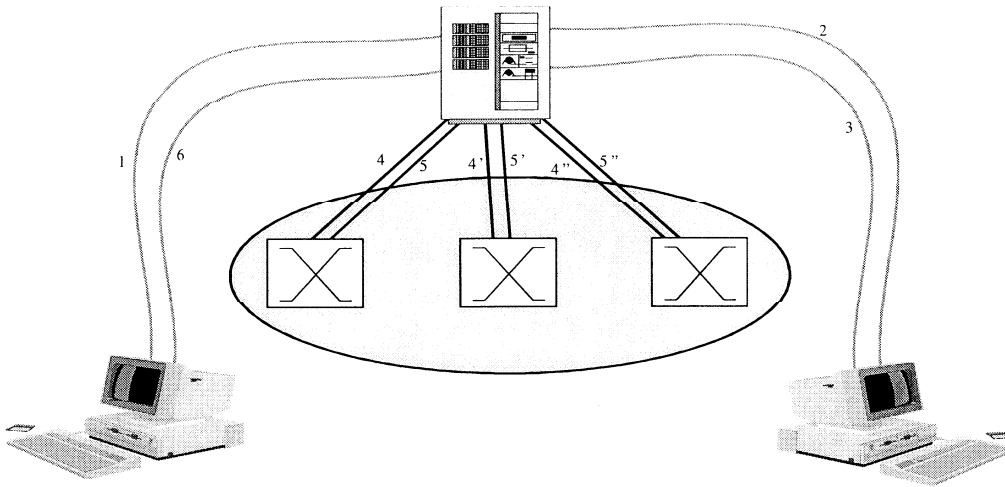[1] A Layer Network is one that consists of a single technology.

Figure 2: Message 1 is a call initiation request. 2 is a setup request to the end terminal. 3 is a call acceptance from the end terminal. 4 are resource allocation messages issued in parallel to network switches. 5 are returned acknowledgements. 6 is call accepted, connection ready.

We have experimented with several different ways of implementing the interface between the switches and the off-switch controller. Broadly, we have used a Fore ASX-100 switch (running SunOS) and two different controller platforms NT and HP-UX. We have been impressed by the use of the Java Virtual Machine on the different controllers to achieve platform portability and also the use of the JVM on the switch to build the server software. The use of 'Java everywhere' has allowed us to drop useful pieces of software onto the VM such as a Corba ORB. Without Java it would have been a significant piece of platform dependent work to embed an ORB on the switch.

We have experimented with different platforms for the controller (HP-UX and NT), using the same IIOP interface to the switch. Figure three shows two examples of the software pieces. The central block represents the switch while the right and left blocks show two alternative controllers. The IIOP interface corresponds to the protocol between the interfaces 4 and 5 in Figure 2:
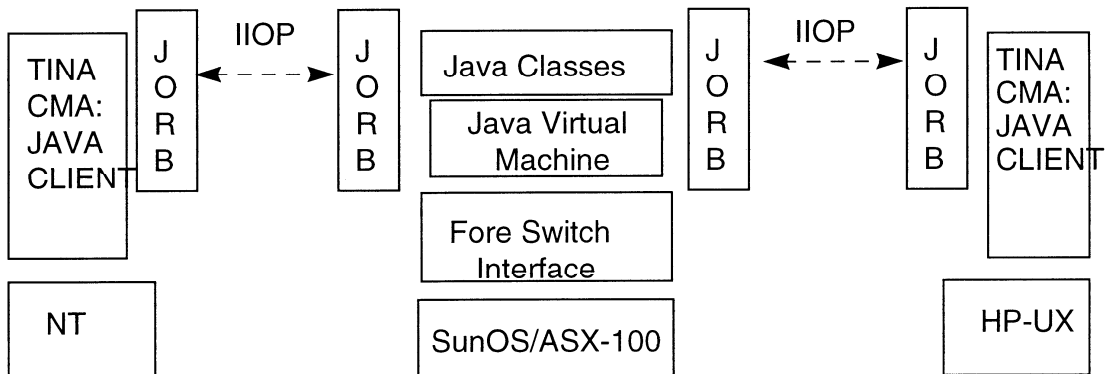

Figure 3: Java allows easy porting of the client software

2. Background Specific to our Experiment

One of the important goals of Retina is to provide a set of DPE services in order to support telecommunication applications. One of these DPE services is a TINA connectivity service. In the context of Retina (release 1), this service is supported by an implementation of TINA Connection Management provided by the Alcatel2 Alcin CM software.

The Alcin software has been deployed by Telenor as a central component of the Retina BVPN (Broadband Private Virtual Network) demonstrator [8]. The BVPN demonstrator investigates how to open up connection management in public networks, in order to allow third-party providers and customers to create "leased lines" on demand, without the need to contact the telecommunications operator.

The experiments described in following sections were prompted by the belief that use of SNMP (Simple Network Management Protocol) constitutes a performance bottleneck in the first BVPN version, which uses SNMP to do link by link setup. I.e., SNMP was suggested as a possible explanation to the inferior performance experienced by Telenor during BVPN experiments. In this paper, we therefore compare SNMP performance to the performance of an alternative switch interface developed at HP labs. The overall conclusion of these measurements is that SNMP performance can indeed be improved by an order of magnitude by means of an alternative switching interface. However, our measurements also suggest that SNMP performance is unlikely to be the primary BVPN bottleneck.

The physical setup used during performance measurements is illustrated in Figure 4.
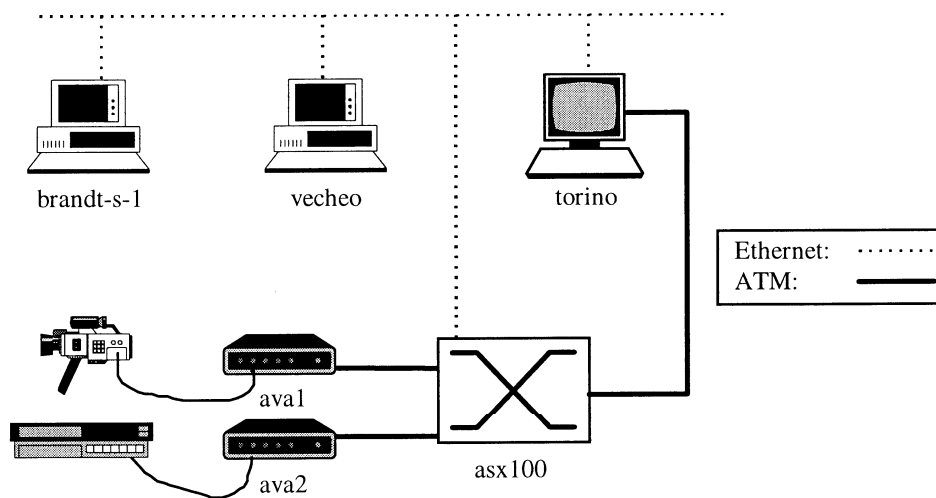

Figure 4: Experiment setup.

Connections through the ASX-100 ATM switch were created by means of either SNMP or our open switching interface. The latter will be described in more detail in Section 3.1. As illustrated in Figure 4, all workstations, including the switch itself, were connected by means of an Ethernet. The SNMP protocol and the open switching protocols both ran over the Ethernet, and did not make use of the ATM connections. Hence, the actual ATM connections were not used during the performance measurements described in the following section. However, to test that connections were actually created and deleted, we used two KNET AVA-200 boxes. The AVA boxes sample analog audio and video inputs, and feed ATM AAL5 streams onto the ATM network. The resulting streams were routed through the switch to software decoded video and audio sinks running on a HP-UX workstation (torino) equipped with an ATM card.

3. Performance Figures

This section describes a series of experiments and performance measurements carried out to assess the achievable performance of connection creation and deletion on an ATM switch. One of the comparisons we make in this section is between the use of SNMP (Simple Network Management Protocol) as a switch control interface and a more direct control interface developed at HP Labs. The overall conclusion of these measurements is that SNMP performance can indeed be improved by an order of magnitude by means of the alternative switching interface.

However, the performance measurements described in this section also provide input to other discussions, as listed below.

1. Performance of SNMP versus alternative link by link setup mechanisms.
2. Performance of different Java implementations.
3. Performance of C versus Java in I/O bound applications.
4. Performance of CORBA versus raw sockets.
5. Deployment of TINA CM computational objects.

Item 5 requires a bit of explanation: We saw in section 1.2.1 that the TINA CM framework consists of a number of computational objects, each described by CORBA IDL interfaces. At the bottom layer, so-called EML-CP (Element Management Layer Connection Performer) objects represent the physical network components, such as individual switches. However, the TINA framework does not specify where to actually deploy these objects, leaving this decision to actual implementations. At one extreme, all objects could be instantiated in a central computer, communicating with switches by means of whatever protocol is available, including SNMP. At the other extreme, an object representing a physical entity might run directly on the entity represented. For example, an EML-CP could run directly on the switch, providing a CORBA interface to the outside world. Performance measurements in this paper suggest that the latter solution, of running the EML-CP on the switch, will result in inferior performance, mostly due to the

7

performance of the switch hardware itself. The use of another switch with an up-to-date control station (e.g. a pentium) would probably lead to another conclusion.

3.1 An Alternative Switch Interface

As an alternative to SNMP, a "connection server" process was hosted on a Fore Systems ASX-100 ATM switch running SunOS 4.1.1. The server process allows clients on other hosts to create and delete connections by means of RPC calls. The interface of the connection server is shown below:

```
interface ConnectionServer {
    add_connection(in_port,in_vp,in_vc,out_port,out_vp,out_
vc);
    delete_connection(in_port,in_vp,in_vc,out_port,out_vp,o
ut_vc);
}
```

The connection server is implemented on top of the Vince [11] software, which provides an interface to the switch table by means of memory mapping. Hence, internal to the connection server, the creation or deletion of a connection amounts to a few memory accesses to update the memory mapped switch table. Since the amount of work done in the server to create a connection is therefore negligible, the performance of the RPC mechanism becomes crucial to the overall performance. To quantify this claim, we measured the processing time used internally in the server to create or delete a connection. When calls to add or delete a connection was done from inside the server process itself, on the order of 10,000 calls per second where achieved with the server implemented in Java. That is, the time spent on work proper inside the server is on the order of 0.1ms per connection created or deleted, and can be ignored as compared to communication overheads.

The negligible cost of work internal to the server meant that we could run the server in dummy mode during performance measurements. That is, the server did not actually create or delete connections, but simply returned success to the calling client. This allowed us to run the server on machines different from the switch itself, in order to compare switch performance to the performance of other workstations.

3.2 Switch Interface Test Configurations

A number of different Java implementations of the connection server was created, allowing comparison of several 1) Java implementations, 2) communication mechanisms, and 3) hardware platforms. It was possible to run Java programs on the switch by means of a port of the Java virtual machine to the switch, as was illustrated in Figure 3. Finally, an implementation of the connection server in C allows us to compare the performance of Java and C.

3.2.1 Hardware

The hardware platforms involved in performance measurements are listed in the following table.

Machine	Model
asx100	FORE Systems ASX-100 ATM switch
torino	HP PA 9000/755
brandt-s-1	HP Vectra Xu 6/200 (Pentium Pro)
vecheo	HP Vectra Xu 6/200 (Pentium Pro)

All machines, including the switch, were interconnected by an Ethernet LAN as was illustrated in Figure 1.

3.2.2 Languages

The connection server and its client was implemented in Java. To assess the performance differences between different Java implementations, several Java virtual machines were used, as listed below.

Virtual Machine	JDK version	Used on	Notes
JDK 1.01	1.01	asx100	SunOS port by HP Labs
JDK 1.01	1.01	torino	HP-UX 9.x port (alpha)
JDK 1.02	1.02	brandt-s-1/vecheo	Official Sun release
Kaffe 0.6.0	1.02	asx100	Public Domain [10]
Symantec Café 1.51	1.02	brandt-s-1/vecheo	Just-In-Time compiler

Furthermore, to compare Java to C, a version of the connection server was implemented in C.

3.2.3 Communication Mechanisms

Physically, network connections were provided by an Ethernet LAN. Logically, communication was done by means of two distinct mechanisms:

1. A CORBA/IIOP interface.
2. A raw interface based on TCP/IP sockets.

The connection server can be configured at start-up to provide either or both of the two interfaces, allowing clients to choose the means of communication. During measurements, the server was configured to only provide one of the two possible network interfaces at a time, although provision of multiple interfaces is unlikely to affect server performance.

Measurements on pure CORBA solutions suggested that CORBA is too heavy-weight for communication with the physical switch. This realisation prompted implementation of a raw socket interface for comparison. However, to be plugged directly into the TINA

framework, the server is required to provide a CORBA interface. Therefore, a hybrid solution was tested, using a CORBA proxy running on a fast workstation, and forwarding requests to the switch by means of the raw socket interface.

Two different ORBs were used, both written entirely in Java, and both providing a full CORBA 2.0 implementation:

1. JORB (Java ORB) from HP, in alpha 3 release.
2. OrbixWeb2.0 from Iona, providing the Java equivalent of Orbix 2.1.

3.3 Measurements

The following table lists performance measurements of Java based solutions.

	Network Interface	Client	Server	RPC/sec
1	JORB	torino/JDK 1.01	asx100/JDK 1.01	14
2	Socket	torino/JDK 1.01	asx100/JDK 1.01	106
3	JORB	torino/JDK 1.01	brandt-s-1/JDK 1.02	107
4	Socket	torino/JDK 1.01	brandt-s-1/JDK 1.02	597
5	JORB	brandt-s-1/Symantec	asx100/JDK 1.01	16
6	Socket	brandt-s-1/Symantec	asx100/JDK 1.01	121
7	JORB	brandt-s-1/Symantec	vecheo/Symantec	401
8	Socket	brandt-s-1/Symantec	vecheo/Symantec	1330
9	JORB	brandt-s-1/Symantec	brandt-s-1/Symantec	418
10	Socket	brandt-s-1/Symantec	brandt-s-1/Symantec	1818
11	JORB	brandt-s-1/JDK 1.02	brandt-s-1/JDK 1.02	159
12	Socket	brandt-s-1/JDK 1.02	brandt-s-1/JDK 1.02	998
13	JORB	brandt-s-1/Symantec	asx100/Kaffe	10
14	Socket	brandt-s-1/Symantec	asx100/Kaffe	105
15	OrbixWeb2.0	brandt-s-1/Symantec	brandt-s-1/Symantec	600

As shown in row 1, JORB communication with the asx100 switch allows around 14 RPCs per second. This number is clearly limited by three main factors:

1. The performance of the switch itself.
2. The performance of the Java virtual machine.
3. The performance of the JORB implementation.

The impact of switch performance can be seen by comparing rows 1 and 3, respectively rows 2 and 4[3]. The performance gain from running the server on brandt-s-1, as opposed to asx100, is 7.6 using JORB, and 5.6 using sockets. Hence, performance of the switch is a clear bottleneck. That performance gain is larger in the JORB case must be contributed to the larger overhead of parameter marshalling, resulting in more time spent in the Java virtual machine.

---

[3] This assumes that JDK 1.01 and JDK 1.02 exhibit similar performance. This has not been tested.

10

The impact of virtual machine performance can be seen by comparing rows 9 and 11. Both rows reflect JORB performance with the client and server running on the same machine, in this case a 200MHz Pentium Pro PC. In row 9, the Symantec virtual machine is used, which includes a just-in-time (JIT) compiler. In row 11, the JDK 1.02 virtual machine is used. The performance gain from the JIT compiler is a factor 2.6 with JORB communication. Similarly, comparing rows 10 and 12 we get a factor 1.8 with raw socket communication. The difference in speed-up must again be contributed to the different times spent in the Java virtual machine, with the socket solution being the most I/O bound of the two solutions. In any case, similar speed-ups can in principle be achieved on the switch by applying a better Java virtual machine, although one is currently not available.

The impact of using CORBA is apparent from comparing any odd-numbered row to its following even-numbered row. For example, the difference between rows 1 and 2 is a speed-up of 7.6 from replacing JORB communication by socket communication. However, this overhead also depends on the CORBA implementation. For example, as can be seen by comparing rows 9 and 15, OrbixWeb performs roughly 50% better than JORB, and would therefore allow around 20 RPCs per second against the switch.

It will be interesting to compare these two CORBA implementations to the future ReTINA real-time DPE based on the Chorus-Cool platform when it is available.

3.4 A Hybrid Solution

Given the above measurements, it might be worthwhile to try to combine the best of the two worlds: The speed of plain sockets with the plugability of CORBA/IIOP. To that end, we tried to insert a proxy object between the client and the server: By means of JORB, the client makes calls to a proxy running on a fast PC on top of a fast Java implementation. The proxy then forwards the request to the real server, using the raw socket interface. Measurements are shown below:

	Client	Proxy	Server	RPC/sec
15	brandt-s-1/Symantec	brandt-s-1/Symantec	asx100/JDK 1.01	97

As can be seen, this approach does in fact combine the best of the two solutions. Combined with a fast JIT based Java implementation on the switch, around 200 connection creations per seconds should be comfortably within reach.

3.5 A Server Implementation in C

Finally, we implemented the server in C. The performance of the server written in C was measured in two different configurations: With the client connecting directly to the server using sockets, and with the client connecting via JORB to a proxy server, which then connects to the server using sockets. The C server does not provide a CORBA interface. Performance is shown below:

11

	Client	Proxy	Server	RPC/sec
16	brandt-s-1/Symantec	brandt-s-1/Symantec	asx100/C	251
17	brandt-s-1/Symantec	Direct (no proxy)	asx100/C	600

Comparing rows 15 and 16, we see that rewriting the connection server in C provides a speed-up factor of 2.6 in a proxy based solution. With the client calling the server directly by means of raw sockets, the C server provides a speed-up factor of 5, as can be seen by comparing rows 6 and 17. It should be noted that this speed-up would shrink to between 2 and 3 if a faster virtual machine could be used on the switch. However, this relatively small speed-up reflects the I/O bound nature of the server. Hence, if the computational complexity of the server was larger, a larger speed-up could be expected.

3.6 SNMP

The performance of connection creation and deletion by means of SNMP was measured by Matthieu Goutet of HP Labs. Using a Smalltalk client and communicating with the switch by means of SNMP, around 50 connection creations or deletions per second was achieved. Assuming that Smalltalk exhibits performance similar to Java (this is ignoring the fact that many Smalltalk systems include a compiler), this number is comparable to row 17, providing around 600 connection creations per second. Hence, using a connection server instead of SNMP, an order of magnitude better performance can be achieved.

Although we have only performed measurements on the ASX-100, others quote SNMP performance around two connection creations or deletions on other switches. This seems to suggest that there are large differences between the performance of SNMP on different switches, and that SNMP implementations can therefore in general be improved with respect to performance. We would expect to see optimised implementations of SNMP on switches where the interface is being used for switch control.

Another optimisation would be an attempt to overcome the SNMP feature that multiple get/set attribute operations have to be made in multiple calls to the interface. One obvious way to address this would be to have an object on the switch that could handle multiple attribute operations as a bundle and that could then make the multiple SNMP calls locally.

To be fair to SNMP, we should not forget that it is significantly more general than our protocol, applies to most or all switches for private networks (switches for public networks may support the CMIP protocol), and was furthermore not conceived as an alternative to connection setup by means of signaling. Nevertheless, although it can be improved upon, SNMP performance on the ASX-100 is probably sufficient for many purposes, maybe even including the implementation of "open switching" approaches such as TINA and Xbind. In any case, as we will discuss below, SNMP is unlikely to be a major bottleneck in the BVPN demonstrator.

4. Discussion

4.1 SNMP Performance

Our measurements demonstrate that SNMP performance is indeed inferior to the connection server used in this experiment. On the test equipment used, SNMP achieves around 50 connection creations per second, which should be compared to the around 600 per second achieved by the connection server implemented in C.

However, only a few switches actually allow the implementation of a connection server by means of an open switching interface such as used within HP Labs. Our ability to put this interface onto the ASX-100 is thus an exception.

4.2 Deployment of TINA Computational Objects

The inferior performance of the Fore switch hardware as compared to a modern PC or UNIX workstation suggests that it would be unwise to run a full EML-CP on the switch itself, although this is indeed possible. A more viable approach is to run the EML-CP on a high-performance PC or workstation, only communicating with the switch itself when necessary to actually create or delete physical connections. This separation of concerns also makes sense from a software engineering point of view, since it allows different switches to be controlled by the same (powerful) controller hosting the corresponding EML-CP object implementations.

The nature of the communication from the EML-CP object to the physical switch can then be achieved by means of SNMP or other mechanisms such as GSMP [9] or the connection server used in this experiment. The latter may achieve an order or magnitude better performance than SNMP.

4.3 The TINA CM Software Bottleneck

Given our SNMP performance measurements, it is our conclusion that the overhead experienced by users of the CM software is not caused by SNMP.

The Alcin CM software has been experienced mainly by Telenor within the BVPN demonstrator. This software includes computational objects: Connection Co-ordinator (CC) that controls the whole underlying network, Layer Network Coordinator (LNC) and Network Management Layer - Connection Performer (NML-CP) described in section 1.2. The EML-CP objects that control the Fore switches have been provided by Telenor.

The performance of the connection creation by means of Alcin was measured within Alcatel lab. A client object, the Alcin package and one "dummy" EML-CP were executed on top of ORBIX 2.0 MT on one Ultra1 Sparc station (running Solaris 2.5 with 64Mb

Ram). The EML-CP is called "dummy" because it is not interconnected to the actual corresponding switch (see figure 5).
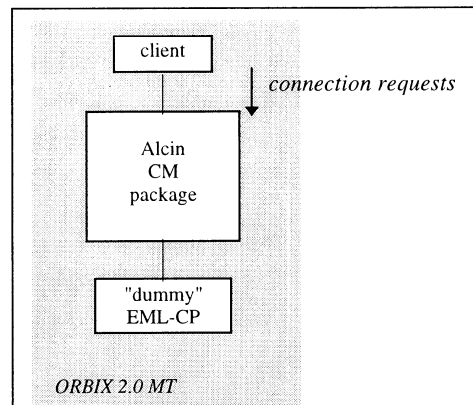

Figure 5: Experiment configuration

A (point-to-point) connection was created in around 28ms, which would correspond to 35 creations per second. This figure has to be compared to the 50 creations per second provided by SNMP. This result can be explained by the number of remote operations required to establish one connection: 9 are required between the client and the Alcin package and 4 between Alcin and the EML-CP object. Indeed, a remote interaction between two ORBIX objects on an Ultra1 Sparc station takes approximately 2ms. Therefore approximately 90% of the time is spent in ORBIX transport. Furthermore we noted that this result had remained valid when the client, the Alcin package and the EML-CP were executed on different computers.

In order to validate this conclusion, we made the same experiment with the Alcin package and the EML-CP co-located in the same address space (one process). With such a configuration a connection was created with 9 invocations in around 20ms (50 creations per second). Again approximately 90% of the time was spent in transport. Therefore the first conclusion is that the use of Alcin on top of SNMP to control one switch may not be seen as a real critical bottleneck (factor 1.4).

However, Alcin allows the control of a global layer network, i.e. of a set of switches. In its current version, the Alcin software controls the set of EML-CPs in a sequential manner. Thus the duration of an end-to-end connection is proportional to the number of involved switches. Since 4 interactions (8ms) are required to establish a sub-connection on each EML-CP, we may estimate the duration of the creation for an end-to-end connection crossing 3 EML-CPs to approximately 44ms (24ms plus 20ms for the interactions between the client and Alcin). This "sequential" behaviour is the main cause of the overhead experienced by Alcin users and creates a bottleneck in the BVPN demonstrator.

Alcin was provided as the ReTINA CM first release. It is important to note that no specific effort has been made to support non-functional requirements such as latency of

14

connections establishment or real-time control. The improvement of the CM non-functional behaviour is one of the objectives for the second release. This will be done through the re-engineering of the CM computational objects, reducing the number of needed invocations, processing multiple requests concurrently and parallelizing the establishment of sub-connections.

Further experiment with this new CM should take place in order to measure the overheads imposed by the use of the CM software, both for the control of one switch and for the establishment of an end-to-end connection. However the quantitative results presented above allow us to be optimistic. Indeed if we could reduce the number of required invocations to 2 (one between the client and Alcin, and one between Alcin and the EML-CP), the average time for the creation of a connection on a switch could be around 5ms, or maybe 10ms if the process time was not so negligible. Thanks to this approach and the introduction of parallelism, the use of ReTINA CM with an effective switch control interface such as e.g. the proxy based hybrid solution would become conceivable.

5. Future Directions

We now have a fairly complete understanding of the link-by-link CM issues and the next phase of the project will address the wider and more difficult issue of end-to-end connections and the QoS attributes of these connections. Within ReTINA we will also implement a complete application that can use the mechanisms provided by the QoS manager. This application will be a BVPN (Broadband Private Virtual Network). The BVPN demonstrator addresses the problem of how to open up connection management in public networks in order to allow third-party providers and customers to create "leased lines" on demand without the need to contact the telecommunications operator.

Two of the design goals of ATM were to support multiple traffic classes simultaneously and to make all resources in the network available to the client services. ATM uses a statistical multiplexing approach to the simultaneous support of multiple connections. Also, connections do not reserve the network resources they need for the entire duration of their lifetime. These efficiency-motivated aspects have two important effects in an ATM network:

- congestion is possible when excessive amounts of traffic contend for limited buffer resources (this results in lost cells)
- variable cell delays occur as the cells traverse the network. These are caused mainly by random queuing delays at each switch and multiplexer (the queues themselves are there to avoid excessive cell loss).

Our challenge will be how to protect the Quality of Service (QoS) of those multiple traffic classes, while simultaneously optimizing the statistical sharing of the network resources.

5.1 ReTINA Quality of Service Management

ReTINA's focus is on real-time services and so we are concentrating on quality of service required in the context of high speed networks and distributed multimedia applications. The objective of the QoS Manager is to assure the quality of connections in accordance with the requirements of the session customer on whose behalf the connection is being made. ReTINA will adopt a contractual approach to the establishment of connections with requested QoS attributes.

Under our architecture a user application makes a connection request based on a particular traffic type and a set of parameters characterizing the kind of quality of service desired. The request goes to a Connection Manager and this request is hierarchically partitioned across the network until the decision is taken locally at each switch involved in the path. At each switch local CAC and resource reservation decisions will be made. The strategies differ from switch to switch. We will develop simple mathematical models of switch behaviour and a calculus for co-ordinating the local QoS capacities into end-to-end estimates. We are also developing algorithms for the aggregation of the figures over our subnetwork hierarchy. These QoS capacity estimates will be used in high-level admission control and in QoS-based routing.

6. References

[1]   "Asynchronous Transfer Mode", Martin De Prycker, Prentice Hall, 1995.

[2]   "ATM Switching Systems", Thomas M. Chen and Stephen S. Liu, Artech House Publishers, 1995.

[3]   "A Binding Architecture for Multimedia Networks", Aurel Lazar, Shailendra K. Bhonsle and Koon-Seng Lim. Center for Telecommunications Research, Columbia University, New York. 1995.

[4]   "Project Xbind", Columbia University in the City of New York. February 1996.

[5]   TINA. http://www.tinac.com

[6]   "Towards an Open API for ATM Switch Control", Aurel Lazar, Franco Marconcini, Comet Group, Center for Telecommunications Research, Columbia University, New York. March 1996.

[7]   "The Recommendation for the IP Next Generation Protocol". Bradner, S and Mankin A. RFC 1752, January 1995.

[8]   "BVPN Service Enterprise Specification", Eirik Dahle and Håkon Solbakken, Telenor, March 1996.

[9]   "IP Switching: The Intelligence of Routing, The performance of Switching", Ipsilon, February 1996.

[10]   http://www.tjwassoc.demon.co.uk/kaffe/kaffe.htm

[11]   Research Networks Section, Naval Research Laboratory (ftp site hsdndev.harvard.edu:pub/mankin)

[12]   Network Resource Information Model Specification, No. TB_LR.001_2.1_95, TINA-C, December 1994.

[13]   "DPE Services Specifications (Telecom Services)", ReTINA ACTS Project AC048 Deliverable, RT/TR-96-10.1/WP3, September 96

[14]   "ATM Solutions for Enterprise Networking" David Ginsburg, Addison-Wesley, 1996.

Acknowledgements