A NETWORK ARCHITECTURE FOR SUPPORTING PARALLEL COMPUTING OVER ATM

A DOCTORAL THESIS

Author: **Joan Vila i Sallent**
Advisor: **Josep Solé i Pareta**

UNIVERSITAT POLITÈCNICA DE CATALUNYA
Departament d'Arquitectura de Computadors

Barcelona,
September 1997

Contents

List of Figures

Chapter 5

Chapter 6

List of Tables

Preface

This thesis presents a network architecture addressed to the execution of parallel computing applications on top of a computer network based on the ATM (Asynchronous Transfer Mode) technology. Recent advances in network performance and workstation technologies have encouraged the research in strategies to provide support to the communication needs of network-based parallel computing environments. We understand that the use of such platforms for supporting parallel computing makes sense when the environment takes advantage of a previously existing network that supports a base of networking applications. In many cases, it will be possible to achieve satisfactory performance for parallel computing applications without the need of incurring in the cost motivated by the acquisition and maintenance of a dedicated multiprocessor, which may be difficult to amortize unless a highly intensive use of parallel computing is performed. For this reason, parallel computing will no longer be limited to critical applications but rather the availability of less costly environments will allow for a wider range of applications to benefit from parallelism. Thus, the existence of both options can lead to a popularization of parallel computing. All these issues make our case of network-based distributed computing be conditioned by:

- The need that parallel computing applications and traditional networking applications share a common ATM network, in order to avoid the unnecessary allocation of resources that would result from the use of a dedicated network for parallel computing.
- The utilization of ATM as the networking technology, since this technology is targeted at integrating the various communication services in the near future and, as a consequence, it will be present in both LAN (Local Area Network) and WAN (Wide Area Network) environments.
- The requirement that any mechanism for supporting parallel computing applications has to rely on standard definitions of ATM, since the addition of network-level specific mechanisms would lead to increase the complexity in the network and difficult network sharing among diverse application types.

As ATM-based networks of workstations are a particular instance of distributed memory parallel systems, the bottleneck is found in the communications through the interconnection network. Therefore, the contribution from communications to performance degradation has to be minimized in order to allow for the achievement of satisfactory performance to ATM-based parallel computing environments. Under all these conditions, our proposed ATM-based network architecture will have to satisfy the major requirement of communications in parallel computing applications, that is, the provision of a low-latency communication service while guaranteeing the delivery of all the data submitted to the network. For this purpose, we will take advantage of the special characteristics of parallel computing applications, which include:

- The short PDUs that parallel computing applications interpret —the elementary data types integer, float, etc.— over which more complex structures such as vectors and matrices are built.
- The moderate average bit rate due to a fairly low frequency of communications, but with large bandwidth requirements when communications effectively take place.

- The long execution time —hours or days— that reality parallel computing applications usually experience.

- The fact that communications in parallel tasks consists of a sequence of requests and responses, because the importance of throughput as a measure of performance becomes relative and, consequently, we can sacrifice some throughput if this allows for a gain in latency.

The work on the proposal of a network architecture for supporting parallel computing applications included three phases: (1) the assessment of the potential possibilities of ATM-based environments; (2) the proposal of an architectural model enabling the integration of specific mechanisms for parallel computing on top of an ATM network that is shared with other networking applications; and (3) the design and evaluation of concrete mechanisms which cover the transition between the communication requirements of parallel computing applications and the ATM transport capabilities.

With regard to the potential performance achievable using ATM, we assess that ATM-based parallel computing environments outperform environments based on other network technologies such as Ethernet and, depending on the application, the performance can be comparable to that of multiprocessors. We also assess that the achievement of the maximum performance over ATM networks requires the elimination of the bottlenecks found in the endpoint hosts.

As an integrated network architecture, we propose a model in which several application types can coexist, each one with its specific network architecture. In the case of parallel computing applications, we consider an architecture including three levels: an *application level* where the specific requirements of parallel computing applications are managed, a *network level* containing the functions provided by the ATM technology, and a *convergence level* including those functions that are required for supporting parallel computing applications and are not supported in the network level.

The concrete mechanisms that we suggest for the network architecture focus on minimizing the impact of the protocol stack on latency. This is accomplished by tightly coupling communications from parallel computing applications with ATM, so that heavyweight protocol stacks such as TCP/IP are avoided. Thus, AAL-level lightweight mechanisms supply the robustness required by parallel computing applications that ATM networks do not provide, including procedures to reduce the need of recovering from information loss. The evaluation of the mechanisms has been performed by simulation, due to the need of a highly-configurable environment, as well as the unavailability of suitable equipment in some cases. In particular, our suggested mechanisms involve:

- *Coupling of a transport mechanism to parallel computing*. The loss recovery mechanism of an existing lightweight transport protocol, namely SSCOP (Service Specific Connection-oriented Protocol), has been modified in order to avoid the occurrence of useless retransmissions by considering PDUs (Protocol Data Unit) not longer that 48 bytes (the payload of an ATM cell). This modified SSCOP is viewed by the application as a novel, specific AAL.

- *Exploitation of ATM's standard transport services.* The ATM Forum defines several classes of service, according to the quality of service, namely UBR (Unspecified Bit Rate), ABR (Available Bit Rate), VBR (Variable Bit Rate), and CBR (Constant Bit Rate). The introduction of ABR can improve the communications latency, but at a cost that may be excessive as compared to an UBR service when the network is not congested. Thus, we propose that ABR be used only when the latency experienced in the network is significantly high while in the rest of time UBR is used instead. The purpose of this strategy is to reduce the cell loss probability experienced by modified SSCOP, and therefore the need of retransmissions, in a cost-effective fashion.

The main results from the evaluation of these mechanisms are that (1) the mechanism restricting the retransmitted information to the effectively lost cells succeeds in providing a robust operation and, in addition, the short PDUs accelerate the delivery of data, thus reducing latency; and (2) the exploitation of ATM service categories leads to achieve cost-effective performance, as it is possible to get values for latency similar to those obtained when relying on ABR all the time, but with a significantly lower utilization of the ABR service category. Thus, we have achieved an ATM-based network architecture whose impact of the run-time performance of parallel computing applications is reasonable. Another issue that is important for building real ATM-based parallel computing environments is the configuration of the environment itself. Further research is needed on the signaling procedures for this purpose.

The internal organization of the thesis is as follows: Chapters 1, 2, and 3 are devoted to introduce ATM-based parallel computing environments and the related literature on them. In particular, Chapter 1 contains a brief summary and introduces some background concepts on ATM technology and measurement techniques that are relevant to the understanding of the architectures and mechanisms presented in the work. In Chapter 2, the performance of an ATM-based parallel computing environment when running real parallel algorithms under ideal conditions, that is, without overheads, is compared with the performance achieved by other parallel computing environments, including multiprocessor and Ethernet-based platforms, and the relative importance of bottlenecks is also studied. Chapter 3 summarizes the diverse strategies undertaken by many authors in order to introduce ATM in parallel computing environments, focusing on their adaption to the characteristics of parallel computing applications and their ability to be integrated in a network shared with other applications.

Chapter 4 draws the framework on which our proposal of an ATM-based network architecture for supporting parallel computing is situated. For this purpose, the characteristics of the model on which we base our proposals are discussed, by taking into account the aforementioned requirements of specificity and ability of integration.

Chapters 5 and 6 are focused on the particular mechanisms we suggest to be implemented within the proposed network architecture. In Chapter 5 we discuss the specific AAL for parallel computing that couples an existing protocol to the special characteristics of communications in parallel computing environments. In Chapter 6, we present how we enhanced the specific AAL in order to take advantage of ATM service categories. These two chapters include simulation-based performance studies.

Finally, Chapter 7 presents the conclusions of the work and displays the future work that will be carried out to complete the present work and to widen its scope.

The appendices contain complementary aspects of the work. Appendix A contains a description of the characteristics of the benchmarks used in the performance evaluation measurements carried out that have been carried out throughout the work. Appendix B is a description of a particularly complex simulation environment that we used for emulating an ATM network in the experiments discussed in Chapter 2.

Acknowledgments

The work that has led to the present thesis would not have been possible without the help of many people. First of all, I specially thank Josep Solé i Pareta, who has been the advisor all these years, for his dedication on the supervision of the work in this thesis. I also thank Jordi Domingo i Pascual for having dedicated some of his time on my doubts about measuring, as well as Jordi Torres and Teodor Jové for their help in concepts concerning distributed systems and parallel computing, as well as their collaboration in the part of assessing the potential performance of ATM-based environments. I acknowledge the technical orientations from Prof. Ian F. Akyildiz, Ramón Beivide, and Víctor Viñals.

I do not want to forget the research assistants in the department, without whom the fine atmosphere that have enabled the realization of this thesis would not have been possible. Some of the informal discussions have been taken into account throughout the thesis. The latter applies also for the rest of members in the department.

An essential part of the work has been carried out with the support of the machines in the DAC's computing center (LCAC), the CEPBA (European Center for Parallelism in Barcelona), the CESCA (Supercomputing Center of Catalonia), and the CCABA (Center for Advanced Broadband Communications). I acknowledge the invaluable support from the staff in these centers.

Finally, I want to thank my family, for their support along the work.

During the preparation of the present thesis, the author has been supported by a research training fellowship from the Spanish Education Ministry (FPI program), reference code AP92 39346379. The work was carried out under the following basic research contracts by the Spanish Government's CICYT (Interdepartmental Commission for Science and Technology): TIC 92-1298-PB and TIC 95-0982-CO2-01.

1 Introduction

This chapter is an introduction to the scenario on which all the work in this thesis is based. After overviewing the objectives and contents of the work, we present the characteristics of the ATM-based platform for supporting parallel computing on which we rely for our proposals. The rest of the introduction is devoted to background technical aspects that will be used throughout the work: the study of some concepts regarding the services provided by ATM networks, and the discussion of the parameters on which the performance studies contained in this work are based.

1.1 Motivation and contributions

The ultimate aim of this work is to contribute to the convergence between two major technological areas such as communications and computing. This convergence is currently accelerating thanks to a number of technical factors, namely (1) the progress in hardware technology, in particular microprocessor technology, which is leading to an increasing popularization of parallel computing; (2) the increasing availability of large amounts of bandwidth in the networks, which is allowing that communications no longer be the hardest bottleneck in distributed computer applications; and (3) the current trend of integrating the various communication services provided by networks with a single networking technology that is capable of supporting all these services will reduce the costs of the networking infrastructure. These factors are leading to a future panorama where large computing systems composed by distributed machines interconnected via high speed links will be able to support a wide variety of applications.

One of the services to be included within this future platform is the support to communications in parallel computing environments, where the processes are located in a number of computers connected to the integrated network. The availability of such an environment with sufficiently adequate performance is particularly interesting since many types of parallel applications can execute on top of such a platform without incurring in the costs involved with dedicated parallel computing environments —multiprocessors and multicomputers. Although some applications will always require the power provided by these dedicated environments, it is also true that some other applications will suffice with a network-based parallel computing platform, specially if the organization does not have a very intensive need of parallel computing. We assume that the integrated network will be based on ATM (Asynchronous Transfer Mode), as the international standardization bodies have adopted this technology for this purpose. Thus, ATM networks are expected to be very common and, as any previously existing ATM-based network can be used as an infrastructure for supporting parallel computing, it will be possible to run complex parallel

applications with a very small cost.

The contributions in the present work are addressed to provide ATM-based parallel computing platforms with a set of communication mechanisms that, by taking advantage of the features supplied by ATM networks, lead to minimize the performance degradation that is inherent to communications between parallel tasks in different nodes. In particular, our work includes:

- The study of the potential performance to which ATM-based environments will tend in an asymptotic manner, in order to assess the usefulness of all the research involved in ATM-based parallel computing platforms.
- The definition of the network architecture model, in which parallel computing will be supported by specific mechanisms that will share the ATM network with specific network architectures for other networking applications.
- The proposal of an ATM Adaptation Layer (AAL) that will specifically support communications generated by parallel computing applications. This specific AAL takes advantage of the cell-relying feature of ATM equipment in order to reduce the impact of congestion in ATM networks.
- The enhancement of the specific AAL by exploiting the diverse service categories provided by ATM. The objective is to improve performance in a cost-effective fashion with the introduction of expensive mechanisms only when strictly required.

In the rest of the introduction, we deal with the characteristics of the parallel computing environment on which the present work is based, and relate this platform with the concept of "Networks of Workstations" that is found in the origin of our proposals. This discussion is completed with the summary of background concepts that are important to understand the rest of the work. In particular, we include a study of all aspects of ATM technology that can potentially be used by parallel computing environments. Another issue that is covered in the introduction is the description of the particular characteristics of communications in parallel computing environments, which are originated by the generation and reception by each tasks of messages from all the other tasks in a discontinuous fashion, and the specific procedures to measure performance in these environments that result form the specificities of parallel computing applications.

1.2 The scenario

Until recently, running parallel applications has been a matter reserved to supercomputers. The current progress in both microprocessor and networking technologies in beginning to build cost-effective parallel systems with on-the-shelf workstation and local area networks. In the following we introduce general concepts of parallel computing before discussing the specific characteristics and concerns found in networks of workstations, and then we present the characteristics of the ATM-based parallel computing environment on which we have based.

1.2.1 Parallel computing and Networks of Workstations (NoWs)

The advent of multiprocessor systems has enabled the possibility of taking advantage of parallel execution flows in order to reduce the execution time of applications. This enhanced performance can be exploited either on an instruction basis, a thread basis, or a process basis. Many scientific and engineering applications are designed to run on task-level parallel systems, thus achieving coarse-grain parallelism. Although in theory it is possible to multiply the performance achieved by a uniprocessor by the number of processors

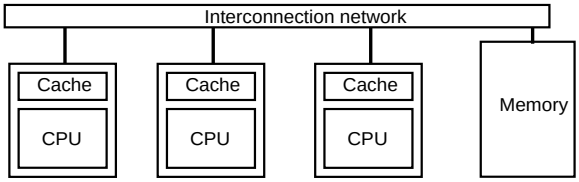


Figure 1.1: *Architecture of a shared-memory system.*

in the parallel computer, in practice this is not possible due to a number of reasons. The first reason is related to the characteristics of the parallel algorithms where, as the Amdahl's law illustrates [Amd67], the performance delivered by the sequential parts of the code significantly condition the actual speedup achieved by the system. The second reason is subsequent to the need of parallel processes to communicate and synchronize among themselves. Thus, the communication mechanisms have a significant impact on the actual performance achieved by parallel computers. In fact, communications are found to be one of the most important bottlenecks.

Architectures of parallel computers are traditionally classified in two main classes: shared-memory and distributed-memory. For each class, several implementations have been proposed which trade off performance and cost. As far as distributed-memory architectures are considered, one particular case is represented by the networks of workstations, which provide a cost-effective solution since they enable the support of parallel computing by using components not specifically designed for this purpose [Fos95]. In the following, each of the three studied environments are discussed. In particular, their relevant characteristics are reviewed, and the implementation corresponding to each environment is presented. Emphasis is made on communication issues.

Shared-memory multiprocessors: In this architecture, displayed in Figure 1.1, part or the entire memory space is shared by all the processors. Thus, communications between tasks running on different processors can communicate through these common memory locations, usually with the help of mutual exclusion procedures. Shared-memory multiprocessors make extensive use of caches, since the memory latencies can be very high, specially when memory modules are distributed. For this reason, the characteristics of the caches and the coherence algorithm, together with the locality of the data exchanged by the tasks, determine in a significant fashion the performance of shared-memory communications. An example of this architecture is the SGI PowerChallenge, whose architecture is similar to that of the Challenge system, discussed in [HP96].

The SGI PowerChallenge is a shared memory multiprocessor. The interconnection between processors and modules takes place through a wide bus (256 data bits and 40 address bits). The cache coherency is maintained through an enhanced snooping protocol. Each bus cycle corresponds to 21 nanoseconds. As 22 bus cycles are required to satisfy a read miss, 462 nanoseconds is the latency for a processor module to get 128 bits from main memory. The particular implementation used in the measurements in this work uses R10000 processor modules.

Distributed-memory multiprocessors: In distributed memory systems, each processor has direct access to local memory that cannot be directly accessed by the rest of processors, as shown in Figure 1.2. As far as communications between tasks are concerned, the predominant cost comes from the characteristics of the network interconnecting the processors.

Analogously to shared memory environments, locality is an important factor since it determines the need to communicate to remote processors. Distributed memory systems are simpler than shared memory systems in the sense that no mutual exclusions or cache coherence algorithms have to be implemented. However, programming these environments is more difficult since the programmer has to be aware of the

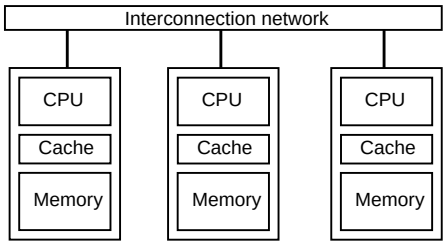


Figure 1.2: *Distributed memory architecture.*

architecture. The IBM SP2 is an example of this type of machines. Networks of workstations, which are discussed below, are another case of distributed memory systems. Its architecture is described in [A+95]. The computing nodes consist of RS/6000-based modules.

Unlike centralized shared-memory systems, represented by the SGI PowerChallenge, in the SP2 memory modules are local to each processors. Thus, a complex interconnection is needed which, in the particular SP2 configuration tested in this work, consists of a multistage packet-switched network so-called *high performance switch* [S+95]. Requests of remote memory require to access the I/O bus attached to each processor, so remote memory accesses exhibit much higher latency than local ones. As mentioned in [MABG96], latencies are lower than 1.2 μs.

Programming parallel systems involves a high amount of issues to take into account, namely interprocess communication, synchronization, etc., as opposed to sequential programming. In order to ease the development of parallel programs, there exist the so-called *programming models*, which consist of a set of abstractions providing transparency to the programmer with respect to the particular architecture of the parallel system. The most popular programming models include the *Shared-Memory model*, where details of communications are hidden to the programmer, and the *Message-Passing model*, where communication takes place through explicit messages. There are other popular programming models, for instance the *Data-Parallel* model, but compilation can reduce them to shared-memory or message-passing.

Although the shared-memory model is well-suited for running on shared-memory architectures and the message-passing model on distributed-memory systems, these models are independent of the underlying architecture. Indeed, it is possible to build a shared-memory programming model over a distributed memory architecture, with the help of the operating system [MYW94], thread libraries, or object-oriented abstractions [Fos95], as well as to use a message-passing model over a shared memory architecture, as possible in the SGI PowerChallenge machines [SGI96]. Thus, many parallel algorithms are designed to be run on a message-passing model since it enables a higher degree of interoperativity. This model is generally supported by a user-level *message-passing library*. The most popular libraries are PVM (Parallel Virtual Machine) [G+94] and MPI (Message Passing Interface) [DOSW96].

The challenge of interprocessor communication is to minimize the delay introduced by the interconnection networks. Thus, in the SP2 a message header can reach its destination in just 39 μseconds for a 32-node system. Unless a failure in the network, the reliable delivery of data is assured by means of a hardware-level protocol. Thus, in case of contention, data can be slightly delayed but not lost, thus avoiding complex software-level protocols.

Networks of workstations, also known as Workstation Clusters, are in fact a special case of distributed memory systems. Their particularity relies on the fact that, as displayed in Figure 1.3, typical workstations are used as processor nodes and the interconnection network is implemented by standard local area networking technologies including Ethernet, FDDI (Fiber Distributed Data Interface) and, recently, ATM (Asynchronous Transfer Mode), described in the next section. The utilization of such technologies leads

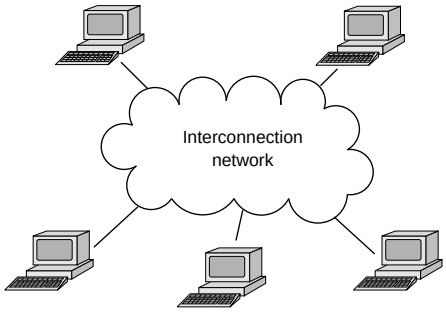


Figure 1.3: *Network of Workstations (NoW).*

to cost-effective solutions for a price in performance degradation that can be affordable for a wide range of applications, as opposed to more expensive systems for which the higher performance is achieved at the expense of specific components, as is the case of the IBM SP2 [A+95].

Using networks of workstations for parallel computing is raising increasing interest because there is no need of dedicated equipment. The same networked workstations used as file servers or individual desktop computers can be used as parallel nodes when these workstations are idle —usually at nighttime, although in Berkeley it was found that as many as 60% of workstations were available for running parallel jobs, even during daytime hours [ACP+95]. However, communications represent a hard bottleneck for networks of workstations. In particular, using Ethernet as the interconnection technology seriously limits the performance gain achieved by applications with respect to the sequential execution of an equivalent algorithm —this gain is known as the *speedup*. The reasons are the low bandwidth offered by Ethernet —10 Mb/s, and the fact that concurrent communication requests are serialized because all hosts share the transmission medium, which specially impacts configurations with many processors since concurrent communications are more likely. Using FDDI can partially improve the performance of networks of workstations, thanks to its 100 Mb/s bandwidth. Nevertheless, FDDI is another shared-medium technology where communications should be serialized. This fact, together with the high latency involved in transmitting data due to the high overhead, precludes the achievement of the performance that the increased bandwidth promised. These observations are supported by a number of studies, for example [FW94].

A consequence from the statements above, the most popular LAN technologies like Ethernet and FDDI are expected not to have adequate connectivity and/or bandwidth as well as low enough latency to carry out some of the applications. But the networking technology is not the sole responsible of the bottleneck. The evolution of networking technologies has achieved very important increases in bandwidth, but the time invested in processing within the communicating peers has not improved similarly. Therefore, the communications bottleneck is not eliminated but simply moved from the networking technology to the host-level processing. This effect is illustrated in [C+94], where the effective bandwidth utilization is measured along a 3-minute job transferring 32 MB of data. While an Ethernet achieved utilizations between 20% and nearly 100%, a FDDI ring, offering ten-times-higher bandwidth, experienced utilizations between 8% and 25% only.

1.2.2 A public platform based on the NoW concept

The most relevant feature of the NoW concept is the possibility of taking advantage of previously existing equipment in order to support parallel computing. Thus, the typical configuration of workstations connected to a local area network that is found in many organizations can be used as a parallel computing

platform. Actually, the network need not be a local area network; MANs and WANs can be an adequate infrastructure if parallel computing applications can afford the extra delays involved in such wide areas. However, much of the work on Networks of Workstations rather assume a dedicated environment[1]. Instead, we explicitly focus on platforms using non-dedicated equipment, whose operation is consequently conditioned by the need of parallel computing applications to interoperate with other applications. For this purpose, we assume the following conditions for our model of networks of workstations as a logical result:

1. *Parallel computing applications will share the network with traditional networking applications*. Parallel computing could exploit dedicated networking technologies, such as Myrinet [B+95], and also other technologies requiring a tight coupling with processors, as in Fiber Channel [SV96] and Memory Channel [Gil96]. These technologies only become affordable when the network of workstations is intensively devoted to parallel computing, due to the important amount of resources that are required to be allocated for a single application type. In contrast, taking advantage of an existing network that is already used by other networking applications (file transfers, multimedia, etc.) introduces a more efficient exploitation of the resources available in the organization.

2. *In the expected scenario, ATM technology will be commonplace*. Current progress in networking is addressed to achieving faster technologies so that the spectrum of applications taking advantage of networking becomes broader. For this purpose, many networking technologies have been being used in order to increase the available bandwidth in both the local and the wide areas, namely 802.x LANs (Ethernet and Token Ring), X.25, FDDI (Fiber Distributed Data Interface), DQDB (Distributed Queue Dual Bus), Frame Relay, and ATM (Asynchronous Transfer Mode). ATM has become a more serious candidate to become the future networking technology for both wide and local area networks. The reason is that although ATM was originally proposed to support the B-ISDN (Broadband Integrated Services Digital Network) infrastructure, today is a major option to replace legacy LAN technologies. We foresee the existence of a large number of workstations connected to ATM networks in the near future.

3. *The adaptation to parallel computing specificities will be done above ATM*. Although doing the adaptation inside the network level as done in [SBT96] could lead to better performance, we do not find this approach appropriate in the context of this work. Instead, in order to fulfill the preceding hypothesis, in particular the fact that many application types will share the ATM network, we assume that the network is a standard ATM network. For this purpose, all the functions that are specific to parallel computing applications will be implemented in the endpoint hosts, above ATM. The mechanisms implementing the proposed network architecture will rely on the standard service categories defined by the ITU-T (Telecommunications Standardization Sector of the International Telecommunication Union) and the ATM Forum.

It becomes clear from these hypothesis that we prioritize the search of a cost-effective network of workstations that balances performance and cost over the maximization of the performance of such configurations. Networks of workstations as specified above convey a number of overheads in the procedures for configuring them that prevent applications running very short from being advantageous. Nevertheless, many scientific applications execute longer, during hours or even days, so in these cases a non-dedicated platform can be advantageous. In any case, for achieving competitive performance it is necessary to build communication mechanisms that benefit as much as possible from the features provided

[1]The boundary between a multicomputer and a network of workstations is not clear then. For example, the IBM SP2 could be considered as a network of workstations.

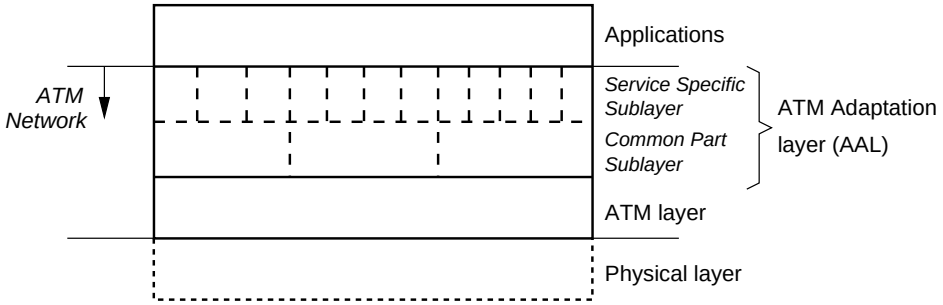


Figure 1.4: *Architecture of ATM networks.*

by ATM networks in order to reduce the impact of communications on performance. For this purpose, it is important to study the characteristics of ATM networks in order to find out which features of ATM can be exploited by the proposed network architecture for supporting parallel computing.

1.3 Concepts of ATM technology

ATM (Asynchronous Transfer Mode) is a communications technology designed and recommended by ITU-T (Telecommunications Standardization Sector of the International Telecommunication Union) for supporting the B-ISDN (Broadband Integrated Services Digital Network) infrastructure. As the B-ISDN has to provide satisfactory support to a wide range of applications, including both real-time services such as voice and video, and data services like file transfers, data base querying, etc. However, ATM's most successful applications today include the replacement to existing, traditional LAN technologies, as well as the interconnection of LANs through a backbone network. Thus, in the future, ATM is expected to integrate both LAN and WAN environments. For this reason, ATM becomes a serious candidate technology to support communications in parallel computing applications. This section summarizes the characteristics of ATM which are more relevant to the present work. In particular, we describe the layer structure of ATM networks, focusing on the description of the service categories provided by ATM and the functions provided above this layer in order to interface with higher-level mechanisms. We also discuss the possibility of virtual subnetworks enabled by the virtual circuit scheme defined for ATM.

1.3.1 Architecture of ATM networks

The ability of ATM to adapt to the different services comes from the compromise between the low delay achieved by circuit-switching mechanisms and the flexibility inherent to packet-switching technologies. This behavior has been made possible with the encapsulation of data into fixed-sized, very short packets (a 5-byte header plus a 48-byte payload, as defined in [ITU93c]), known as *cells*, whose transmission is carried out through a hierarchy of virtual connections described in [ITU93b]. The final objective is to efficiently exploit the high bandwidth enabled by current transmission media (fiber optics). ATM-based networks involve a fairly complex architecture in order to allow for a wide range of applications to take advantage of ATM. As shown in Figure 1.4, ATM networks include two layers: the *ATM layer* and the *ATM Adaptation Layer (AAL)*.

The ATM layer is in charge of providing the cell transport service. As data are transferred by using virtual connections, cells are delivered in order to the receiving peer. In fact, ATM is a packet-switching technology dealing with small packets having all of them the same short length. As a consequence, like in all packet-switching networks, the delay experienced in the virtual connection is not constant but varies

according to the cell. In addition, for the same reason, congestion situations may occur in which some cell may be lost.

As most of the candidate applications to be supported by ATM have further communication requirements, the ATM Adaptation Layer adds some functionality to the ATM layer in order to facilitate the interface application-ATM. For this purpose, two sublayers are considered in the AAL, as defined in [ITU93e] and shown in Figure 1.4:

- *Common Part Sublayer*, which contains some functions that are common to a number of services.
- *Service Specific Sublayer*, which contains the functions required for supporting a higher-level service that are not included in any defined common part sublayer.

In the following subsections, the facilities provided in both of the layers are described, emphasizing those functions that are more likely to be used by communication procedures in parallel computing applications.

1.3.2 Services provided in the ATM layer

As mentioned above, the ATM layer provides a non-reliable cell transport service which does not disorder cells. For some applications, this behavior is not a serious problem. For instance, in the real-time transmission of video, the loss of a cell results in an alteration of the attributes of a few pixels which is usually unnoticed by the human receiver. In contrast, other applications require the full delivery of all submitted data. All services involving file transfer, for example, require that reliability in order to avoid the corruption of files.

Generalizing this observation, each type of applications expects characteristic features in the service provided by ATM, that is, a characteristic "Quality of Service (QoS)". To specify the Quality of Service, the following parameters are used, as defined in [ATM96c]:

- *Maximum Cell Transfer Delay*. It determines the maximum delay a cell should experience in the network.
- *Cell Delay Variation*. Cells belonging to the same transfer may experience different delays, due to the characteristics of cell relaying. If the variations are too large, some services might degrade their quality, specially interactive and real-time services.
- *Cell Loss Ratio*. The highest fraction of lost cells that a service can tolerate for guaranteeing a good quality.

These parameters are subject to negotiation between the end-system and the network when establishing an ATM-level connection. There are still other parameters that are not negotiated. The protocol for this negotiation is defined in the UNI Signaling specification [ATM96a]. The combination of different values for the QoS parameters determine a wide range of types of service. In the ATM Forum [ATM96c] and the ITU-T [ITU95a], these types of service have been classified in a number of families, so-called *Service Categories* by the ATM Forum or *Transfer Capabilities* by the ITU-T. In the following, these services are discussed.

Some service categories provide a *guaranteed* service, meaning that the ATM network is willing to allocate the necessary resources in order to enable the desired service characteristics. The service will perform as requested provided that the traffic generated by the end-system is really consistent with the traffic description supplied when establishing the connection. In contrast, other services, mostly data

transfer services, cannot specify their resource needs when establishing the connection, since the usage of bandwidth and buffers varies during the connection. Thus, these services share the resources that have been not allocated for the guaranteed services. This operation is known as *best-effort*, meaning that the network will provide the best service as possible according to the state of the network.

The ATM Forum and the ITU-T define the following guaranteed service categories: CBR, rt-VBR, nrt-VBR and ABT. The former two are devoted to services requiring real-time operation. The latter two categories are not so strict, as long as time requirements are concerned.

CBR (Constant Bit Rate).

Known as "Deterministic Bit Rate" (DBR) in the ITU-T literature. This service category is intended for real-time applications where it is reasonably efficient to assign a fixed bandwidth to a single source. This is usually the case of applications containing video or audio information. In this service category, the end-system specifies a cell rate to be allocated, and the application is assumed to offer traffic constantly at this rate. A real-time constraint applies, in the sense that excessively delayed cells are considered to be lost. As noted in [Gar96], the CBR service category is simple to be defined but not so simple to be implemented, specially when several administrative domains have to be traversed. The ITU-T and the ATM Forum have adopted a Leaky Bucket algorithm to measure and "police" the cell rate entering a network, either from the end-system or another network. The candidate applications for using the CBR service category include telephone, video conferencing, and television (entertainment video) [Jai95].

rt-VBR (Real-Time Variable Bit Rate).

Not supported in the ITU-T service architecture. Many real-time applications use coding algorithms generating variable rate traffic. In this case, sources may be statistically multiplexed —i.e. the sum of the average rates may be higher than the output rate—, at the cost of a non-zero cell loss probability. The strategy to allocate resources for a rt-VBR service quality is not an established issue, since it may depend on the particular characteristics of the coders and the user perception of the service. The main difficulty in building implementations of his service category is the tradeoff between rate variability and the real-time constraint. If all the difficulties are solved, all types of video and audio applications can accommodate to this service and therefore they may gain bandwidth efficiency. Some applications, however, will not be able to move from CBR because they cannot afford any cell loss.

nrt-VBR (Non-Real-Time Variable Bit Rate).

Known as "Statistical Bit Rate" (SBR) in the ITU-T specification. It is similar to the rt-VBR service category, except in the lack of a real-time constraint, so applications not generating a continuous flow of bits but well-delimited amounts of data can be supported. Thus, sources may be statistically multiplexed and a certain cell loss probability does exist, analogously as above. As a guaranteed service, nrt-VBR requires explicit reservation of resources. Implementations of this service category are easier to build than rt-VBR since there is no tradeoff with the real-time constraint. The applications expected to use this service category include response time critical transaction processing (for instance, airline reservations, banking transactions, process monitoring, etc.), and Frame Relay interworking.

ABT (ATM Block Transfer).

This service category is only defined in the ITU-T specification. It provides a service where the QoS parameters are negotiated on an ATM block basis. Within an ATM block accepted by the network, sufficient resources are allocated so that the QoS received by the ATM block is equivalent to the QoS that a CBR connection would receive if the same peak cell rate was specified. An ATM block is a group of cells delineated by two special cells devoted to resource management. Two types of ABT are defined: (1) ABT with Delayed Transmission (ABT/DT), where renegotiation of the peak cell rate need an acknowledgment from the network prior to be effective, and (2) ABT with Immediate Transmission (ABT/IT), where ATM

blocks are transmitted without waiting for acknowledgment from the network so cells may be discarded if sufficient resources are not actually available. ABT is designed to support applications generating data as well-delimited bursts of cells.

All the service categories described above required some sort of explicit resource reservation, particularly bandwidth. In the following, the rest of defined service —the best-effort ones— are discussed.

Unspecified Bit Rate (UBR).

Not supported in the ITU-T service architecture. This service category does not involve any negotiation with the network. Sources submit and cells as fast as the applications generate the data. However, if the guaranteed services do not leave sufficient bandwidth, the buffers in the switches may easily become congested and therefore the probability of cell loss can be very high. Cells belonging to UBR connections are the first ones to be discarded in case of congestion because there is no guarantee of maximum delay. If some QoS has to be provided, the higher levels over ATM have to care about it. This service category can be used for any data transfer applications not requiring time-critical response. It is also used by applications relying on an Internet protocol stack (TCP/IP or UDP/IP), since the functionality achieved by the UBR service category resembles that of IP.

Available Bit Rate (ABR).

Supported by both ITU-T and ATM Forum under the same name. The objective of this service category is to reduce the cell loss that is experienced with UBR. For this purpose, a feedback mechanism is added so that the source can adapt the generated traffic to the amount of bandwidth available in the network —hence the name of this service category. The feedback covers either all the end-to-end path or the path between two nodes only. Apart from the desired cell loss ratio, no other guarantees are specified[2], although the end-to-end delay is attempted to keep low. This service category is addressed for the same applications as UBR in the cases where lower cell loss is desired, as well as critical data transfer applications.

Two major approaches have been proposed, namely credit-based and rate-based [New94b]. The credit-based approach is a link-by-link window flow control scheme. At the receiving end of each link, a certain number of cell buffers is reserved depending on the propagation delay and the maximum transmission rate. The sending end of the link keeps track of the empty cell buffers at the receiving end in order to set a limit upon the maximum allowed amount of cells to be sent. Periodically, the account of empty cell buffers is updated by the receiving end of the link. This approach was rejected by the ATM Forum in favor of the rate-based approach, because it was considered too expensive and inflexible.

Rate-based schemes use feedback information from the network to control the rate at which each source emits cells into the network on every virtual connection. Three types of rate-based schemes have been proposed: forward explicit congestion notification (FECN), backward (BECN), and explicit rate control. In FECN, when a path through a switch becomes congested, all the cells in the path are marked. The destination end system monitors the congestion status and sends congestion notification cells in the reverse direction to inform the source of the congestion status. In BECN, backward congestion notification cells are generated directly by the congested switch instead of the receiving end. In both FECN and BECN, the cell rate is heuristically determined (multiplicatively decreased when congestion detected and additively increased while congestion is not experienced). In explicit rate control, a Resource Management cell periodically explores the path and, at each switch, the optimal cell rate is computed so that the lowest computed rate is returned to the sender. The definition of the ABR service in the ATM Forum's Traffic Management specification allows for the three schemes, and therefore the source algorithm has to support all of them [JKGF96].

[2]In fact, this is not true. Current specifications of ABR allow to specify a Minimum Cell Rate in order to avoid blocking the ABR connection in case of severe congestion. Thus, ABR is not a pure best effort service category since it provides some bandwidth guarantee.

Explicit rate control schemes allow for a faster reaction time [CCJ95] but at the cost of much operation in the switches. The ATM Forum has not adopted any scheduling algorithm for the ABR-supporting switches but instead has left this decision as implementation-specific, in order to enable flexibility for switch manufacturers. However, some examples of algorithms are briefly described in [ATM96c]: EPRCA (Enhanced Proportional Rate Control Algorithm), ERICA (Explicit Rate Indication for Congestion Avoidance), and CAPC (Congestion Avoidance using Proportional Control). These algorithms are summarized in [Jai95]. The ERICA algorithm is fully described in [JKG+96].

1.3.3 Services provided in the ATM Adaptation Layer

The ITU-T Recommendation I.362 [ITU93d] defines the function of the ATM Adaptation Layer (AAL) as to enhance the services provided by the ATM layer in order to support the functions required by the next higher levels. The functions performed in the AAL depend on the higher level requirements. The AAL supports multiple protocols to fit the needs of the different AAL service users. Examples of services provided by the AAL include handling of transmission errors; handling quantization effect due to cell information field size; handling of the lost and misinserted cell condition; and flow control and timing control. With the AAL, the higher layers can be isolated from the specific characteristics of the ATM layer.

ITU-T is considering four different types of AAL to provide a range of basic functions: AAL type 1 (AAL1) for constant bit rate services; AAL type 2 (AAL2) for variable bit rate services; AAL type 3/4 (AAL3/4) and AAL type 5 for packet transport services. In some cases, these basic functions can be extended to a more service-specific functionality. Another possibility enabled by current definitions of AAL is to directly provide ATM-layer service, which may be required by some applications. In the following the functionality of standard AAL types is outlined. Further information can be obtained from ITU-T Recommendation I.363 [ITU93e].

AAL1.

This AAL type is addressed to applications generating constant bit rate traffic with strong time relations between both communicating peers. There exist functions for supporting circuit transport, video signal transport, and voice-band signal transport. The services provided by AAL1 include:

- Transfer of data with a constant bit rate and delivery of them with the same bit rate.
- Transfer of timing information between source and destination.
- Transfer of structure information between source and destination.
- Indication of lost or errored information which is not recovered by AAL1, if needed.

The AAL1 accepts data on a bit-by-bit or a byte-by-byte basis. The constant bit rate is ensured by means of a buffer. Loss of data may be detected by sequence violation. The timing information transfer mechanisms has been included for applications requiring recovery of source clock frequency at the destination end. For this purpose. For video and high quality audio signal transport, forward error correction (FEC) may be performed to protect against bit errors. This may be combined with interleaving of AAL user bits to give more secure protection against errors. The standardization process of this AAL type is fairly complete. Some vendors are currently supporting it in their equipment.

AAL2.

AAL type 2 is targeted at supporting all services that require the multiplexing of information from multiple sources into a single ATM connection. These services currently include voice, data and video

telephony, in both wire-line and wireless transmission. Unlike the earliest proposals for AAL2, this AAL type is not targeted at supporting VBR video. Major objectives of AAL2 are to achieve low cell assembly delay, coupled with high bandwidth utilization efficiency [Cav97]. As being defined in [ITU96], this adaptation layer is separated into two parts, a Service Specific Convergence Sublayer (SSCS) and a Common Part Sublayer (CPS).

The CPS provides for dynamic allocation of slots within each cell, where each slot is prefixed by a three-byte header. Slots can be shorter than one ATM cell, so that several micropackets, as used for mobile trunking, are multiplexed. In addition, AAL2 allows for AAL-level switching, what means that switches along a virtual connection can deal with particular slots. Unlike AAL1, no end-to-end time relation is supported in the CPS. The SSCS enables different types of services to take advantage of the CPS. In the ATM Forum several SSCS for AAL2 are being defined, corresponding to voice trunking and mobile communications.

AAL3/4.

This AAL type is the result of the integration of the former AAL3 and AAL4 types. The service provided by the AAL3/4 is addressed to the transfer of packetized data between peers without time relation between them. Two sublayers are considered:

- *Service-Specific Convergence Sublayer (SSCS).* It includes those functions required for supporting a particular packetized data-oriented service. As each service involves its respective SSCS, their definition is not included in the AAL specification [ITU93e].
- *Common Part Sublayer (CPS).* Its objective is to provided those functions common to most packetized data-oriented services. In particular, the service involves the non-assured transfer of user data frames with any length. The rest of the comments on the AAL3/4 apply to the CPS only.

The specification considers two types of service for the CPS: *message mode service*, where the CPS processes the whole data packet at a time; and *streaming mode service*, where the CPS operates with fragments of a data packet —fragmentation has to be done by an SSCS— so that transmission occurs concurrently with AAL processing. The data packet is delimited by a header and a tailer. Each cell resulting from the segmentation process includes a 10-bit CRC. Another field in each cell allows for multiplexing several connections in a single virtual connection. This feature was included for enabling support of connectionless service through the CLNAP (ConnectionLess Network Access Protocol) described in [ITU93f]. This AAL is being abandoned because of the large amount of overhead involved in it. Its function is widely replaced by AAL5, described below, which offers almost the same functionality with much fewer overhead.

AAL5.

This AAL type is addressed to the transfer of packetized data between peers without time relations, as AAL3/4. It was proposed by the LAN industry in order to achieve an adaptation layer that were both simpler and more efficient than AAL3/4. For this purpose, all “per-cell” encapsulation is removed and “per-packet” encapsulation is kept to a minimum. Thus, as instead of a 10-bit per-cell CRC, a per-packet 32-bit CRC will be used, very low error rates on the underlying ATM virtual connection are assumed.

The internal structure of AAL5 is the same as AAL3/4, that is, there are a Service Specific Convergence Sublayer and a Common Part Sublayer, both with the same scope as in AAL3/4. The CPS includes equivalent message mode and streaming services. The delimitation of a data packet is carried out by marking the cells with a bit in the ATM header, the ATM User-to-User indication (AUU), which is devoted to carry end-to-end information, as defined in [ITU93c]. When using AAL5, this bit is ‘0’ for every cell but the last one, which is ‘1’. The last cell contains the 8-byte trailer, which is the only overhead in AAL5,

including a 32-byte CRC.

From the functionality point of view, there are two major differences between AAL3/4 and AAL5. The first one is the impossibility in AAL5 of multiplexing several flows in a single virtual connection, as all cells received between two 'end cells' are considered to belong to the same packet. Thus, different strategies have to be considered for supporting connectionless service. The other difference is in the error notification; in AAL3/4, the AAL user can be notified of an errored cell as soon as occurred, thanks to the per-cell CRC; in AAL5, the user will not be notified until the receipt of the last cell, because the CRC is per-packet and resides in a trailer.

Higher-level AALs.

The possibility of defining service-specific sublayers for several AAL types allows to increase the adaption to the characteristics of particular applications. These are many proposals of mechanisms in this direction. Of some of them can be of interest of a wide audience, the ITU-T has recently defined several of these specific sublayers. All these specifications have been designed to operating over AAL5, but obviously other definitions of service-specific sublayer may rely on different standard AALs.

- *Support to signaling*. This sublayer is specified in Recommendation Q.2100 [ITU93a]. It relies on a lightweight protocol so as to provide reliable transfer of signaling data. This protocol is defined in Recommendation Q.2110 [ITU94b], and is known as SSCOP (Service Specific Connection Oriented Protocol). We will refer to this protocol later in this work, since it will be the basis of our proposals.

- *Support to Frame Relay*. This sublayer is defined in Recommendation I.365.1 [ITU93g]. It emulates the bearer service of Frame Relay networks, and can be used as an interworking strategy between Frame Relay and B-ISDN as well. Several Frame Relay connections may share a single AAL5 connection. This sublayer includes procedures for managing congestion situations.

- *Support to Connection-Oriented Network Service (CONS)*. This sublayer is defined in Recommendation I.365.2 [ITU94a]. It provides support to the network service as defined in the OSI model [ITU92]. For this purpose, like the sublayer for supporting signaling, it relies on SSCOP in order to achieve reliable transfer of data.

- *Support to Connection-Oriented Transport Service (COTS)*. This sublayer is defined in Recommendation I.365.3 [ITU95b]. It provides support to the transport service defined in the OSI model [ITU93h]. For this purpose, like the sublayer for supporting signaling, it relies on SSCOP in order to achieve reliable transfer of data.

In addition to these service-specific sublayers, some other proposals have been made in the ATM Forum, in particular for supporting MPEG video over AAL5 and for supporting voice trunking over AAL2.

1.3.4 Possibility of virtual subnetworks

In ATM networks, two levels of connections are defined: *Virtual Path Connections (VPC)*, and *Virtual Channel Connections (VCC)*. Virtual Path Connections are labeled paths which can be used to transport, process, and manage a bundle of Virtual Channel Connections, as illustrated in Figure 1.5. Thus, if the user is provided with a VP connection, he or she can multiplex several virtual channels within the VP with considerable autonomy with respect to the network. This characteristic of ATM is useful for reducing the difficulties inherent to the heterogeneity of ATM traffic, as it enables the subdivision of the traffic into more homogeneous and, therefore, more easily manageable groups. This feature is known as *virtual subnetworking*.

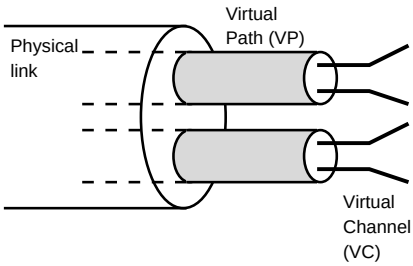


Figure 1.5: *The VP concept: Virtual Channels multiplexed within Virtual Paths.*

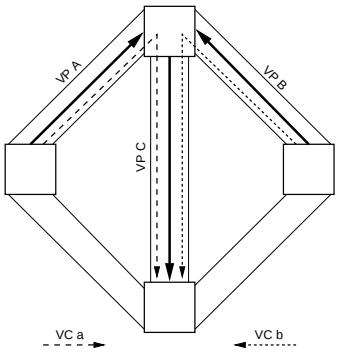


Figure 1.6: *Example of virtual subnetwork by concatenating VPs.*

Virtual subnetworks on top of ATM can be built by using two strategies, as discussed in [FGCF95]: (1) using end-to-end virtual paths, and (2) concatenating two or more virtual paths between every two sites. The use of end-to-end virtual paths is the simplest solution, and has the advantage that the user has full control of the bandwidth provided by the VP and therefore he/she can multiplex several VC in a way totally transparent to the ATM network. The main drawback of this scheme is that the virtual subnetwork may contain many virtual paths, some of which possibly sharing the same physical link and, consequently, limitations in bandwidth could appear. In addition, the scalability is somewhat poor. For this reason, the use of concatenated VPs is the approach generally adopted for supporting virtual subnetworks [FHW96, ATTD94].

Figure 1.6 shows a small example of an ATM virtual subnetwork using concatenated VPs. Two virtual channels —*VC a* and *VC b*— on an end-to-end basis within the subnetwork, which is formed by the virtual paths —*VP A, VP B, and VP C* . *VC a* is embedded in the concatenation of *VP A* and *VP C*, while *VC b* is built over the concatenation of *VP B* and *VP C*. Note that, in this case, *VP C* is shared by two VCs using different end-to-end paths, which is not possible with the scheme of end-to-end virtual paths.

In ATM virtual networks, there are two main problems to solve, as discussed in [FHW96]: (1) the topology of VPCs, and (2) the capacity allocation. The choice of a particular VPC topology greatly impacts the connection setup and switching costs, as well as the network's resilience to unexpected traffic conditions and component failures. Some constraints affect the topology, namely the configuration of the physical layout, and the maximum number of available VPIs (Virtual Path Identifiers). Regarding capacity allocation, this problem can be viewed in two levels: a *network level*, where it is determined how the network capacity is shared between VPCs, and a *VPC level*, where it is determined how much bandwidth is allocated to the individual VCCs within a particular VPC, so that the quality of service of all of them is maintained. A third problem present in ATM virtual networks is connection signaling and rerouting. As topologies can be dynamically reconfigured, protocols for this purpose may be designed. In [CS94] this

Table 1.1: *Round-trip delay and peak delivered throughput for a variety of commercial local area network environments. Platform: Solaris 2.4 on SS-20.*

Network	UDP/IP Round-trip Time (μs)	Peak TCP/IP Throughput (Mb/s)	TCP/IP Half-Power Point (Bytes)	Value of n to equate per-message & per-byte contributions
Ethernet: AMD Lance Ethernet NI and Bay Networks EtherCell	$1638 + 2.18n$	9.0	108	752
ATM: SunATM-155 NI and Bay Networks LattisCell Switch	$1261 + 0.32n$	84.3	2194	3941
ATM: Fore SBA-200 NI and ASX-200 Switch	$1432 + 0.32n$	81.2	1196	4446
Myrinet	$1492 + 0.36n$	75.3	2847	4145

topic is discussed.

1.4 Measuring communications performance

Traditionally, the performance of communications is measured in terms of consumed bandwidth, or throughput. This habit is inherited from the times when the cost of sending information through the network was so high that applications were tuned for minimizing the amount of data added to the transmitted information. The much higher speeds available today, together with the wider diversity of applications making use of networking, are leading to the revision of the validity of the sole use of throughput as *the* performance measure in communications.

1.4.1 Performance parameters

The LogP model proposed by Keeton et al in Berkeley [KAP95] suggests a number of parameters to characterize the performance of network-based applications, apart from throughput or bandwidth: overhead and latency. These parameters impact on performance on a per-message basis, regardless of its length, as opposed to throughput, which affects performance on a per-byte basis. Besides, if the network is not reliable (as happens in most general purpose networking technologies, including legacy LANs and ATM), the need of recovering from lost or errored information has an impact on performance as well.

By taking these considerations into account, we can express the performance of a network as $p = r(M + nB) = rM + rnB$, where M represents the per-message cost, and B the per-byte cost. n indicates the size in bytes of messages, and r the number of times a message has to be recovered. The per-message cost includes the time required by the protocol and the network interface to process a message, as well as the delay introduced by the switches (buffering, scheduling, etc.). Per-byte costs essentially include the transmission time. It is clear that in slow networks, the importance of the second component is more dominant but, as the first component is more difficult to reduce that the second, the dominance of the second component decreases as far as the available bandwidth increases. Table 1.1, displayed in [KAP95], illustrates this behavior.

The Half-Power Point measures the message size with which half of the peak throughput is achieved. Both ATM and Myrinet get the higher peak throughput, but rather large message sizes are required for taking actual advantage of the available bandwidth, so only applications managing large messages are really capable to achieve significant performance gains. Thus, the characteristics of messages, as well as the communications pattern of each application, will condition the sensitivity of communications to the

Table 1.2: *Average throughput achieved by real parallel programs (Mb/s).*

Parallel kernels	Average throughput (Mb/s)
CG	20.66
FT	53.20
IS	39.95
MG	7.29

various parameters, as noted in the next subsection.

1.4.2 Performance of parallel computing applications

The adequate procedure for measuring communication performance of a particular application type depends on the characteristics shown by the communications generated in the application. In [KAP95], the following issues are considered for deciding the right performance measure:

- *Overhead & latency*. Overhead includes the per-packet cost of communications in the endpoint hosts, while latency, in [KAP95], is restricted to the per-packet cost in the network. Unlike these conventions, in this work we refer to "latency" as all the per-packet costs, either in the host or in the network.
- *Communication pattern*. In operations where messages are sent in a one-way exchange between sending and receiving hosts, the performance is not seriously conditioned because it is relatively easy for the application to overlap communication and computation. In contrast, for request-response operations like Remote Procedure Call (RPC), the requester cannot resume computation until a response has been received, so in this case latency introduces a significant limit in communication performance.
- *Message sizes*. Large messages amortize per-message overhead over a large number of bytes, thus making per-byte costs and bandwidth the more important factors in determining performance. For smaller messages, per-message costs dominate since more of the communication time is spent in processing on the host than in actually transmitting the data onto the link.

In the particular case of communications from parallel computing applications, we have to determine, as far as communication pattern and message sizes are concerned, to which category they belong in order to suggest the appropriate measures. Regarding communication patterns, parallel computing applications lie in a hybrid case between one-way communications and request-response communications, since each parallel task transmits messages to other tasks in a one-way fashion, but periodically parallel tasks periodically have to wait for responses from the rest of tasks. This behavior indicates that latency will indeed impact on performance. With regard to message sizes, they depend on the particular parallel application, and may vary from very few bytes to some 260 KByte per message.

Table 1.2 shows the average throughput —computed in a multiprocessor— generated by a number of programs from the NAS benchmark suite, described in Appendix A. The highest value (53.20 Mb/s) corresponds to the kernel generating the longest messages, but even in this case the throughput is significantly lower than the peak achievable throughput with ATM and Myrinet as displayed in Table 1.1. Thus, it becomes clear that throughput is not sufficient to characterize network performance; therefore,the per-message component has to be taken into account as well. This need is also discussed in [HP96]. As a consequence, in this work we use latency —which includes all per-message components of performance— as the primary network performance measure.

2

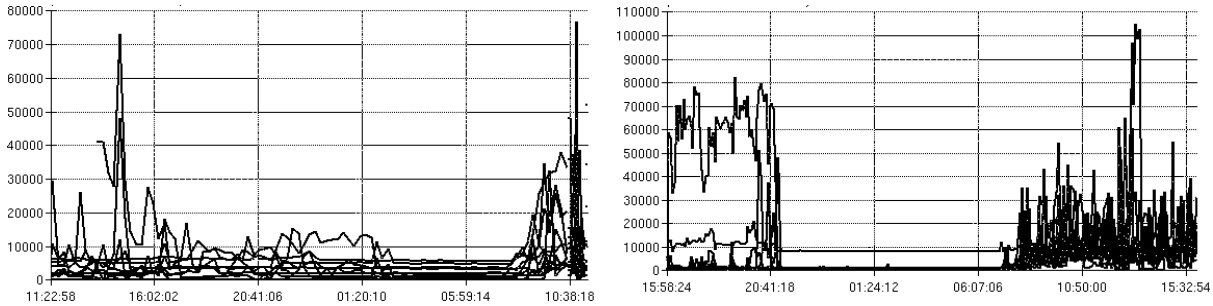
Potential performance of ATM in Networks of Workstations

This chapter analyzes the potential capabilities of ATM networks for supporting parallel computing applications. The non-specific focus of current ATM standards, in addition to other technical issues including hardware architecture, protocols and operating systems, are limiting the potential performance of these environments. We evaluate the impact of ATM by considering a possible evolution of the performance-degrading factors, as compared to the performance achieved by an Ethernet-based cluster and two multiprocessor environments. Performance degradation principally is found to be caused by delays in the hosts, although the network load and the cell loss recovery mechanisms contribute to the degradation as well. The subject of this chapter is covered by the papers [VSSPJT97] and [VSSPTJ97].

2.1 Performance under ideal conditions

The adoption of a high-speed networking technology such as ATM allows for improving performance thanks to the alleviation of the impact of communications. However, communications are just one of the factors in determining performance, so the actual improvement in performance will be related to the importance of communications in the particular parallel application. We are interested in quantifying the improvement of performance that the advent of ATM networks can provide to network-based parallel computing environments. Not only the present advantages, but also the advantages that the technological evolution will tend to achieve in the future.

In order to undertake this quantification, we consider a number of real parallel computing environments —including an Ethernet-based environment and two types of multiprocessors— and compare the performance achieved under these platforms with the results from an ATM-based environment under ideal conditions. As ideal conditions we mean: a one-switch network, no delays in the hosts and no load in the network. The only considered delay is the transmission delay. This comparison will allow us to situate the contribution of the bandwidth to overall performance. If the comparison shows that the performance is clearly inferior to multiprocessors while keeping close to the performance achieved by Ethernet-based environments, that would mean that communications do not have much importance to communications and, consequently, the proposal of higher bandwidth to networks would not lead to significant performance improvements. In contrast, a close performance to multiprocessors would indicate that increasing the available bandwidth for parallel computing applications would be sufficient for them to reach adequate



(a) Input traffic (b) Output traffic

Figure 2.1: *24-hour Ethernet traffic in the department, measured in the internal subnetwork attachment.*

Table 2.1: *Processor performance (5 CPUs).*

Environment	CPU	GFLOPS	SPECint per CPU	SPECfp per CPU
Ethernet	SuperSPARC	0.22	87.0	100.0
SGI	R10000	3.88	111.35	310.57
SP2	RS/6000	1.33	122.0	260.0
ATM	AXP 21064	1.35	84.0	101.0

performance without the cost inherent to multiprocessors.

The measurements carried out for this purpose have used a set of real algorithms that are described in Appendix A in detail: *PDE1, PDE2, SOLVER, EP, CG, FT, IS,* and *MG*. The comparison has been performed with executions of these applications over various environments. These environments use different types of CPUs, whose relative performance can be observed in Table 2.1. In the following we give a brief description of the considered environments:

- *A network of workstations based on Ethernet.* The measurements have been carried out on five SUN 4 workstations attached to the Ethernet-based department LAN. Two cases have been considered: silent Ethernet and loaded Ethernet. The former measurements were taken at night and the latter during daytime. The network activity experiences significant variations between both periods, as displayed in Figure 2.1.

- *A shared-memory multiprocessor*. A Silicon Graphics PowerChallenge multiprocessor has been used for these measures, whose interconnection network is bus-based.

- *A distributed-memory multiprocessor*. The multiprocessor used for these measurements is an IBM SP2 using a dedicated interconnection network based on the purpose-specific *high-performance switch* [S$^+$95]. multiprocessor has been used for these measures.

- *A network of workstations based on ATM*. As reasoned above, ideal conditions have been assumed for this environment. For this purpose, the ATM network has been simulated by an emulator that allows to run real parallel computing applications over a simulated ATM network. The characteristics of the emulator are detailed in Appendix B.

Table 2.2 shows the results of these measurements. For each parallel code, they include the total execution time, the time invested in communication-related issues, and the significance of the communications time The total execution time is a measure of the overall performance achieved by each application. The time invested in communication-related issues include, in addition to the actual transmission time,

the delays introduced by the application in order to build the transferred messages. This parameter, as well as the measure of its significance with respect to the total execution time, indicates the impact of communications on performance. According to the results, we observe that:

- The lowest total execution time is experienced by the shared-memory multiprocessor in most cases. In *PDE1* and *SOLVER*, the best performance corresponds to the ATM environment, and in *CG* the lowest execution time is experienced by the distributed-memory multiprocessor. In addition, the total execution time in the ATM-based environment is much better than in Ethernet-based environments.

- Analogously, the time invested in communications achieves the lowest values in the SGI measure in most cases, except for the *CG* and *FT* codes, where the lowest communication times are found in the SP2 measure. Again, as expected, the measured times in the ATM-based environment are much better than in Ethernet-based environments.

- Despite the absolute values, the contribution of communications to performance is very high in the SGI measure for some applications. In general, nevertheless, the impact of communications is higher in Ethernet-based environments, lower in both multiprocessors, and intermediate in the ATM-based environment.

These results show that the performance achieved in the ATM-based environment under ideal conditions is closer to multiprocessors than to Ethernet-based environments, even better in some particular cases. Two conclusions may be drawn from these observations: (1) networking introduces a significant contribution to performance, so the reduction of the delays involved with communications leads to important improvements; and (2) communications in current multiprocessors still involve excessive delays, which make some parallel computing applications to perform like environments that in theory are less powerful. We have to note, however, that not all the differences can be explained with the different behavior of communications in each environments. Other issues, including the different performance of the respective CPUs and the way how communications are distributed along the execution time, have to be considered as well.

The contribution of the different CPUs in each environments is clearly significant, as illustrated in the measure of the *EP* parallel code, where no interprocess communications take place. Other issues impacting on performance are found in the length distribution of messages, as well as the temporal distribution of messages, in each parallel code, both of which are discussed in Appendix A. These reasons can explain the behaviors of *PDE1, SOLVER, CG,* and *FT,* as well as the rest of irregularities appearing in Table 2.2. All these factors may hide the contribution of latency to communications time by facilitating the delivery of data to the receiver prior to their consumption.

In summary, we observe that the reduction of the relative importance of the delays associated with ATM can accelerate the convergence of network-based environments with multiprocessor technology. Obviously, multiprocessors will always remain a step forward network-based environments, but the step will become shorter with the time. Applications with fine- or medium-grained parallelism are most likely to require dedicated environments such as multiprocessors, while most of applications with coarse-grained parallelism will suffice with ATM-based environments. The fewer the bottlenecks in communications, the higher the range of applications that can be supported. As a consequence, it is interesting to attempt the reduction of the overheads currently present in ATM-based environments that are precluding the approximation to the ideal conditions considered in this section. For this purpose, we have to determine where these overheads reside and which relative importance they represent.

Table 2.2: *Performance achieved by real algorithms over different environments (ATM under ideal conditions).*

Measurement		Ethernet (silent)	Ethernet (loaded)	Multiproc. (SGI)	Multiproc. (SP2)	Ideal ATM (simulated)
PDE1	Total time (s)	427	509	47.2	34.4	24.5
	Comms. time (s)	158	196	7.0	8.61	12.0
	% Comms.	37.0%	38.5%	14.8%	25.0%	49.0%
PDE2	Total time (s)	407	671	31.7	47.0	32.8
	Comms. time (s)	162	321	3.1	6.66	8.25
	% Comms.	39.8%	47.8%	9.78%	14.2%	25.1%
SOLVER	Total time (s)	341	454	28.7	30.1	13.2
	Comms. time (s)	8.6	283	0.6	2.83	1.4
	% Comms.	2.52%	62.3%	2.09%	9.40%	10.6%
EP	Total time (s)	53.6	68.2	9.3	42.7	38.5
	Comms. time (s)	N/A	N/A	N/A	N/A	N/A
	% Comms.	N/A	N/A	N/A	N/A	N/A
CG	Total time (s)	32.5	39.7	1.7	1.66	5.84
	Comms. time (s)	16.6	18.9	0.8	0.29	2.8
	% Comms.	51.1%	47.6%	47.1%	17.5%	47.9%
FT	Total time (s)	46.5	52.4	3.0	8.37	16.4
	Comms. time (s)	35.7	40.7	1.7	0.59	11.3
	% Comms.	76.8%	77.7%	56.7%	7.05%	68.9%
IS	Total time (s)	46.8	54.4	1.6	1.98	3.6
	Comms. time (s)	33.6	39.1	0.3	0.41	0.9
	% Comms.	71.8%	71.9%	18.8%	20.7%	25.0%
MG	Total time (s)	65.9	107	7.0	8.14	16.8
	Comms. time (s)	27.8	48.7	1.0	1.32	3.8
	% Comms.	42.2%	45.5%	14.3%	16.2%	22.6%

2.2 Bottleneck analysis in ATM-based environments

For assessing the relative impact of bottlenecks over performance, we have classified the usual overheads according to the situation when their are produced, and then we artificially vary the delay introduced by each of these situations in order to study the response of the ATM-based environment. In particular, we consider that ATM-based environments can suffer from three causes of bottleneck: (1) the network size; (2) the network load; and (3) the delay introduced in the host. We have performed three experiments covering these issues. The first experiment measures the influence of the network size and the network load. The second experiment estimates the incidence of limited buffering under different network loads and sizes. Finally, the third experiment, for several network loads, explores the performance degradation introduced by different values of delay in the hosts.

2.2.1 Experiment 1: Impact of network load and network size

The size of the network is a potential cause of performance degradation, because of the accumulation of transmission and, principally, buffering delays. As the buffering delay is more important the higher the load in the network, we observe that the influences of network load and network size are closely related. To illustrate the influence of these magnitudes, we have measured the performance achieved by two parallel algorithms chosen from those described in Appendix A, *PDE1* and *PDE2*, for different network sizes and network loads. The measurements have been obtained by means of the ATM emulator described in Appendix B. Note that in the present experiment we do not assume ideal conditions. Indeed, no host delays have been considered, but network load has been set to $\rho = 0.0$, $\rho = 0.3$, $\rho = 0.6$, and $\rho = 0.7$. The network sizes have been modeled by considering the network as a chain of switches of variable length.

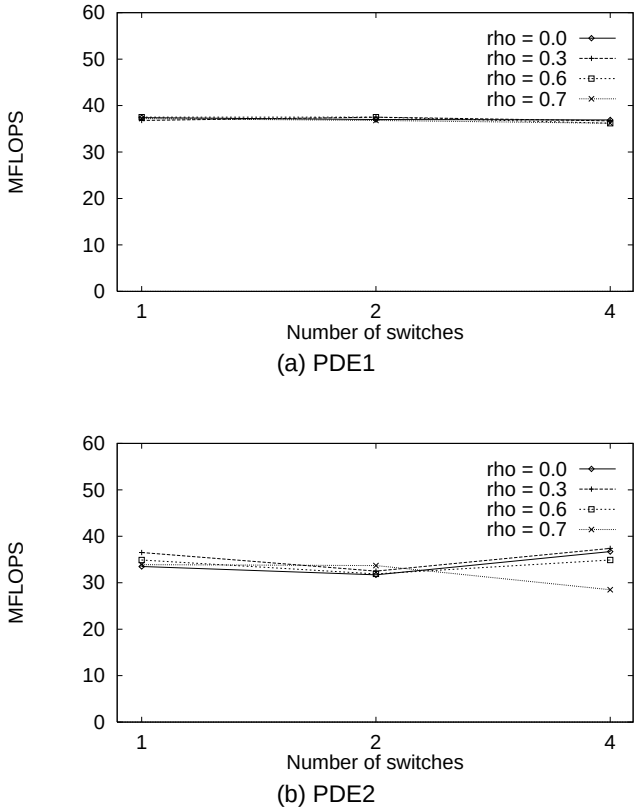


Figure 2.2: *Performance for different network sizes.*

In particular, sizes of 1, 2 and 4 switches have been measured. The results are displayed in Figure 2.2.

The impact of both the network size and the network load on performance is very low in the case of *PDE1*. In contrast, the results of *PDE2* exhibit a slight dependence on the network, specially when the cross-traffic load is high. This behavior is related to the different message lengths and temporal distributions observed in *PDE1* and *PDE2*, (see Table A.1 in Appendix A). In order to highlight to effects of these characteristics, we have carried out a simple experiment concerning the influence of the length distribution and the network load on the time required to transfer data.

The experiment consists of the measurement by simulation of one synthetic 5-second cell sequence of 2500 cells crossing a switch suffering from background traffic. Two intensities of background traffic have been considered: $\rho = 0.7$ and $\rho = 1.2$, and is generated by four identical ON-OFF sources whose parameters are tuned to produce cell sequences with the desired value of ρ. The 2500 cells are grouped in bursts, and we have repeated the measurement for different burst sizes. The results depicted in Figure 2.3 show that the sensitivity to increases of network load is higher when bursts of cells are short. The figure also shows that the lower the network load the more apparent the influence of burst size on latency. The question is now whether this trend occurs in traffic from real parallel applications.

The results shown in Figure 2.3 are consistent to the behavior experienced by *PDE2* as depicted in Figure 2.2, according to the characteristics of communications, since *PDE2* generates cell sequences with low burst sizes. Thus, the higher sensitivity to background traffic occurring in *PDE2* involves an increase in the time required for transferring data and, consequently, to performance degradation. Note that in this experiment the cost of recovering lost cells has not been taken into account. Loss recovery can introduce important increases in latency, depending on the load of the network and the characteristics of

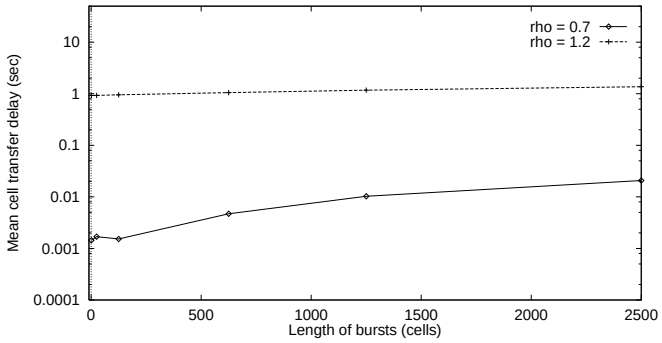


Figure 2.3: *Impact of network load and burst size on communications latency.*

Table 2.3: *Performance of different situations for the ATM-based environment, with respect to multiprocessors.*

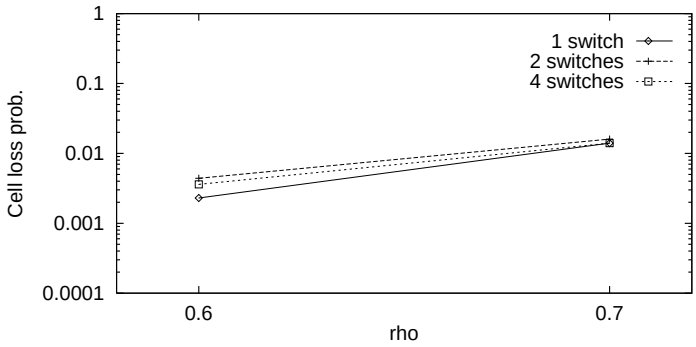
Measurement		Multiproc. (SGI)	Multiproc. (SP2)	ATM (1 sw, silent)	ATM (1 sw, loaded)	ATM (4 sw, loaded)
PDE1	Total time (s)	47.2	34.4	24.5	24.6	25.4
	Comms. time (s)	7.0	8.61	12.0	12.0	12.8
	% Comms.	14.8%	25.0%	49.0%	48.8%	50.4%
PDE2	Total time (s)	31.7	47.0	32.8	32.5	38.6
	Comms. time (s)	3.1	6.66	8.25	9.75	14.5
	% Comms.	9.78%	14.2%	25.2%	30.0%	37.6%
CG	Total time (s)	1.7	1.66	5.84	5.99	6.01
	Comms. time (s)	0.8	0.29	2.8	3.03	2.94
	% Comms.	47.1%	17.5%	47.9%	50.6%	48.9%
FT	Total time (s)	3.0	8.37	16.4	17.7	18.1
	Comms. time (s)	1.7	0.59	11.3	13.0	13.3
	% Comms.	56.7%	7.04%	68.9%	73.4%	73.5%

the protocols.

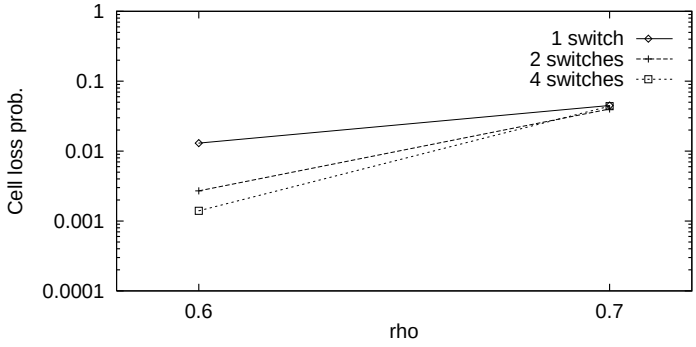
In order to realize the importance of both the network size and the network load, four representative algorithms have been chosen, according to their characteristics of communications, and the performance achieved by an ATM-based environment with the minimum and the maximum size of the network, for a load $\rho = 0.7$ is displayed in Table 2.3. These results are compared in the same table with the corresponding issues in Table 2.2. Excepting the case of *CG*, the influences of both the load and the size of the ATM network are significantly confirmed. The contribution of each issue depends on the characteristics of each algorithm. In particular, the more bursty the traffic generated by the algorithm, the more influential the network size as compared to the network load. The importance of the contribution from communications on performance also grows with a similar pattern. The high burstiness of *CG* explains its special behavior, since the impact of the network size has very low significance.

2.2.2 Experiment 2: Estimation of cell loss

In the previous experiment, as it reflects an ideal case, no cells from parallel applications are supposed to be lost. In real environments, the cost of recovering from cell loss can significantly degrade the performance of the ATM-based environment. In order to determine the importance of cell loss, we have measured the amount of cells each switch would lose for a finite buffer capacity limit. As no transport protocol is emulated, the excess cells from parallel applications are computed at each switch but not actually discarded. As a consequence, the results do not represent the exact cell loss ratio but an approximation.



(a) PDE1



(b) PDE2

Figure 2.4: *Cell loss for different configurations.*

Figure 2.4 displays the achieved cell loss, computed as the average of the excess cells, for the network loads producing cell loss, and several network sizes. The results show that the network load has a strong influence on losses: when ρ grows from 0.6 to 0.7, the cell loss ratio increases about an order of magnitude. In actual systems, these lost cells increase and therefore the overall performance of the ATM environment will degrade. Figure 2.4 also shows that the network size affects the results only for $\rho = 0.6$, which reflects that the cell loss is more homogeneous for high values of ρ. The higher sensitivity of *PDE2* to the cross-traffic load exhibited in Experiment 1 is reflected also in the present experiment. In addition to the higher cell loss ratio experienced by *PDE2* for $\rho = 0.7$, the influence of the network size is also higher in *PDE2*. This behavior can also be attributed to the particular characteristics of communications in *PDE1* and *PDE2*.

Another observation from Figure 2.4 is the fact that, when $\rho = 0.6$ cell loss depends on the network size in the sense that the larger the network the lower the loss. This behavior does not apply to $\rho = 0.7$. When several switches have to be crossed, it is easier that the background traffic leaves free slots in the buffer of some switch at some instants. When $\rho = 0.7$, it is more difficult to get these free slots and, consequently, the cell loss probability becomes independent of the network size.

As a conclusion for the experiment, we observe a sensible influence of the network load on the cell loss probability which significantly interacts with the network size unless the load is extremely high. Although the Experiment 1 has not shown an important influence of network size and network load on performance, we have to note that one of the sources of performance degradation, namely the need of retransmitting lost information, was not considered. The results of the present experiment show that the likelihood of having to perform retransmissions can be high, and that this likelihood depends more on network load than on network size. Thus, we observe that network load can an important contributor to the bottleneck, although

not as influential as endpoint host delay, as shown in the next experiment.

2.2.3 Experiment 3: Impact of host delay and network load

The previous experiments have been addressed to the study of delays originated within the network. Unlike them, in the present experiment we consider the delays occurring within the host, immediately before the actual transmission and after the actual reception of data. In particular, these delays include the cost of protocol processing and the overhead in the host-network interface. All these delays are expected to reduce in the future, so it is interesting to assess how this reduction will be translated to the performance of ATM-based parallel computing environments.

In the present experiment, we have measured the performance of *PDE1* and *PDE2* over a simple one-switch ATM network by means of the ATM network emulator used in all the preceding experiments. As this emulator allows to configure the delay introduced by the hosts, we have performed measurements with four different values of the host-introduced delay. These values range from 0 to 500 microseconds, and cover situations from the ideal conditions —0 μs— to a current TCP/IP-based environment —500 μs. In addition to the host delay, we have also considered different background traffic intensities in order to assess how the network load interact with the effects of the host delay.

As shown in Figure 2.5, the influence of the host delay affects the performance achieved by both *PDE1* and *PDE2* in a similar way, with an exponential pattern. This means that the improvements in performance achieved by the reduction of delays in the host will increase their significance as long as the importance of this reduction increases. As in Experiment 2, *PDE1* has little sensitivity to network load, in contrast to *PDE2*. Note that the degradation in *PDE1* achieves a higher impact than in *PDE2*, due to the longer time required to process a message from *PDE1*. As shown in Table A.1, the messages generated by *PDE1* are much longer than those from *PDE2*. In summary, it is observed that the host delay is the primary contributor to performance degradation.

Analogously as in Experiment 1, we have selected four representative algorithms in order to assess how host delay affects execution and communication times. Likewise, the performance values corresponding to the multiprocessors and to the ATM-based environment under ideal conditions are taken from Table 2.2. These values are displayed in Table 2.4, whose last column represents the behavior achieved with a host delay of 500 μs and a network load of $\rho = 0.7$. As expected, the contribution of communications to performance is very high, much higher than the impact of the network size displayed in Table 2.2. The relationship between this impact and the characteristics of communications —specially burstiness— is also confirmed by the results. The applications experiencing higher burstiness —*PDE1* and *FT*— exhibit a higher sensitiveness to both network load and endpoint delay than the applications with lower burstiness —*PDE2* and *CG*, which manifests as a higher impact of communications on the total performance. All these results reflect the strong impact of endpoint host delays on performance. Thus, the reduction of the impact of this issue is crucial for allowing ATM-based parallel computing environments to reach the asymptotic performance as discussed in Section 2.1.

2.3 Discussion

Until recently, the idea of supporting parallel computing applications by means of commodity workstations interconnected to a general-purpose network has not had a competitive viability because the limited bandwidth inherent to traditional networking technologies has represented a hard bottleneck. However, the introduction of ATM with its comparatively higher bandwidth leads to question this viability. Nevertheless, the comparison of multiprocessor with environments based on commercially available ATM equipment

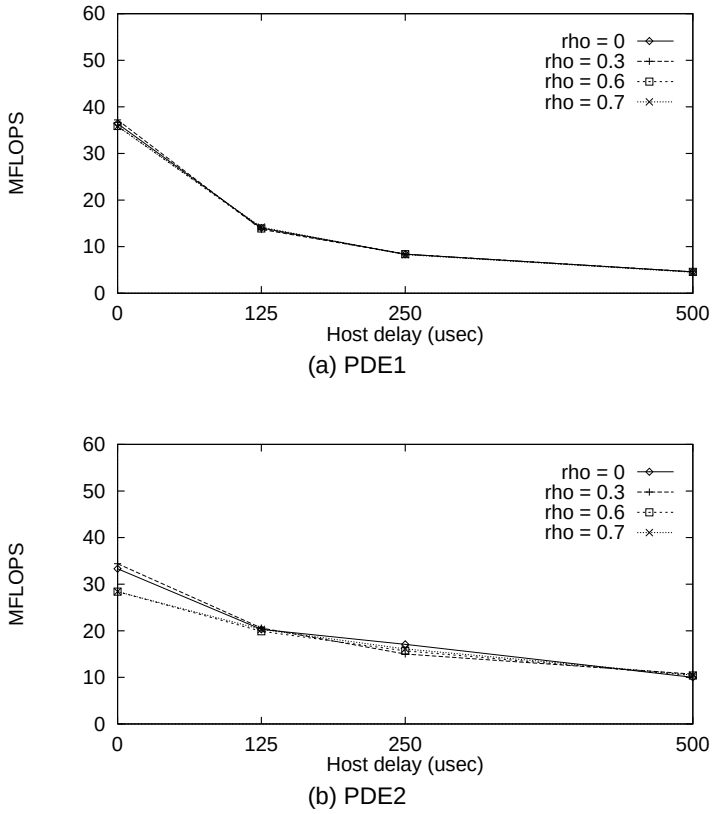


Figure 2.5: *Performance for different host delays.*

Table 2.4: *Performance of different situations for the ATM-based environment, with respect to multiprocessors (1 switch only).*

Measurement		Multiproc. (SGI)	Multiproc. (SP2)	ATM (no delay, silent)	ATM (no delay, loaded)	ATM (500μs, loaded)
PDE1	Total time (s)	47.2	34.4	24.5	24.6	199.5
	Comms. time (s)	7.0	8.61	12.0	12.0	187.0
	% Comms.	14.8%	25.0%	49.0%	48.8%	93.7%
PDE2	Total time (s)	31.7	47.0	32.8	32.5	103.0
	Comms. time (s)	3.1	6.66	8.25	9.75	81.1
	% Comms.	9.78%	14.2%	25.2%	30.0%	78.7%
CG	Total time (s)	1.7	1.66	5.84	5.99	144.5
	Comms. time (s)	0.8	0.29	2.8	3.03	104.9
	% Comms.	47.1%	17.5%	47.9%	50.6%	72.6%
FT	Total time (s)	3.0	8.37	16.4	17.7	151.7
	Comms. time (s)	1.7	0.59	11.3	13.0	147.2
	% Comms.	56.7%	7.04%	68.9%	73.4%	97.0%

shows that the performance is still far from what multiprocessors achieve, but the issues limiting the performance may be tied to technological reasons that may evolve over time. For this reason, a study of the viability of ATM-based environments as parallel computing platforms requires us to abstract from these transient technological issues and consider an asymptotic case instead. The results indicate that ATM-based environments are a worthy platform to consider as their performance without accounting for the transient overheads is very close to multiprocessors, so bandwidth is no longer a problem for achieving competitive performance. However, this *potential* performance is not reachable without the minimization of the bottlenecks present in current ATM environments.

The rest of experiments in this chapter have been addressed to determine the relative impact of the main causes of bottleneck in ATM-based environments: network size, network load, and endpoint host delay. The results have shown that the endpoint host is, with difference, the most influential bottleneck, and is due to protocol and driver processing, as well as the host-network interface. Although in a lower extent than the endpoint host delay, the load of the network significantly contributes to degrade performance, since the mechanisms to recover from lost cells introduce additional latency. The contribution of the network size is less important, but for certain applications and topologies could be more significant, specially if very large networks and/or large switch buffers are used.

As a conclusion, the results indicate that any strategy aiming at taking full advantage of ATM has to reduce the combined effects of host delays and network load. This goal can be achieved with the adoption of enhanced host interfaces, and with the improvement of the protocol stack. Many research works are dealing with both issues for this purpose. Regarding host-network interfaces, it is a very active research area. Many proposals exist in the literature, all of them triggered in reducing the latency introduced by the interface itself and the device driver controlling it. See [Dav93, TS93] as examples of studies on this subject. Particularly, there are proposals oriented to systems supporting parallel computing, for example the interface described in [vEBBV95]. In this work, nevertheless, is focused on protocol-related solutions. Within this scope, many other researchers have made suggestions about how an efficient ATM-oriented protocol stack should behave, whose contributions are summarized in the next chapter. The present thesis introduces new approaches which take into account the network load induced by background traffic. In order to achieve as good performance as possible, they attempt to exploit the specific features of ATM.

3

Strategies for introducing ATM in parallel environments

As shown in Chapter 2, ATM-based parallel computing environments are potentially capable of providing satisfactory performance to parallel applications. Many researchers have also had this intuition, and therefore several strategies have been proposed to introduce ATM in parallel computing environments. In this chapter, we present a summary of the strategies addressed to parallel computing over ATM networks that is an extended version of [VSSP96a]. The ideas guiding each of these strategies have been combined in the proposal of a model for the integrated network architecture that is discussed in Chapter 4.

3.1 Traditional protocols over ATM

The attractiveness of parallel computing over LANs has lead to some studies of both the performance of workstation clusters over legacy networks and their limitations due to this assumption, as in [FW94], where several cluster environments, based on legacy LANs, are compared to multicomputers. The achieved performance can potentially be improved with the introduction of a high-speed network technology such as ATM. In order to enable full compatibility of applications, it is possible to replace the legacy networking technology with ATM. There are two popular strategies for achieving this goal: *LAN Emulation* and *IP over ATM*. The performance of the resulting protocol stack has been measured in a number of studies.

3.1.1 Interfaces between existing protocols and ATM

Enabling current protocol stacks to support ATM without significant changes requires to deal with: (1) an interface at some layer of the stack that it is common to both ATM and legacy network, and (2) the interoperation of ATM-based segments with other network segments that are still based on legacy technologies. A good report on the state of the art in both subjects can be found in [All95]. For these purposes, there are two popular strategies, both setting the interface at level 3: *LAN Emulation* and *IP over ATM*. LAN Emulation is a protocol defined by the ATM Forum [ATM95] in which traditional transport protocols contemplate ATM as a 802.x MAC technology like Ethernet and Token Ring. IP over ATM is based on an LLC-level encapsulation, whose schemes are defined in the RFC 1483 [RFC93].

The LAN Emulation protocol defines a service interface for higher layer (that is, network layer)

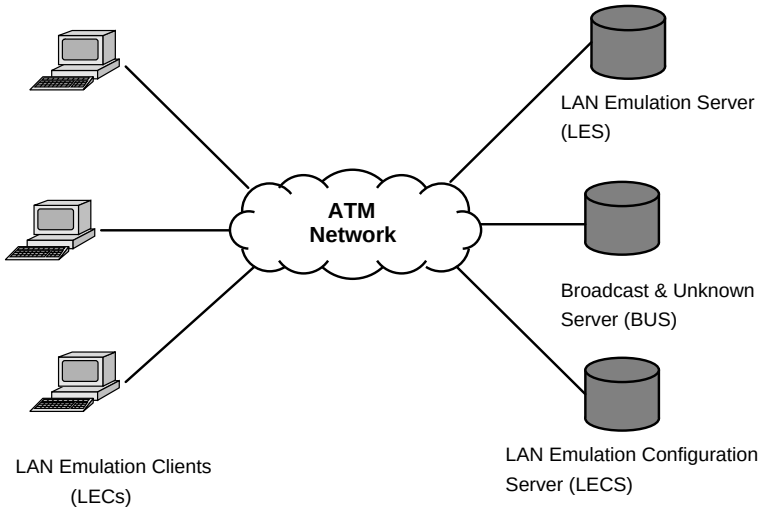


Figure 3.1: *LAN Emulation architecture.*

protocols, which is identical to that of existing LANs, and that data sent across the ATM network are encapsulated in the appropriate LAN MAC packet format. Thus, the emulation involves the interface only; no attempt is made to emulate the actual medium access protocols of the specific LAN concerned (802.3 or 802.5). As the LAN emulation service presents the same interface of existing MAC protocols to network layer drivers, no changes are required in those drivers.

Figure 3.1 depicts the ATM Forum's LAN Emulation architecture. It is based upon the overlay model, so that the LAN Emulation protocol can operate transparently over the ATM network, using only standard ATM signaling procedures. The basic functions of the LAN Emulation protocol is to resolve MAC addresses into ATM addresses. In traditional LANs, address resolution is accomplished by broadcasting the destination address to the LAN. With ATM, the mechanism is more complex. Four entities cooperate in the protocol:

- *LAN Emulation Client (LEC).* LECs are the entities in the end systems. Each LEC is identified by a unique ATM address, and is associated with one or more MAC addresses.
- *LAN Emulation Server (LES).* There is one LES per emulated LAN, and includes the control function. Each LES is identified by a unique ATM address.
- *Broadcast and Unknown Server (BUS).* The BUS is a multicast server that is used for implementing broadcasting and multicasting within the emulated LAN, including the search of unknown MAC addresses. It is identified by a unique ATM address and associated to the MAC's broadcast address.
- *LAN Emulation Configuration Server (LECS).* There is a LECS per administrative domain, and contains information assigning individual LECs to particular emulated LANs under its domain.

The LECs are aware of their own ATM address by consulting the LECS (through well-known ATM address or permanent connection). The LECS also provides information about the emulated LAN, including the ATM address of the LES. Then, the LEC can register its own MAC and ATM addresses with the LES. Thus, any LEC can know the corresponding ATM address to a particular MAC address through the LE-ARP (LAN Emulation Address Resolution Protocol). This procedure is used to find out the ATM address of the BUS (corresponding to the MAC broadcast address). The BUS then typically adds the soliciting LEC as a leaf in a point-to-multipoint connection.

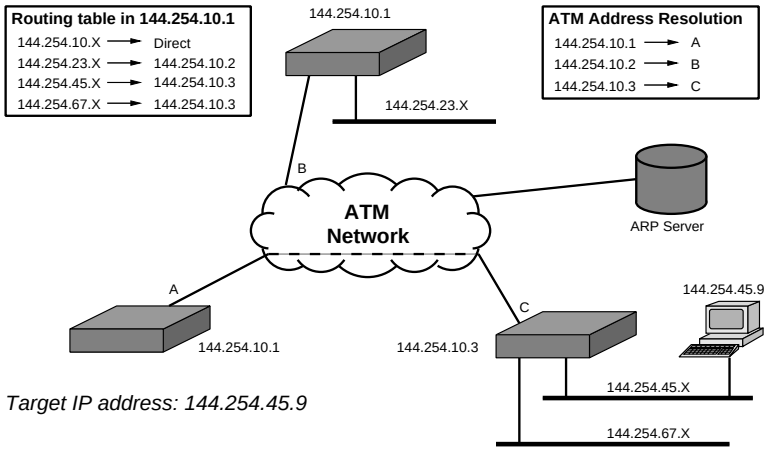


Figure 3.2: *IP routing across an ATM network.*

During data transfer, the sender LEC gets the ATM address for the destination MAC address by connecting to the LES through the LE-ARP. If the destination belongs to the same emulated LAN, there will be an immediate response. If not, the destination may be located behind a bridge (it might even be non-ATM). Thus, the LE-ARP asks for the address to all LECs in the emulated LAN (through the BUS), in order to get the ATM address of the responsible bridge. When the target ATM address is finally known, a virtual circuit can be set up (if not already established) through which data transfer will take place.

As far as IP over ATM is concerned, there are two main components: a *packet encapsulation* procedure and an *address resolution* procedure. Packet encapsulation occurs at the LLC level, and allows for a received AAL5 packet to determine which application or higher layer protocol entity should receive it. The IETF (Internet Engineering Task Force) defines in RFC 1483 [RFC93] two strategies for doing this: LLC/SNAP encapsulation, where the IP packet is prefixed by a standard LLC/SNAP header, and VC multiplexing, in which only a single protocol is carried across a single ATM connection.

Prior to transfer data, it is necessary to resolve IP addresses to their corresponding ATM addresses. For this purpose, the RFC 1577 [RFC94] defines a protocol to support automatic address resolution. Each IP subnetwork can contain one or several Logical IP Subnets (LIS), each of them being associated to a single ATM network. Each LIS includes a so-called ATMARP (ARM Address Resolution Protocol) server. Clients wishing to communicate first connect to this ATMARP server in order to register and ask for the ATM address corresponding to the destination IP address —all clients within a LIS know the address of the ATMARP server in advance. If the destination lies within the same LIS, the ATM connection can already be set up. If not, the ATMARP will provide the ATM address corresponding to a default router of the IP subnetwork. Figure 3.2 shows an example of the interactions between IP and ATM networks, according to RFC 1577.

3.1.2 Performance studies

In [LDTM95], the performance achieved with traditional LAN technologies has been measured and compared to that of an equivalent environment when the traditional technology is replaced by ATM.

The performance achieved by the replacement of traditional LAN technologies by ATM is analyzed in [LDTM95, TL93]. Table 3.1 shows the results presented in [LDTM95], where the performance over 100-Mb/s-ATM is compared to Ethernet and FDDI. In the three cases, the messages are generated by a UNIX socket API (Application Programming Interface). The throughput measurement has been computed

Table 3.1: *Performance under sockets and TCP/IP.*

Technology	Capacity (Mb/s)	Throughput (Mb/s)	Startup latency (s)
Ethernet	10	8.40	1.053×10^{-3}
FDDI	100	17.20	1.833×10^{-3}
ATM	100	16.72	1.960×10^{-3}

Table 3.2: *ATM, Ethernet and FDDI under a simple RPC protocol.*

Activity	RPC time (μs)					
	Ethernet (10 Mb/s)		FDDI (100 Mb/s)		ATM (140 Mb/s)	
	Short	Long	Short	Long	Short	Long
System Calls	123	671	153	280	108	560
Interrupt Handling	51	51	112	126	34	37
Total Software	174	722	265	406	142	597
Controller Latency	51	52	97	164	16	88
Time on the wire	115	1278	9	127	6	91
Total Latency	340	2052	371	697	164	776
Software Speedup	-	-	0.6	1.8	1.2	1.2
Hardware Speedup	-	-	0.5	0.3	3.2	0.6
Network Speedup	-	-	12.7	10.0	19.1	14.1
Global Speedup	-	-	0.9	3.0	2.1	2.6

by considering very large messages so that the measurement corresponds to the maximum achievable throughput. In contrast, the latency measurement has been carried out with very short messages —just four bytes, in order to represent the minimum latency experienced by any message. The throughput achieved by FDDI and ATM are similar and significantly higher than the throughput of Ethernet. In contrast, the latency experienced by Ethernet is clearly lower than that of FDDI and ATM, which in turn are similar. Despite the similarities between FDDI and ATM measurements, the performance is always slightly better in the case of FDDI, due to the higher overhead involved in ATM.

Note that the Ethernet network experiences a high utilization —84%— as opposed to FDDI and ATM —approximately 17%. In the former case, the bottleneck is found in the network, while for FDDI and ATM the protocol stack implementation is precluding the achievement of better utilizations. Thus, the protocol stack becomes a bottleneck when the available bandwidth increases. The slightly lower throughput that is experienced in ATM with respect to FDDI means that the aggregate impact of overheads on performance is slightly more apparent in ATM that in FDDI, basically due to the need of segmenting data in small cells that is inherent in ATM.

Table 3.2 shows the results presented in [TL93], where the latency experienced by a lightweight RPC (Remote Procedure Call) protocol is analyzed with Ethernet, FDDI and ATM, as before. In particular, the different contributions to latency are separately studied, and two sizes of messages have been considered, which are labeled “short” (about 50 bytes) and “long” (1500 bytes). The speed improvements achieved by each contribution to latency with respect to the performance over Ethernet have been included in Table 3.2. These results show that although the bandwidth with ATM achieves a gain of 14 for short messages and 19 for long messages, the effective speed increases with a factor of only 2. The latency ratio software to time-on-wire is higher for ATM and FDDI than for Ethernet, highlighting the importance of the host processing, as was shown above. In addition, ATM involves significant overhead due to the need of segmenting and reassembling user packets into cells that, consequently, is closely related to the length of user packets. On the other side, each FDDI frame includes a fixed overhead of 20-32 bytes —depending on the addressing scheme— regardless of the length of the information field, which is significantly larger than the 5 bytes per ATM cell. For this reason, the performance for short messages is clearly better in

ATM because fewer bits are injected to the network. In contrast, the performance for long messages can be better in FDDI than in ATM because the amount of overhead bits remains 20-32 bytes in FDDI while in ATM they grow to 160. Thus, unlike for short messages, the information-to-overhead ratio is more favorable in FDDI.

A full advantage of ATM technology will only be achieved in the immediate future with the minimization of the bottlenecks revealed by the measurements in Tables 3.1 and 3.2. The following sections discuss several approaches published recently in the literature which aim at this objective, as far as software is concerned. In addition to these advances, progress in hardware issues is mandatory, so research on high-speed host-network interfaces is currently very active.

3.2 Introducing an ATM-specific API

Although the absolute time required for protocol processing could be reduced by the increased processing capacity of workstations, the relative impact of latency on the overall time may increase. This fact demands strategies to reduce this impact. Intuitively, one of the solutions to overcome the overhead introduced by the protocol structure is to bypass it, allowing for applications to directly access ATM through an ATM-specific API. Several papers are devoted to the study of the ATM API behavior [LDTM95, ZG95, DSBC95]. These papers compare the performance of ATM API against other interfaces and evaluate its integration with existing software mechanisms for supporting parallel computing over LANs. In particular, two approaches for the integration of an ATM API appear: (1) Leaving some transport-layer functionality to applications, without modifying any underlying system software such as operating system and message-passing libraries, and (2) Modifying message-passing libraries so as to obtain ATM API-specific implementations. Both approaches are discussed in the following.

3.2.1 The ATM API

APIs (Application Programming Interfaces) allow to hide the complexity of the network layer from the application layer [Ros95]. Examples of typical APIs are the socket libraries in UNIX systems, RPCs (Remote Procedure Calls), etc. As far as ATM is concerned, two trends in specifying APIs have appeared: (1) *To hide ATM from the API and overlying protocols*, by using procedures such as LAN Emulation and IP over ATM, explained in the previous section, and (2) *Using ATM-aware, vendor-specific APIs*, which enable ATM-specific operations by relying on the architecture depicted in Figure 3.3. The ATM Forum recommends that all vendors delivering APIs subject to a common semantic specification, defined in [ATM96b]. Recently, Microsoft and Intel have jointly developed the Winsock-2 API, which includes ATM-Forum compliant procedures to natively operate over ATM networks [Int96]. So far, probably the most used ATM API is the one supplied by Fore systems, a principal vendor of ATM equipment, which is described in [BCS93] and summarized in the following.

Fore Systems supplies with its host interfaces a software module addressed to applications requiring to exploit ATM-specific capabilities including bandwidth reservation, selection of a specific AAL, multicasting, etc. Fore's ATM API follows the client-server model of distributed computing. Both unidirectional and bidirectional point-to-point connections, as well as multicast connections from one sender to multiple receivers. The ATM API is implemented by a subroutine library, with support from the ATM device driver. Depending on the platform, the communications between the subroutine library and the device driver takes place through a Streams (System V) or Sockets (BSD) interface, so a potential bottleneck appears in the user-kernel boundary, whose effects will be demonstrated in the measurements described below. As these details are hidden to the programmer, the interface described in the following is portable across platforms.

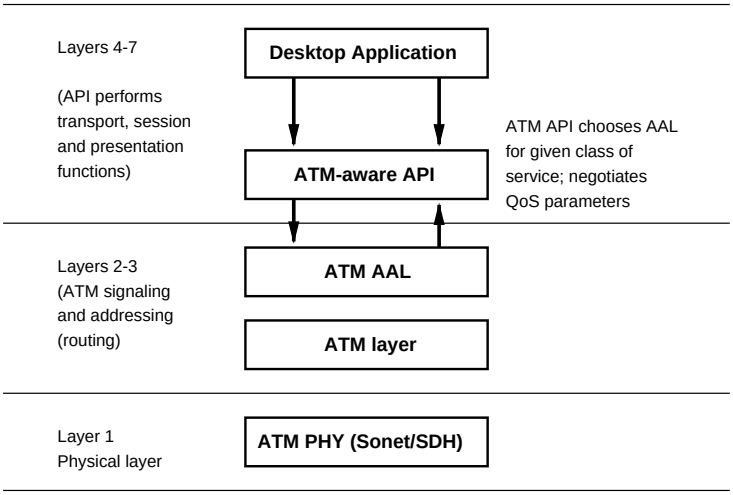


Figure 3.3: *Native ATM API-based architecture.*

A more complete description of each routine function is available on the on-line manual included with the library [For94].

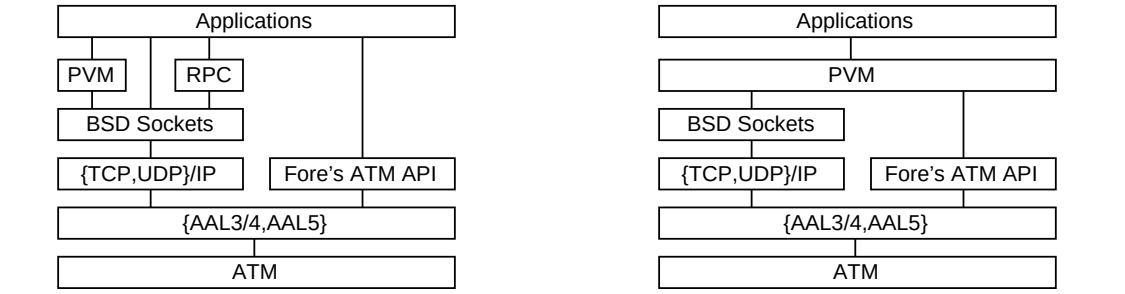
A client program opens an ATM connection by calling the `atm_connect` routine, whose arguments specify the destination ATM address, the desired and minimum acceptable resources, the AAL to use and the type of connection (unidirectional, bidirectional, or multicast). The opened connection is bound to a UNIX descriptor. A server program uses the `atm_listen` operation to receive requests from incoming ATM connections. The server can the choose to accept or reject the connection. The `atm_accept` operation allows the server to place additional restrictions on the resources it is willing to commit to the connection; these restrictions are propagated back to the client.

Data are transferred by giving `atm_send` and `atm_receive` a data buffer and waiting for the transfer to complete. One data packet is transferred on each call. The maximum size of the packet depends on both the AAL selected for the connection and the constraints inherent to the underlying UNIX Sockets or Streams implementation. The data given with `atm_send` are segmented according to the selected AAL, and each cell is prefixed with the outgoing VCI (Virtual Channel Identifier) for its connection as it is transmitted. On the receiving side, the data are reassembled and delivered via the connection descriptor corresponding to the incoming VCI. This implementation of the ATM API assumes the VPI (Virtual Path Identifier) as 0. Note that Fore's ATM API does not include functions not defined in the ATM or AAL such as retransmission of lost cells or flow control. These function have to be provided by the application.

3.2.2 Performance without an adapted message-passing library

In [LDTM95], a comparative study of four APIs has been carried out: PVM, RPC, BSD sockets and the ATM API supplied by Fore Systems, described above. Figure 3.4(a) shows how the protocol structure looks like at all cases. For a simple echo test program, these combinations provide the performance values shown in Table 3.3.

The main result from Table 3.3, obtained from [LDTM95] is that the ATM API achieves both the highest throughput and the lowest latency among the measured interfaces. It is very important to remark, however, that the functionalities provided by these APIs are not comparable. As opposed to RPC, PVM and BSD sockets where fully reliable communications are supported, the ATM API offers no additional features with respect to ATM, and therefore in many cases they have to be provided by upper layer



(a) Protocols considered by Lin et al [LDTM95]. (b) Protocols considered by Dowd et al [DSBC95].

Figure 3.4: *Protocol stacks for ATM-API-based architectures.*

Table 3.3: *Performance of five protocol combinations from a ping-pong test.*

Protocol structure	Capacity (Mb/s)	Throughput (Mb/s)	Startup latency (s)
RPC	100	12.72	2.957×10^{-3}
PVM	100	12.16	2.766×10^{-3}
BSD Sockets	100	16.72	1.960×10^{-3}
ATM API over AAL3/4	100	32.56	1.034×10^{-3}
ATM API over AAL5	100	31.68	0.869×10^{-3}

software or the application itself, which introduces a cause of performance degradation not considered in the measurements. Note that in either case the achieved throughput is very far from the maximum capacity of the ATM network, even in the case of the ATM API. There are many implementation issues yet to be improved in the APIs.

In order to assess the real impact of the introduction of the ATM API, in [LDTM95] measurements using real algorithms have been performed, which are displayed in Table 3.4. It is observed that the execution times experienced with the ATM API are not much better than those experienced with the rest of configurations. This behavior indicates that the extra user-level functionality counteracts the improvements in the raw communications performance enabled by the ATM API. Another issue from Table 3.4 is the fact that PVM over ATM, i.e. PVM-TCP/IP-ATM, performs better than ATM API for small matrices, which indicates that the impact of message length is better for the API rather than the traditional transport protocol.

Table 3.4: *Execution time of matrix multiplication over several APIs.*

Protocol structure	Matrix Size		
	32×32	128×128	256×256
Sequential	0.0988 s	6.6205 s	64.0001 s
PVM over ATM	0.0524 s	1.9493 s	16.4005 s
PVM over Ethernet (silent)	0.0134 s	1.9693 s	16.9130 s
PVM over Ethernet (30% loaded)	0.0341 s	2.0355 s	17.2416 s
Sockets over ATM	0.0736 s	1.9177 s	16.4030 s
Sockets over Ethernet (silent)	0.0627 s	1.9136 s	16.7187 s
Sockets over Ethernet (30% loaded)	0.0714 s	1.9932 s	16.9256 s
ATM API	0.0629 s	1.7758 s	16.2709 s

Table 3.5: *Performance of PVM over several architectures.*

Protocol structure	Capacity (Mb/s)	Bandwidth (Mb/s)	Latency (s)
PVM over AAL5 (API)	100	26.608	1.617×10^{-3}
PVM over AAL3/4 (API)	100	17.776	1.646×10^{-3}
PVM over IP-ATM	100	16.848	1.234×10^{-3}
PVM over IP-Ethernet	10	8.608	1.662×10^{-3}

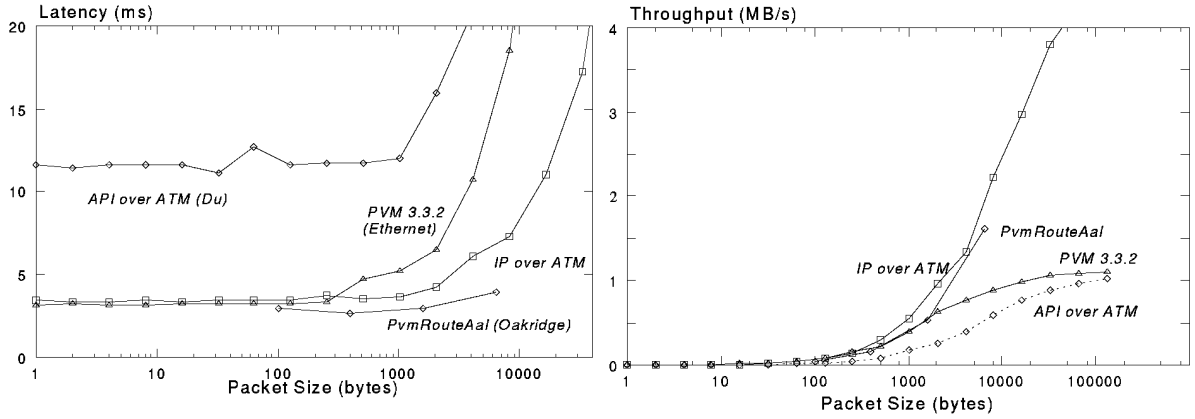


Figure 3.5: *Performance of several software structures obtained by Dowd et al [DSBC95].*

3.2.3 Performance with adapted message-passing libraries

Applications relying on low-level functionality mechanisms like the ATM API involve a high degree of complexity in them because of the need of implementing functions that are usually provided by the communications system in most environments. Therefore, it becomes wise to adopt a more user-friendly interface to ATM, which will manage the additional functionality required by applications. In the particular case of parallel computing, a message-passing library like PVM can incorporate such functions, thus relieving applications from the burden of implementing them. Papers [ZG95, DSBC95] deal with this case, and both are based on the architecture shown in Figure 3.4(b).

In [ZG95], the PVM library has been modified in order to support direct access to the ATM API. In addition to some PVM implementation issues, a guaranteed transmission facility has been incorporated. The performance achieved by this enhancement has been compared to PVM-IP-ATM and PVM-IP-Ethernet. A particular implementation of these functions into the PVM library is discussed in [CDH+94]. This approach has the disadvantage that a specific version of PVM is required for every vendor's ATM API, but in the future this problem can be partially solved thanks to current API standardization efforts in the ATM Forum [ATM96b].

The results in Table 3.5, obtained from a simple ping-pong test, suggest that (1) the introduction of ATM enables faster communications, as expected, and (2) the direct access to the ATM API does not involve significant improvements with respect to the access through IP. Architectures directly using the ATM API experience increases in bandwidth, basically in the AAL5 case since the important amount of overhead inherent to AAL3/4 seriously limit the achieved increase. In contrast, the performance in latency of both direct APIs is worse than in the IP-ATM structure. The reasons of this unexpected behavior can be found in (1) the quality of the implementation of the additional features within PVM, and (2) the implementation of the ATM API. With regard to the ATM API, it is implemented in user space, and the communication with the device driver in kernel space takes place through the standard UNIX communication mechanisms —sockets or streams—. Thus, UNIX communication mechanisms are used twice when using the ATM

API —when applications invoke communication procedures, and within the API— while in the IP-ATM structure are used only once —when applications invoke communication procedures, as IP is already implemented inside the kernel. Standard UNIX communication mechanisms involve a lot of buffering, so in the case of the ATM API, a more intensive use of buffering is performed. In addition, the ATM API requires to cause a context switch, which introduces an important amount of overhead. All these facts explain why the ATM API achieves the best throughput but not the best latency.

In [DSBC95], a similar a study has been carried out. The PVM library has been modified to enable direct access to the ATM API, in order to determine the magnitude of the performance improvement compared to ATM over TCP/IP. Figure 3.5 depicts the latency and throughput achieved by running simple ping-pong tests in several configurations. Both ATM-API and Ethernet-based environments have been considered. The results show that there is little benefit of direct API access with current implementations of ATM networks. The throughput when using ATM through UDP is even superior. All the reasons explaining the results in Table 3.5 apply in this case. The best throughput experienced by IP-ATM with several packet sizes is also related to the fact that TCP/IP and UDP/IP are implemented in kernel space. The slightly different behaviors are a consequence of different measurement conditions, as Table 3.5 bandwidth measurements correspond to a packet size of 262144 bytes.

3.2.4 Discussion

The replacement of the protocol structure by an ATM API, which does not add any functionality to that of ATM, involves that some functions like flow control and error recovery have to be performed elsewhere, specially when AAL5 is used. If these functions are left to applications, their implementation become more complex and the performance improvement is not very significant. If transport-layer functions are implemented in ATM-API-specific implementations of the message-passing library, applications do not get increased complexity, but the achieved performance does not get significant improvements either. In both cases, the reasons for this behavior are twofold:

- The implementation of the transport-layer functions is not specific for supporting parallel computing. Therefore, some of the problems of generic transport protocols still remain.
- The sensitivity to the implementation of the ATM API is very high. In particular, the Fore's ATM API has shown not to be optimized for achieving low latency.

As a result, in order to achieve the desired performance it will be necessary to replace current implementations of the API and the message-passing library by more efficient versions which really take into account the specificities of ATM and parallel computing, specially the need of low-latency communications [MSD94].

3.3 Specific mechanisms for parallel computing

Several authors have proposed mechanisms that reduce the impact of the bottlenecks on the performance of parallel computing applications. In this section, these proposals are classified in three groups, according to the primary issue the enhancement impacts on: (1) *the ATM API*; (2) *the transport protocol*; and (3) *the application context*. Examples of these approaches are [vEBB94], [HPPF94], and [YRHF95, HHM95], respectively.

Table 3.6: *Performance of Active Messages over ATM [vEBB94].*

Machine	Peak BW	Round-Trip Latency
SP-1 + MPL/p	66.4 Mb/s	56 μs
Paragon + NX	584.0 Mb/s	44 μs
CM-5 + Active Messages	80.0 Mb/s	12 μs
SS-20 cluster + SSAM	60.0 Mb/s	52 μs

3.3.1 API-level enhancement mechanisms

In Section 3.2, the implementation of the API adopted for the experiments has shown to be an important bottleneck. Therefore, the research on alternative APIs becomes a key requirement to achieve enhanced performance for ATM-based parallel computing environments. In [vEBB94], a multicomputer communication mechanism, namely *Active Messages* is adapted to ATM. Active Messages offer simple, general purpose communication primitives as a thin layer over the raw hardware, in order to serve as a base for building higher-level communication libraries and to be included in compiled codes generated from parallel languages.

As defined in [vECGS92], the basic communication primitive is a message with an associated small amount of computation (known as a *handler*) at the receiving end. One of the fields in an Active Message is a pointer to the handler associated with that message. On message arrival, the computation on the node is interrupted and the handler is executed. the role of the handler is to extract the message out of the network, by integrating it into the ongoing computation and/or by sending a reply message back, thus the communications take place under the request/reply paradigm. The only buffering provided by Active Messages is that involved in actual data transport. Note that this scheme requires that all the nodes contain the same code image, so it is only suitable for running SPMD (Single Program, Multiple Data) algorithms. There are implementations for several systems, including multiprocessors and networks of workstations. In [vEBB94] an implementation tailored for ATM-based networks of workstations is discussed.

The implementation of Active Messages for ATM networks consists of two parts: a device driver which is dynamically loaded into the kernel, and a user-level library to be linked with applications using Active Messages. The driver implements standard functionality to open and close the ATM device, and a trap-based interface with the library so that the code for sending and receiving individual ATM cells is directly accessed. Nevertheless, all functionality specific to Active Messages is in the user-level library. In addition to the standard functionality of Active Messages, the user-level library includes flow control and buffer management procedures.

The flow control mechanism has been added because of the unreliability inherent to ATM, as opposed to the interconnection network of most multiprocessors. For this purpose, a simple sliding window scheme is used in order to prevent overrun of the receive buffers and to detect cell losses. The window size is dimensioned to allow close to full bandwidth communication among pairs of processors. For each received message, the receiver generates an acknowledgment that can be piggy-backed in the reply, if applicable. The recovery scheme used in case of lost or duplicate cells is standard —retransmission-based, except that the reception of duplicate request messages may indicate lost reply messages which have to be retransmitted. There is no attempt to minimize message losses due to congestion.

Table 3.6 from [vEBB94] shows the performance achieved by a micro-benchmark by several supercomputers and a workstation cluster with Active Messages. The results indicate a significantly high performance for the workstations with Active Messages, as the performance lies in the same order of magnitude as multiprocessors. However, Active Messages involve a number of serious drawbacks compromising this performance.

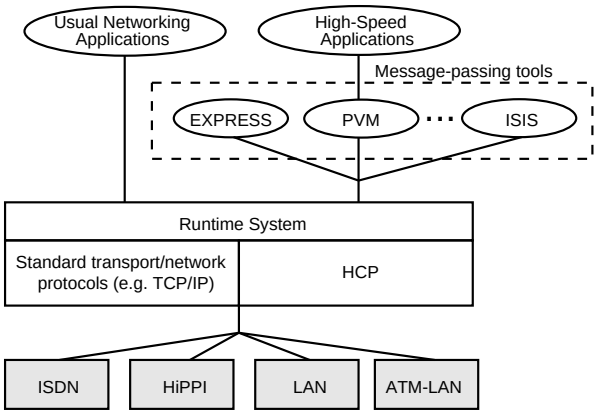


Figure 3.6: *Dedicated protocol environment for parallel computing.*

As noted in [vECGS92], handlers are not allowed to block, since deadlock might result. For example, a handler that attempts to acquire a held lock may cause deadlock as the thread or process holding the lock is suspended by the activation of the handler, hence the lock cannot be released during execution of the handler. Other deadlocks may arise when the handler execution is excessively long, since it precludes the processing of new incoming messages. Other situations leading to network congestion can also produce deadlocks. In general, Active Messages put many burdens on the programmer, who must worry about deadlock in the network, synchronization between messages and computations, and other timing-dependent problems. In order to solve these problems, in [WHJ+95] a solution is proposed for is applicable in multithreaded environments.

The proposal in [WHJ+95] is known as *Optimistic Active Messages*. Unlike other compilers generating Active Messages handlers, it is assumed that every piece of code that is executable as a handler will be effectively compiled as a handler, without checking for deadlocks, etc. —that is why they are called "optimistic". When executing the handlers, if a cause of deadlock is detected, the execution of the affected handler is aborted. Then, three strategies can be considered in order to resume the execution of the aborted code. The particular strategy is selected as a function of the application, the programming language or even the architecture.

3.3.2 Transport-level mechanisms

Traditional transport protocols have regarded the communication bandwidth as a scarce resource and the communication medium as inherently unreliable, therefore they were designed to be very general in order to handle complex failure scenarios. These characteristics have lead to complicated and therefore time-consuming protocol implementations. The high speed and high reliability of current networks allows for simpler protocols. In [HPPF94], a communication environment specifically tailored for supporting parallel computing applications is presented, whose architecture is shown in Figure 3.6. The hardware portion of this environment includes a host interface processor, a high speed network, and a "normal" speed network. The software portion consists of a high-speed communication protocol (HCP) and a HCP API. The API is an interface between a parallel computing application and the HCP services implemented on an interface processor. The keys for the high speed are: (1) implementing the protocol in a special communication processor, thus offloading the host from protocol processing, and (2) making the protocol implement the basic functionality of message-passing libraries.

HCP includes the services that are common to most popular message-passing libraries: point-to-point

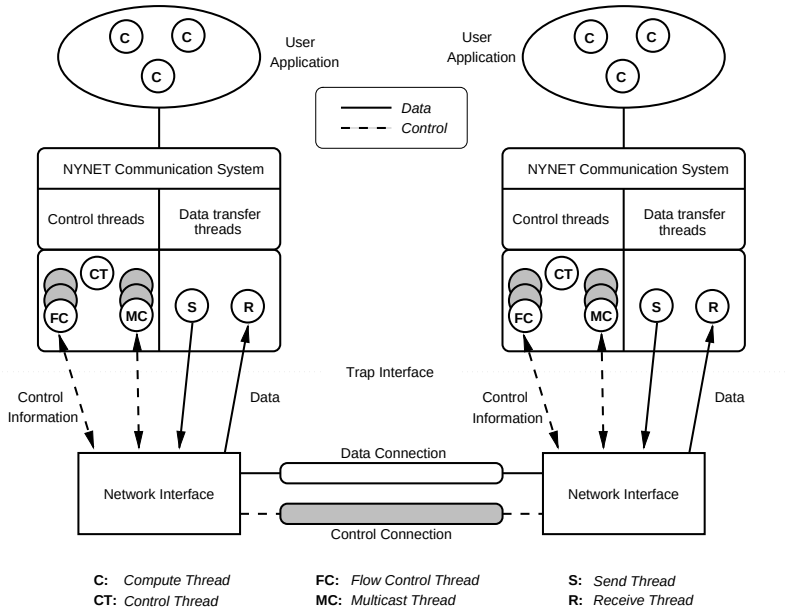


Figure 3.7: *Architecture of a multithreading environment for supporting parallel computing.*

communications, group communications, process synchronization, system control and management, and error handling. Thus, HCP offers a series of primitives for accessing these services. the functions supported by HCP include synchronous and asynchronous data transfers, broadcast, barrier and system configuration.

For transferring data, two schemes are used depending on the message size. Short data packets are transferred in a datagram-like fashion. In contrast, long messages involve setting up a connection prior to the transfer of each of them. Reliability of the transport service is assured by simple error and flow control mechanisms. For each sent frame, the sender waits for an acknowledgment. Upon receipt of a positive acknowledgment, the next frame is sent, while a negative acknowledgment triggers the retransmission of the frame. If the receiver has not enough room for the frame, it responds with a not-ready indication so that the source stops to send data. Thus, HCP implements a very simple window-based flow control mechanism where the frames are individually acknowledged.

This protocol is *not* designed to be implemented over ATM but over a proprietary high-speed network. Nevertheless, the software can be adapted to run for an ATM network embedding the “normal” speed network and the high speed network. Implementing HCP in an ATM host interface card is more difficult.

3.3.3 Application-level mechanisms

Despite the availability of the enhanced mechanisms described so far, a portion of latency can remain unavoidable. Several approaches intend to take advantage of the idle timer for doing more computation. One common technique for this purpose is *multithreading*, which consists of allowing several concurrent execution flows per task, so-called *threads*, sharing all the resources allocated on a per-task basis. Thus, computation and communication can overlap, since while one thread is waiting for a message to arrive, another thread can carry out computations not requiring the data in the expected message. This feature is specially interesting in WAN environments as the propagation delay can be hidden.

An example of an ATM-specific multithreading mechanism is the NCS (NYNET Communication

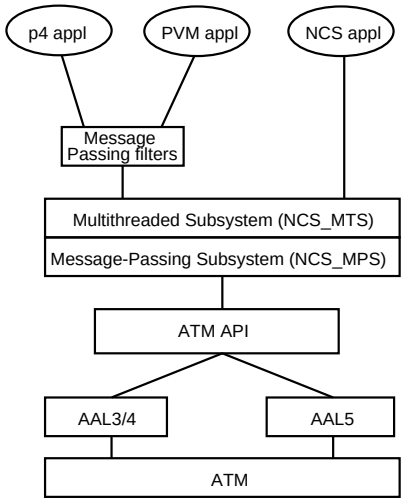


Figure 3.8: *Multithreading support for distributed parallel computing.*

Table 3.7: *Execution timer of Matrix Multiplication (seconds).*

Nodes	p4	NCS_MTS/p4	% Improvement
1	24.89	25.03	-
2	14.4	11.51	20.06%
4	7.52	5.41	28.05%

system) discussed in [YRHF95, PHK+96]. In this environment, a process consists of user threads and system threads. User threads are in charge of performing the actual computations of a given parallel computing application, and are written and activated by the user. Systems threads include control, data transfer, flow control, or multicast threads, and are created on initialization of the environment and activated by the control thread according to a user specification. Thus, for example, in an ATM network, the control thread establishes ATM connections for data transfer and maintains configuration data of each machine; the data transfer threads (send and receive threads) are solely addressed to data transfer. In addition, users may choose among several flow control and multicast algorithms in order to adopt the best for each particular application. All these threads share data structures in memory so that they can interoperate. The architecture of the NCS environment is depicted in Figure 3.7.

The NCS can be viewed as two main subsystems, as shown in Figure 3.8: the MultiThreaded Subsystem (NCS_MTS) and the Message-Passing Subsystem (NCS_MPS). The NCS_MTS provides all thread-related services, including initialization, context switching, scheduling and synchronization issues. the NCS_MPS provides the communication services required by applications. For this purpose, two approaches have been considered: (1) to integrate an existing message-passing library (p4 or PVM), including traditional transport protocols such as TCP/IP, like the first strategy, and (2) to avoid traditional communication protocols and directly access the ATM API, like in the second strategy. Table 3.7 shows the performance gains experimented by a matrix multiplication algorithm over such an environment.

Another work taking advantage of multithreading is [HHM95]. For each task, separate threads are created for computation, data send and data/ack receive. Here, multithreading is exploited for configuring efficient collective communications, such as gather and reduction, with the construction of configurations where communications can occur concurrently. For this purpose, multicast is implemented by using a point-to-multipoint connection, whilst the acknowledgments are carried in a series of point-to-point connections arranges as a spanning tree. The data send and date receive/acknowledge threads concurrently operate

on this configuration. In this work, many-to one connections are implemented by using bidirectional point-to-point connections arranged as a spanning tree, as in multicast. Here a single thread operates on all data transfers, namely send, receive and acknowledge.

3.3.4 Discussion

Two classes of mechanisms have been suggested for enabling support of parallel computing applications over ATM networks. One class of mechanisms attempt to increase the speed of communications but at the same time they try to keep compatible either with protocols addressed to traditional networking applications, such as TCP/IP, or at least with existing message-passing libraries, such as PVM. Thus, unnecessary overhead in communications is introduced that precludes the achievement of the potential performance enabled by ATM. For instance, the mechanisms for replacing the legacy networking technology by ATM (LAN Emulation, IP over ATM and the adaption of an ATM API) that have been presented earlier in this chapter are conceived to be integrated in general-purpose architecture, which experience significant performance degradation. As far as the various ATM API-based strategies, they rely on an interface that is common to all types of applications. In addition, supporting a message-passing over such an API does not take advantage of the special characteristics of both parallel computing applications and ATM, since the communications interface of current message-passing libraries is tailored to be supported by traditional protocols.

The other class of mechanisms involves the proposal of mechanisms specially tailored for supporting parallel computing. As the communication overhead is minimized, the performance achieved by these mechanisms is superior. However, the fact that these mechanisms are efficient for parallel computing applications does not mean that all applications be properly supported. For example, the usefulness of Active Messages for supporting applications other than parallel computing is yet to be proved. With regard to HCP, the physical network is not shared, but rather a high-speed network is crafted aside the “normal” speed network. Inside the high-speed network, HCP should be common to all high-speed applications, which is not the case as HCP is specifically tailored for supporting parallel computing applications. In addition, some of the specific mechanisms require special programming for applications. The Active Messages interface involves such particular programming conditions that the programmer has to be aware that the program is relying on Active Messages. Multithreading-based approaches demand the addition of synchronization and mutual exclusion calls, so applications require specific programming as well.

As a result, some mechanisms intended for supporting parallel computing over ATM networks are conceived to be integrated in a general-purpose architecture. Other mechanisms, in contrast, are specifically designed to optimize performance for parallel computing applications, but without considering the existence of other networking applications sharing the ATM network. For this reason, in this work we propose an architectural model in which mechanisms for supporting parallel computing are integrated, and then they coexist with other architectures that are specific to other types of networking applications. Thus, all mechanisms will be members of an integrated network architecture for a particular application type, one of which being parallel computing. The characteristics of the specific architecture for parallel computing will enable the future development of message-passing libraries that allow to fully exploit the advantages of ATM-based environments while programmers can be hidden most of the programming details that current application-specific mechanisms usually require.

4
The network architecture model

We propose a global framework for supporting parallel computing communications over ATM networks. For this purpose, we assume a model of network architecture that allows to integrate specific mechanisms for parallel computing within an ATM network that is shared with the network architectures of many other application types. We also advance the suggestion of two possibilities for building an overlay network where signaling procedures are decoupled from the execution of parallel computing applications, in order to outline the final appearance of ATM-based parallel computing environments.

4.1 Integration of specific mechanisms over ATM

As pointed out in Chapter 3, the best performance for parallel computing applications can be achieved with the introduction of mechanisms specifically tailored for parallel computing. Nevertheless, this is not exclusive for parallel computing applications; every type of application gets the best performance with the use of specific mechanisms. However, the traditional approach up to now has been the use of a common protocol architecture for all applications. The reasons were (1) the short range of networking applications did not demand for specific mechanisms, and (2) traditional networking technologies did not provide any facilities for developing specific mechanisms, since the bottleneck to minimize was not protocol processing but network-level transmission, as discussed in previous chapters. Currently, with the availability of a technology such as ATM and the consequently increasing base of networking applications, network architectures tend to include diverse application-specific architectures coexisting on top of ATM, each one corresponding to different types of applications, One of these architectures can be devoted to support communications in parallel computing applications.

Some proposals of application-specific network architectures have already been made. As an example, in this section we review specific mechanisms for transporting MPEG-coded multimedia information, and for supporting a video-on-demand service.

A proposal of an AAL specific for multimedia applications is made in [GVH96]. The main characteristics of multimedia data streams are their continuous nature —as opposed to the bursty nature found in data applications— and their very stringent constraints in terms of delay and delay variations —known in this field as *delay jitter*. When a compression scheme is used, traffic from multimedia applications becomes also sensitive to cell loss, although the actual impact of loss depends on the type and location of lost information. The most suitable standard AALs for the transport of multimedia information, AAL1

and AAL5, but they suffer from some inconveniences. In particular, the FEC (Forward Error Correction) scheme included with AAL1 includes some unnecessary overhead. As far as AAL5 is concerned, it is not capable to known the position of lost cells. Then, when cell loss is detected, the corresponding packet is discarded with a consequent increase of loss at application level. For all these reasons, the proposal in [GVH96] consists of a specific AAL, whose organization is displayed in Figure 4.1(a), which includes cell-level granularity to improve error detection, and a selective, specific FEC scheme to selectively protect essential data. Several versions of this AAL can be developed for the diverse coding schemes (MPEG-2, H.261, etc.), as shown in Figure 4.1(b).

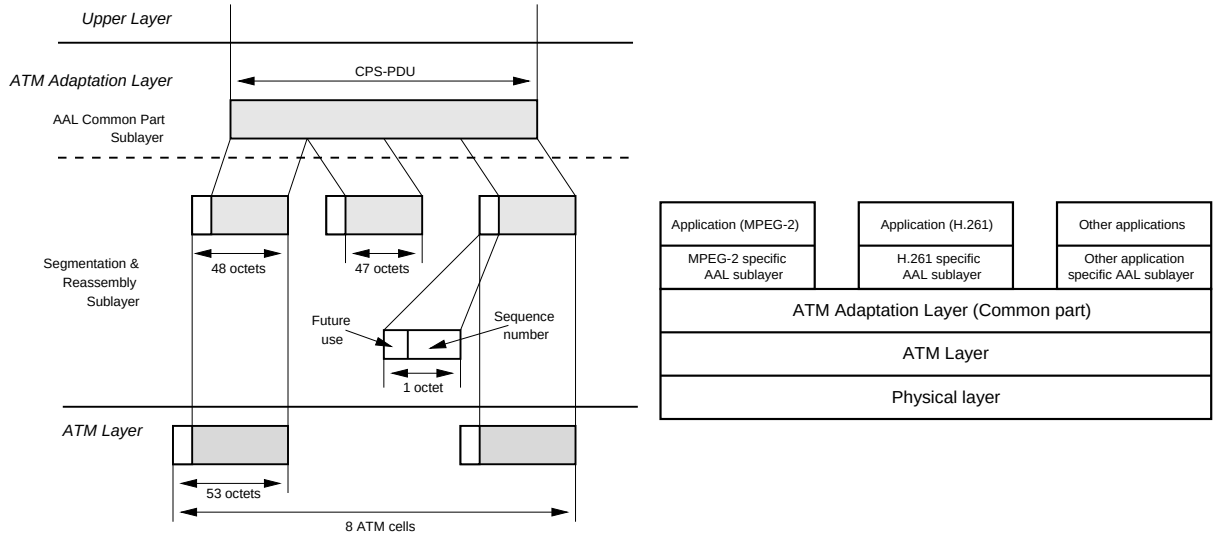
Another example of application for which a specific AAL has been proposed is the environment for supporting the Video-on-Demand service discussed in [CF96]. As displayed in Figure 4.2(a), this service is organized as an ATM virtual network including video servers —called "Information WareHouses" (IWH)— that are in charge of providing the service to the specialized nodes —called "Intelligent Access Periferals" (IAP)— which in turn will deliver the appropriate information to the users. As each IAP can apply for diverse videos in one IWH, a protocol is necessary to determine which demand is serviced at a particular instant, according to the deadlines of the contending demands, which are a consequence of the real-time constraint. This operation mode conveys the need of transmitting information on a per-burst basis. For this purpose, this Video-on-Demand architecture relies on the presence of Fast Resource Management in the network, which is possible when the actual data transfer is performed through a virtual circuit using the ABT (ATM Block Transfer) service category that is defined by the ITU-T in the Recommendation I.371 [ITU95a]. Users explicitly demand each burst of information by using an original protocol, the MTEX (Multi Token EXchange protocol), whose operation is illustrated in Figure 4.2(b). When the IAP issues a request to the corresponding IWH, it waits for the reception of a token. This token can be accepted or rejected. If the token is accepted, the IAP prepares for delivering the service and instructs the IWH to start the FRM procedure in order to transfer an information burst. The rejection of a token occurs when the IAP is already serving a request with a higher priority.

4.2 Specific architecture for parallel computing

All application-specific network architectures like the previous examples can be integrated over a common ATM network, as shown in Figure 4.3. In this section we focus on describing our specific network architecture for supporting parallel computing. Figure 4.3 outlines the internal organization for this network architecture within the general architectural model. The idea of a specific architecture for parallel computing coexisting with other applications appears in [HPPF94], although that proposal is oriented to supporting a particular mechanism. In contrast, the proposal discussed in the work is intended to integrate whatever mechanisms are specifically conceived for parallel computing. Our specific network architecture includes two components: (1) the overlay network over which parallel computing applications will be supported, and (2) the mechanisms within the endpoint stations that will perform the actual data transfers during execution of applications. We outline some possibilities for the overlay network, but we specifically focus on the component within the endpoint stations because it is the component that actually impacts on the performance of parallel computing applications.

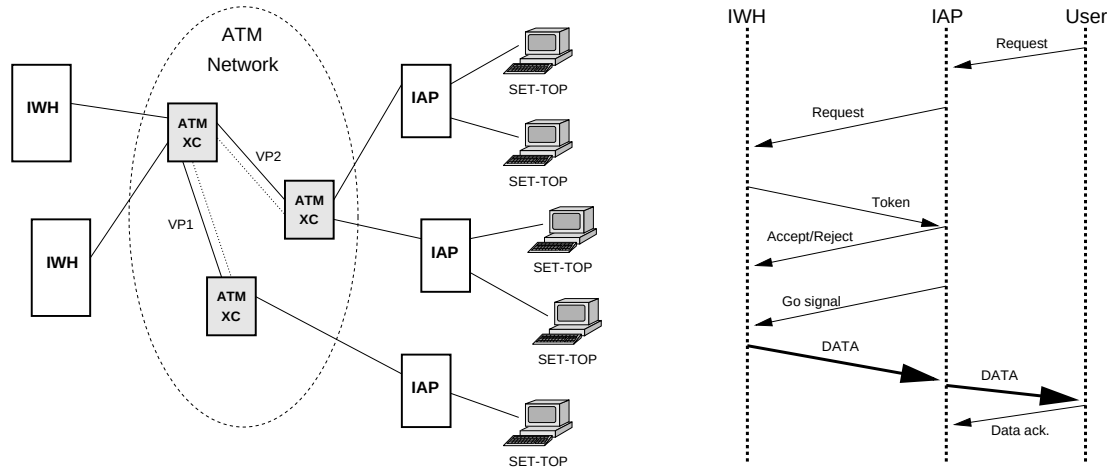
4.2.1 Overlay network for signaling in parallel computing

In order to understand the characteristics of the mechanisms for supporting parallel computing, we first discuss the structure of this service. In order to illustrate over which environment parallel computing applications operate, we assume that parallel computing applications will run over a virtual network interconnecting the endpoint stations participating in the execution of a particular application. This virtual



(a) Structure of the specific AAL (common part) (b) Global view

Figure 4.1: *Specific architecture for supporting multimedia applications suggested in [GVH96].*



(a) ATM virtual network for supporting Video on Demand (b) Operation of MTEX

Figure 4.2: *Specific architecture for supporting the Video on Demand Service as suggested in [CF96].*

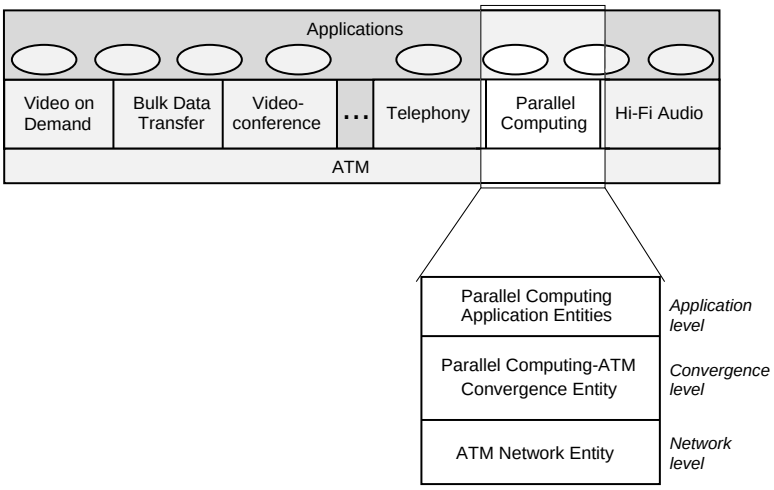


Figure 4.3: *Integration of services over ATM.*

network is set up before the execution of a particular parallel computing application, and released after this execution. Thus, the realization of signaling operations is restricted to be performed outside the execution of the application, so that signaling does not impact on application performance. The effective performance as perceived by the user —which includes both signaling and application performance— is not significantly affected by signaling either, provided that applications run during long periods —hours or days—, as actually expected for real parallel computing applications, because otherwise the signaling delay would quit the advantages of introducing parallelism in a particular application.

The conceptually easiest strategy to build a virtual network is to establish virtual paths among all communicating pairs of stations. In parallel computing applications it is very easy that each station needs to communicate with the rest of stations. This is feasible when the parallel computing environment is composed by a few stations but, as long as the number of stations grows, the maintenance of so many connections becomes rapidly infeasible [New94a]. For this reason, in order to enable a smooth scalability of ATM-based parallel computing environments, we have to consider more complex virtual networks. Thus, a virtual network for supporting parallel computing will include two types of nodes: (1) the endpoint stations where parallel computations are performed, and (2) parallel computing servers, which are in charge of supporting signaling procedures in order to build virtual topologies. The latter procedures include the setup of data transfer connections and the selection of the ATM service categories over which these connections need to rely. As an example of such virtual networks, we will present two possible overlay networks for supporting parallel computing over ATM networks.

The *parallel computing servers* receive the invocation of the service by the users. In the example shown in Figure 4.4, the parallel computing servers are permanently interconnected with a ring topology. Each server, is permanently connected with a number of endpoint stations. If the requested configuration is sufficiently small, the server will ask the respective endpoint stations to setup data connections among them. If more stations are required, the server will setup a data connection with another server and tell it to establish connections between the remote server and the respective stations. The signaling connections and the data connections between servers can be contained within a single Virtual Path between the concerned pair of servers. Thus, when a station wants to exchange data with a station not connected to the same server, it forwards the data to the server so that it routes the data to the appropriate server, which in turn will deliver these data to the destination station. This strategy for building a virtual network is inspired by the 'virtual machine' model used in PVM [G+94], and is quite simple. However, much delay may be involved in building the virtual network, as well as during data transfers to remote stations, if the number

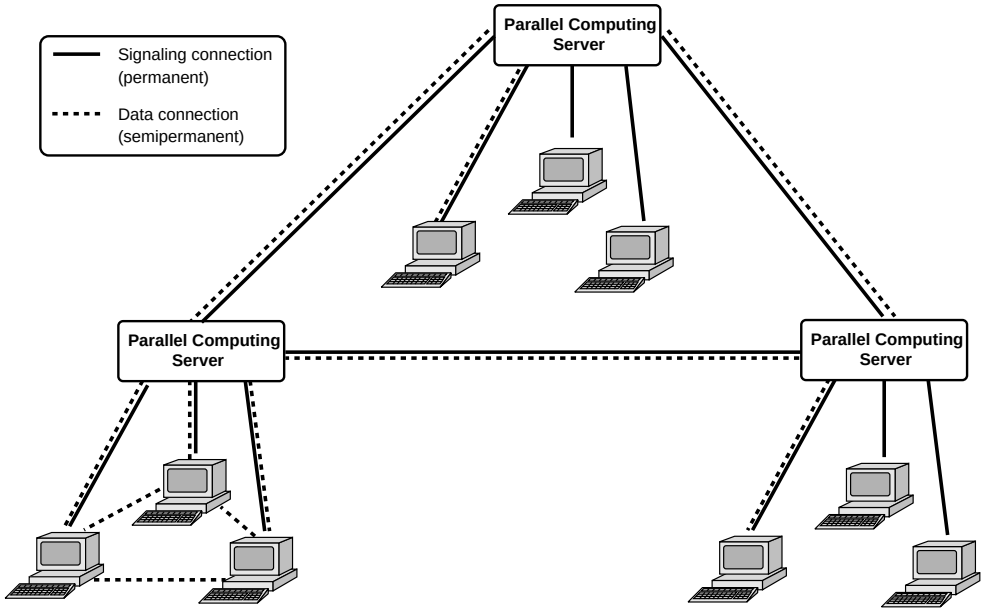


Figure 4.4: *A virtual ring overlay network.*

of parallel computing servers is large, due to the use of a virtual ring topology. The second example attempts to overcome this limitation, at the cost of increased complexity.

The example virtual network shown in Figure 4.5 attempts to reduce the number of servers to be crossed for building the virtual network, as well as to enable direct connections between either endpoint stations or parallel computing servers. Each server can be connected to a number of endpoint stations, like in the previous example, but also to other servers. Thus, we obtain a hierarchical configuration. As before, servers —in fact, only those servers connected to endpoint stations— receive invocations from users, and ask the endpoint stations for building the virtual network. When endpoint stations connected to other servers are required, the local server requests the address of the remote station by accessing the remote server through the tree, in order to allow for the direct connection between the concerned stations. In case direct connections are not possible because there are no free VCI identifiers left, the data will be transferred through a data connection via the server. All these procedures are valid for the servers, in the sense that direct data connections may be established between the servers in order to enable the communication between two remote stations that have run out of VCIs.

In both schemes, addressing is performed on the basis of the VPI/VCI pair in each cell in the communications between two stations connected to the same parallel computing server. For accessing stations connected to remote servers, an additional field is required since several connections can share the virtual channels associated to the servers, analogous to the MID field found in AAL3/4 packets, which will be used by the servers for routing and for the remote endpoint stations to determine the origin of the messages. In addition to routing functions, parallel computing servers may contain other functions, for instance mechanisms for the automatic selection of stations for building the virtual network. These mechanisms can operate by using a distance criterion, or even load-balancing schemes. Although so far in this subsection we have considered a single data connection between communicating pairs, actually several connections may be established. This is the case described in Chapter 6, where several connections using different ATM service categories are setup in order to take advantage of the features of each service category in the adequate moments.

We do not attempt to make a deeper study of virtual networks for supporting parallel computing over

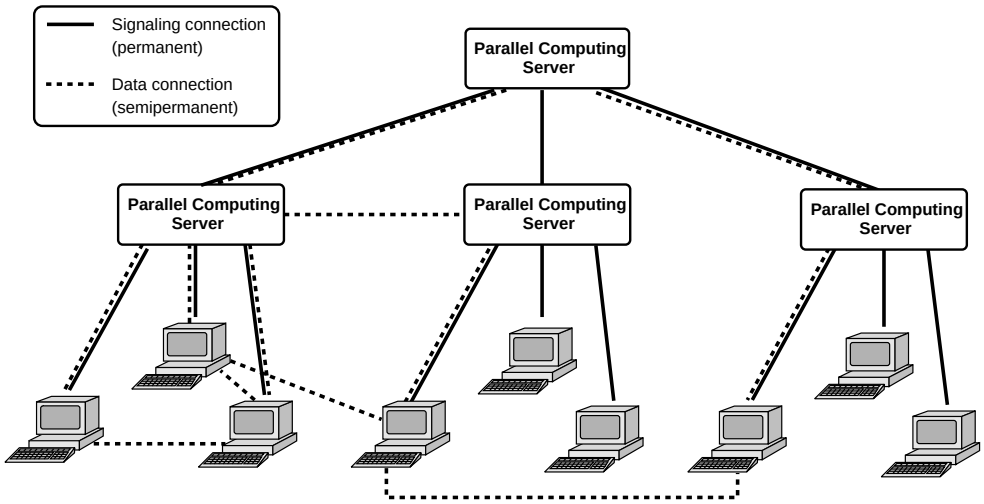


Figure 4.5: *A virtual tree overlay network.*

ATM networks as they are out of the scope of the present thesis. We have suggested these examples in order to give an idea about the characteristics of the environments over which we expect that parallel computing applications will be executed. This thesis is more focused on mechanisms that are present in the endpoint stations for performing the actual data transfers. The knowledge on characteristics of the virtual networks will then facilitate the understanding about some of the assumptions made in the mechanisms discussed in the rest of the work, whose characteristics are outlined in the remainder of this chapter.

4.2.2 Protocol architecture

In this subsection we review the architecture in which the mechanisms supporting run-time communications during the execution of parallel computing applications. As we assume that run-time communications and signaling are neatly separated, the network architecture for supporting run-time communications should actually be independent of the design of the overlay network.

The objective of the specific network architecture for parallel computing is to provide applications with a smooth transition to the peculiarities of ATM networks while satisfying the requirements of communications in parallel computing environments, that is, to minimize end-to-end latency with full guarantee of data delivery. Another issue to have into account is the condition that the ATM network used by parallel computing applications is shared with other networking applications. Thus, it becomes necessary to decouple communications as viewed by parallel computing applications from the actual data transfer service provided by ATM. For this purpose, as discussed in [SPVS96, VSSP96b], we consider the network architecture for supporting communications in parallel computing applications as organized in three levels, as depicted in Figure 4.3. In the following we summarize the scope of each level.

- *Application level*. The communicating entities attempt to implement interprocess communication by assuring the correct delivery of data by taking advantage of the special properties of the data generated by parallel computing applications, as shown in Subsection 4.2.3.

- *Network level*. The functionality contained by the communicating entities is facilitated by the networking technology. In the present work, this level implements the standard functionality of ATM, as summarized in Section 1.3.

- *Convergence level*. The communicating entities implement the functions needed for supporting the service requirements of entities in the Application level that cannot be directly provided by the network technology itself. Our work is mainly concerned with this level.

This architecture has to be understood as an analogy to the OSI reference model. Thus, communication between two peer entities in a particular level is transparent to the other levels. As shown in the following subsections, each level incorporates particular protocols with their own message units. Interfacing between two adjacent levels involves the adaption of the their respective message formats. The view of communications as understood in each level is discussed in the next subsections.

4.2.3 Requirements for the Application Level

In order to achieve low-latency communications, we attempt to take as much advantage as possible of the nature of communications generated by parallel applications. At the highest level, we observe that the data exchanged by parallel computing applications consists of chunks of some elementary types of data. For instance, mathematical applications exchange arrays of floating-point numbers.

The message generation procedures of current message-passing libraries, such as PVM and MPI, are designed to pack the elementary data types into large messages. The reason is that these libraries were conceived to run on traditional protocols like TCP/IP on LAN environments (then, usually Ethernet and Token Ring). In these environments, as the cost of communications is dominated by the capacity of the network, the objective is to maximize the throughput by avoiding as much overhead as possible. Thus, parallel applications seek to encapsulate as much information as possible in a single message although sometimes it were necessary to delay the transmission or the reception of some of the components of the message.

In ATM, the higher speed significantly reduces the importance of the network capacity in the overall cost of communications. Thus, the natural data unit is no longer a variable-size large packet but a small fixed-size cell. For this reason, there is no sense in delaying the transmission and reception of data, so we can associate an elementary type of data (which is shorter than an ATM cell) as the basic application-level PDU (herein PC-PDU, after Parallel Computing PDU). Given this assumption, we can now classify applications according to the characteristics of PDU sequences and their distribution along the time. As far as the sizes of PC-PDUs are concerned, we consider:

- *Basic PDUs*. They correspond to elementary data types (such as `float`, `int`, `double` in C language), as mentioned above, and therefore their length does not go beyond 16 or 24 bytes in usual workstation systems. These are the shortest data unit that are understandable by parallel computing applications in a semantic sense. They are the predominant type of PDUs in message-passing systems.
- *Uncertain-length PDUs*. They correspond to whole memory blocks that cannot be structured as sequences of basic PDUs. These PDUs include strings and cache lines, the latter being used in shared memory programming models. These PDUs are rarely used in message-passing systems.

The distribution of the elements within the sequences of PC-PDUs is also important to determine the behavior of communications in parallel computing applications. In this work we consider two main distributions of these sequences:

- *Non-correlated*. PC-PDUs are smoothly distributed along the time, with a regular and significant spacing between them. This situation corresponds to uniformly distributed transmission of basic PC-PDUs.

- *Correlated*. PC-PDUs are transmitted in bursts, where the spacing within the PC-PDUs is rather short, interleaved with periods of low activity. This behavior occurs when arrays of PC-PDUs are transmitted.

As a summary, parallel computing applications using a message-passing model essentially generate basic PC-PDUs which in many cases belong to highly correlated sequences, since many algorithms deal with arrays and matrices. All the services provided by both the network entity and the convergence entity have then to be tuned to provide an efficient transmission of this traffic. Some other characteristics of communications in parallel computing applications will help for this purpose:

- Parallel computing applications execute for fairly long periods of time —hours or days. Communications do not occur continuously during this period but rather they consist of sparse transactions.

- As a consequence, the average bit rate is moderate due to a fairly low frequency of communications, although the bandwidth requirements are much more important when communications effectively take place.

This behavior of communications, together with the requirement of a total reliability that is common to all applications dealing with data, conditions the characteristics of the convergence level. In particular, the relative importance of keeping a high throughput can be exploited for reducing the latency in communications, which is more relevant to parallel computing applications.

4.2.4 Architecture of the Convergence Level

In parallel computing applications, like usual data applications, but unlike real-time applications including video, audio, etc., the reliable delivery of the transferred data is mandatory. The characteristics of ATM make it a suitable technology for transferring short messages such as PC-PDUs. Unfortunately, ATM networks, as defined in the specifications by ITU-T and the ATM Forum, are not capable of providing guarantee of data delivery, so this function has to be implemented elsewhere. As we assume the hypothesis that only standard definitions for the ATM service are considered, the functions needed to provide the service requested by parallel computing applications will be implemented in the convergence level. For this purpose, some of the facilities discussed in Section1.3 can facilitate the deployment of efficient convergence level mechanisms. In particular, the following capabilities have been considered for use in the convergence level:

- *Service categories*. Among the service categories that have been standardized for ATM, namely CBR, rt-VBR, nrt-VBR, UBR and ABR [ATM96c], as well as ABT [ITU95a], we consider that for the data transfer service the only eligible categories are the best-effort ones, namely UBR and ABR, since the traffic generated by different parallel applications have no common properties allowing for its characterization *a priori*, as would be required for the rest of service categories. For the control flows, in contrast, any of the categories could be used, depending on the desired operation of the convergence level mechanisms.

- *Adaptation layers*. As currently standardized, AAL1, AAL3/4, AAL5 and the user-defined AAL can be used to smooth the transition between the convergence level and the raw ATM. AAL adds some functionality for real-time constant-rate services, so it is not appropriate for parallel computing applications. As AAL3/4 is currently being abandoned, the alternative is between AAL5 and the user-defined AAL. The size of the PDUs is the criterion to decide the finally adopted AAL.

Table 4.1: *Suitable mechanisms to support convergence level functions.*

Functions	Mechanisms	Examples
PDU mapping	ATM API	Message-based mapping Cell-based mapping
Loss Recovery	Retransmission	Selective ARQ Go-back-N ARQ
	Addition of Redundancy	Standard FEC Connection replication
Flow Control	Congestion Evaluation	
Routing	Simultaneous Replication	
	Adaptive Routing	Alternative paths Indirect connections
	Burst Distribution	

- *VC/VP connectivity*. The structure of virtual paths and virtual channels inherent to ATM allows for the setup of private virtual subnetworks and redundant paths in order to provide fault tolerance.

With these facilities, the convergence level has to implement a number of mechanisms whose final objective is to provide a reliable service to parallel applications by minimizing the impact on performance. As argued in [Fel93], for achieving adequate performance the convergence level has to be strict in eliminating unneeded or replicated functionality, as well as taking care of the implementation of the mechanisms. By taking these criteria into account, the convergence level can give support to the following functions:

- *Service interfacing*. In general, this function consists of the adaption of the PDU from an overlying application entity to the data unit managed by the corresponding network entity. If the application-PDUs are longer than the network-PDUs, this process involves segmentation. In the particular case of parallel applications, PC-PDUs are shorter than one ATM payload, so no segmentation is done. In addition to the adaption of PDUs, the service interfacing functions solicit to the network level the facilities required by the rest of functions in the convergence level.

- *Loss recovery*. They are very important in parallel computing communications since they allow for the fulfillment of the reliability requirement of this type of communications. The contribution to end-to-end latency from loss recovery mechanisms can be highly significant; therefore, special attention must be paid to both the algorithm and the implementation of these mechanism in order to achieve the desired performance for parallel applications.

- *Flow control*. Since the cost in latency from the loss recovery mechanisms is very high, the flow control function shapes the traffic so that the occurrence of losses is reduced and, consequently, the cost of loss recovery decreases. As the reduction is usually achieved by decreasing the speed at which data are sent, the mechanisms implementing flow control have to be precisely tuned in order to optimally tradeoff the contributions to latency from flow control and loss recovery.

- *End-to-end routing*. The possibility of redundant paths enabled by the virtual path/virtual channel hierarchy allows for the reduction of the costs of loss recovery and flow control functions by routing the traffic through alternative paths.

Some suitable mechanisms for implementing the functions in the convergence level are outlined in Table 4.1. For service interfacing functions, there are some ATM APIs (Application Programming Interface) already defined. In subsection 3.2.1 Fore Systems's API is discussed. The mechanism included in this particular API is oriented to large frames and, although it allows for single cell-sized payloads, it

is not optimized for this purpose. As a consequence, parallel computing communications require a more specialized mechanism in order to optimally dealing with short PC-PDUs.

Regarding loss recovery functions, there are two basic types of mechanisms available to improve reliability: retransmission-based and redundancy-based. The former achieve reliability with the retransmission of data that were not correctly received by the receiver. For this purpose, the transmitter and the receiver have to exchange state information about the status of individual messages. At least one round-trip latency is added for each retransmission. On the other hand, redundancy-based mechanisms provide robustness by adding redundant information to the original data. If a sufficiently low amount of data has been lost, the added redundancy enables a reconstruction of the original data at the receiver, thus avoiding the need of retransmitting it. The ability to recover lost information strongly depends on the degree of redundancy. Redundancy-based mechanisms seem attractive for applications with latency constrains, since they enable the saving of additional latencies induced by retransmissions, as stated in [Bie93]. Unfortunately, they involve considerable processing overhead and, in addition, they result in increased bandwidth consumption, which in turn leads to cause higher information loss, so a well designed and tuned retransmission-based mechanism may also offer adequate performance, even in time-sensitive applications [DLW94].

Retransmission-based mechanisms are known as ARQ (Automatic Repeat Request). There are two classes of ARQ mechanisms: selective mechanisms, in which the retransmitted information strictly corresponds to the lost information, and go-back-N mechanisms, in which, when a failure is detected, the subsequent information is retransmitted, independently of it was lost or not. ARQ mechanisms have been implemented in most popular protocols, including TCP, X.25, etc. Regarding redundancy-based mechanisms, the most relevant are the FEC mechanisms, which consist of encoding the transmitted information in order to introduce redundancy. Such a FEC strategy is included in the specification of the AAL1, since it is targeted at supporting real-time services [ITU93e]. A similar proposal has been discussed in the ATM Forum to be used with AAL5, in order to cover a wider range of applications [ECG+95]. In this case, since FEC can recover some of the lost information but cannot guarantee the delivery of all transmitted data, an ARQ mechanism has to be present. Despite this, FEC has been proposed for some data applications [ECD+95].

In order to minimize the impact of loss recovery mechanisms, the convergence layer can make use of both flow control functions and routing functions. Flow control mechanisms adapt the traffic characteristics to the state of the network so that cell loss can be reduced. Although standard approaches like sliding window can be used, in ATM-based environments it is interesting to take advantage of the flow control facility provided with the ABR service category [ATM96c]. Finally, routing mechanisms can make use of redundant paths and information about the network in order to use the fastest path at each moment. An example of such a mechanism is proposed in [SPSLA95], where the virtual path carrying the information is dynamically assigned to a data flow among several redundant paths.

4.3 Summary

In this chapter we presented a network architecture which allows to integrate the specific architectures of a wide diversity of applications on top of a common ATM network. As examples of specific architectures, we summarized the characteristics of a multimedia data transport service and a video-on-demand system.

Parallel computing is one application more among those to be integrated on top of a common ATM network. Its associated specific network architecture includes two components: (1) an overlay network, and (2) the endpoint run-time mechanisms. We outlined the characteristics of the overlay network, but we focus on the endpoint runtime mechanisms as they directly affect the performance experienced by parallel computing applications.

The endpoint component of the network architecture for supporting parallel computing applications is organized in three levels: (1) the application level, (2) the network level, and (3) the convergence level, which contains the functions needed by the application level that are not provided directly by the network level, in out case ATM networks.

The immediate chapters are addressed to the proposal of concrete mechanisms to be implemented within the convergence level. In Chapter 5 we suggest an AAL specific for parallel computing —the PC-AAL— in which an existing transport protocol for providing end-to-end reliable data transfer, the SSCOP (Service Specific Connection Oriented Protocol) is modified for it to adapt to the special characteristics of the data generated by parallel computing applications, in particular to the short length of the basic data units —the PC-PDUs. In Chapter 6 we discuss a strategy to introduce the diverse service categories supported by ATM in order for the network architecture to benefit from their features. The final goal is to optimize the performance to cost ratio. For this purpose, the more expensive service categories, as is ABR with respect to UBR, are activated only when the advantages really compensate the additional cost. A real-time monitoring process is used to determine the periods when this activation can yield positive results.

5

Coupling SSCOP with ATM-based parallel computing

The first phase of the implementation of the convergence level involves the proposal of a mechanism for providing reliable delivery in parallel computing communications. For this purpose, we have adopted an existing lightweight transport protocol as a basis for developing this mechanism, in particular SSCOP. The modifications introduced in SSCOP allow for the removal of AAL5, in such a way that a novel, specific AAL for parallel computing has resulted. In this chapter, we first review the operation of standard SSCOP, and then we describe the modifications introduced in order to achieve the specific Parallel Computing AAL. After that, the results of the performance evaluation study of these modifications are exhibited. This work is covered in [VSSP96b, SPVS96].

5.1 Description of SSCOP basic functionality

The Service Specific Connection Oriented Protocol (SSCOP) has been recently proposed by the ITU-T [ITU94b] in order to support certain types of data transfer over ATM networks which require assured service. In particular, one of these services is the support to signaling [ITU93a] and to a reliable connection oriented transport service (COTS) [ITU95b], both mentioned in previous chapters. ATM has increased the available bandwidth of communication networks and, in parallel, emerging applications are consuming more and more bandwidth. Under these conditions, some currently used data communication protocols extended beyond their assumed operating environment and, consequently, cause service degradation, as noted in [Hen95]. These concerns justify the adoption of an ATM-specific approach such as SSCOP.

5.1.1 Interest of SSCOP in ATM networks

In this subsection we discuss the reasons why a novel protocol such as SSCOP has been developed for supporting reliable data transfers over ATM networks, instead of considering other existing and well-known transport protocols like TP4 or TCP. These widespread protocols were designed by considering the available bandwidth as the dominating bottleneck. For this reason, they use a generous amount of processing power in order to reduce transmission costs, with the addition of extra processing to minimize the bandwidth required to recover from errors and incorporating relatively simple flow-control algorithms. The current availability of networks offering greater bandwidths, together with the higher reliability

provided by fiber optic cables, claim to minimize processing requirements, even if transmission bandwidth has to be sacrificed, to optimize protocol processing for the case of error-free communication, and to include more effective flow-control algorithms [DDK+90].

Some authors argue that, in high speed networks, most of inefficiencies found in TCP come from the implementation and environmental factors; thus, they believe that by optimizing the critical path in the protocol implementation is sufficient to achieve adequate performance [CJRS89, MPBO96], with the advantages introduced by the full compatibility with all current applications relying on TCP. In contrast, many other researchers have focused on designing new protocols from scratch that intend to satisfy the requirements pointed out above. Thus, the transport protocols are specifically tailored to take advantage of high-speed networks, and therefore the relative importance of implementation is lower and consequently the advantages are more sustainable as long as the speed offered by networks increases. As these protocols focus on reducing processing, they are known as "lightweight protocols". A survey of lightweight protocols has been published in [DDK+90]. Examples of these protocols are NETBLT (Network Bulk Transfer) [CLZ87], VMTP (Versatile Message Transaction Protocol) [CW89], Datakit [FM89], XTP (eXpress Transfer Protocol) [SDW92] and SNR (named after its inventors) [NRS90, LT94].

For providing a reliable data transfer service, the standardization bodies chose to include a lightweight protocol which takes advantage of the functionality offered by AAL5. The first application that was in mind of the designers was the support of signaling procedures. Thus, ITU-T chose SSCOP as the protocol to be supported by ATM. SSCOP is based on SNR and Datakit, and the main difference between them and SSCOP is the fact that status reporting is initiated by the sender instead of the receiver; this behavior is convenient for supporting signaling since most signaling operations are analogously initiated by the originator. SSCOP relies on AAL5: it does not include checksumming as AAL5 already provides it.

As mentioned above, SSCOP was first conceived to be the heart of the so-called Signaling ATM Adaptation Layer (SAAL), which is a protocol stack defined by ITU-T in Recommendation Q.2100 [ITU93a]. Later on, ITU-T has agreed on suggesting SSCOP for supporting other applications, such as the Connection-Oriented Network Service (CONS) (Recommendation I.365.2 [ITU94a]) and the Connection-Oriented Transport Service (Recommendation I.365.3 [ITU95b]). As we are also interested in supporting reliable data transfer on top of ATM, we have considered SSCOP as well for the particular case of parallel computing applications. Although the final version of the mechanism comprises many modifications with respect to the standard SSCOP, we have captured the retransmission mechanism, which is the most characteristic feature of SSCOP. In the following subsection, this mechanism is outlined.

5.1.2 Operation of the loss recovery mechanism

SSCOP transfers data to its peer in variable data length protocol data units (usually referred to as "frames"). The sent frames are stored for potential retransmission until the receiver has acknowledged them. The main contribution of SSCOP is the correction of errors and losses by selective retransmission of missing frames via Automatic Repeat Request (ARQ) procedures. In particular, frames are numbered sequentially, and the receipt of frames is explicitly acknowledged. If the receiver determines, through the examination of received sequence numbers , than one or more frames are missing, it explicitly requests the retransmission of the missing frames. Unlike other protocols like TCP, no timeouts are used to detect the possible losses.

SSCOP defines four basic frames for data transfer: SD (Sequenced Data) for user data, and the POLL, STAT and USTAT frames for control flow. Figures 5.1 to 5.4 show the formats for these frames. Table 5.1 displays the SSCOP protocol variables in both sender and receiver peers, as well as the associated PDU parameter when applicable. Other important fields of the SSCOP PDUs are shown in Table 5.2. In the following the function of the SSCOP PDUs that are relevant to the loss recovery mechanism is described.

Table 5.1: *Relevant protocol variables and PDU parameters.*

Peer	*Variable*	*PDU Parameter*	*Meaning*
Sender	VT(S)	N(S)	Sequence number of the current/next SD PDU
	VT(PS)	N(PS)	Current value of the POLL sequence number
	VT(A)	-	Next expected SD PDU sequence number to be ack'ed
Receiver	VR(R)	N(R)	Next expected in-sequence SD PDU sequence number
	VR(PR)	N(PR)	Last received POLL sequence number
	VR(MR)	N(MR)	Maximum acceptable SD PDU sequence number

Table 5.2: *Meaning of the fields common to several PDUs.*

Field	*Meaning*
PAD	Unused octets complementing PDU length to 4-multiple
PL	Pad Length (in octets)
Reserved	32-bit alignment. Other functions, for further study
List	Returned status of the receiver peer

- SD (Sequenced Data) frames, whose format corresponds to Figure 5.1, convey the user data to transfer. Each SD frame contains up to k octets, where k is currently set to 65528 octets, so that SD frames do not exceed the maximum capacity of an AAL5 payload. The *N(S)* field contains the current sequence number, which will be used to detect losses.

- POLL frames are periodically issued by the transmitter in order for the receiver to respond with an update of its status. As shown in Figure 5.2, POLL frames contain in the *N(S)* frame the sequence number of the next new SD frame to be issued by the sender, as well as a "POLL sequence number" in the *N(PS)* field, which functions as a timestamp.

- STAT (Solicited Status) frames are issued by the receiver as a response to the receipt of a POLL frame. They contain the receiver status as far as received and pending SD frames are concerned, by means of the format depicted in Figure 5.3. This information is contained in the "list element" fields in the following way: the first element contains the sequence number of the first missing SD frame; the second one contains the sequence number of the next successfully received SD frame; the third one is the sequence number of the next missing SD frame, and so on. The *N(PS)* field conveys the "POLL sequence number" of the POLL frame to which the STAT frame responds. The With all this information, the sender can acknowledge the successfully received frames and retransmit the missing frames. *N(MR)* and *N(R)* fields are used by the flow control mechanism of SSCOP, which we are not considering in the present work.

- USTAT (Unsolicited Status) frames are issued by the receiver when the sequence number of the last received SD frame is higher that the expected, thus meaning that one or more SD frames are

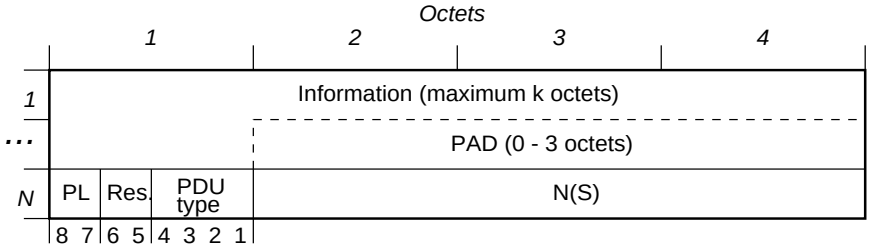


Figure 5.1: *SSCOP SD frame format.*

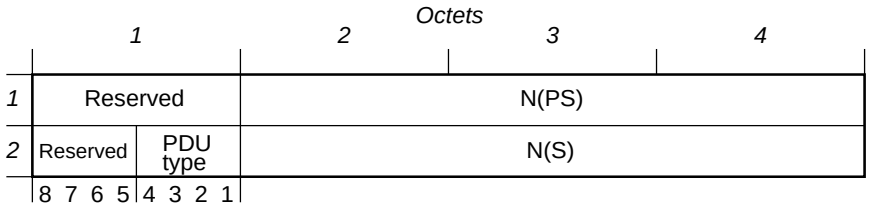


Figure 5.2: *SSCOP POLL frame format.*

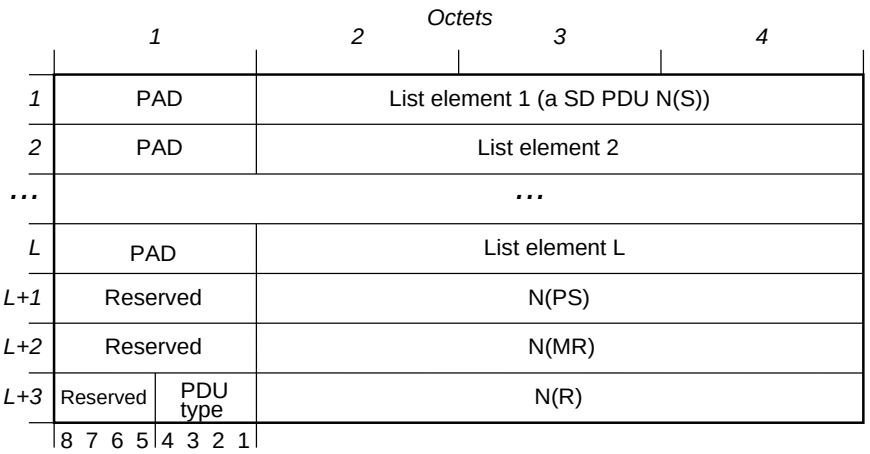


Figure 5.3: *SSCOP STAT frame format.*

missing. In order to accelerate the retransmission of these frames, the USTAT is immediately sent, containing in the list element field the currently expected SD sequence number and the sequence number that the receiver was expecting prior to the receipt of the offending SD frame. The format corresponds to Figure 5.4. As in STAT frames, the *N(R)* and *N(MR)* parameters are used for the flow control mechanism.

By following this scheme, unnecessary retransmissions might occur. In particular, it could happen that the receiver gets a frame retransmitted upon a USTAT frame just after the issue of a STAT frame, which will be received by the sender *after* the retransmission. Thus, the STAT frame will report on some missing frames whose retransmission has been already issued. In order to avoid a new retransmission triggered by the STAT frame, for retransmissions triggered by USTAT frames the next "POLL sequence number" is recorded. If the received STAT frame corresponds to a POLL sequence number that is lower than the

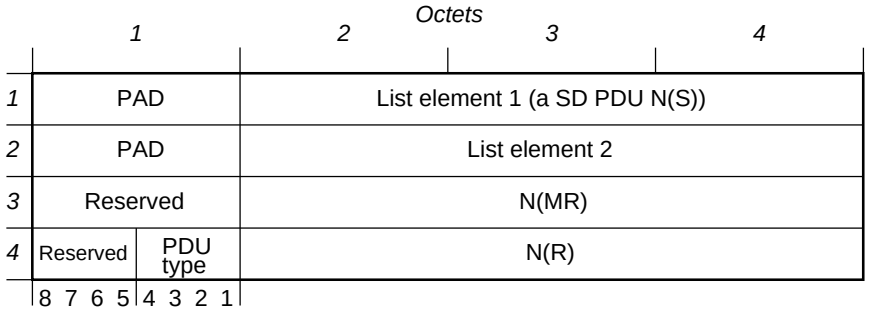


Figure 5.4: *SSCOP USTAT frame format.*

record, the concerned SD frame is not retransmitted, giving an opportunity to receive the retransmission issued upon the USTAT.

Figure 5.5 displays the flowcharts representing the sender behavior in the data transfer service of SSCOP. The main sender loop is depicted in Figure 5.5(a), where three possible events are managed:

- *Sending a new SD frame*. Prior to effectively sending the frame, the record of the current sequence number is increased (variable *VT(S)*), and a copy of the frame is stored in the sender window, together with the current value of the POLL sequence number (variable *VT(PS)*), in order to enable the avoidance of unnecessary retransmissions.

- *Receiving a STAT frame*. The fields of interest are the sequence number of the corresponding POLL frame (parameter *N(PR)*) and the list of sequence numbers reflecting the status of the receiver peer. The sender reads the list and acknowledges the successfully received frames and triggers the retransmission of the missing frames. After reading the list, the sender window is updated. The *N(PR)* parameter is used to avoid unnecessary retransmissions, as explained above.

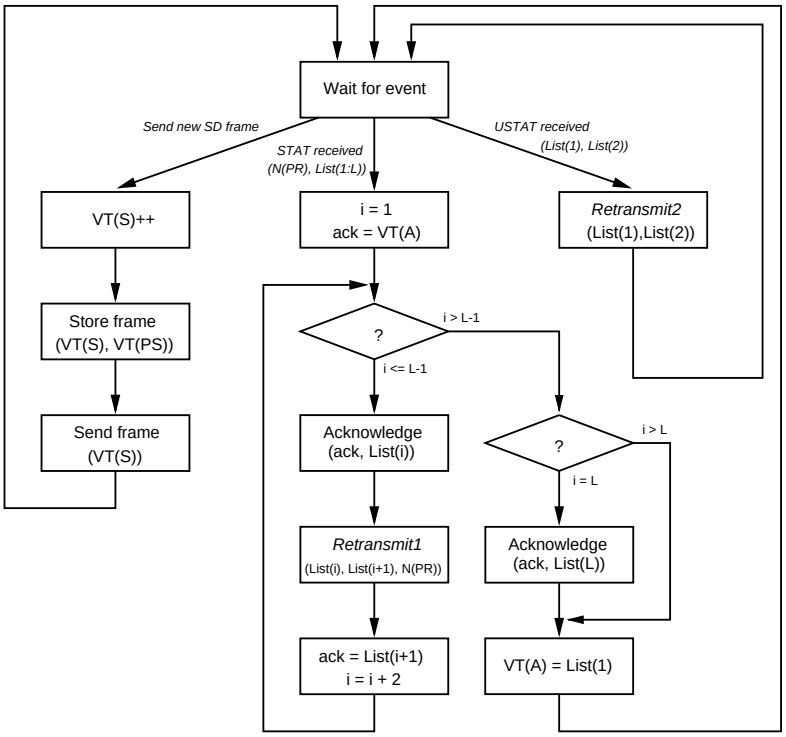
- *Receiving a USTAT frame*. In this case, the only task to do is to retransmit the frames indicated in the list —actually, it contains just two elements. It is not necessary to care about unnecessary retransmitions, since USTAT frames are not supposed to trigger them.

The retransmission procedures used by the main sender loop are represented in Figure 5.5(b). The leftmost flowchart is tailored to avoid unnecessary retransmissions. Thus, for each frame considered for retransmission, the record of the POLL sequence number that was stored in the sender window during the last transmission is retrieved and compared with the POLL sequence number provided with the STAT frame (parameter *N(PR)*. The retransmission is effective only when the parameter *N(PR)* is higher than the record, meaning that the receiver is not expecting any previous retransmission. In case the retransmission is carried out, the record in the sender window is updated to the current value of the POLL sequence number (variable *VT(PS)*). The rightmost flowchart is similar to the leftmost one, excepting for the fact that there is no testing for unnecessary retransmissions, since USTAT frames are not supposed to trigger them. The sender window is then always updated to the current value of the variable *VT(PS)*.

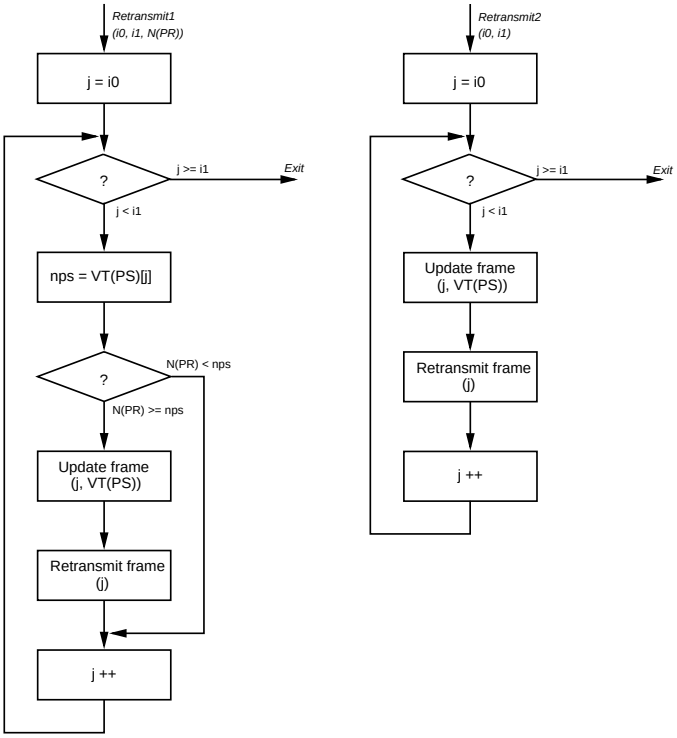
The process issuing POLL frames runs concurrently with the main sender loop, as indicated in Figure 5.5(c). POLL frames are sent with a periodicity T_{POLL}, and include the next in-sequence SD sequence number, as well as the current POLL sequence number. The variable *VT(PS)* containing the latter is incremented with each issue.

The flowchart in Figure 5.6 illustrates the behavior of the receiver peer. Note that it is significantly simpler that the sender behavior, as all functionality of SSCOP is governed from the sender peer. The loss recovery mechanism of SSCOP processes two events in the receiver:

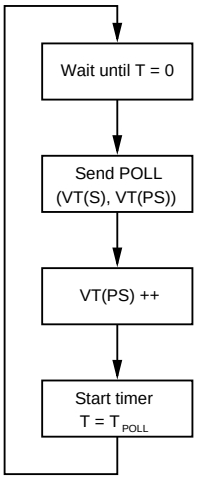
- *Receiving an SD frame*. The sequence number in the received SD frame (parameter *N(S)*) is compared to the expected one (in the variable *VR(R)*), in order to detect possible missing frames. If both numbers are equal, no missing frames have been newly detected and then the expected SD sequence number is increased and the receiver window is updated. If *N(S)* is lower that *VR(R)*, the received SD frame corresponds to the retransmission of a previously missing SD frame and then no action is undertaken apart from updating the receiver window. Finally, the parameter *N(S)* can be higher that the expected number *VR(R)*), meaning that the SD frames with sequence numbers between *VR(R)* and *N(S)* - 1 are possibly missing. In this case, an USTAT frame is build and the variable *VR(R)* is set to expect the next frame after the received one (*N(S)*). The receiver window is updated in the three possibilities.



(a) SD frame transfer



(b) Retransmission procedures



(c) POLL frame generation

Figure 5.5: *SSCOP sender behavior flowchart.*

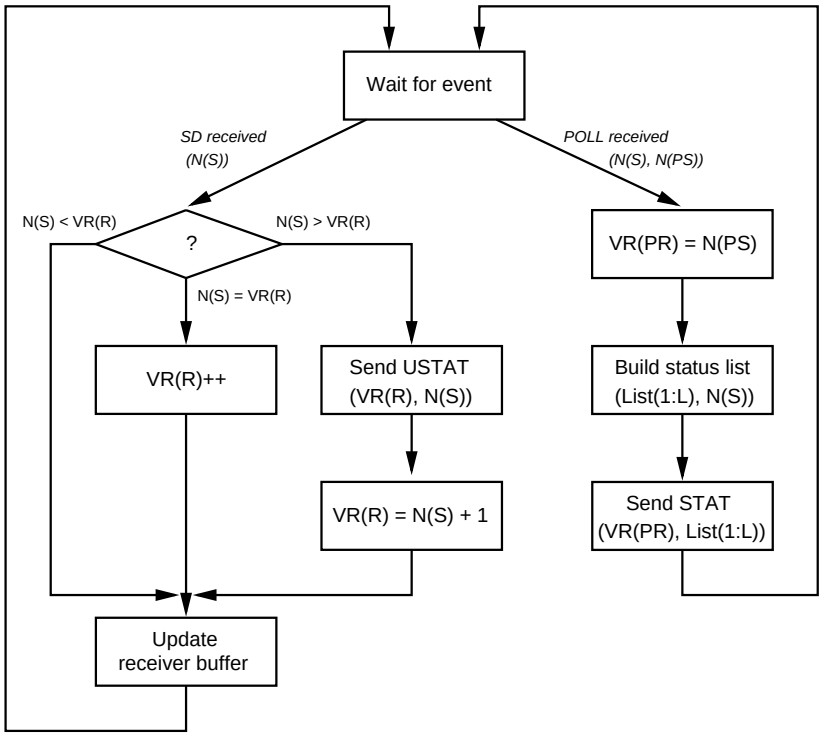


Figure 5.6: *SSCOP receiver behavior flowchart.*

- *Receiving a POLL frame.* With the receiver status stored in the receiver window, together with the *N(S)* parameter corresponding to the next in-sequence SD frame to be issued by the sender, the receiver peer builds the status list indicating both the successfully received and the missing SD frames. This list is sent through a STAT frame including the received POLL sequence number (parameter *N(PS)*) in order to avoid unnecessary retransmissions.

A simple example for the operation of SSCOP is illustrated in Figure 5.7. After the first 4 frames have been sent, a POLL frame is issued. After a while, a STAT frame has been received which acknowledges the correct receipt of those SD frames. After issuing the POLL frame, the transmitter has sent the SD frames numbered from 5 through 9, prior to the next POLL frame, but the frame number 7 has been lost. The receiver detects this loss when frame number 8 is received, and then a USTAT frame is issued requesting the retransmission of frame number 7. The transmitter receives the USTAT frame and immediately retransmits frame number 7, with priority over new data. Note that when the POLL sent right after frame number 9 is received, the retransmission of frame number 7 has not reached its destination yet, so the STAT frame again requests the retransmission of frame number 7. When the transmitter receives the STAT frame, it observes that a retransmission of frame number 7 has occurred after the issue of the POLL frame corresponding to the current STAT frame, so a new retransmission is inhibited expecting that the retransmission issued by the USTAT frame will successfully reach the receiver. If it were not the case, the next STAT frame would stimulate a second retransmission.

5.2 Modifications to SSCOP

As SSCOP is a protocol specifically designed to be supported by ATM networks, it allows to achieve better performance than that of other traditional transport protocols. However, SSCOP is not application-specific,

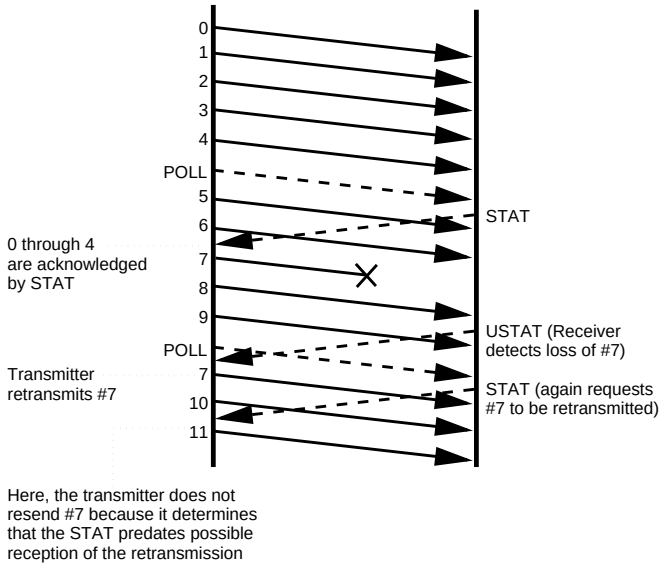


Figure 5.7: *Example of SSCOP operation.*

and therefore it is not adapted to the characteristics of any particular application type. Thus, the specific characteristics of traffic from parallel applications may be captured in order to achieve a new version of SSCOP that be well tuned for cell sequences like those from parallel computing applications.

5.2.1 The "frame corruption by cell loss" problem

Traditionally, networking applications try to maximize the throughput of the networks, specially those managing large amounts of data. For this reason, most packet-based transport protocols rely on variable size packets, whose length applications attempt to keep as long as possible. In case of congestion, whole packets are discarded and, in the case of ARQ protocols, they have to be retransmitted. This is the behavior experienced by common protocols such as TCP.

When packet-based transport protocols are introduced on top of an ATM network, the behavior varies. Indeed, when congestion is experienced, individual cells are lost, as opposed to complete packets, so the characteristic effect of congestion on packet-based transport protocols over ATM is the *corruption* of packets, instead of their *loss*. Thus, the destination endpoint can receive an amount of cells corresponding to corrupted packets. As the whole packet has to be retransmitted, the successfully received cells will be transmitted again, and therefore a significant amount of bandwidth may be wasted, depending on the size of the packets. To reduce the impact of this problem, two approaches can be observed: (1) to alter the network so that useless cells are discarded; and (2) to set the transmission unit in such a way that useless be minimized or avoided.

In [RF94] two strategies altering the network are described: *Partial Packet Discard (PPD)*, and *Early Packet Discard (EPD)*. Partial Packet Discard is an algorithm implemented on ATM switches that consists of dropping all subsequent cells from a packet as soon as one cell has been dropped. It can be implemented on a per-virtual connection basis. Thus, if AAL5 is used, once the switch drops a cell from a virtual connection, the switch continues dropping cells from the same virtual connection until the switch observes that the AUU (ATM User-to-User) bit in the ATM header is set indicating the end of the AAL packet. This cell indicating the end of an AAL packet is not dropped. This approach reduces the amount of useless cells in the network, but does not eliminate them, because all the cells belonging to the corrupted packet that

were received in the switch earlier than the first lost cell will continue in the network. Thus, the amount of useless cells is reduced, but the number of corrupted packets is not.

Early Packet Discard enhances the operation of PPD by discarding entire packets. If the first cell of a packet is received in an EPD switch whose buffer occupancy exceeds a fixed threshold, all subsequent cells belonging to this packet are dropped, even if buffer occupancy falls below the fixed threshold during the reception of the packet. In contrast, if the first cell of a packet comes across a buffer occupancy that is lower than the fixed threshold, all subsequent cells belonging to this packet, even if buffer occupancy grows until exceeding the fixed threshold, unless there is no physical room in the buffer for the cell. Thus, the behavior of ATM switches becomes similar to packet-based switches. In [RF94], the threshold is set to half the buffer size. With this approach, ATM switches have to include mechanisms for monitoring the buffer occupancy. The effective throughput achieved with EPD is clearly better than with PPD and, in the case of sufficiently large buffers, the throughput gets very close to the ideal. Both PPD and EPD are supported by a number of ATM switch manufacturers, for example in Fore's switches [For96].

The simulation results presented in [LST+95] indicate that UBR+EPD does not provide fair bandwidth allocation to the concurrent sources. The operation of EPD favors the connections finding a buffer occupancy lower than the fixed threshold, which is more apparent the longer are the packets. Thus, in case virtual connections conveying short and long packets compete for the buffer, the latter are allocated most of the bandwidth. Both PPD and EPD, in addition, are designed to maximize throughput and bandwidth utilization in the switches, but are not specially conceived to reduce end-to-end delay. Indeed, retransmissions are not directly avoided, so neither EPD nor PPD are not good mechanisms for supporting parallel applications. Adopting a modified version of those mechanism involves all the inconveniences inherent to network-level mechanisms, including the need of implementing the mechanism in all switches in the path. Another solution is to consider end-to-end convergence-level mechanisms.

As shown in the following subsections, the introduction of useless cells can be avoided by encapsulating the "packet" in one ATM cell. Thus, it is not necessary to discard cells in the switches and the receiver. This approach involves a higher degree of overhead —a packet-level overhead in each ATM cell, but the gain in latency may compensate for the lower throughput. In the following subsection, several strategies to build cell-based PDU mapping schemes are discussed.

5.2.2 Cell-based PDU mapping

Current message-passing libraries are designed to be supported by traditional transport protocols, as mentioned in Subsection 4.2.3. Thus, their communication procedures involve encapsulating a certain number of PC-PDUs in larger messages. The PVM (Parallel Virtual Machine) library, one of the most popular ones, operates this way, so we can assume that PVM generates PVM-PDUs which in turn are composed by a number of PC-PDUs. Using a frame-based protocol, as is the case of TCP and the standard SSCOP described in Section 5.1, involves encapsulating a PVM-PDU into a transport packet. To avoid the problems demonstrated in the previous subsection, these PVM-PDUs should be segmented so that each fragment can accommodate in one cell, thus avoiding the generation of useless cells. Two different strategies can be considered for the fragmentation of PVM-PDUs.

- *Unstructured fragmentation.* The PVM-PDU is considered as an unstructured piece of data. Thus, the fragments do not necessarily have any relationship with the PC-PDUs contained in the PVM-PDU, as depicted in Figure 5.8(a).
- *PC-PDU-based fragmentation.* The PVM-PDU is fragmented in such a way that each cell contains an integer number of complete PC-PDUs. Figure 5.8(b) shows an example.

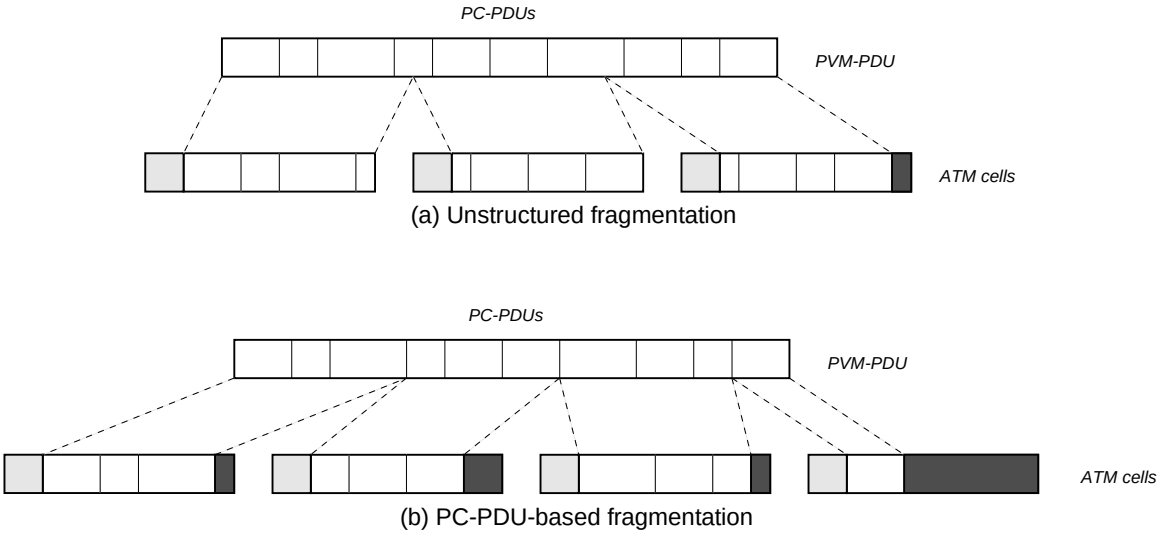


Figure 5.8: *Fragmentation strategies.*

Unstructured fragmentation strategy achieves a higher efficiency as far as the amount of generated cells is concerned, but leads to the reception of incomplete PC-PDUs. Thus, if we are interested in allowing the receiving endpoint to use the received PC-PDUs as soon as received, extra complexity in the receiver is needed to reassemble PC-PDUs, which contributes to increase the processing latency. In contrast, PC-PDU-based fragmentation ensures that all received data are immediately usable by the receiver endpoint, without further complexity. The operation of PC-PDU-based fragmentation enables the deployment of novel message-passing libraries that are based on streaming of PC-PDUs. In the rest of the work, the PC-PDU-based fragmentation is adopted. Although in Figure 5.8(b) all the cell payload is dedicated to encapsulate PC-PDUs, in practice part of the payload will contain some overhead as well as PC-PDUs.

With the fragmentation strategy, the frames to be used by SSCOP can be accommodated in one cell. As standard SSCOP is designed to rely on AAL5, one-cell AAL5 packets can encapsulate the fragments. Nevertheless, this scheme incurs in unnecessary overhead. In particular, the 32-bit CRC is excessive for a one-cell AAL packet —a 10-bit CRC like that included in cells generated by AAL3/4 is probably overdimensioned [AA93]. In addition, the ‘packet length’ field does not need two bytes —6 bits are sufficient. For this reason, we can skip the AAL and submit the fragments from SSCOP directly as ATM cells, without further adaptation. This procedure in fact is equivalent to the definition of a new AAL specialized in conveying PC-PDUs, which we refer to as “Parallel Computing AAL” (PC-AAL).

5.2.3 Encapsulation schemes

As a result of the guidelines expressed above, the SSCOP requires some modification in order to build the PC-AAL. The modifications we performed to SSCOP involve the removal of AAL5-related fields and the addition and/or adaption of further fields that are more related to PC-PDUs or the application-level library supported by the PC-AAL. The key point in the proposed modifications is that the AAL-level PDU will consist of just one cell. With this particular characteristics, SSCOP frames have to be modified. Some modifications apply to all SSCOP frames while others are particular to a certain frame. In the following we discuss the common modifications:

- The SD sequence numbers are allowed 4 bytes (32 bits) instead of 3 bytes (24 bits). This increase is motivated by the presumably higher number of frames transferred as a consequence of the fact that

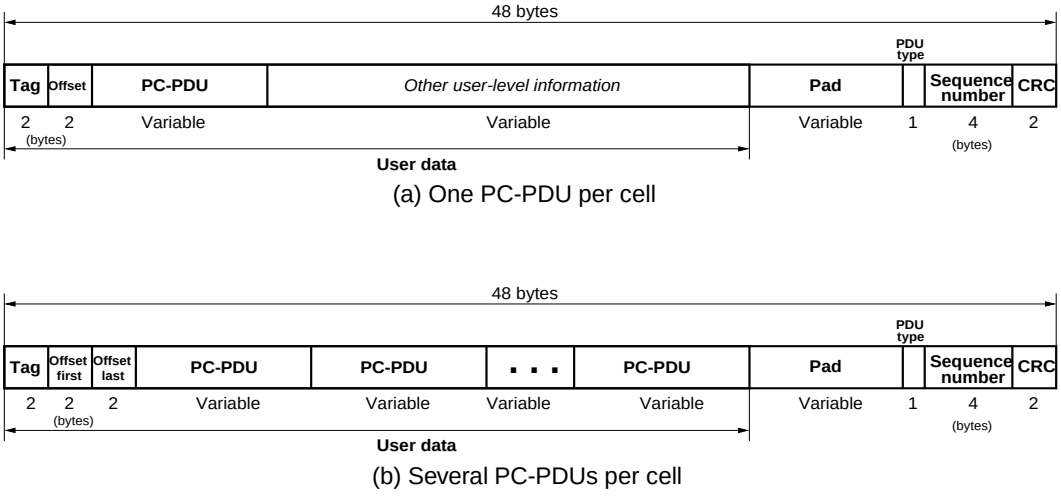


Figure 5.9: *Encapsulation schemes for SD frames (sizes in bytes).*

one frame is accommodated in one cell.

- CRC is set to 16 bits. Although 10 bits would have been sufficient, an integer number of bytes has been preferred for byte alignment purposes. In any case, it is half of AAL5's standard 32-bit CRC.
- No variation have been considered for the 'PDU Type' field (4 bits). In the rest of the discussion, we include four additional 4 bits that are reserved for future purposes in this field, hence the 1 byte length shown in Figure 5.9 below.

After discussing the common modifications, we revisit each SSCOP frame type in order to propose modifications to their format. These modifications are required for letting the information contained in each AAL-PDU be processed without having to rely on the receipt of other AAL-PDUs. In particular, the modifications include the addition of some fields in order to enable compatibility with current upper-level message-passing libraries —which mostly generate large SDUs as they are designed to run over traditional networks such as TCP/IP-based LANs— as well as slight differences in the STAT list generation procedure in order that, when the list has to be contained in more than one cell, the sender can retransmit missing cells without requiring the receipt of all cells composing the STAT list. In the following paragraphs, each frame type is discussed their modifications and particular characteristics in more detail.

As far as SD (Sequenced Data) frames are concerned, two schemes of PC-PDU encapsulation have been considered: encapsulating one PC-PDU per cell, shown in Figure 5.9(a), and encapsulating several PC-PDUs per cell, depicted in Figure 5.9(b). Allowing only one PC-PDU leads to degrade the throughput and to increase the amount of generated cells, but the generation of SSCOP frames is simpler and therefore it is possible to reduce the latency, provided that the achieved performance be not much lower than that obtained when encapsulating several PC-PDUs per cell. In addition, when encapsulating one PC-PDU per cell, the unused space can be dedicated to other functions (for example, adding redundancy). It is highly intuitive that the encapsulation of several PC-PDUs will lead to achieve better performance than the alternative of encapsulating just one PC-PDU. The comparison of both encapsulation schemes will allow to separately determine the contributions of the cell-based PDU mapping strategy and the PC-PDU-based fragmentation.

The 'user data' field in the frame formats shown in Figure 5.9 include the PC-PDUs (one or several, depending on the scheme), together with other user-level informations in case of encapsulating one PC-PDU per cell. In addition, other fields are included that are necessary to support PVM. For each

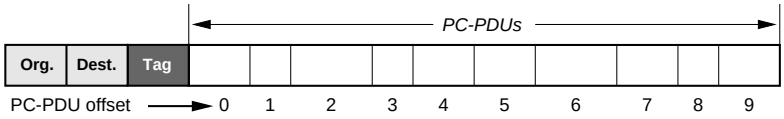


Figure 5.10: *Structure of a PVM message.*

message generated by PVM (PVM-PDU), the programmer associates an integer value (known as a *tag*), that is used by the receiver to distinguish between different incoming messages. Thus, the 'tag' fields in Figure 5.9 contain the tag number of the PVM PDU to which the PC-PDUs in the frame belong. the 'offset' fields indicate the relative position of PC-PDUs in the PVM message, as illustrated in Figure 5.10. When encapsulating one PC-PDU per cell, only one 'offset' field is required; when several PC-PDUs are accommodated in one cell, two 'offset' fields are considered, one for the first PC-PDU and the other for the last PC-PDU. Thus, one cell can contain PC-PDUs from one PVM message only. This scheme is also valid for other message-passing libraries such as MPI (Message-Passing Library) [DOSW96]. Other programming models will require alternative schemes. In particular, if the message-passing library directly generates PC-PDUs, no "offset" field would be required. The development of an efficient PC-PDU-aware message-passing library may enhance in a great extent the features of the PC-AAL.

POLL frames can be accommodated in one cell, even with standard SSCOP. Thus, the only modifications in the POLL frame format include the addition of the 16-bit CRC and the use of 4-byte sequence numbers. The same considerations apply for the USTAT frames.

The STAT frame, in contrast, possibly spans several cells, due to the unspecified length of the status list. Recalling the format of the status list, explained in Subsection 5.1.2, it is organized in pairs of numbers: the first one indicates the start of a sequence of missing frames and the second one represents either the start of a sequence of successfully received cells or the sequence number of the next frame to be issued by the sender —known by means of the POLL frame. An unpaired sequence number may be added if the receiver infers from the information contained in the POLL frame that cells are being successfully received, and therefore it is not possible to determine the end of this sequence. These pairs can be distributed in several cells so that each cell contains complete pairs —plus an unpaired element in the last cell, if necessary. Unlike standard SSCOP, a STAT sequence may be partially received. If the first cell of the sequence is lost, the receipt of a subsequent cell would acknowledge the receipt of all cells until the first element contained in the cell, including the missing frames indicated in the lost STAT cell. To avoid this effect, each cell contains a field with the last element in the STAT sequence indicating the start of a sequence of successfully received frames —or directly the last acknowledged SD frame if the cell contains the first elements in the STAT sequence—, which we call LAF (Last Acknowledgeable Frame), so that the acknowledgment of missing cells is precluded. When a cell containing part of a STAT sequence is received, all SD frames whose sequence number is higher than LAF but lower than the first element in the cell are acknowledged. Figure 5.11 illustrates the operation of this mechanism. Note that, although only one pair per cell is represented, in reality one cell can accommodate as many pairs as possible.

5.3 Experiments

For assessing the performance of the specific AAL for parallel computing, we want to test the response of this AAL to the load in the network produced by the rest of networking applications sharing the ATM network. In frame-based mechanisms, congestion situations may become worse due to the increased number of cells generated as a consequence of the retransmission of large frames. By forcing that the retransmitted cells correspond exclusively to lost cells, we expect to smooth congestion situations. Another

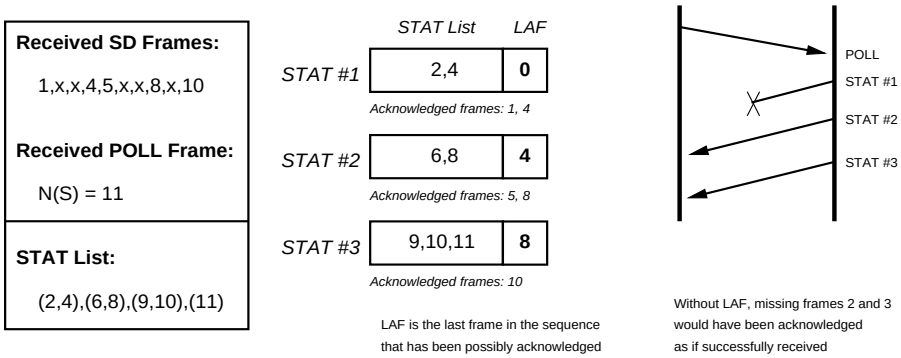


Figure 5.11: *Example of issuing a STAT sequence.*

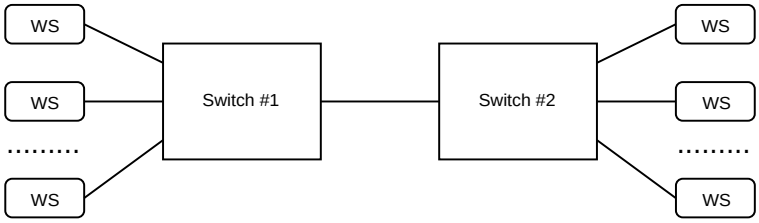


Figure 5.12: *Simulated environment.*

effect of frame-based mechanisms is that the receipt of the whole frame is required for processing the contents. With the PC-AAL, received PC-PDUs can be processed immediately, so latency is expected to decrease. Finally, we will determine the advantage obtained when encapsulating several PC-PDUs per cell, with respect to the performance achieved when encapsulating one single PC-PDU per cell. All the experiments have been performed with simulations. One reason is that we did not have suitable equipment available when experiments were carried out. Another reason is the flexibility in configuring the scenario that the use of simulation allows for.

5.3.1 Measurement scenario

As discussed in Section 1.4, the most indicative performance measure in parallel computing environments is latency. Latency includes the delays introduced in the host and the network along the retransmissions required for the correct reception of data. The contribution of cell-based PDU mapping scheme goes in the direction of reducing cell loss, so performance improvements are expected to be related to the cell loss ratio experienced in communications.

For measuring communication latency when emphasis is made on cell loss, it is sufficient to adopt a simple environment with a bottleneck link for carrying out the measurements. Using a simple environment is also interesting for letting simulation time keep within reasonable limits. Thus, the scenario displayed in Figure 5.12 is sufficient for our purposes. It consists of a number of workstations connected to a network of two switches, where the link interconnecting both switches acts as a bottleneck link. The modeled environment, as specified earlier, provides for the support of traffic from both parallel computing and traditional applications sharing the ATM network. The impact of this background traffic over the network is user-configurable, what enables the study of the response of the PC-AAL under different conditions. As far as parallel computing applications are concerned, each task of a parallel computing applications is assumed to execute on a different workstation, and is allowed to communicate with any of the remaining

Table 5.3: *Parameters of simulation.*

Parameter	Value
Link capacity	155.52 Mb/s
Protocol processing delay	100.0 μs per transport PDU
Additional delay per retransmission	50.0 μs per transport PDU
Interrupt processing delay	75.0 μs per received packet
Bus transfer delay	0.85 μs per cell

tasks. All the necessary connections between communications are assumed to be established prior to execution. A particular communication can be either local (to a task attached to the same switch) or remote (to a task attached to the other switch).

Table 5.3 shows some of the most relevant simulation parameters. The protocol processing delay corresponds to to half of the time invested in processing TCP/IP, as shown in [DDP94]. The reason for considering half of TCP/IP's delay is the fact that the PC-AAL is intended to require less processing than traditional protocols. For each retransmission, the latency experienced by the concerned message is added half the protocol processing —i.e. 25% of TCP/IP processing time— in order to include the part of operations that are performed on each transmission. The values for the rest of delay parameters are taken directly from [DDP94].

The traffic from parallel applications is modeled by traces attached to each source that contain references to calls to communication functions of a message-passing library, which have been obtained from executions of real algorithms over a LAN of SUN 4 workstations. In this work we have considered one algorithm running over the PVM (Parallel Virtual Machine) message-passing library [G+94]: *PDE1*, from the GENESIS benchmark suite [AGH+91], whose characteristics as far as communications are concerned are detailed in Appendix A. In PVM, the basic communication procedures of PVM are `pvm_send` and `pvm_recv`. In standard operation, the sequence for sending a message is `pvm_initsend`, several `pvm_pk`*x*, and `pvm_send`. For receiving a message, the sequence is `pvm_recv`, and several `pvm_upk`*x*. `pvm_initsend` initializes a buffer for transmitting a message; `pvm_pk`*x* adds one or several PC-PDUs of type *x* to the current buffer; `pvm_send` effectively sends the message. In the receiver process, `pvm_recv` retrieves a full message and stores it in a buffer; `pvm_upk`*x* extracts one or several PC-PDUs of type *x* from the current buffer. As observed, this operation is clearly oriented to build large messages, since PVM, like most of the message-passing libraries, was designed to be supported by traditional protocols like TCP/IP.

The encapsulation schemes for the PC-AAL described above, however, do not rely on the transmission of large messages, as the AAL-PDU is as long as one ATM cell. Instead, communications are actually issued by the `pvm_pk`*x* and `pvm_upk`*x*, that is, in the instants when PC-PDUs are explicitly involved. The operation of the PVM calls now becomes like this:

- `pvm_initsend`: A communication (i.e. a buffer) is initiated. Unlike the standard call, the *destination* and *tag* fields, supplied in the `pvm_send` call in the standard operation, have to be provided because the actual transmission occurs *before* `pvm_send`, as shown below.
- `pvm_pk`*x*: In the one-PC-PDU-per-cell encapsulation scheme, as many cells as PC-PDUs are sent, each of them containing just one PC-PDU of type *x*. In the several-PC-PDUs-per cell encapsulation scheme, the PC-PDUs are accumulated in a one-cell-wide buffer that is sent when there is no room for the next PC-PDU.
- `pvm_send`: In the one-PC-PDU-per-cell encapsulation scheme, simply closes the communication. In the several-PC-PDUs-per cell encapsulation scheme, the buffer is first flushed.
- `pvm_recv`: The reception of expected data is initialized.

Table 5.4: *Values for the high state average rate and average link utilization.*

Average peak rate (Mb/s)	Average minimum rate (Mb/s)	ρ	Average peak rate (Mb/s)	Average minimum rate (Mb/s)	ρ
150	10	0.4	400	10	1.1
200	10	0.5	450	10	1.2
250	10	0.7	500	10	1.4
300	10	0.8	550	10	1.5
350	10	0.9			

- `pvm_upkx`: One or several PC-PDUs or type *x* are explicitly demanded. This involves the reception of as many cells as PC-PDUs in the case of encapsulating one PC-PDU per cell, while when encapsulating several PC-PDUs per cell the number of expected cells will be lower.

In addition to the traffic from parallel computing applications, the ATM network is assumed to drive traffic from other networking applications as well. Thus, a variable intensity of background traffic is considered that will compete for networking resources with parallel computing communications. This background traffic is modeled by means of ON-OFF sources, analogously to the model used for the ATM network emulator discussed in Appendix B. Like in this emulator, the attachment of ON-OFF sources is not made on a per-output basis but on a per-switch basis, in order to avoid unreasonable simulation times and, for the same reason, when a cell is generated by the ON-OFF source in a particular switch, it is determined for each output of this switch, with a uniform probability of 50%, whether this outputs holds a copy of the generated background cell. Thus, the risk of correlation that would appear if all outputs hold a copy of the generated cell is reduced. The parameters for the ON-OFF traffic are set for representing the result of multiplexing many background traffic sources on a single switch input.

The average network load is represented by the parameter ρ, which represents the quotient between the traffic using the network and the capacity of this network. The operation of networks is straightforward when $\rho < 1$ because the network is capable of absorbing all the traffic submitted to it although in some periods congestion is experienced. However, when $\rho > 1$, the network is not capable of absorbing all the traffic injected to. The consequence is that finishing a particular communication with such an extreme congestion situation will be considerably difficult. For example, in the subsections below we will observe that standard frame-based mechanisms are not capable of transmitting all the data. The fact that we rely on ATM networks, however, will allow to overcome this effect. Table 5.4 contains the values for the parameters of the ON-OFF sources that have been used to produce the values of ρ used in the experiments.

For validating this environment, we have measured the latency experienced by an 'echo' program, the same as in the validation study of the ATM emulator discussed in Subsection B.2.2 of the Appendix B, in the three studied mechanisms: standard SSCOP and the two encapsulation schemes for the PC-AAL. The conditions are set to be similar to the round-trip study in [TL93]. Thus, by approximating the simulated rout-trip time by twice the latency, we compare the achieved values. For the standard SSCOP measurement, we consider the two packet sizes used in [TL93]. In contrast, in both PC-AAL encapsulation schemes

Table 5.5: *Values considered for the validation study, in μseconds.*

Component	Value
Controller latency	4
Control/data transfer	4.25
Sum of per-cell components	8.25
Vectoring the interrupt	12.5
Sum of per-packet components	12.5

Table 5.6: *Results of validation study, in μseconds.*

Environment	Cells/packet	Measure	Latency	Round-trip time
Standard SSCOP	1	Original measure [TL93]		73
		Simulation	37.257	74.514
Standard SSCOP	29	Original measure [TL93]		746
		Simulation	337.686	675.341
PC-AAL (1 PDU/cell)	1	Original measure [TL93]		73
		Simulation	37.257	74.514
PC-AAL (several PDU/cell)	1	Original measure [TL93]		73
		Simulation	37.257	74.514

we only consider a one-cell packet because the concept of 'packet' does not apply to the PC-AAL as it operates on a per-cell basis. Table 5.5 is the same as Table B.3 and contains the values adopted for the simulation parameters. The comparison of the results, shown in Table 5.6, confirms the validity of the ATM model in the simulator.

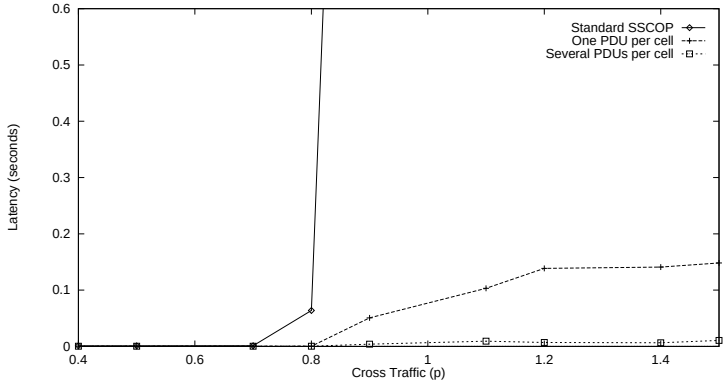
5.3.2 Behavior of PDE1

Our first experiments are intended to compare the performance achieved by one particular application when using both proposed encapsulation schemes with the performance resulting from using the standard SSCOP. The communications pattern generated by the particular application, *PDE1*, consists of a sequence of bursts around 8 KByte long that are sparsely generated along the execution time, as deduced from the information in Table A.1 in Appendix A. In the measurements we consider a wide range of values for the background network load as an indication of the degree of occupancy that the ATM network experiences due to all the applications sharing the network. In particular, we refer to values of $\rho > 1$ in order to determine the behavior of each communication mechanism for extremely high load.

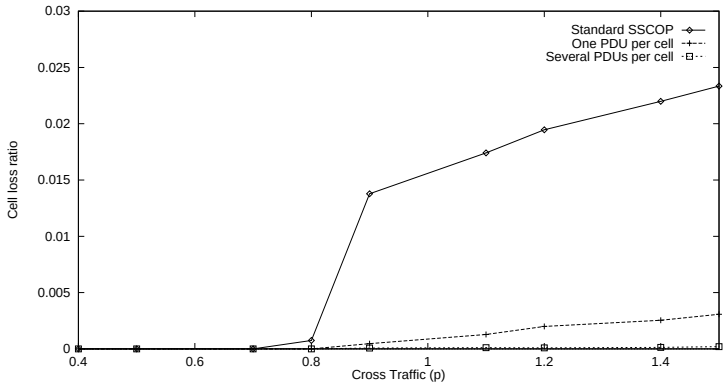
The measured parameter that indicates the performance achieved by communications is the average latency. This latency can be originated by a number of issues, but the proposed PC-AAL is designed to improve the contribution due to retransmission. Thus, we display the experienced cell loss in order to assess the relationship between latency and retransmissions. Finally, we consider the execution time of the algorithm as a measure of the impact of each communication mechanisms that the user will actually observe.

Figure 5.13 displays the results for the first experiment. The two switches in the simulated scenario have been allocated a capacity of 200 cells. Another parameter to set is the interval between successive POLL frames in the SSCOP operation. As it is recommended to use a value as high as possible whenever the offered performance is satisfactory, we first tried with a value of 0.5 seconds for this POLL interval. As expected, as long as the load in the network (ρ) approaches to 1, the standard SSCOP rapidly tends to get unstable. Indeed, the cell loss ratio becomes so high that some cells cannot ever be retransmitted, hence the infinite latency experienced in this case and, as a consequence, the infinite execution time. When using an cell-based encapsulation scheme for our PC-AAL, however, it is possible to successfully finish the execution of the application in all cases, even under extremely congestion. Thus, the first advantage observed to occur with the PC-AAL is the high robustness with respect to the load experienced in the ATM network.

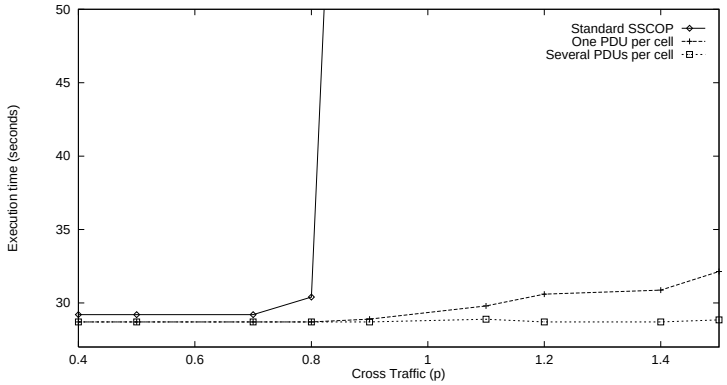
It may be surprising to observe a stable behavior for very high values of ρ. The reason is quite simple though. Figure 5.14 illustrates an equivalent example for three cases: (a) a byte-oriented, frame-based protocol, like BSC; (b) a cell-oriented, frame-based protocol, like SSCOP; and (c) a cell-oriented, cell-based protocol, like the PC-AAL. In all cases, $\rho > 1$ implies that the input cell rate is on average higher than



(a) Average task-to task latency



(b) Experienced cell loss by application



(c) Total execution time

Figure 5.13: *Performance of PDE1 for a buffer capacity of 200 cells and a POLL interval of 0.5 seconds.*

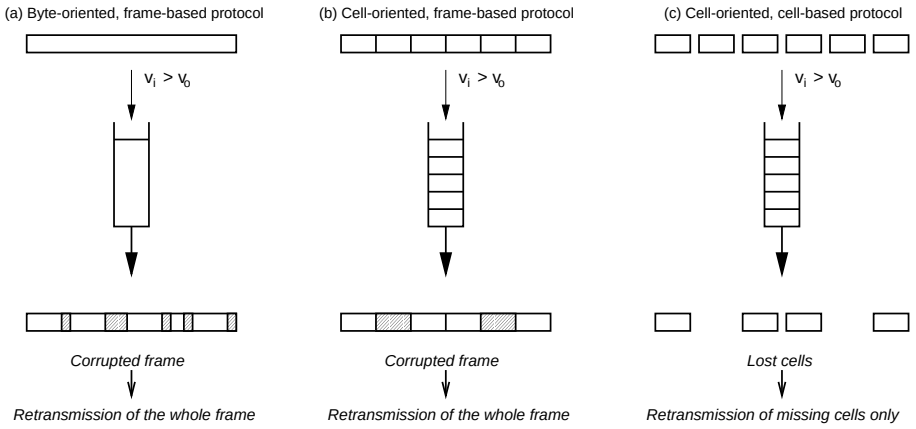


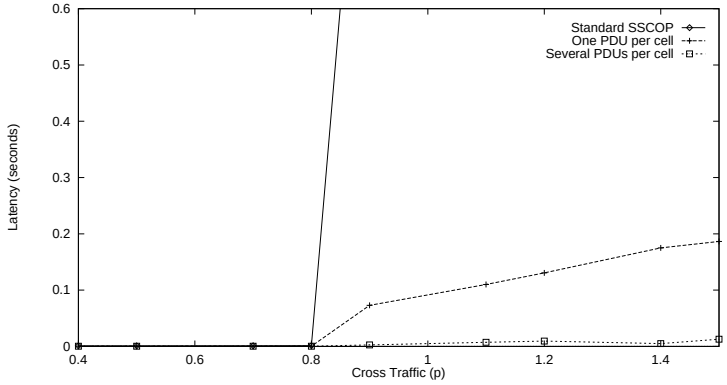
Figure 5.14: *Why cell-based fragmentation leads to stability.*

the output rate. In the first case, in each frame sent under these conditions a part of the bytes is lost due to congestion in the buffer. Thus, these frames become corrupted and therefore they have to be retransmitted, and the successfully received data are useless. If congestion conditions do not cease, it will be impossible to successfully receive the frame, hence the instability. The second case is analogous, except for the fact that the buffer elements are added and removed in groups of 53 bytes —the size of a cell—, instead of individual bytes. Again, the successfully received cells cannot be usefully exploited by the receiver. In the third case, however, the "frames" are not longer than the buffer elements. Thus, information can either be received or not, but never be corrupted, so the receiver is sure that the successfully received data are useful. As a consequence, only missing cells have to be retransmitted.

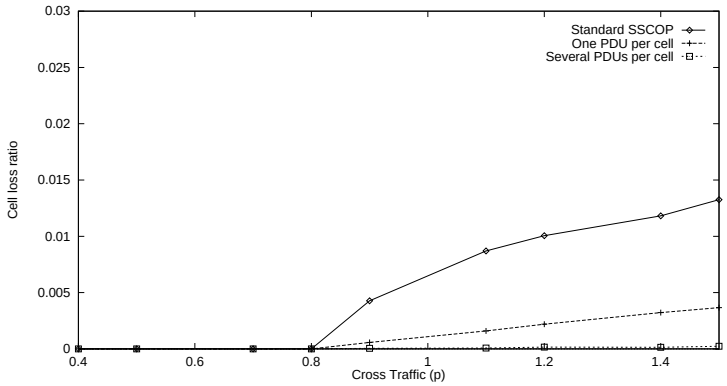
As far as latency is concerned, both encapsulation schemes achieve better performance than standard SSCOP, specially when the background load begins to grow. Note that there is a direct relationship between the latency measurements in Figure 5.13(a) with the cell loss measurements in Figure 5.13(b), what confirms the importance of the impact of cell loss and the subsequent retransmissions on latency. The latency performance of each environment is also reflected in the execution time measurement, where the relationship with latency is very clear for *PDE1*.

The results also confirm the better behavior when encapsulating several PC-PDUs per cell, as compared to encapsulating one PC-PDU per cell. Although the cell loss of the latter encapsulation scheme is not much higher, both latency and execution time are substantially longer. The reason is the large amount of cells generated by the one-PDU-per-cell encapsulation scheme. The comparison of both encapsulation schemes shows that the contribution of the cell-based mapping is as significant as the contribution of the encapsulation scheme, so therefore it is worthwhile to adopt the several-PDUs-per-cell encapsulation scheme for the PC-AAL. In the next measurements we will assess these observations under different environmental conditions.

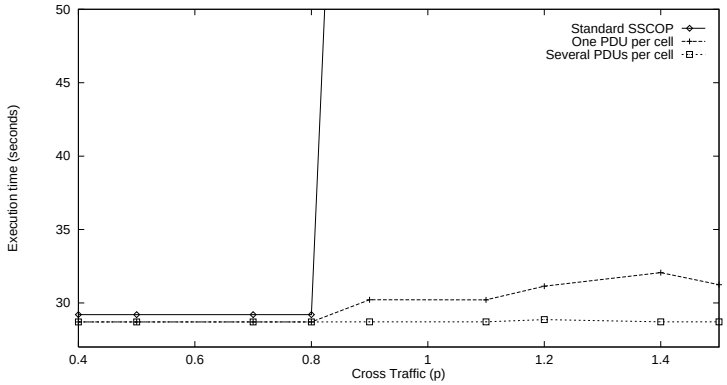
In the first experiment, we fixed the values of the buffer capacity in the switches, as well as the interval between consecutive POLL frames. Since both factors intuitively have an impact on performance, we have repeated the first experiment with new conditions. At first, we consider the influence of the buffer size in the switches, since a larger size can lead to increase latency and to reduce the cell loss rate. In particular, we have assigned a capacity of 1000 cells to the buffers in the switches, instead of 200 cells. A first look to the results, which are displayed in Figure 5.15, shows that the behavior is very similar to the 200-cell case: Standard SSCOP becomes unstable when the background load grows; both encapsulation schemes are stable even for extremely high background loads; and the performance achieved by the several-PC-PDUs-per-cell encapsulation scheme is clearly better than the other encapsulation scheme.



(a) Average task-to task latency



(b) Experienced cell loss by application



(c) Total execution time

Figure 5.15: *Performance of PDE1 for a buffer capacity of 1000 cells and a POLL interval of 0.5 seconds.*

A closer appreciation on the results uncovers the particular influence of the buffer size. The first issue is the lower cell loss experienced by all mechanisms, particularly by the standard SSCOP. The effect of this lower cell loss varies upon the mechanism. In the standard SSCOP, the background load has to reach higher values to produce significant increase in latency, as compared to the 200-cell case. When encapsulating one PC-PDU per cell, the increase of latency as a consequence of network load is higher due to the longer delay occurring in the switches. When encapsulating several PC-PDUs per cell, the effects are not noticeable as the influence of cell loss has little significance in both cases —200 cells and 1000 cells in the switch buffers. The execution times experienced by *PDE1* in this case reflects the trend of the discussed latency measurements.

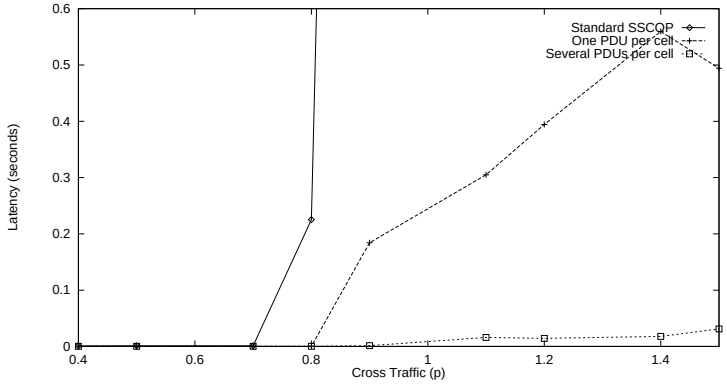
In the next experiment, we study the influence of the interval between POLL frames. A longer POLL interval, for example, is expected to delay the notification of many cell losses and, consequently their retransmission, so latency can increase. In order to assess this behavior, we have measured the three scenarios again. For this purpose, the switches have been allocated a capacity of 200 cells each, as in the first experiment, but the POLL interval has been set to 1.5 seconds, instead of 0.5 seconds. Figure 5.16 shows the results achieved under these conditions. Unlike the previous experiment concerning buffer capacity, a significant influence of the new POLL interval is demonstrated. As expected, there are no significant differences in the behavior as far as cell loss ratio is concerned, in contrast to latency. In particular, standard SSCOP experiences higher latency prior to become unstable, with respect to the first experiment. In the cell-based mechanisms for the PC-AAL, latency also increases but the application remains stable as in the previous cases. The magnitude of the increases has very little significance when encapsulating several PC-PDUs per cell, as opposed to the much higher importance of the increase experienced when encapsulating one PC-PDU per cell. The higher the amount of generated cells, the higher the absolute amount of lost cells and, consequently, the higher the number of retransmissions. Thus, the expected impact of POLL interval on latency will be higher as more retransmissions are required.

The experiments discussed so far consider a single application, *PDE1*. We should check if the conclusions drawn from the results of these experiments can be generalized to other applications. For this purpose, in the next subsection we discuss the behavior experienced by other applications under the same conditions as those experienced by *PDE1* in the measurements studied in the present subsection.

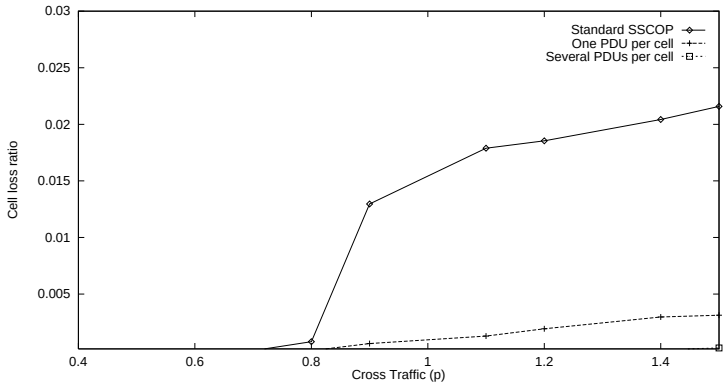
5.3.3 Behavior of PDE2 and SOLVER

PDE2 and *SOLVER* are two applications belonging to the same benchmark suite as *PDE1*. The main differences between the three parallel kernels lie in their communication patterns. While *PDE1* involves the sparse generation of 8 KB-long bursts, in *PDE2* the traffic is more uniformly distributed along the execution time. *SOLVER* reflects an intermediate situation between *PDE1* and *PDE2*, although in fact it is closer to *PDE1* that to *PDE2*. The main objective of the measurements of *PDE2* and *SOLVER* is to assess whether the most outstanding property enabled by the PC-AAL when running *PDE1*, namely the stability of latency even for extremely high background load, is verified by other different applications. For this purpose, we have performed the measurement under the conditions of the first experiment —a switch buffer capacity of 200 cells and the interval between POLL frames set to 0.5 seconds— with other applications.

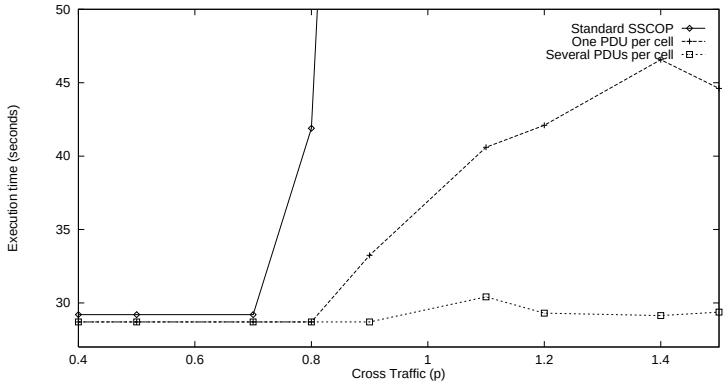
Figure 5.17 depicts the latency and the cell loss ratio experienced by *PDE2* and *SOLVER* and, as a reference, it is compared to the measurement of *PDE1* displayed in Figure 5.13(a). These measurements have been carried out with an encapsulation of several PC-PDUs per cell. The latency results indicate that (1) a expected, communications remain stable when extremely background load in the network is reached, and (2) the characteristics of communications have a significant impact on performance as well, hence the different behaviors experienced by each application. Nevertheless, the relationship between latency and



(a) Average task-to task latency

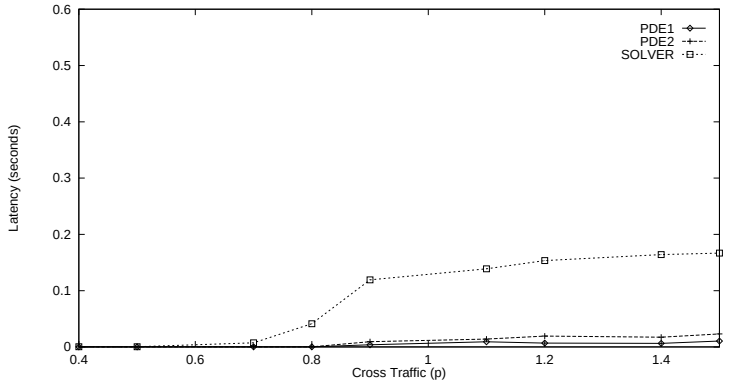


(b) Experienced cell loss by application

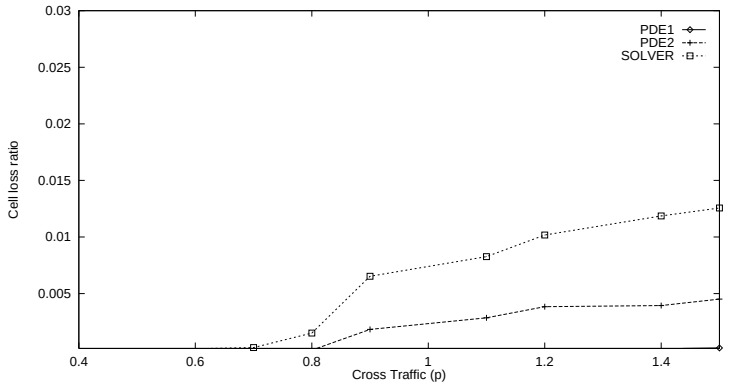


(c) Total execution time

Figure 5.16: *Performance of PDE1 for a buffer capacity of 200 cells and a POLL interval of 1.5 second.*



(a) Average task-to-task latency



(b) Experienced cell loss by application

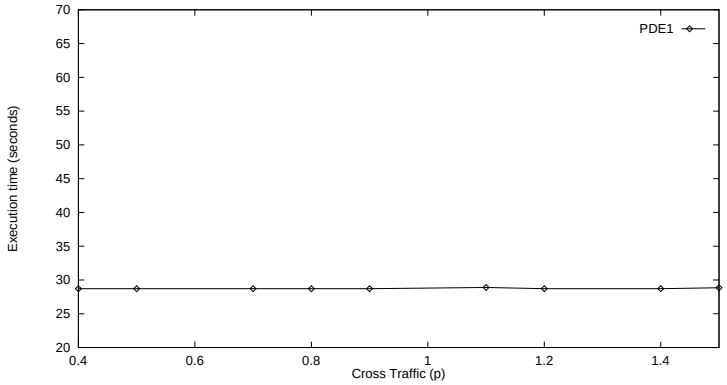
Figure 5.17: *Latency and cell loss of PDE1, PDE2, and SOLVER for a buffer capacity of 200 cells and a POLL interval of 0.5 seconds. Encapsulation scheme: several PC-PDUs per cell.*

cell loss is shown to be very close, as happened with *PDE1*.

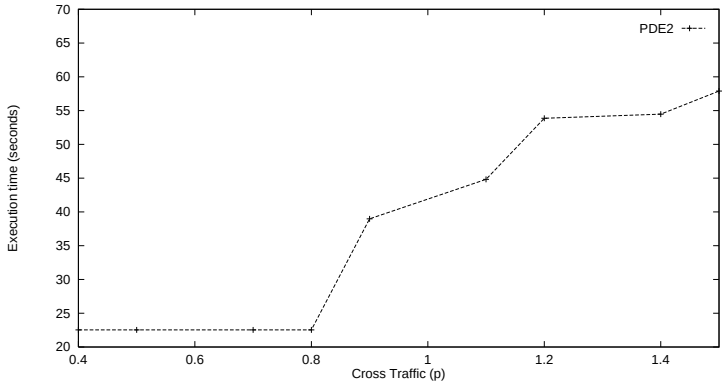
Figure 5.18 reflects the effects of latency on the execution time. These times cannot be compared to one another as each application has its own time scale. The most important result is that latency does contribute to increase the execution time but the impact of other factors has an equivalent impact. This is not apparent in the measure of *PDE1* but is clearer in *SOLVER* and, specially, in *PDE2*. This behavior has to do with the absolute amount of information exchanged by the parallel tasks, as well as the way in which this information is distributed along the execution time. The latter two issues are precisely what make difficult to build analytical models of the kind of traffic generated by parallel computing applications, as each particular application has its own specific characteristics.

5.3.4 Confidence of results

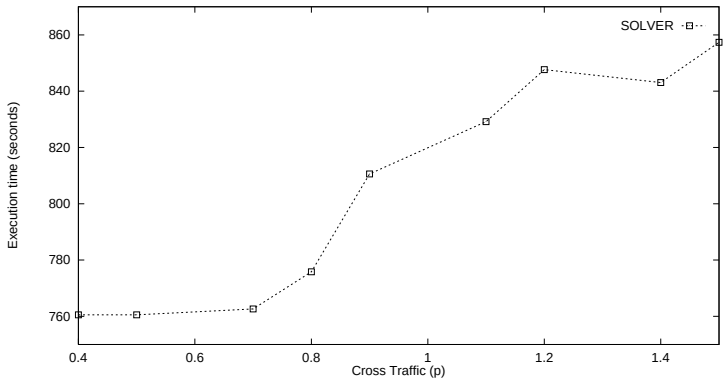
A confidence interval has been determined for the measurements in order to assess the applicability of the conclusions drawn from the results. In experiments where some conditions depend on random parameters, the results are in fact random variables, and the actual value of the parameters is just one of the possible values belonging to the distribution of the random variable. Usually, the mean value of the random variable population is adopted as the representative value of this population. Thus, if the results of two or more measurements belong to populations with different means, we can assure that the



(a) PDE1



(b) PDE2



(c) SOLVER

Figure 5.18: *Execution time of PDE1, PDE2, and SOLVER for a buffer capacity of 200 cells and a POLL interval of 0.5 seconds. Encapsulation scheme: several PC-PDUs per cell.*

Table 5.7: *Confidence intervals for the measured mechanisms (confidence level: 90%).*

Mechanism	Confidence interval (% over mean)
Standard SSCOP	7.60%
PC-AAL (one PC-PDU per cell)	5.16%
PC-AAL (several PC-PDU per cell)	12.30%

results of these measurements are different from one another. However, the population mean is usually unknown. As explained in textbooks on statistical techniques (for example [Jai91]), an estimation of the population mean can be obtained by repeating n times the experiment and computing the average values of the measured parameters. As this estimation cannot be perfect, statistical bounds should be given, so an interval around the sample mean —the confidence interval— is computed. If we have the sample means of some measurements of a parameter whose confidence intervals do not overlap with a certain probability, we can assert that these measurements correspond to different populations with this probability.

As the simulations performed for evaluating the PC-AAL are quite long, we have not determined the confidence interval for all the parameters under all the measured network loads in every experiment. Instead, we have considered the average latency measurements only with the network loads on which the conclusions of the measurements particularly rely. This is possible because the initial conditions are the same in all cases, and the parameters are closely related to one another. Thus, we have repeated 10 times the experiments on which we compute the confidence interval. By using the t distribution and a confidence level of 90%, we observe the radiuses of the confidence intervals displayed in Table 5.7.

It is difficult to obtain good confidence intervals for these measurements because the network load is a random variable as well. As the network load directly conditions latency performance, the samples for estimating the population mean do not strictly rely on equivalent conditions. Nevertheless, in the present experiments, the obtained intervals are good enough to enable the differentiation of the latency performance of the three measured mechanisms. Note that, for the PC-AAL when encapsulating one PC-PDU per cell, the particular confidence interval is $(0.157, 0.175)$, while in the case where several PC-PDUs are encapsulated in one cell, the interval is $(0.0098, 0.0126)$. Thus, both confidence intervals are far from being overlapped, so we can assure that the latencies achieved in both encapsulation schemes of the PC-AAL are different (with a probability of 90%).

5.4 Discussion

The mechanisms presented in this chapter are intended to implement one of the most important functions in charge of the convergence level of the network architecture for supporting parallel computing over ATM, the support to reliable delivery of the data that parallel tasks exchange. Like all the mechanisms in the aforementioned architecture, the latency involved in the operation of the mechanisms is the parameter to minimize. For this reason, a lightweight transport-level protocol has been adopted as the basis of the mechanisms. Instead of designing a whole protocol from scratch, we have used one of the many lightweight protocols already defined. The particular protocol, the SSCOP, has been chosen because (1) it relies on ATM, therefore it is adapted to the addressing characteristics of ATM —VC/VP, etc.; and (2) as it was designed for supporting signaling, it is adequate for services whose data generation pattern is sparse, which is the case for parallel computing applications as well.

Standard SSCOP relies on AAL5, which is best suited for transporting large frames. However, the fact

that the data exchanged between the tasks of parallel computing applications is structured as a sequence of PC-PDUs —each PC-PDU represents, as mentioned earlier, an elementary data type whose length will not exceed a few bytes— allows for the frames to be as short as to fit in single ATM cells. As the overhead in AAL5 becomes useless for one-cell frames, we have defined a specific AAL for parallel computing —the PC-AAL— which relies directly on ATM and provides support to the encapsulation of PC-PDUs and to overlying message-passing libraries. The selected encapsulation scheme consists of including in an ATM payload as many complete PC-PDUs as possible. The other considered encapsulation cell, namely one single PC-PDU per cell, involves the generation of such a large number of cells that the lower delay for generating the convergence-level PDU is very far from compensating the high latency consequent to the harder impact on buffers.

The results from the measurements show that the decision of replacing the AAL5 by a specific PC-AAL whose elements are not longer than one cell leads to achieve a high degree of robustness when the network becomes highly loaded. This is an important result because these situations will eventually occur due to the fact that parallel computing applications share the ATM network with other networking applications. In addition, all the data successfully received can be immediately used by the received as no incomplete PC-PDUs are contained in the cells. Thus, the latency is lower than in mechanisms like standard SSCOP, where the whole frame has to be received to guarantee the complete delivery of the data.

The performance achieved by the PC-AAL is possible because it relies on the fact that switching in ATM networks is cell-based and, consequently, only uncorrupted cells are delivered to the received —the rest are dropped. Thus, the PC-AAL is a step forward to couple SSCOP with ATM. Nevertheless, ATM networks provide much more features that are not exploited by the PC-AAL. In particular, the various ATM service categories can satisfy diverse traffic requirements that eventually can be useful for parallel computing applications. The definition of the PC-AAL presented in this chapter does not provide for a particular ATM service category. The experiments have been carried out by assuming the simplest —and worst-performing— service category, namely UBR (Unspecified Bit Rate). Considering the adoption of other service categories will eventually contribute to enhance the performance of the PC-AAL and, according to the particular strategy adopted to introduce these service categories, the cost involved in the enhanced PC-AAL can remain very reasonable. In the next chapter, these issues are discussed in detail.

6
Exploitation of ATM services

In this chapter, we propose a mechanism specifically conceived for optimizing the cost-performance tradeoff in fairly long parallel executions. The proposed mechanism relies on a modified version of the loss recovery procedure of SSCOP, which is enhanced by means of a more intensive exploitation of ATM service categories in order to reduce the occurrence of cell loss. For this purpose, we make use of both the UBR and ABR service categories, with ABR being only introduced in the periods of high latency. These periods are determined by periodically monitoring the experienced latency. This chapter is based on [VSSP97].

6.1 Enhancing the Parallel Computing AAL

The PC-AAL as discussed in Chapter 5 replaces AAL5 and improves performance by avoiding the retransmission of more cells than those effectively lost. With this AAL, applications are less sensitive to the network load induced by the rest of applications sharing the ATM network and communications achieve better latency performance. This AAL, however, does not rely on any particular ATM service category. If we consider particular service categories for the PC-AAL, we can benefit from their specific properties in order to improve the performance and/or the cost of communications.

6.1.1 Introducing ATM service categories

An important consideration in parallel computing applications is the fact that the exchanged messages are not very long and that they are sparsely issued along execution time, usually hours or even days. This behavior indicates that UBR (Unspecified Bit Rate) and ABR (Available Bit Rate) are the most appropriate service categories to support communications in ATM-based parallel computing environments, since their cost will mostly rely on the effective consumption of bandwidth, as opposed to other service categories where the length of connection period will be a more important issue. UBR is the least expensive service category, but the latency can be excessively high due to the cell loss occurring as the network load increases, while ABR is more expensive but faster, as the built-in flow control mechanism allows to achieve lower latency thanks to the fewer retransmissions needed. However, during the execution time of parallel applications some periods with low network traffic may appear in which the performance of UBR is sufficient and, as a result, the higher cost of the ABR service category is not amortized. Thus, for achieving cost-effective performance the data transfer should be conveyed through UBR when the latency experienced in the network and, when latency through UBR is excessively high, data transfer should be

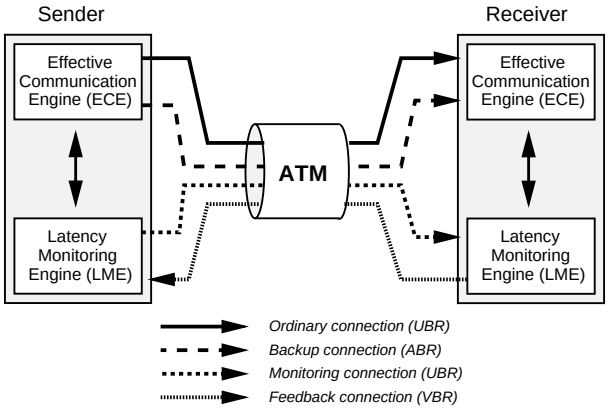


Figure 6.1: *Mechanisms for extending the Parallel Computing AAL.*

moved to an ABR-based connection. As long as the execution time of applications increases, the advantage in cost with respect to the use of an ABR-based connection all the time is more significant.

We focus on the data transfers occurring during execution time by assuming that the necessary connections have been established prior to the execution, in such a way that the VCI/VPI (Virtual Channel/Virtual Path Identifiers) fields in ATM cells will be sufficient for one endpoint to address all the remote endpoints. We will consider that three connections have been previously established between each pair of communicating endpoints: one using the UBR service category with an unlimited peak rate; another using an ABR service category with a limited peak rate and a minimum bit rate set to zero; and a VBR (Variable Bit Rate) service category with a guaranteed low peak rate. These three connections are used by two mechanisms to be included in each endpoint: (1) The *Latency Monitoring Engine (LME)*, which monitors the latency in the network in order to determine the periods in which high latency is experienced, and (2) the *Effective Communication Engine (ECE)*, which performs the actual data transfers according to the information supplied by the LME. The ECE uses the UBR-based connection —herein, *ordinary connection*— for transferring data when latency is low, and the ABR-based connection —herein, *backup connection*— as an alternative connection when the LME indicates that latency is high. The LME monitors latency through the UBR-based ordinary connection, and returns feedback information through the VBR-based connection, since this information requires a fast and reliable delivery. Later in the paper we will observe that the adoption of a VBR-based connection on cost does not significantly impact on performance of parallel computing applications. Figure 6.1 displays a scheme of this configuration. The mechanisms enhancing the PC-AAL will obviously introduce additional overhead (including scenario setup and run-time signaling) but, as noted in Subsection 1.2.2, parallel computing applications are considered to execute during quite long periods —hours or even days. Thus, the impact of the overheads will not introduce a substantial performance degradation.

6.1.2 The Latency Monitoring Engine (LME)

The goal of the LME is to provide an estimation of the latency experienced in the ordinary connection. This estimation can be done concurrently with the data transfer process performed by the ECE. In order to propose a concrete operation for the LME, we considered different options for (1) computing the latency; (2) when the operation mode is actually switched; and (3) which particular implementation is the best suited, as discussed in the following:

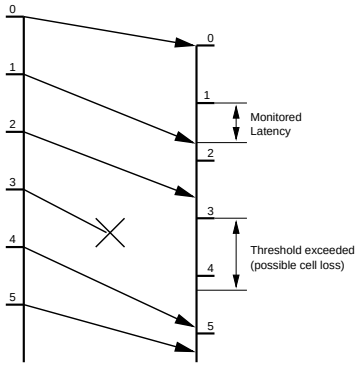


Figure 6.2: *Operation example for the time-stamped LME.*

- *Averaging vs. instantaneous monitoring*. Latency can be monitored by computing the average latency over a period of time. This is well suited for applications dealing with large chunks of data, like video and file transfers, but as this procedure has a slow response time, it is not convenient for applications generating more bursty traffic patterns. Therefore, we believe that instantaneous monitoring is a more adequate approach for parallel computing applications.

- *Asynchronous vs. periodic activation*. Latency can be monitored either before a burst of messages or in a periodic fashion. The former case forces the ECE to defer the transmission until the latency is monitored, so it involves a significant amount of latency. In contrast, the latter approach enables the ECE to avoid this delay. For this reason, we believe that a periodic LME is more adequate, despite the extra bandwidth required to support periodic monitoring. Nevertheless, the operation of the periodic LME can be improved by allowing monitoring to be performed on each data transfer as well, so in fact we use a hybrid approach.

- *The monitoring mechanism*. We considered the following options: (1) using network-level information; (2) computing the Round Trip Time (RTT); and (3) synchronizing peers and using time-stamped information. The first approach requires the use of a ABR-like network level mechanism providing accessible feedback information, which is not currently standardized within ATM. In the second case, the computed time depends on the latencies of both the monitored connection and the returning path, which are not necessarily equivalent. In the third approach, the experienced latency is monitored by the receiver LME peer, so there is no influence of the returning path on the computed value. As a result, we adopt the third approach as we find that it suits better the requirements of parallel computing communications.

The operation of the adopted approach for the LME is as follows: the sender periodically submits a cell containing a time-stamp. When the receiver gets this cell, it compares its time-stamp to the time the receiver expected to get the cell. The measured latency corresponds to the difference between both times, and then the measurement is passed to the ECE so that it takes the appropriate action, which in the implementation of the ECE discussed above consists of replying to the sender if the monitored latency exceeds a threshold. As the time-stamped cells might be lost, when a certain amount of time T_L has elapsed since the expected time, the receiver warns the ECE of that circumstance, meaning that a monitoring cell is possibly lost. Figure 6.2 illustrates the operation with an example. As an enhancement to this basic procedure, the cells issued by the ECE through the ordinary connection are also monitored their latency in order to reduce the response time of the whole mechanism. As in the periodic part, the result is passed to the ECE. In this asynchronous part, no action is carried out by the LME if the ECE cell is lost.

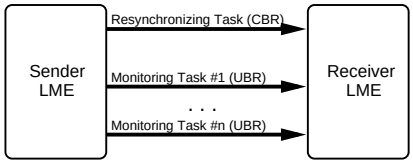


Figure 6.3: *Tasks and connections used in the LME.*

It is important to note that both the sender and the receiver must be synchronized to each other in order for the measurements to be significant. For this purpose, one of the peers has to report the other one on its current time with a certain periodicity. Thus, we consider two tasks included in the time-stamp LME: (1) *Monitoring task*, which deals with both the periodic and the ECE-originated time-stamped cells; and (2) *Resynchronizing task*, which guarantees that time values are consistent for both communicating peers. We can make use of the different service categories provided by ATM in order to implement these tasks.

The Monitoring Task is carried out over the same connection as the ordinary data transfers in the ECE, so it is supported by a UBR service. The Resynchronizing Task requires also high priority and, as it is periodic, a CBR service is more adequate. Note that the peak rates for the CBR service should keep low in order to avoid the allocation of an excessive amount of resources. The concrete value of the period depends mostly on the characteristics of the system clocks in both communicating peers, since the more diverging the clocks are, the more frequently the Resynchronizing Task should be activated. Figure 6.3 shows the tasks and the services used to implement them. In the experiments presented below, we will assume both endpoints as perfectly synchronized and, therefore, no resynchronizing task is considered.

6.1.3 The Effective Communication Engine (ECE)

The Effective Communication Engine (ECE) consists of an extension of the Parallel Computing AAL —PC-AAL— described in Chapter 5 that allows to exploit the information supplied by the LME in order to achieve low latency communications. This mechanism is based on a modification of the selective retransmission procedure of SSCOP. A summary of the standard retransmission mechanism of SSCOP can be found in Section 5.1. The modification to SSCOP is addressed to limit the length of the frames to one cell. Thus, unlike standard SSCOP, the amount of retransmitted cells corresponds exactly to the lost cells and, as a consequence, applications become less sensitive to network load. This modification is possible thanks to the short length of PC-PDUs, as characterized in Subsection 4.2.3. This mechanism succeeds in providing robust operation against network load but its performance can be improved with the reduction of the required retransmissions.

The ECE enhances the preceding mechanism by considering two operation modes: *low-latency mode*, and *high-latency mode*. In low-latency mode, the operation of the ECE reduces to the mechanism of the Parallel Computing AAL as discussed in the previous chapter. The transfers of data take place over the ordinary connection, so a UBR service is used. Latency monitoring by the LME takes place also over this ordinary connection using a UBR service. When the LME detects a significant growth in the latency through the ordinary connection, the high-latency mode is entered, where the backup connection is activated. The action undertaken with the ordinary connection in high-latency mode determines several versions for the ECE. One possibility is to deactivate the ordinary connection, so cells solely use the backup connection in high-latency mode. We call this strategy the *Switching ECE*. Another possibility is not to deactivate the backup connection, so cells can be redundantly submitted through both the ordinary and the backup connection. We know this strategy as the *Duplicating ECE*. Figure 6.4 outlines the operation of these versions of the ECE.

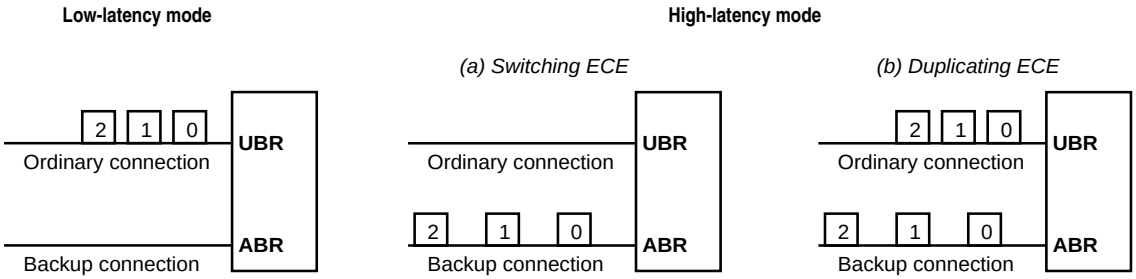


Figure 6.4: *Operation of ECE's latency modes.*

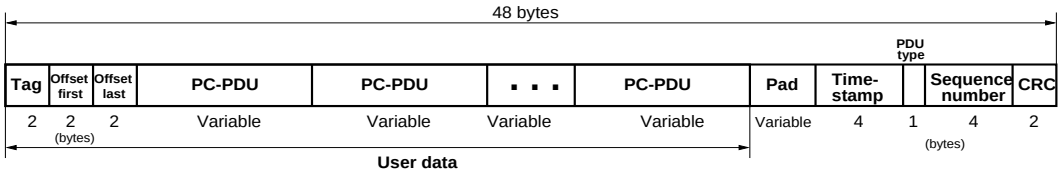


Figure 6.5: *Modified encapsulation scheme for the ECE.*

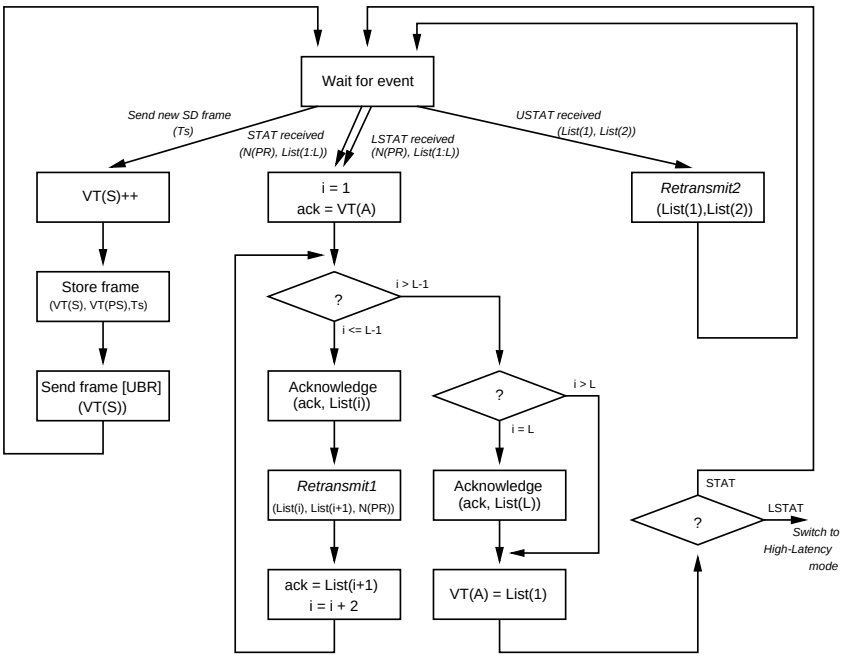
The higher amount of data managed by the Duplicating ECE will result in a higher bandwidth consumption as compared to the Switching ECE but, on the other hand, some advantage in latency is potentially possible. As the ordinary connection is based on the UBR service category, the cell rate can be faster that in the ABR-based backup connection. Thus, if a cell sent through the ordinary connection in high-latency mode is not dropped, it can reach its destination faster than its replica sent through the backup connection. Anyway, if that cell is lost, it is not necessary to trigger its retransmission as the replica in the backup connection will then play the role of the retransmitted cell. Both phenomena can contribute to accelerate the delivery of cells. The actual extent of these theoretical advantages will be discussed in the performance evaluation study in order to assess whether the possible better performance is sufficient to compensate for the extra bandwidth consumption with respect to the switching ECE.

As discussed in Section 6.1.2, we have adopted a LME with a a periodic nature. Ideally, this monitoring should be done very frequently so that the estimation be sufficiently realistic, but then substantial overhead traffic is introduced in the network. For this reason, in order to achieve accurate enough estimations without very high monitoring periodicity, we have added a timestamp field to the PC-AAL frame format. Then, the reaction to latency increases will be faster, while periodic monitoring will be useful for keeping a record of the experienced latency in the periods where no traffic from parallel computing applications is generated. Figure 6.5 shows how the encapsulation scheme in Figure 5.9(b) is modified. In the following we detail the operation of the two versions of the ECE in both operation modes.

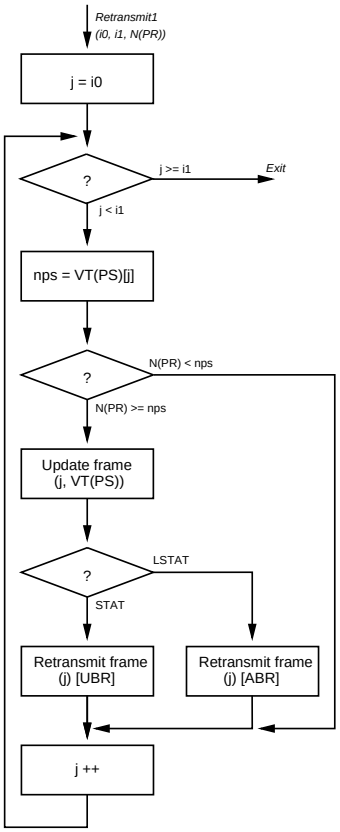
(a) Switching ECE, low-latency mode

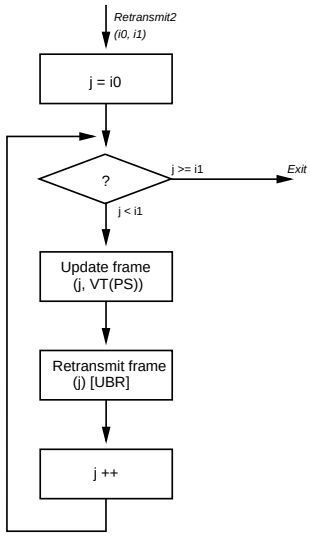
The behaviors displayed in Figures 6.6 and 6.7, corresponding to the sender and receiver respectively, are shown to be similar to the operation of SSCOP as described in Subsection 5.1.2. As far as the sender is concerned, the actions undertaken when sending an SD frame or receiving STAT and USTAT frames are substantially similar. The differences lie in the POLL frame generation and the reception of a new frame type, LSTAT (Latency-triggered STAT). These differences are effective when switching from low-latency mode to high-latency mode.

The modification in the POLL-generating process consists of including LME data in the POLL frame, thus avoiding redundant traffic, as both POLL frames and LME are periodic. In Figure 6.6(c), the periodicity of POLL frames is assumed to be a multiple of the LME period, N_{POLL} being the multiplicity.

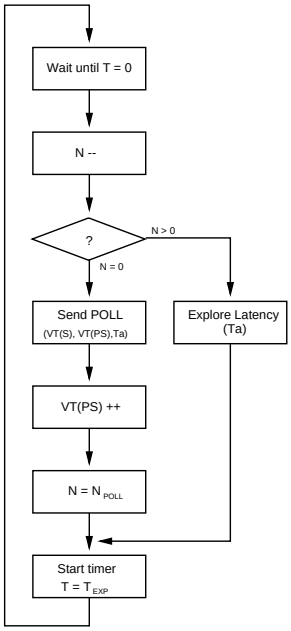


(a) SD frame transfer





(b) Retransmission procedures



(c) POLL frame generation

Figure 6.6: *Sender ECE behavior in low-latency mode (switching ECE).*

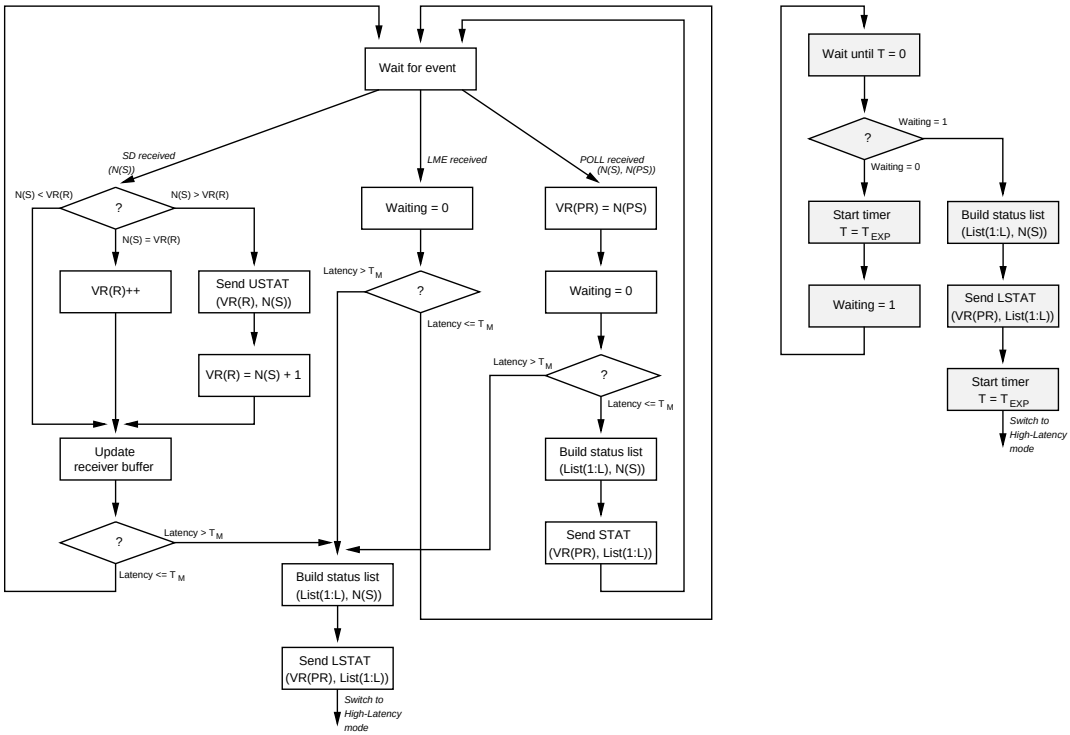


Figure 6.7: *Receiver ECE behavior in low-latency mode (switching ECE).*

Thus, after $N_{POLL} - 1$ monitorizations of latency, the next one is performed via a POLL frame. The T_a parameter represents the current timestamp, and is used by the LME.

LSTAT frames are sent by the LME when a change of operation mode is required. They contain the same information as STAT frames, as well as a timestamp (the parameter T_o) indicating the instant that the cell whose experienced latency motivated the change of operation mode. In the switching ECE, the action undertaken is the same as for the reception of STAT frames. The only difference is found in the retransmission procedure, where retransmissions are performed through the backup connection (using ABR) instead of the ordinary connection (using UBR) as done when a STAT frame is received.

The receiver behavior shown in Figure 6.7 includes the action undertaken by the LME, which is described below. The other two possible events continue to be the reception of SD and POLL frames. The procedures in both cases have been modified in order to allow for them to interact with the LME. Thus, the LME uses information of both frame types for monitoring latency. When latency exceeds a threshold T_M, an LSTAT frame us built, analogously as when POLL frames trigger the generation of STAT frames, except for the timestamp added to LSTAT frames indicating the instant when the offending SD or POLL frames where issued. In low-latency mode of the switching ECE, when an LSTAT frame is triggered by a POLL frame, the STAT frame is not generated in order to avoid redundant information. Another difference between the ECE and the original PC-AAL is that all STAT, USTAT, as well as LSTAT frames, are issued through VBR in order to accelerate their delivery.

(b) Duplicating ECE, low-latency mode

The operation of the duplicating ECE in low-latency mode is essentially the same as the switching ECE, which in turn is similar to the PC-AAL as described in the previous chapter. In the sender size, the modifications with respect to the original PC-AAL are concerned again with the POLL frame generation and the processing of LSTAT frames. The operation of generating POLL frames is identical to the process

in the switching ECE, where POLL frames take advantage of LME's monitoring process.

The action undertaken on receipt of an LSTAT frame is clearly different. The most outstanding difference, as shown in Figure 6.8, is the use of a new retransmission procedure —labeled *Retransmit3*— that is executed prior to the standard retransmission process. This new process involves the retransmission *through the backup connection* of all the SD frames issued from the sender between the instant when the SD frame triggering the mode change was sent —indicated by T_o in Figure 6.8— and the instant of reception of the LSTAT frame —represented by T_a. The reason behind this mechanism is the fact that, in the duplicating ECE, all cells issued in high-latency mode are issued through both the ordinary and the backup connection. The retransmitted SD frames correspond to those frames issued immediately behind the frame or monitorization experiencing a latency higher then T_M, which should be issued in high-latency mode but were not because the sender was not notified yet of this circumstance. Thus, this retransmission scheme realizes the duplication that was not performed earlier. Another difference lies in the *Retransmit1* mechanism where, in case of being triggered by an LSTAT frame, the retransmission is performed through both the ordinary and the backup connections, as opposed to the use of the backup connection only carried out by the switching ECE, which is another consequence of the particular characteristics of the duplicating ECE.

As far as the receiver is concerned, the operation is the same as in the switching ECE, illustrated in Figure 6.7. The only difference is the possibility of receiving duplicated SD frames, through both the ordinary and the backup connections, which is not possible in the switching ECE, when the low-latency mode is recently entered. Therefore, the duplicating ECE needs special procedures to deal with this case.

(c) Switching ECE, high-latency mode

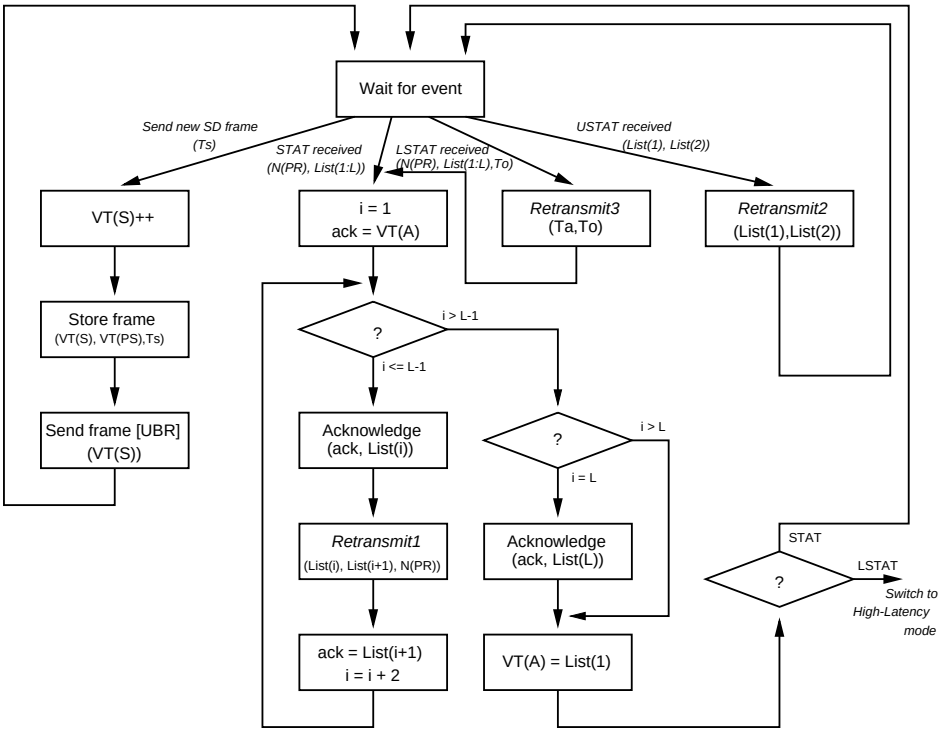
In high latency mode, the behavior of the sending peer of the switching ECE does not experiment strong variations with respect to low-latency mode, apart from the fact that all data transfers take place through the backup connection —using ABR— instead of the ordinary connection —using UBR—, which is illustrated in Figure 6.9. The differences are found in the retransmission procedures —particularly, when retransmission is triggered by the reception of STAT and LSTAT frames— and the POLL frame generation process.

Regarding retransmission, the only variation is the exchange of roles of backup and ordinary connections. In particular, receiving an STAT frame involves that the retransmitted SD frames will be delivered through the backup connection while LSTAT frames will use the ordinary connection, as this event is related to a change to low-latency mode. Nevertheless, the most outstanding difference is the fact that, unlike low-latency mode, POLL frames are not used by the LME to monitor latency. The reason is that these POLL frames are issued through the backup connection, like the rest of data, while monitoring is carried out over the ordinary connection only.

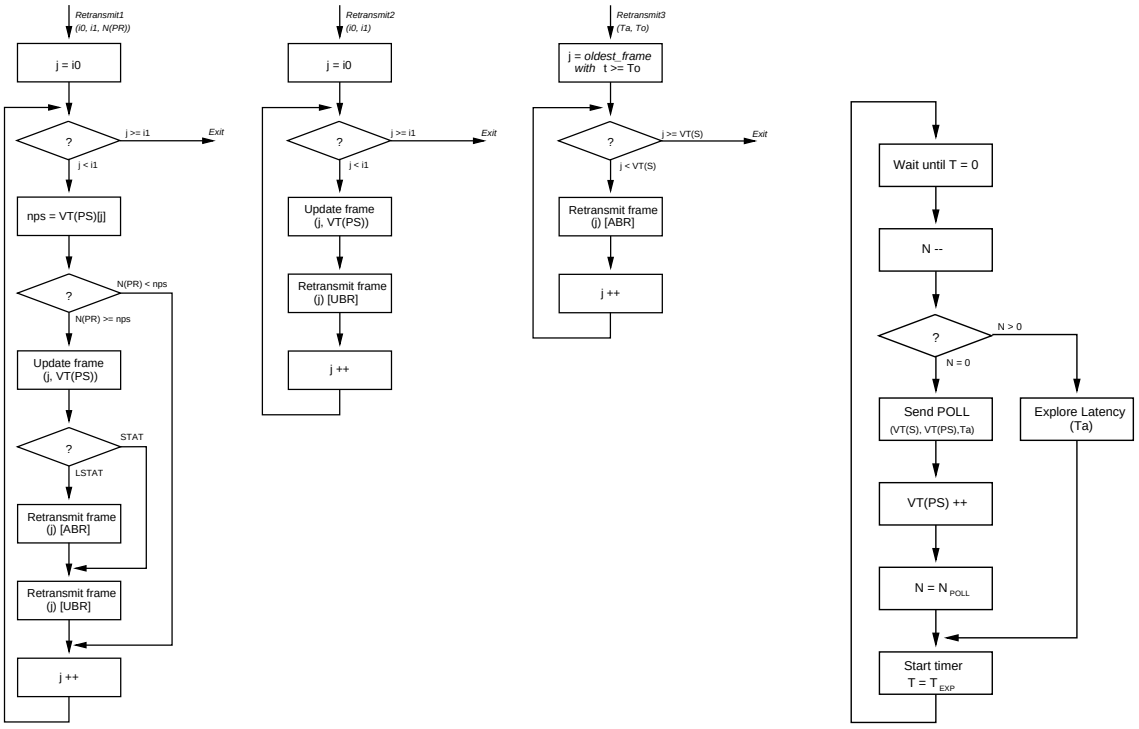
As far as the receiving process is concerned, shown in Figure 6.10, it is practically equivalent to the receiving process in low-latency mode. The process in high-latency mode, nevertheless, is simpler because neither SD frames nor POLL frames are used by the LME for monitoring latency. Thus the generation of LSTAT frames can be solely triggered by the periodic monitoring process. The transition to low-latency mode is realized when the monitored latency falls below the threshold T_m, which is necessarily lower than the threshold T_M that applies for the transition between low-latency and high-latency modes.

(d) Duplicating ECE, high-latency mode

The operation of the duplicating ECE in high-latency mode a little more complicated than the preceding modes, specially for the receiving peer. The sender part, however, is very similar to the low-latency operation mode of the same duplicating ECE, as reflected in Figure 6.11. As a characteristic of the duplicating ECE, each SD frame is issued simultaneously through both the ordinary —UBR— and the

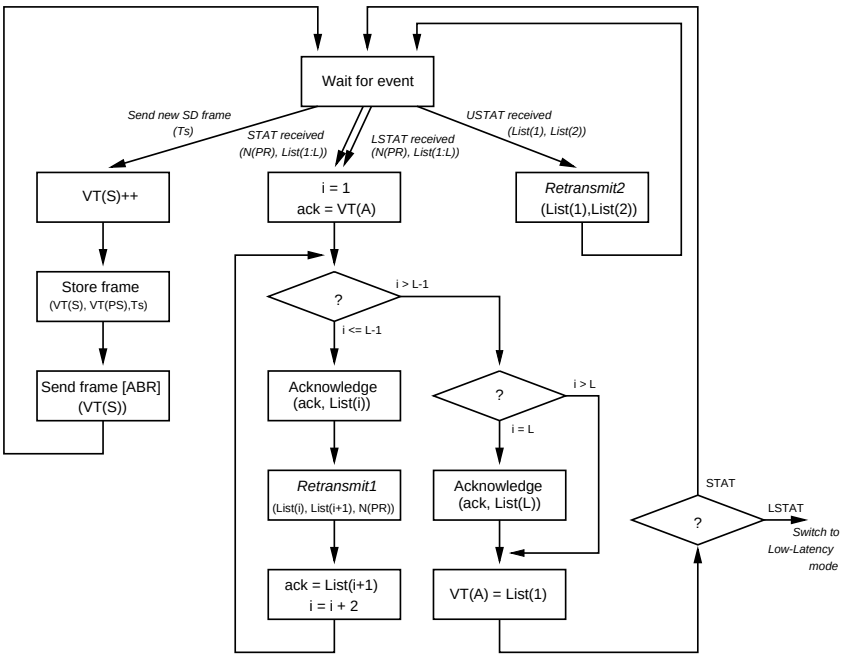


(a) SD frame transfer

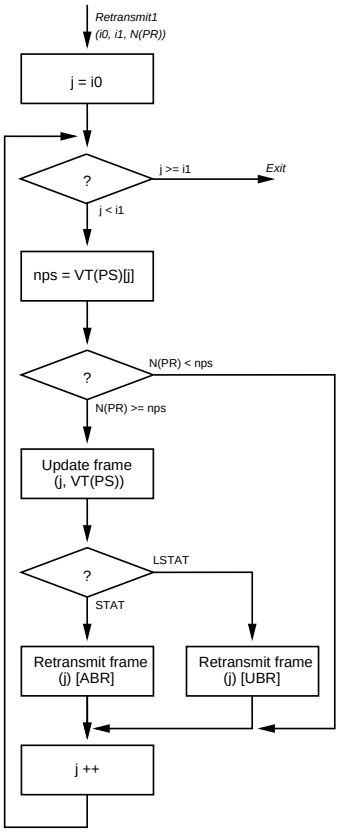


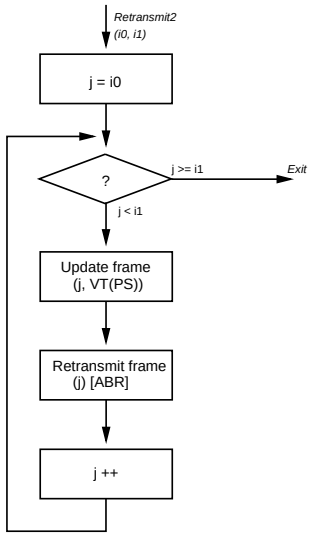
(b) Retransmission procedures (c) POLL frame generation

Figure 6.8: *Sender ECE behavior in low-latency mode (duplicating ECE).*

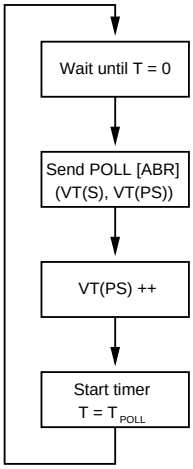


(a) SD frame transfer





(b) Retransmission procedures



(c) POLL frame generation

Figure 6.9: *Sender ECE behavior in high-latency mode (switching ECE).*

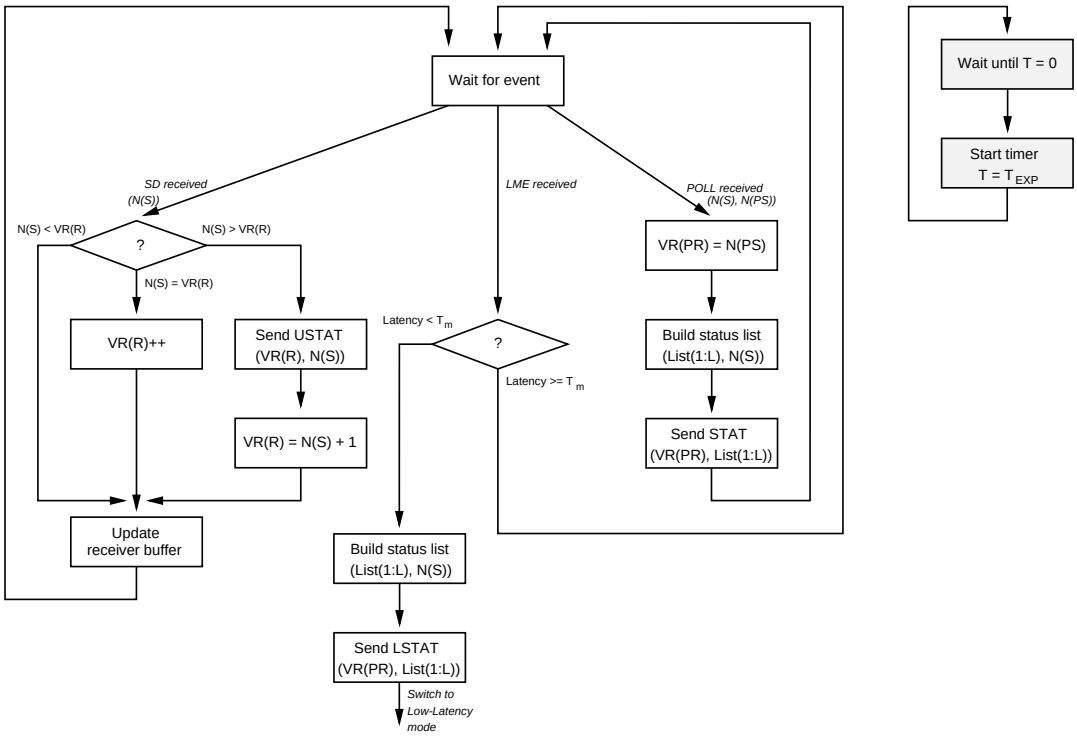


Figure 6.10: *Receiver ECE behavior in high-latency mode (switching ECE).*

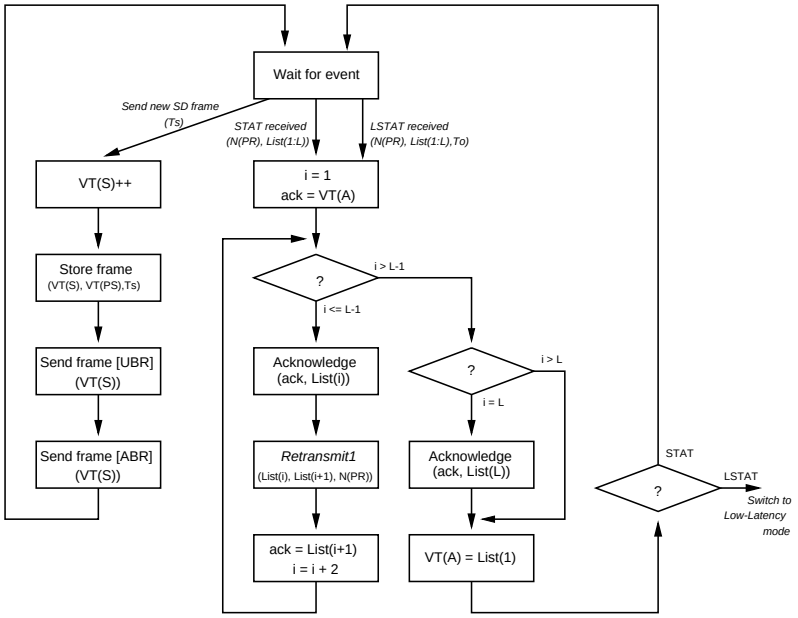
backup —ABR— connections. Analogously, retransmissions triggered by STAT frames are issued again through both connections, while retransmissions subsequent to LSTAT frames are supported by the ordinary connection only, since the ECE is switching to low-latency mode. The most remarkable difference is that USTAT frames are not processed in this case. The reason is that, in the duplicating ECE, there is no need of requesting the retransmission of missing SD frames in the ordinary connection, as a replica has been already issued through the backup connection, hence the lack of USTAT frames.

Monitoring latency, unlike in the switching ECE, is carried out by POLL frames and the replicas of SD frames that are issued through the ordinary connection, in addition to the periodic monitoring that is independently performed. POLL frames use the ordinary connection, like low-latency mode and unlike high-latency mode in the duplicating ECE.

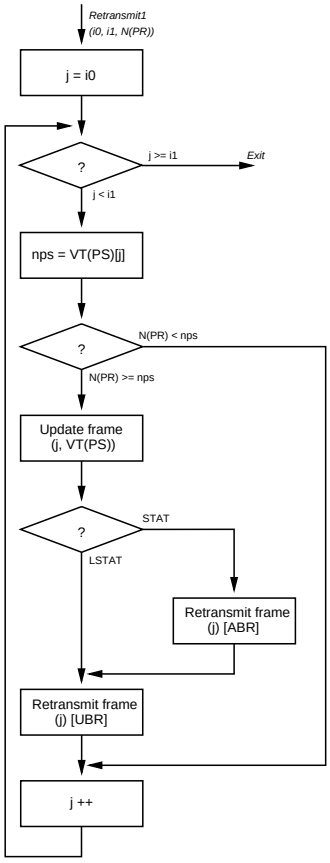
The receiving part, which is displayed in Figure 6.12, is slightly more complicated that in the low-latency mode due basically to the need of covering the duplicates. As two copies of each SD frame are expected, it is likely that in some arrivals through one connection it is found that the frame has been successfully received through the other connection. For this reason, a duplicate detection process has been added. The rest of procedures is similar to the low-latency operation mode, except for the fact that USTAT frames are not generated when the sequence of SD frames is broken. We adopted this behavior as the replicas of the lost frames have probably been already issued and consequently, as expressed above, there is no need of requesting their retransmission.

6.2 Performance measurements

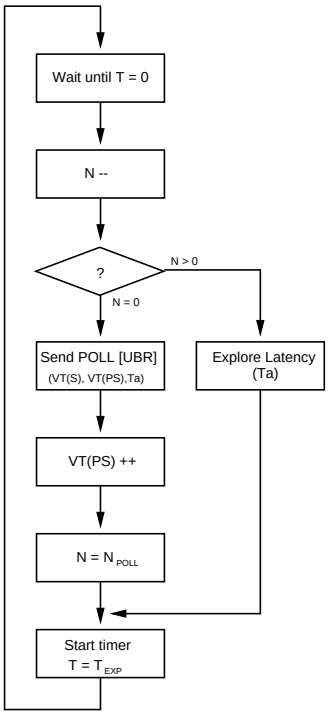
To characterize the performance, the average end-to-end latency has been measured in a simple configuration, in order to realize the impact of the mechanism. The cost of the mechanism is also determined



(a) SD frame transfer



(b) Retransmission procedures



(c) POLL frame generation

Figure 6.11: *Sender ECE behavior in high-latency mode (duplicating ECE).*

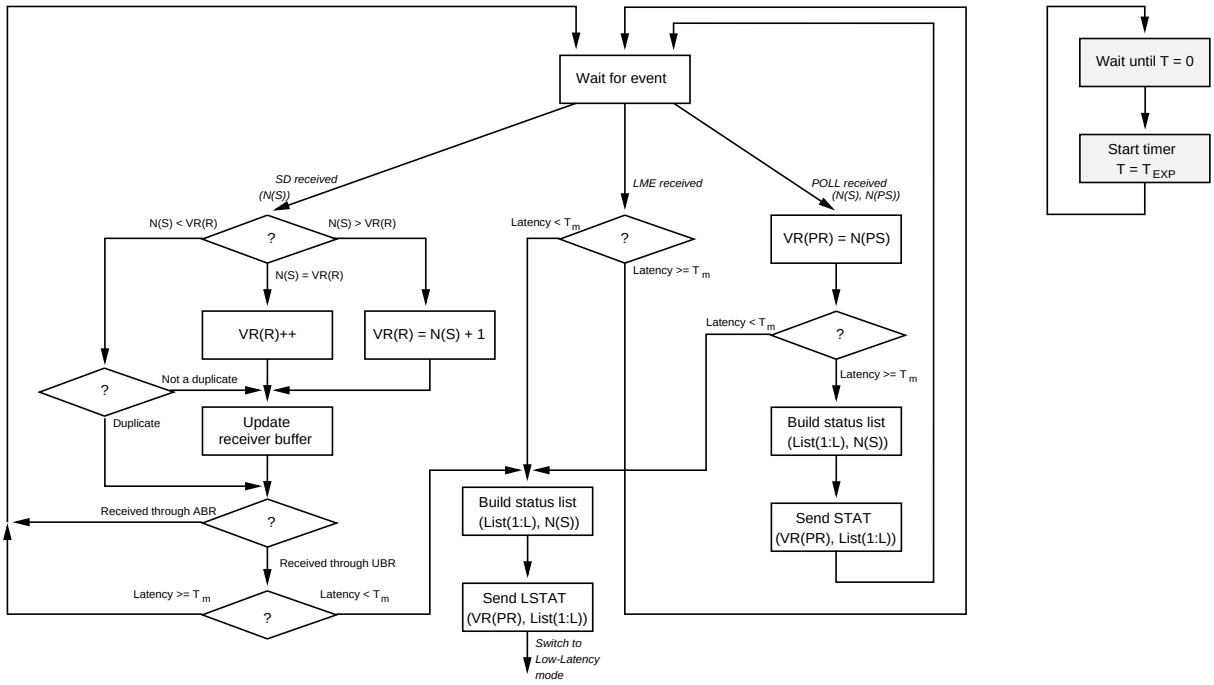


Figure 6.12: *Receiver ECE behavior in high-latency mode (duplicating ECE).*

and compared to that of the standard ABR service. Other measurements have been performed in order to decide which ECE is best suited for supporting communications of parallel computing applications. In additions, we have assessed the impact of several issues concerning the ABR service category, namely the peak and the minimum cell rates. All the measurements have been carried out by simulation because of the lack of standard ABR compliant equipment.

6.2.1 Experiment configuration

For moderate network sizes and buffer capacities, the most significant contributions to latency come from the bottleneck links in the ATM network, due to the cell loss and subsequent retransmissions occurring when becoming congested. Thus, the configuration shown in Figure 6.13 is sufficient for evaluating the performance of the proposed mechanism, and is simple enough to allow for simulations to keep within a reasonable duration. All the links have a capacity of 155 Mb/s. Two types of sources are considered: one *data source* modeling traffic from a real parallel computing application by means of a trace, and a number of *background sources* modeling traffic from traditional networking applications, by means of ON-OFF sources. The traffic generated by the data source corresponds to the messages generated by one task of the parallel computing application. In contrast, the traffic from each background source represents the result of multiplexing many sources of traffic from traditional networking applications.

The traces for the data source have been collected from the execution of parallel codes from the GENESIS benchmark suite [AGH+91], whose description can be found in Appendix A. In particular, the considered codes have been *PDE1* and *PDE2*, which are two parallel equation solvers using different algorithms. The traffic generated by *PDE1* consists of relatively long bursts (around 8 KB). In contrast, bursts from *PDE2* are much shorter (50–100 Bytes). As a result, different behavior is expected for each code. Actually, the traces are the same as those used in the experiments of the previous chapter. In particular, we have considered the trace file corresponding to one source. All the data is assumed to be

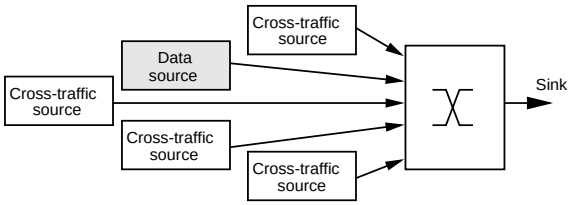


Figure 6.13: *Simulated environment.*

Table 6.1: *Values for the relevant parameters of the ABR service.*

Element	Parameter	Value
Switch	Target Utilization	0.9
	Measurement Interval	30 cells
Source	Nrm	32 cells
	ADTF	0.5 sec
	Peak rate	50 Mb/s

forwarded to the same output despite the various destinations in the trace, so as to simulate the effect of a bottleneck link. The `receive` instructions are by-passed, so the measured execution time is optimistic. This is not a problem because we are interested in other parameters that are collected on a per-cell basis which, as a consequence, are independent of the execution time[1].

As far as background traffic sources are concerned, the values for the parameters of both the ON and OFF states are exponentially distributed. In the measurements, several sets of values have been used in order to obtain diverse aggregate input rates. In particular, the network utilization ρ ranges from 0.3 to 1.1, with respect to the output link capacity. As each background source models the result of multiplexing several sources, as mentioned above, we do not want a very aggressive background traffic. Thus, the parameters of the ON-OFF models generate a traffic pattern with a burstiness not higher than required to capture the characteristics of multiplexed cell streams. As demonstrated in several papers, for example [RV91, dM95], their burstiness decreases as long as the number of multiplexed sources grows.

The switch is modeled as output-queued. Two priority levels are considered: one for guaranteed service categories (in particular VBR), and the other for best-effort service categories (ABR and UBR). The buffer space is shared by the logical queues associated with each priority level. The buffering scheme is basically drop-tail, except for the case of a full switch buffer, where the arrival of a non-UBR cell forces the dropping of an UBR cell already queued in the switch. The aggregate incoming traffic is arranged in order for the switch to contemplate it as a mixture of UBR and ABR traffic. The ABR scheduling algorithm adopted in the measurements in based on ERICA (Explicit Rate Indication for Congestion Avoidance), fully described in [JKG+96]. Table 6.1 shows the values for the most relevant parameters in the switch and the sources, which in turn are mostly based on the defaults suggested in [ATM96c, JKG+96, JKVG95]. Table 6.2 displays the values for the parameters used in the performance evaluation study presented in this section.

For validating this environment, we have carried out two tests. The first one measures the latency experienced by an ‘echo’ program, the same as in the validation study of the ATM emulator discussed in Subsection B.2.2. The conditions are set to be similar to the round-trip study in [TL93]. Thus, by approximating the simulated rout-trip time by twice the latency, we compare the achieved values. We only consider a one-cell packet, as all the simulated mechanisms operate on a per-cell basis. Table 6.3 is

[1]This is not exactly true, since in the real situations the blocking caused by data requests alter the message generation pattern and, consequently, the conditions are not strictly equivalent. The difference would lie essentially on a longer spacing between some messages.

Table 6.2: *Parameters for our low-latency mechanism.*

Parameter	Value
SSCOP POLL interval	0.1 sec
LME monitoring interval	0.1 sec
LME loss threshold T_L	0.1 sec
ECE latency threshold T_M	0.0001 sec
ECE latency threshold T_m	0.00009 sec

Table 6.3: *Values considered for the validation study, in μseconds.*

Component	Value
Controller latency	4
Control/data transfer	4.25
Sum of per-cell components	8.25
Vectoring the interrupt	12.5
Sum of per-packet components	12.5

the same as Table B.3 in Appendix B and contains the values adopted for the simulation parameters. The comparison of the results, shown in Table 6.4, confirms the validity of the ATM model in the simulator.

The second test is concerned with checking that the behavior experienced by the modeled ABR-based connections corresponds to what is expected for real ABR connections. For this purpose, we have considered a greedy ABR source generating traffic with an intensity of 80 Mb/s. As background traffic, we have considered a variable number of sources generating traffic according of the Poisson style. The more congested the switch gets, the effective rate experienced by the ABR source should tend to a "fair share" that will depend on the number of active background sources. Figure 6.14 displays the results of the test, and we observe that the variation of both the traffic generated by the background sources and the number of active sources leads to the expected results. Note that no cell loss is experienced by the tested ABR-based connection.

6.2.2 Task-to-task latency

Task-to-task latency is the measure determining the effective impact of communications on the performance of the parallel environment. As we assume that the ATM network is shared with other applications, we expect important variations on performance according to the load of the ATM network. Figure 6.15 shows task-to-task latency as a function of the different values for the background load. We have compared our proposals for enhancing the Parallel Computing AAL with the AAL without these enhancements, the latter by considering both UBR and ABR as the service categories conveying the data.

Both of our mechanisms achieve equivalent performance as that obtained by relying on an ABR service all the time. However, we have to consider other facts, such as the effective utilization of the ABR service and the bandwidth consumption for assessing the actual advantages achieved by our mechanisms, as well as the advantages of each proposed ECE. The relative performance of the measured approaches depends on the particular characteristics of the communications in each application —traffic from *PDE1* is much

Table 6.4: *Results of validation study (1-cell packet), in μseconds.*

Measure	Latency	Round-trip time
Original measure [TL93]		73
Simulation	35.557	71.114

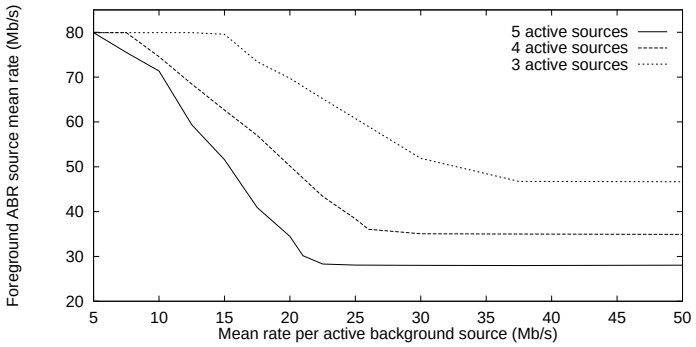
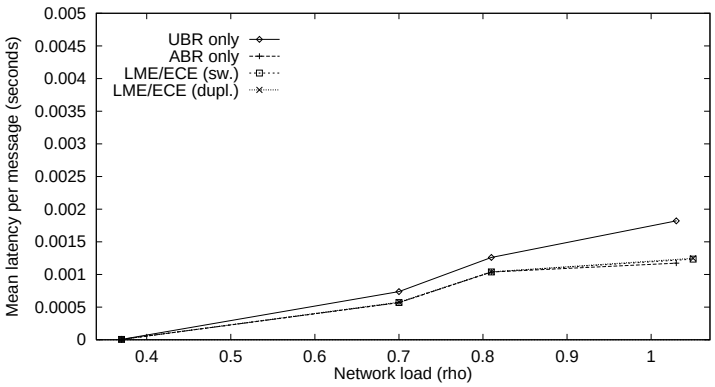
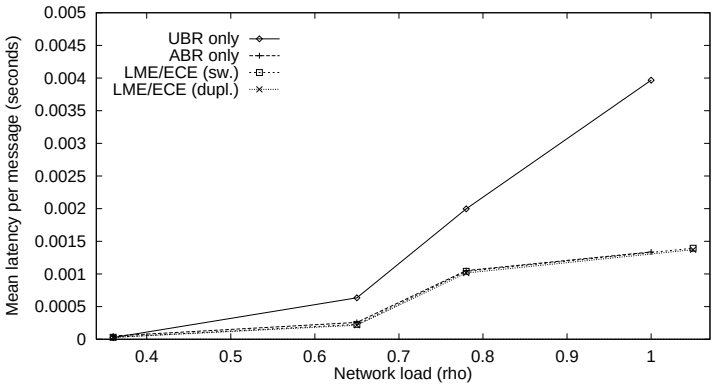


Figure 6.14: *Validation of the ABR service category model. Peak rate: 80 Mb/s.*

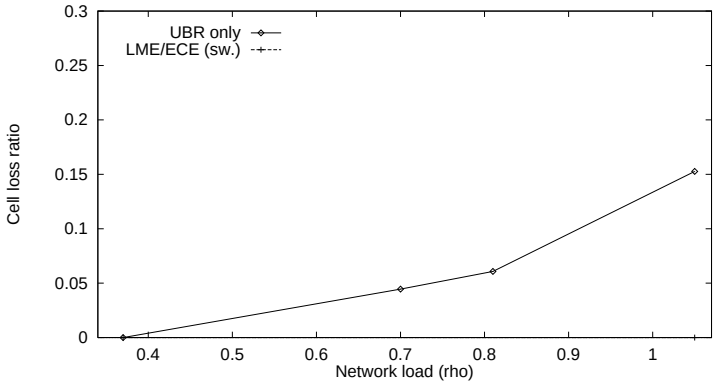


(a) Parallel code: PDE1

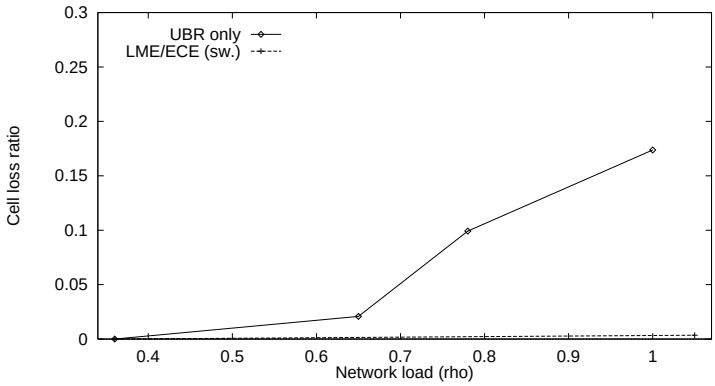


(b) Parallel code: PDE2

Figure 6.15: *Latency measurements.*



(a) Parallel code: PDE1



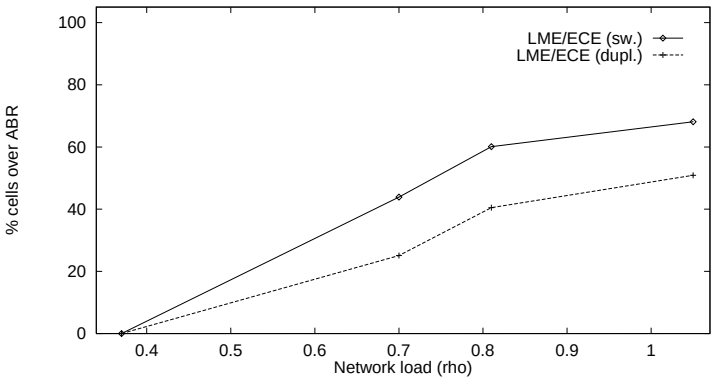
(b) Parallel code: PDE2

Figure 6.16: *Cell loss ratio experienced by applications.*

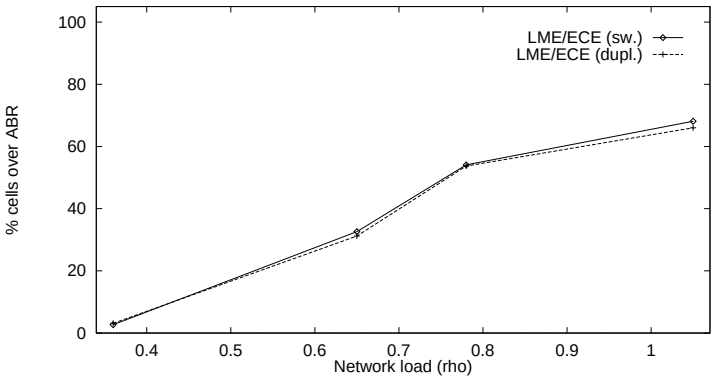
more bursty than traffic from *PDE2*, as stated earlier. Nevertheless, in the next subsection it is observed that, as shown in Figure 6.17, the ABR service is used only by the 30%–70% messages, depending on the application and the network load. Therefore, in addition to equivalent performance, great efficiency in resource exploitation may be achieved.

In order to assess the relationship between the performances achieved by both the original UBR-only mechanism and the switching ECE and the need of retransmissions, we have measured the experienced cell loss ratio with these configurations. We have not measured the cell loss in the duplicating ECE because retransmissions are not always triggered by lost cells, but rather in high-latency mode each cell is automatically replicated. The results in Figure 6.16 confirm that retransmissions are a major cause of latency in ATM-based parallel computing environments, and also that our proposal of ECE succeeds in reducing the amount of required retransmissions, with a lower cost than necessary for achieving the same effect with the sole use of the ABR service category.

Due to the random component of the background traffic, several repetitions of the latency measurements have been performed. When considering a confidence level of 90%, the maximum radius for the confidence interval is 14% of the mean value in the worst case, which indicates a clear difference between UBR-only results and the rest.



(a) Parallel code: PDE1



(b) Parallel code: PDE2

Figure 6.17: *ABR service utilization measurements.*

6.2.3 ABR service utilization

We consider the fraction of the cells generated by a parallel task that used the ABR service as a measure of the utilization of this service. As the ABR service requires more resources from the network (a flow control mechanism, some kind of priority, etc.) than the UBR service (which just takes advantage of the bandwidth not consumed by the other service categories, so no particular resources are allocated for it), the cost of information sent through ABR is also higher.

Figure 6.17 displays the results of this measurement. *PDE1* and *PDE2* exhibit different behavior, as expected for the different characteristics of communications. The following observations can be extracted:

- In *PDE1*, the switching ECE achieves 40% utilization for $\rho = 0.7$ and 70% for $\rho = 1$, and the duplicating ECE achieves 30% utilization for $\rho = 0.7$ and 60% for $\rho = 1$. These results show a highly cost-effective service achievable by both mechanisms. Thus, parallel applications whose communications follow a similar appearance as those of *PDE1* can obtain a performance equivalent to that of the plain ABR service but with a higher efficiency in resource usage. the different results achieved by the switching and the duplicating ECEs should be attributed to the special characteristics of *PDE1* traffic —which involves the generation of bursts of cells— since in the case of *PDE2* this different behavior does not appear.

- In *PDE2*, our proposals achieve a slightly higher utilization of the ABR service —40% for $\rho = 0.65$, and 70% for $\rho = 1$, in both the switching and the duplicating ECEs. In this case, the service remains

Table 6.5: *Bandwidth consumption experienced by* PDE1 *(Kb/s).*

ρ	Service	UBR only	ABR only	LME/ECE (sw.)	LME/ECE (dupl.)
0.7	UBR	266.4	-	154.0	253.9
	ABR	-	275.0	122.0	69.1
	Total	266.4	275.0	276.0	322.0
	VBR	-	-	5.0	7.1
1.05	UBR	266.4	-	88.9	215.4
	ABR	-	275.0	190.2	140.1
	Total	266.4	275.0	279.1	355.5
	VBR	-	-	5.1	5.5

Table 6.6: *Bandwidth consumption experienced by* PDE2 *(Kb/s).*

ρ	Service	UBR only	ABR only	LME/ECE (sw.)	LME/ECE (dupl.)
0.65	UBR	280.0	-	216.0	310.9
	ABR	-	289.0	107.6	105.1
	Total	280.0	289.0	323.6	416.0
	VBR	-	-	7.1	11.2
1.05	UBR	279.0	-	103.6	248.9
	ABR	-	289.0	224.5	222.0
	Total	279.0	289.0	328.1	470.9
	VBR	-	-	6.0	7.9

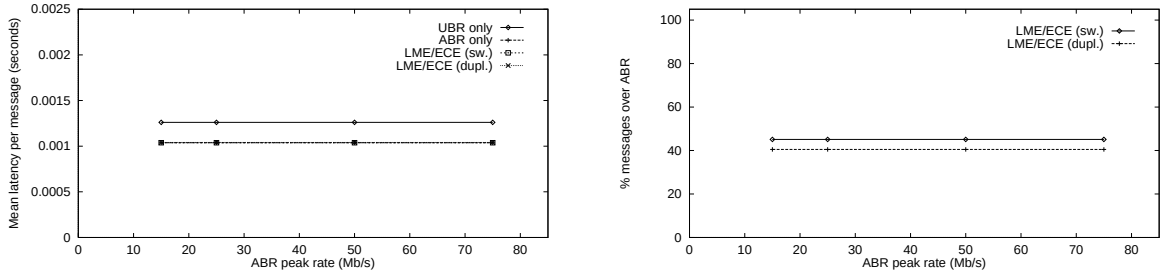
cost-effective —although slightly less than *PDE1*. The utilization of the ABR service is much less dependent on the application, as opposed to *PDE1*. Thus, applications whose communication pattern is similar to that of *PDE2* can equally achieve cost-effective communications. The fact that communications in *PDE2* are more sparse throughout its execution, the differences between the switching and the duplicating ECEs are much less relevant than in *PDE1*.

In order to realize the effective cost of our mechanism, we should take into account the cost of the VBR service conveying the feedback information. As illustrated below, its performance depends on the network load as well, so we can lose some of the advantage in cost-effectiveness, specially in a highly loaded network.

6.2.4 Bandwidth consumption

This measurement serves to solve two pending problems. The first problem to solve is to decide which of both ECEs is best suited for parallel computing. So far, their performance has been shown to be equivalent, although it is intuitive that the bandwidth consumption of the switching ECE will be more important than that of the duplicating ECE. If the consumption of the duplicating ECE is much more important than in the switching ECE, we will adopt the switching ECE for the enhanced PC-AAL. The second problem is to assess the impact of using the VBR service category for conveying feedback information on the cost of our proposed ECEs. A high utilization of VBR would compromise the cost advantages introduced by constraining the use of ABR to the periods where is is really necessary, as the cost of the VBR service is much higher that the ABR service.

Tables 6.5 and 6.6 reflect the bandwidth consumed in both *PDE1* and *PDE2* by the services carrying the actual data, for different two network loads in each case. The first observation from Tables 6.5 and 6.6 is that the fraction of bandwidth spent by the ABR service is closely related to the ABR service utilization displayed in Figure 6.17, specially for the switching ECE. This relationship does apply to the duplicating ECE as well, but is is hidden by the presence of the duplicate cells in high-latency mode.



(a) Task-to-task latency (b) ABR service utilization

Figure 6.18: *Measurements for different ABR peak cell rates (PDE1).*

The results show that, for the switching ECE, the total consumed bandwidth is slightly higher with the switching ECE than with ABR or UBR only, and the difference is lower in *PDE1*. In contrast, the bandwidth consumption experienced by the duplicating ECE is significantly higher than in the switching ECE. As a consequence, the use of the duplicating ECE does not involve significant advantages with respect to the switching ECE, and therefore we conclude that the switching ECE is the most adequate ECE.

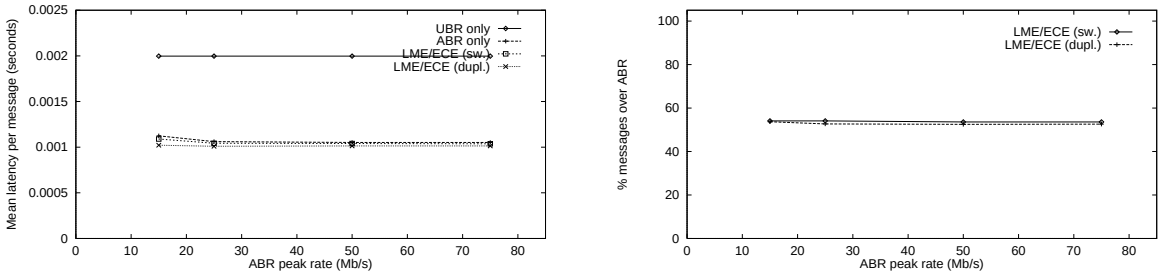
Regarding the bandwidth spent by the VBR service, we recall that the VBR service conveys the STAT frames, which are periodically generated upon receipt of a POLL frame, as well as USTAT and LSTAT frames which are generated asynchronously. Thus, as expected, the spent bandwidth strongly depends on the cell loss ratio, which in turn is related to ρ. In particular, the higher the background load, the lower the consumed bandwidth, due to the increased length of high-latency periods. Note that the significance of the bandwidth consumed by the VBR service is lower than the impact of the ABR service —it is equivalent to 3%–7% of the bandwidth consumption from ABR. Thus, the total cost for the evaluated approach remains advantageous.

In order to achieve a wider generalization of the results, we have to consider different values for some cost-determining parameters of the ABR service category. In particular, all the experiments have been performed, as far as ABR connections are concerned, with a peak cell rate of 50 Mb/s and a zero minimum cell rate. Presumably, the performance (and the cost) can vary if we change these conditions. This is discussed in the following subsections.

6.2.5 Influence of the ABR peak rate

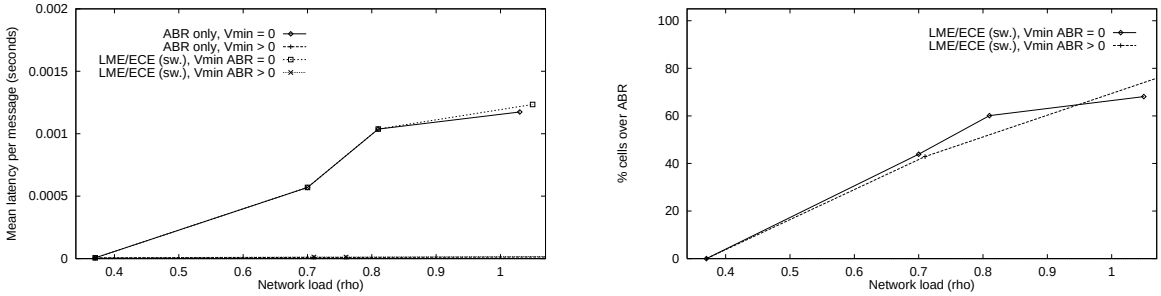
All the measurements discussed so far in this chapter have adopted a fixed peak cell rate of 50 Mb/s for the ABR-based connections. As this option is somewhat arbitrary, we have tested the impact of performance of different values for the ABR peak cell rate. In particular, we fixed the network load to $\rho = 0.8$, and then we have measured the task-to-task latency and ABR service utilization for both *PDE1* and *PDE2*. In each case, we tested the switching ECE, the standard PC-AAL using ABR, as well as the standard PC-AAL using UBR that serves as a reference.

Figures 6.18 and 6.19 show that the influence of the ABR peak cell rate is negligible for both *PDE1* and *PDE2*. This is a consequence of the fact that, in this environment, latency is mostly determined by cell loss and retransmission, issues that are not involved in this measurement. The sensitivity to ABR peak cell rate is more apparent in *PDE2* than in *PDE1*, as a consequence of the different communication patterns. Thus, as peak cell rate in ABR is not a parameter with a significant impact on performance, the conclusions drawn from the preceding measurements, carried out with a fixed ABR peak cell rate, can be generalized to a wide range of ABR peak cell rates.



(a) Task-to-task latency (b) ABR service utilization

Figure 6.19: *Measurements for different ABR peak cell rates (PDE2).*



(a) Task-to-task latency (b) ABR service utilization

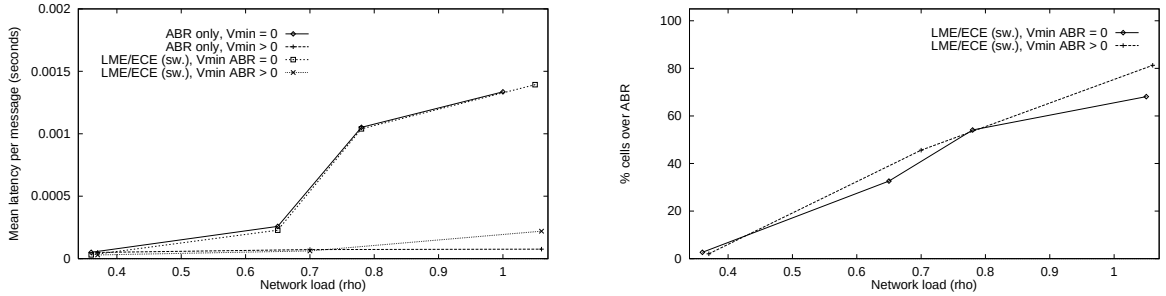
Figure 6.20: *Performance of PDE1 when using ABR with a minimum bit rate of 35 Mb/s.*

6.2.6 ABR with a minimum guaranteed cell rate

Throughout this chapter, we have considered that the ABR service category does not guarantee any bandwidth, that is, ABR has been assumed as a full best-effort service category. Nevertheless, the definition of ABR in the ATM Forum's Traffic Management specification [ATM96c] allows ABR to partially behave as a guaranteed service category. This is accomplished with the possibility of assuring a minimum bit rate. Intuitively, the use of such a feature can lead to achieve a significant improvement in both the PC-AAL using ABR and the enhancements of the PC-AAL that are proposed in the present chapter. The cost involved with this ABR service category with minimum bit rate will be higher than the cost experienced without bit rate guarantees, but the increase in the performance experienced by parallel computing applications may be sufficient to compensate this additional cost. The cost will depend on the minimum cell rate adopted for the particular connection.

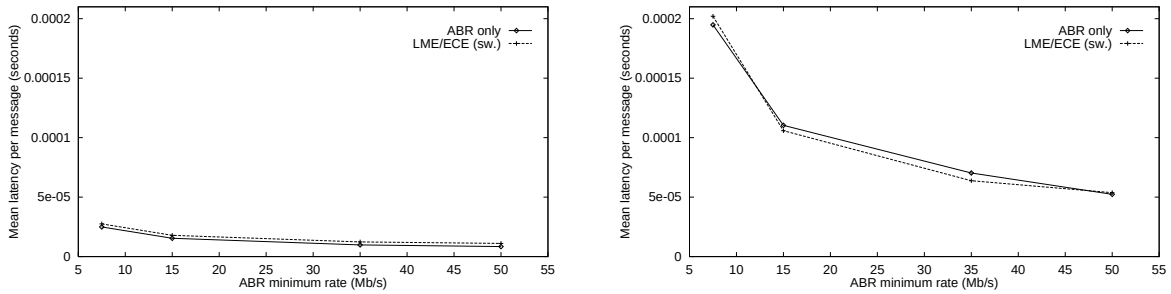
Figures 6.20 and 6.21 show the task-to-task latency and the utilization of the ABR service, as in Figures 6.15 and 6.17, when the ABR service category is guaranteed 35 Mb/s. The results are compared with equivalent mechanisms without guarantee in the ABR service category. The improvement in latency experienced parallel applications is very important, about one order of magnitude. The latency using the switching ECE is close to what is achieved when relying on the ABR service category all the time. With regard to the utilization of the ABR service category, we observe that the adoption of an ABR service category with a minimum bit rate does not result in significant variations of its utilization.

In order to assess the effect of different values for the ABR minimum bit rate, we have measured the task-to-task latency for both *PDE1* and *PDE2*. We have considered values from 7.5 Mb/s to 50 Mb/s —in fact, this is CBR as the peak bit rate is also 50 Mb/s. The results, shown in Figure 6.22, show a slight influence, which is more important in *PDE2* due to its particular communication characteristics. Nonetheless, the magnitude of the variations is very low. Regarding the utilization of the ABR service, no significant variation has been experienced.



(a) Task-to-task latency (b) ABR service utilization

Figure 6.21: *Performance of PDE2 when using ABR with a minimum bit rate of 35 Mb/s.*



(a) PDE1 (b) PDE2

Figure 6.22: *Task-to-task latency experienced under different ABR minimum bit rates.*

6.2.7 Confidence of results

Like in Chapter 5, a confidence interval for the measurements has been determined in order to assess their applicability. As the simulations performed in the work described in the present chapter are very long —even much longer than in Chapter 5—, we have not determined the confidence interval for all the parameters in every experiment, but only those particular cases in which the conclusions rely. We computed the confidence interval with two different strategies. in the first one, we repeated each experiment 10 times, and considered all 10 samples in order to build the confidence interval. However, as the network load is a random variable, not all the samples have been measured under exactly equivalent conditions, thus getting excessive variability in the samples. For this purpose, we considered to take the 5 samples where the network load was most similar.

Table 6.7 shows the confidence intervals for the mean latency when the network load is 0.8. The displayed values correspond to the radius of the confidence interval with respect to the mean value. The values achieved for $\rho = 1.2$ are similar. The difficulties in getting sufficiently good confidence intervals are even greater than in Chapter 5. For $\rho = 0.8$ a certain overlap of confidence intervals occur between the UBR and LME/ECE measurements in *PDE1*, which is not the case in *PDE2*. For $\rho = 0.8$, we can then

Table 6.7: *Confidence intervals for the measured mechanisms (confidence level: 90%).*

Mechanism	Confidence interval (% over mean)			
	PDE1		*PDE2*	
	10 samples	5 samples	10 samples	5 samples
UBR only	11.42%	12.87%	15.77%	13.75%
ABR only	13.05%	18.04%	10.34%	10.39%
LME/ECE (dupl.)	10.80%	14.90%	11.20%	10.04%

state that, with a 90% probability, the switching ECE/LME mechanism performs better than the PC-AAL without introducing ATM service categories. We cannot make such an affirmation for *PDE1* with this network load with a confidence level of 90%, we can only do it with a 80%. In contrast, for $\rho = 1.1$ the statement is possible with a confidence level of 90%.

6.3 Discussion

In this chapter, we have described and evaluated two mechanisms whose aim is to enhance the operation of the SSCOP-based PC-AAL as defined in Chapter 5. In particular, the enhancements are achieved by means of the exploitation of the service categories provided by ATM. Typically, data applications use an ABR service to reduce the occurrence of cell loss, but the use of a UBR service when the network is unloaded can lead to similar performance. Thus, we suggest to use UBR as the basic transfer service but to introduce an ABR service when latency experiences a significant increase. By means of this operation, we achieve low latency in communications and a cost-effective service. For this purpose, we have explored two strategies for introducing ABR when the latency experienced in the ordinary UBR-based connection is sufficiently high: (1) to deactivate the UBR-based connections and switch the traffic to the ABR-based connection, and (2) to use the ABR-based connection as a redundant channel through which the data issued through the UBR-based connection are replicated.

To evaluate the performance of our proposals, as well as to decide on the most adequate of them for supporting communications in parallel computing applications, we have undertaken a number of simulation-based experiments. In particular, we have measured the end-to-end latency and cell loss ratio experienced by communications, the utilization of the ABR service category, and the bandwidth consumption, particularly of the VBR service category. In view of the results yielded by these measurements, we observe that (1) the latency achieved by our mechanism is equivalent to the latency experienced when conveying all communication through ABR-based connections; (2) As in the worst case only 70% of cells use the ABR service category, the cost of communications with our mechanism is much lower than the cost inherent to the full use of ABR-based connections; (3) the LME succeeds in determining the high-latency periods, since our mechanism has been able to avoid most of cell loss; and (4) the bandwidth consumption and the requirements for the VBR service category are so low that the cost of communications is not significantly affected. As a summary, our mechanism allows for parallel computing applications that execute for a significantly long period to achieve cost-effective performance.

Regarding which strategy is best suited to the characteristics of parallel computing applications, the results have shown that their performance is similar as far as latency and ABR service utilization is concerned. However, as the bandwidth consumption experienced by the duplicating approach is significantly higher than the consumption measured in the switching approach, we conclude that the switching ECE is more adequate that the duplicating ECE. It is possible, nevertheless, that in environments where the computing nodes are spread in a very large network, the duplicating ECE could be more advantageous, since its main feature is the fact that retransmissions are “advanced” when in high-latency mode, so the contribution to latency from the notification of cell loss is reduced.

7
Conclusions and future work

In this final chapter, we summarize the conclusions drawn as a result of the diverse proposals made in the thesis. After that, we mention further works that may be undertaken in order to extend the scope and the functionality of the architectures and mechanisms presented so far.

7.1 Summary of contributions

The main topic addressed in the thesis is the proposal of a network architecture oriented to the support of parallel computing over ATM networks. The main originality in this work is the explicit assumption that the ATM network is shared with other networking applications and, consequently, the network can experience load variations and congestion situations. The particular contributions include: (1) the assessment of the promising possibilities reachable with the use of ATM for parallel computing; (2) the proposal of a network architecture for supporting parallel computing applications; and (3) the design and evaluation of concrete mechanisms to be implemented within this architecture.

7.1.1 Potential performance of ATM-based platforms

The potential performance achievable with ATM networks for supporting parallel computing have been studied in order to assess whether further research on such platforms can lead to fruitful conclusions. The performance of current ATM-based environments is far from multiprocessors, due to the presence of important overheads in network-based environments. In a simulation study, we have determined which would be the performance in ATM-based environments if all bottlenecks were removed. The results showed that, in many cases, the performance could reach multiprocessors, so we concluded that it was possible for ATM-based environments to deliver competitive performance provided that the impact of bottlenecks was minimized.

As a complementary study, the relative importance of the various bottlenecks present in current ATM-based environments has been determined. In particular, we considered the network size –in terms of number of nodes—, the network load —consequence of sharing the network with other applications—, and the delays in the endpoint hosts —caused by protocol processing and the host-network interface. Among these bottlenecks, the most influential of them has been resulted to be the delays experienced within the endpoint hosts. The network load has shown a significant influence as well. Regarding the

network size, in moderate sizes the importance is fairly limited. A consequence of this results is that any mechanisms attempting to approach the performance of ATM-based environments to the potential will have to pay special attention on minimizing the processing within the endpoint host —by using a reduced-processing but robust protocol, by means of special, high-performance host-interfaces, etc. These mechanisms will have to take the network load into account, since the processing involved in loss recovery as a consequence of high load situations has a significant impact on performance. Another consequence shows that the low influence of network size allows for the use of end-to-end mechanisms without a significant performance degradation. All these issues have been considered in the proposal of a network architecture for supporting parallel computing over ATM networks.

7.1.2 The network architecture

The promising possibilities of ATM for supporting communications in parallel computing applications have motivated the proposal of many strategies for introducing ATM in such environments. Among the strategies, many of them simply consider the adaption of existing protocols for usual networking applications, or rather ATM-oriented mechanisms which nevertheless are not application-specific but shared by all applications running on top of the ATM network. This is the case of the mechanisms relying on the TCP/IP protocol stack over ATM, as well as those proposals of using an API that is general-purpose. With these strategies, it is not possible to take advantage of the particular characteristics found in parallel computing applications. Other strategies have been specifically conceived for supporting parallel computing applications. They can achieve better results than general-purpose applications, as a consequence in many cases of the fact that no other traffic apart from parallel computing is occupying the ATM network. This assumption allows them to implement elementary mechanisms that do not consider the high load or congestion situations occurring in shared ATM networks. In addition, some of these mechanisms require that applications include additional programming. In active messages, it is hard to incorporate the new interface because much low-level burden is left to the programmer. Likewise, multithreading-based approaches require that the programmer bothers on mutual exclusions and thread management, which are issues that the most commonly available parallel computing applications do not take into account.

The ideal mechanisms should be specific for parallel computing applications, but simultaneously other applications should be able to make use of the same ATM network and, in addition, it should be easy for existing parallel computing applications to be moved to the new environment. For this purpose, we have proposed a model for the network architecture that allows for each type of applications to rely on specific mechanisms. In this model, support to parallel computing is one of the application types. Thus, the specific mechanisms will have to rely on standard features of ATM and, therefore, the adaption between parallel computing applications and the ATM network —the so-called *convergence level*— is performed outside the ATM network. The mechanisms in this convergence level have to guarantee the full delivery of all data submitted to the network, by optimizing the performance. The principal measure of performance in parallel computing applications is latency, rather than throughput.

The key point of the mechanisms implementing the convergence level is that they can take advantage of the special characteristics of parallel computing applications, which include:

- The short PDUs that parallel computing applications interpret —the elementary data types integer, float, etc.— over which more complex structures such as vectors and matrices are built.

- The moderate average bit rate due to a fairly low frequency of communications, but with large bandwidth requirements when communications effectively take place.

- The long execution time —hours or days— that reality parallel computing applications usually experience.
- The fact that communications in parallel tasks consists of a sequence of requests and responses, because the importance of throughput as a measure of performance becomes relative and, consequently, we can sacrifice some throughput if this allows for a gain in latency.

In order to adapt the needs of parallel computing applications to the characteristics of the shared ATM network, the mechanisms in the convergence level benefit from the small size of PDUs —called PC-PDUs after Parallel Computing PDUs— in order to obtain more efficient schemes for loss recovery than those found in existing general-purpose protocols. In addition, ATM networks include several service categories whose features may be exploited by the convergence level in order to enhance performance while keeping the cost within reasonable limits.

7.1.3 The mechanisms for implementing the network architecture

The most important contributions of the present thesis are the proposals of concrete mechanisms to be implemented in the convergence level. Among the functions that can be included in the convergence level, we have chosen to build mechanisms for assuring the reliable delivery of data. For this purpose, we have taken advantage of characteristics of both parallel computing applications and ATM network. Our first mechanisms benefits from the short length of PC-PDUs in order to improve the operation of a retransmission mechanism. The second mechanism enhances the previous one by making use of the features provided by different ATM service categories.

The fact that a single ATM cell can contain one or several PC-PDUs has led to consider the possibility of modifying an existing protocol which is already oriented to operate over ATM networks. In particular, if the frame unit is the same as the minimum losable data unit —in this case, the ATM cell—, it is possible that only lost cells be retransmitted. This is not the behavior in traditional frame-based protocols, where the loss of one cell triggers the retransmission of all the cells belonging to the same frame as the lost cell. By following this guideline, we have modified a frame-based protocol, the SSCOP (Service Specific Connection Oriented Protocol), by converting it into a cell-oriented protocol. We chose SSCOP because it was a protocol designed for running over ATM networks and its initial application scope was the support to signaling, which requires reliable delivery of data as well. The modification of SSCOP has led to build a new AAL specific for parallel computing —the PC-AAL— to replace AAL5. The performance evaluation of this resulting PC-AAL yielded that:

- The reliance on short PC-PDUs allows applications to use the received data immediately, without requiring the complete reception of a large frame. As a result, latency improves with respect to standard SSCOP.
- The reduction of retransmitted information in the network provides communications with a high robustness with respect to the network load, what indicates satisfactory performance of the PC-AAL under the conditions experienced by parallel computing applications.

We compared two encapsulation strategies: (1) encapsulating one PC-PDU per cell, and (2) encapsulating several PC-PDUs per cell. All the tests have demonstrated the best performance of the second strategy, since the lower processing required in the first alternative does not compensate for the effects of the much higher number of generated cells.

The PC-AAL as discussed above does not rely on any particular ATM service category, as the only feature of ATM that has been exploited is the use of small packets as the transmission unit. Service

categories defined for ATM networks can provide additional features, for instance guaranteed bandwidth and a network-level flow control mechanism, with diverse costs for the user. Given the characteristics of the traffic exchanged in parallel computing applications, the most adequate service categories for supporting parallel computing are those based on a best-effort scheme —i.e. UBR and ABR. The measures of the PC-AAL assumed a UBR service category. The introduction of ABR allows to reduce the cell loss ratio and, consequently, the performance degradation caused by retransmissions. However, we observe that the cost of an ABR service category is higher than UBR, while ABR is useful only in the periods where the network load is high —the remainder periods, the performance of UBR is sufficient. This is the reason why we have considered a mechanism that is capable of switching from UBR to ABR and vice versa according to the monitored latency in the network. The complexity involved in this approach requires that parallel applications have a long execution time, in order to take advantage of the variability in the network load.

The mechanism proposed as an enhancement for the PC-AAL includes two components: (1) the data transferring mechanism itself, known as ECE (Effective Communication Engine), and (2) a mechanism for monitoring latency in the network, the LME (Latency Monitoring Engine). The LME indicates when the latency exceeds a threshold in order for the ECE to introduce ABR. For the ECE we have considered two alternatives when ABR has to be introduced: (1) deactivating the UBR-based connection (switching ECE), and (2) duplicating the data through both the UBR- and the ABR-based connections (duplicating ECE). In the following, the most outstanding results of the evaluation of this approach are enumerated:

- The ECE/LME mechanism achieves cost effective performance. The mean latency is similar to what is achieved when using the PC-AAL with ABR all the time. The cost of LME/ECE is lower as the more expensive ABR service category is used by only between 30% and 70% of messages, depending on the load in the network and the parallel application.

- Regarding both schemes for the ECE, they achieve similar performance while the duplicating ECE is more bandwidth-consuming. Therefore, we considered the switching ECE as the best scheme for supporting parallel computing applications.

These conclusions are valid for any peak rate allocated to the ABR service category. In addition, the allocation of a minimum bit rate for the ABR service category improves the performance for both the LME/ECE and the PC-AAL with ABR. As the utilization of the ABR service category remains similar, we conclude that the ECE/LME mechanism is able to bring equivalent performance as that achieved when using ABR all the time, but with a lower cost. Thus, we succeed in providing cost-effective performance to our network architecture for supporting parallel computing applications.

7.2 Future work

The network architecture and the mechanisms discussed along this thesis are an important component to allow for the support of parallel computing applications over ATM-based networks, but they are not sufficient to achieve a real operating environment. In particular, all the necessary procedures to configure the environments have yet to be investigated. The virtual network schemes presented in Subsection 4.2.1 are an advance of a deeper study which should include the specification of the functions performed in the so-called *Parallel Computing Servers* as well as the definition of the signaling procedures that will be in charge of building the topologies for executing parallel applications, downloading the code and the data to the appropriate computers, collecting the results, and deallocating the reserved resources once the execution is finished.

All the mechanisms in this thesis are assumed to operate on top of an all-ATM network. Although this is the expected scenario for future communications, during a long time the communications panorama will be dominated by the presence of separate ATM networks whose interconnection is performed via non-ATM technologies such as FDDI, N-ISDN, Frame Relay or Ethernet. In the virtual networks for supporting parallel computing applications will possibly have to include some nodes in order to allow for the interworking of all these technologies through bridging and routing, where mechanisms like LAN Emulation could be used. This research could then be extended to permit the execution of parallel tasks on machined connected to a networking technology different from ATM.

The mechanism discussed in Chapter 6 relies on real-time latency monitoring through the LME module. Although one possible implementation has been suggested, a better operation may be possible by taking advantage of information on network status. This information could either be obtained from the feedback information provided by ABR-based connections, or be explicitly supplied by the network management plane. Working on alternative implementations for the LME will be possible when the standards become more mature on those aspects.

Another issue that is necessary for enabling the operation of ATM-based environments is the provision of a suitable interface between parallel computing applications and our network architecture. Traditional message-passing libraries like PVM or MPI have been designed for operation on traditional protocol stacks, where the trend is to build packets whose size is as long as possible. When using single PC-PDUs as the transfer units instead of the large packets generated by message-passing libraries, a novel message-passing library designed for supporting PC-PDU-based data transfers could pass by the step of building and decomposing the message and, consequently, it could send data as soon as generated and process data as soon as received.

Finally, a longer-term work can be the consideration of the Distributed Shared Memory (DSM) paradigm for parallel computing applications. This would allow for ATM-based environments to execute all the base of parallel computing applications using a shared-memory programming model. In addition, this could provide distributed operating systems with a more efficient communication subsystem. The problems to be solved are very different as those discussed in the message-passing programming model. For instance, it is not clear that PC-PDUs as considered in this thesis are the fundamental data transfer units, and the communication pattern is expected to be radically different. In addition, issues such as mutual exclusion, cache coherency, and synchronization that are not relevant to message-passing systems like the discussed network architecture will surely be crucial in DSM systems.

A
Benchmarks

This appendix summarizes the characteristics of the different real algorithms that have been used in the simulations and measurements in the present work. In addition to the description, some statistics concerning the messages generated by each application are discussed, since they have shown to have significant influence on the results.

A.1 Description

For the measurements in this thesis, we had to select real algorithms that could be executed on a wide range of parallel computing environments. This requirement, together with the fact that among the diverse environments we have distributed-memory architectures —in particular, the SP2 multiprocessor, the Ethernet-based environments, as well as the ATM-based platform, all of them discussed in Chapter 2— has led us to consider a message-passing environment as a common environment in order to enable direct performance comparison. For this purpose, we adopted PVM (Parallel Virtual Machine) [G+94], as it is available for all the concerned environments and we have an easy access to documentation on this library. thus, we have selected those algorithms from the GENESIS release 3.0 (July 1994) [AGH+91] and NAS 1.0 [WÅS95] benchmark suites whose adaption to the studied environments were less difficult. From the GENESIS suite we have used the following kernels:

- *PDE1*. It solves the Poisson Equation on a three-dimensional grid by using red-black successive over-relaxation (SOR) with Chebyshev acceleration. The Partial Differential Equations (PDE) are represented by a large set of (non)linear equations, each of which couples values at neighboring grid points with each other. The number of floating point operations per gridpoint is quite small, so the ratio of computation to communication is rather low.
- *PDE2*. This benchmark solves a two-dimensional Poisson equation using a multigrid method. A mixture of fine and coarse grids are used to accelerate the solution process. The parallelization is performed by grid splitting. A part of the computational grid is assigned to each processor. After each computational step, values at the boundary of the subgrids are exchanged with nearest neighbors.
- *SOLVER*. It is part of an ongoing software development exercise. The full application generates quark propagators from a background gauge configuration and a fermionic source. The benchmark actually performs a cut-down version of this operation. Instead of performing all the required

Table A.1: *Distribution of message lengths (in bytes).*

Parallel kernels	Number of messages	Length of messages (mean)	Length of messages (median)
PDE1	5608	8180.360	8192
PDE2	8068	116.827	40
SOLVER	4792	7107.595	6144
EP	N/A	N/A	N/A
CG	7116	1963.389	8
FT	189	233200.75	262144
IS	599	71364.320	130016
MG	848	30226.525	2592

iterations, a fixed number of them (50) is actually carried out in order to generate accurate timing information.

The NAS Parallel benchmarks include a number of kernels and simulated applications whose operation imitates the computation and data movement characteristics of large scale computational fluid dynamics applications. Five of these kernels have been ported to PVM, and have been used in the present work:

- *EP*. A large number of iterations is executed in which a pair of random numbers are generated and tested for whether Gaussian random deviates can be made from them according to a specific scheme. This kernel does not incur any data or functional dependencies and requires no communication between processors. Thus, a reference point for peak performance can be established.

- *CG*. It uses the power and conjugate gradient methods to approximate the smallest eigenvalue of a symmetric, positive definite, sparse matrix with a random pattern of nonzeros. The communication patterns in this kernel are long-distance and unstructured.

- *FT*. It executes FFTs (Fast Fourier Transforms) on a complex array to solve a three-dimensional partial differential equation. Communication patterns are structured and long-distance in nature.

- *MG*. A few iterations of the V-cycle multigrid algorithm are executed to obtain an approximate solution to the discrete Poisson problem on a three-dimensional grid with periodic boundary conditions. This application rigorously exercises both short- and long-distance communications, and the communication patterns are highly structured.

- *IS*. 10 ranks of a large number of integer keys are performed. the keys are uniformly distributed in the local memories of the parallel machine. Communication in this environment is frequent and relatively low-volume, and the pattern of communications is a fully connected graph.

A.2 Characteristics of communications

Table A.1 illustrates the characteristics of the eight algorithms as far as communications are concerned. For this purpose, the distributions of the length of messages are characterized with two parameters: the mean value and the median. Similar values for both parameters indicate that the mean value is truly representative of the distribution. A high mean value and a low median can be associated to a distribution with many short messages and a few very long messages. A low mean value with a high median can be obtained with many extremely short messages and a few not very long messages.

Table A.2: *Message temporal density.*

Parallel kernels	Density (msg./sec.)	Av. throughput (Mb/sec.)
PDE1	228.9	14.96
PDE2	246.0	0.23
SOLVER	363.0	20.64
EP	N/A	N/A
CG	1218.5	19.14
FT	11.5	21.50
IS	166.4	95.00
MG	50.5	12.21

PDE1, *FT* and *SOLVER* are the most homogeneous message distributions, since most of messages exhibit around 8100, 7000 and 230,000 bytes, respectively. The distribution is less balanced in *PDE2*, *CG*, and *MG*, where short messages predominate (40, 8 and 2500 bytes, respectively) although some significantly longer messages are also generated. Finally, in *IS* the average is shorter than the median, so many short messages are expected (shorter than 71,000), while those messages longer then 130,000 bytes will not actually fall much beyond this magnitude. Note than some kernels, particularly *FT*, *IS*, and *MG*, generate very large messages. Thus, the burstiness experienced by the networks is expected to be very high in those cases. The actual impact of burstiness depends on the temporal distribution of the instants when messages are issued to the network. This magnitude depends on the networking technology. In the case of an ideal ATM network, evaluated in Section 2.1, the density of communications is shown in Table A.2. For similar throughput values, a lower message density indicates a more aggressive communications pattern.

B
Emulation of an ATM network

This appendix describes the simulated environment used for the study of the potential capabilities of ATM discussed in Chapter 2. This environment is not a simulator in the traditional sense of the term, but an emulator of an ATM-based environment. Thus, it is possible to establish direct comparisons between real environments and ATM-based environments. In particular, we discuss here the architecture of the simulated environment, as well as the validation of the model of the ATM network.

B.1 Objectives and description

The emulated ATM network environment is addressed to those studies where performance is compared to that of real environments, since the simulation study and the real measurements can be based upon the same workload. Using such an emulated environment is also appropriate for those simulation studies requiring so many data that trace-based simulation becomes infeasible. Thus, the emulation of an ATM network includes an ATM simulator which allows for user-configurable topologies and traffic conditions, and an interface with real applications so that they can use the simulated ATM network as if it were a real network. Figure B.1 depicts the architecture of the emulated ATM network.

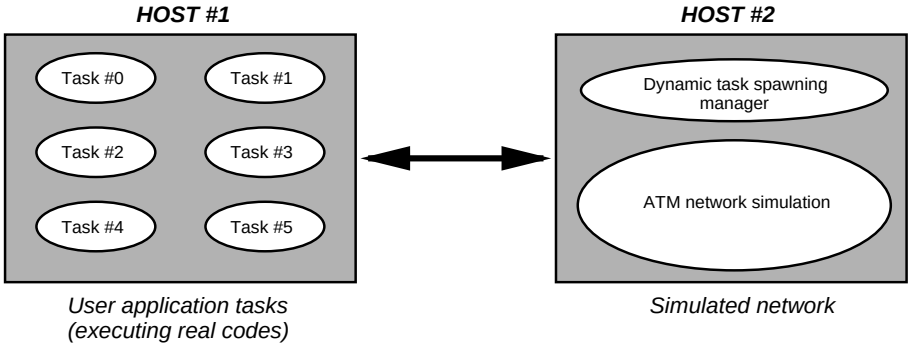


Figure B.1: *ATM network emulation architecture.*

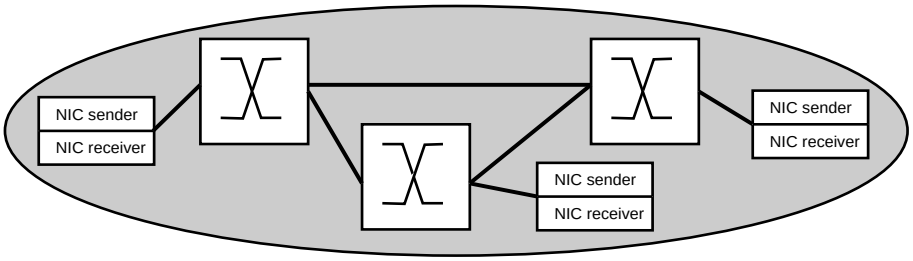


Figure B.2: *ATM network simulator structure.*

B.1.1 ATM network simulation

The ATM network is implemented as an event-driven simulator, and is contained in a process in a separate host from applications. As shown in Figure B.2, the simulated network elements include ATM switches and Network Interfaces. The ATM network emulator allows to freely configure the network topology. The network elements are modeled as follows:

- The switches use output-queuing policy, but the buffer space is shared by all output ports. Routing is implemented by means of a static, user-configurable table. At present, only virtual-channel connections are supported. The scheduling policy is strictly FIFO. Only one service category, as defined in the ATM Forum's Traffic Management specification [ATM96c], is currently supported, namely the UBR (Unspecified Bit Rate) service category. Buffers are assumed as infinite in the current version, since there is no support for protocol testing. The delay introduced by the switch is configurable by the user.

- The sending part of the network interface is simply modeled with a FIFO queue that holds the cells resulting from the segmentation process. Segmentation is done according to AAL5. Consistently with the switch models, only the UBR service category is supported. The delay spent in the network interface is user-configurable.

- The receiving part of the network interface is significantly more complex than the sending part. In addition to the reassembly buffers, message storage has also been implemented in order to simplify the implementation of the interface library. This storage facility supports reassembly of messages longer than the AAL5 payload. The final messages are classified according to their origin and destination. The communication with the interface library takes place by means of a request list. When the library requests a message, the network interface delivers it to the library in case it has been already stored; otherwise, the request is queued. When a message is received and fully reassembled, it is immediately delivered to the library if some request of this message is queued; otherwise, the message is stored in the network interface.

The simulated network includes the emulation of traffic from other applications sharing the network (cross-traffic), which is modeled by ON-OFF sources. ON-OFF models, as guessed from this term, comprise two states: a high-load state (ON) and a low-load state (OFF). Each state is characterized with two parameters: the traffic intensity —in fact, the time between cell arrivals— and the duration. The values for both parameters are exponentially distributed around a specified mean.

If an ON-OFF model is implemented for each output in the switches, the emulation of an ATM network can become very inefficient. For this reason, we have adopted an alternative approach, in which only one ON-OFF model is implemented on each switch, regardless of the number of outputs that is present in the

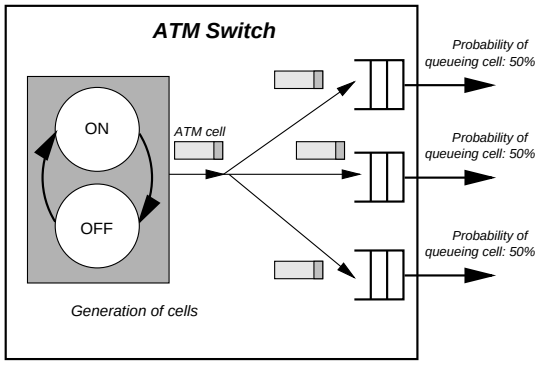


Figure B.3: *Scheme for generating background traffic.*

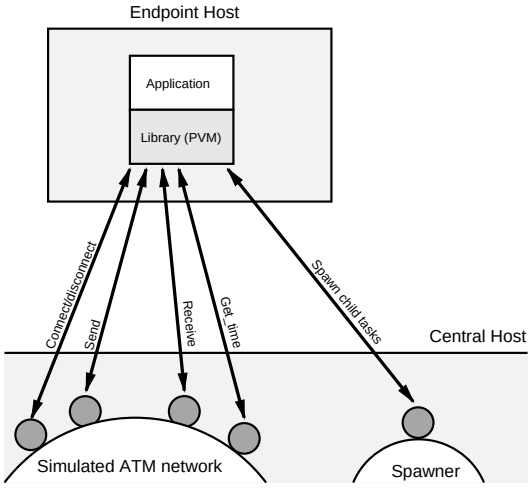


Figure B.4: *Implementation of the interface between the simulated network and the real applications.*

switches. when a cell generated by one ON-OFF model, it is queued into several of the outputs in the corresponding switch. Each output has a probability of 0.5 to hold a copy of the generated cell. With this procedure, the correlation that would appear if the cell were queued into all outputs in the switch is avoided. Figure B.3 illustrates this procedure.

B.1.2 Interface with real applications

Figure B.4 details the structure of the interface between the real applications and the simulated ATM network. In the host executing the real applications, there is a library emulating the primitives of a message-passing library which are more directly related to communications. In particular, we have implemented a few primitives of the PVM message-passing library, particularly those involving the most simple types of communication and task spawning —the real PVM-based applications we have available only use a very basic subset of PVM primitives. The emulated primitives are listed in Table B.1. In addition to standard PVM primitives, the library includes two primitives for obtaining the current simulated time, which are listed in Table B.1 as well, namely `timer` and `msecond`, which are used by the programs in the benchmark suites described in Appendix A.

Figure B.5 illustrates the operation of the PVM primitives by using the interface depicted in Figure B.4. When a task enrolls in PVM –which is done in the first call of a PVM primitive, the application passes the

Table B.1: *PVM Primitives implemented in the application/network interface.*

Communications	Packing data	Unpacking data	Control	Time monitoring
pvm_initsend	pvm_pkbyte	pvm_upkbyte	pvm_mytid	timer
pvm_send	pvm_pkshort	pvm_upkshort	pvm_spawn	msecond
pvm_recv	pvm_pkint	pvm_upkint	pvm_mytid	
pvm_mcast	pvm_pkfloat	pvm_upkfloat	pvm_exit	
	pvm_pkcplx	pvm_upkcplx	pvm_perror	
	pvm_pkdouble	pvm_upkdouble	pvm_bufinfo	
	pvm_pkdcplx	pvm_upkdcplx		
	pvm_pkstr	pvm_upkstr		

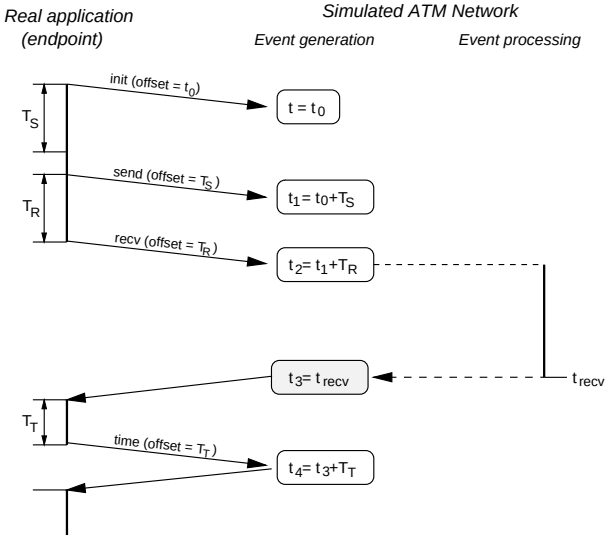


Figure B.5: *Example of the real application-emulated ATM network interface operation.*

initial time to the ATM emulator through the *connect/disconnect* channel. Later on, when PVM attempts to send a message, the whole packet is passed through the *send* channel to the ATM emulator, where it is appropriately segmented into ATM cells. Together with the packet, the amount of time elapsed since the last interaction with the ATM emulator is passed as well, without including the time used for processing the send primitive itself. This amount of time is used by the emulator for updating the internal emulator timer, and convenient events are generated. The operation is analogous when soliciting to receive data, except that the *receive* channel is used. If the requested data are already available, they are passed to the application. If not, as is the case illustrated in the figure, the application blocks until the receipt of the expected data. In the latter case, the ATM emulator processes the pending events, until there is a delivery of data to some application and, if one application has requested the received data, they are passed to the application. In either case, prior to passing the data to the application, the internal timer is updated according to the time when data were received. All the channels have been implemented by means of Internet sockets.

The time primitives interact with the emulator as well. Through the *get_time* channel, the application requests the current time recorded by the emulator. For this purpose, the application passes the time elapsed since the last interaction with the emulator, as before, and then the emulator immediately returns the updated time. The time primitives allow to obtain the same measures as the benchmarks over real environments, which is interesting for comparative studies like that in Chapter 2.

One of the features of PVM is the possibility of dynamically spawning tasks in parallel environments.

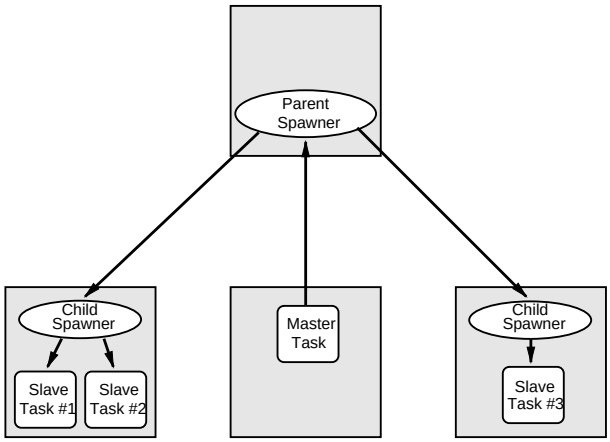


Figure B.6: *Dynamic spawning of tasks.*

The simulated environment includes a mechanism that allows the simulation of the most common cases, which is illustrated in Figure B.6. One of the hosts executes the "Parent Spawner" process, which centralizes the management of spawn requests and task identifiers. The host executing the Parent Spawner need not execute any of the real application's; we have associated it to the same host executing the ATM network emulator. Each host executing one or several tasks of the real application execute a "Child Spawner" that is in charge of starting the tasks when instructed to by the Parent Spawner. The connections between hosts are implemented with Internet sockets, like the interface between real application tasks and the emulated ATM network.

When a task encounters a `pvm_spawn` primitive, it connects to the Parent Spawner and passes the parameters, namely the hosts and the tasks' filenames. The Parent Spawner then assigns task identifiers for each new task and passes them back to the soliciting task. Meanwhile, the Parent Spawner connects to the Child Spawners in the target hosts, and passes the relevant parameters, including the respective task identifier. On receipt of this information, the Child Spawner forks into a new process and prepares it so that it gets the correct parameters when the task enrolls in PVM. This mechanisms is best suited for applications where a master initializes all slaves when starting computation.

B.2 Validation and verification

In this section, we show the results from some tests carried out in order to assess the validity of the results obtained form the ATM emulator. Some tests are addressed to determine the correctness of the emulator implementation, while other tests determine if the measurements from the emulator are realistic.

B.2.1 Verification of message sequence & data integrity

The idea behind these trials is to execute the same application over the ATM emulator and over a real PVM-based environment, so that we can compare the output generated in both cases. This output has to be coincident in both environments; if not, we would know that the ATM emulator includes some error to correct. For example, we have run one of the example programs provided with the PVM distribution, namely the *master-slave* pair in the ATM emulator and an IBM SP2 multicomputer. Their respective output is displayed in figure B.7.

In both tested cases, the output is equivalent —apart from the task identifiers, whose assignation

```
 How many slave programs (1-32)?
4
 tid           0           1
 tid           1           2
 tid           2           3
 tid           3           4
 I got   300.000000000000    from      0
 I got   100.000000000000    from      1
 I got   300.000000000000    from      2
 I got   500.000000000000    from      3
```

(a) Output generated by the ATM emulator

```
 How many slave programs (1-32)?
4
 tid 0 268566529
 tid 1 268632066
 tid 2 268697603
 tid 3 268763140
 I got 100.000000000000000   from 1
 I got 300.000000000000000   from 2
 I got 500.000000000000000   from 3
 I got 300.000000000000000   from 0
PVMD:
        New epoch 1.
```

(b) Output generated by the IBM SP2

Figure B.7: *Comparison of the output yielded by the* master-slave *programs.*

Table B.2: *Round-trip components for ATM, according to [TL93] (in μseconds).*

Component	Packet size: 53 bytes	Packet size: 1537 bytes
Time on the wire	6	176
Controller latency	16	161
Control/data transfer	17	458
Vectoring the interrupt	25	25
Interrupt service	9	20
Sum of components	73	840
Measured round-trip time	73	746

procedure is different, what means that the messages have been transferred in the correct sequence, and also that no loss, duplication or disordering have appeared.

B.2.2 Validation of measurements

The validation has been done by referring to the results displayed in [TL93]. Table I in that paper shows the results of the measurement of hardware-level round-trip packet exchange times. For each networking technology —Ethernet over DEC, Ethernet over Sparc, FDDI, and ATM— two packet sizes are considered, 60 and 1514 bytes. In the ATM case, they considered 53 bytes and 1537 bytes, in order to consider integral ATM cells (1 cell and 29 cells, respectively). The bandwidth of their ATM network is 140 Mb/s. Table B.2 shows the results. *Time on the wire* is the transmission time, where propagation delays are neglected. *Controller latency* is the sum of the delays required for data to cross the controllers in the sending and the receiving endpoints. *Control/Data transfer* captures the time required by the host to transfer the data from/to the controller to/from the host memory. *"Vectoring the interrupt"* includes the time required by the receiver to process incoming interruptions. Finally, *Interrupt service* covers controller-specific operations related to interrupt processing.

For validating the ATM network emulator, we have built a PVM-based 'echo' program that performs N iterations. A message is sent on each iteration and the time elapsed until its return is computed. The final measure is the average of the N iterations. For this purpose, we set the network capacity to 140 Mb/s. As the controller latency and the control/data transfer include the total delay experienced in one round-trip, and in a round-trip a controller is crossed 4 times —a send and a receive in each end—, we consider the fourth part of the values in Table B.2 in the validation study. As far as vectoring interrupt and interrupt service are concerned, we consider half the value for vectoring interrupt in Table B.2 for the validation

Table B.3: *Values considered for the validation study, in μseconds.*

Component	Value
Controller latency	4
Control/data transfer	4.25
Sum of per-cell components	8.25
Vectoring the interrupt	12.5
Sum of per-packet components	12.5

Table B.4: *Results of validation study, in μseconds.*

Measure	Value (53 bytes)		Value (1537 bytes)	
Original round-trip time	73		746	
RTT in the emulator				
(average)	146		671	
(most frequent value)	72	(92% of measures)	652	(96% of measures)

study since interrupt processing impacts specially the receiving end only. We do not consider interrupt service because it is difficult to characterize whether it is a per-cell or a per-packet parameter, according to the value in Table B.2. Thus, the values for the parameters will remain as shown in Table B.3.

Table B.4 displays the results of the validation study. We include the measured value for the round-trip time according to [TL93], together with the average and the most frequent value experienced in the measurements using the ATM network emulator. It is observed that the values measured in the emulator lie in the same order of magnitude as as the original values, although for large packet sizes the emulator tends to be optimistic. The latter in not a problem as the goal of the study where the emulator is used (see Chapter 2) is not to obtain the exact measurement of an hypothetical system but to determine its potential performance. The high discrepancy between the average and the most frequent values is due to a small number of outliers that appeared in a few tests. These outliers do not follow a fixed pattern, and should be attributed to variations in the load in either the measured workstation or the peer workstation.

Bibliography

[A+95] T. Agerwala et al. SP2 System Architecture. *IBM Systems Journal*, 34(2):152–184, 1995.

[AA93] G. J. Armitage and K. M. Adams. Packet Reassembly During Cell Loss. *IEEE Network*, 7(9):26–34, September 1993.

[ACP+95] T. E. Anderson, D. E. Culler, D. A. Patterson, et al. A Case for NOW (Networks of Workstations). *IEEE Micro*, 15(1):54–64, February 1995.

[AGH+91] C. A. Addison, V. S. Getov, A. J. G. Hey, R. W. Hockney, and I. C. Wolton. The GENESIS Distributed-Memory Benchmarks. *Advances in Parallel Computing*, 8 (Computer Benchmarks):257–271, 1991.

[All95] A. Alles. ATM Internetworking. Technical report, Cisco Systems, Inc., May 1995.

[Amd67] G. M. Amdahl. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In *Proceedings of AFIPS Spring Joint Computer Conference 30*, pages 483–485, 1967.

[ATM95] ATM Forum Technical Committee. *LAN Emulation Over ATM, Version 1.0*, January 1995.

[ATM96a] ATM Forum Technical Committee. *ATM User-Network Interface (UNI) Signalling Specification, Version 4.0*. Document ATM_Forum/95-1434R11, February 1996.

[ATM96b] ATM Forum Technical Committee. *Native ATM Services: Semantic Description (Version 1.0)*, January 1996.

[ATM96c] ATM Forum Technical Committee. *Traffic Management Specification, Version 4.0*. Document ATM_Forum/95-0013R10, February 1996.

[ATTD94] S. Ahn, R. P. Tsang, S.-R. Tong, and D. H. C. Du. Virtual Path Layout Design on ATM Networks. In *Proceedings of IEEE INFOCOM*, pages 192–200, 1994.

[B+95] N. J. Boden et al. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, February 1995.

[BCS93] E. Biagioni, E. Cooper, and R. Sansom. Designing a Practical ATM LAN. *IEEE Network*, 7(3):32–39, March 1993.

[Bie93] E. W. Biersack. Performance Evaluation of Forward Error Correction in an ATM Environment. *IEEE Journal on Selected Areas in Communications*, 11(4):631–640, May 1993.

[C+94] K. Castagnera et al. NAS Experiences with a Prototype Cluster of Workstations. In *Proceedings of Supercomputing'94*, pages 410–419, 1994.

[Cav97] K. Caves. *An AAL Type 2 SSCS Proposal for the Support of Synchronous Low Bit Rate Services*. ATM Forum, Contribution ATM_Forum/97-0060, February 1997.

[CCJ95] A. Charny, D. D. Clark, and R. Jain. Congestion Control With Explicit Rate Indication. In *Proceedings of ICC'95*, 1995.

[CDH+94] S-L Chang, D. H. C. Du, J. Hsieh, M. Lin, and R. P. Tsang. Parallel Computing over a Cluster of Workstations Interconnected via a Local ATM Network. Technical report, University of Minnesota, 1994.

[CF96] P. Crocetti and L. Fratta. Video on Demand Service Based on ATM Virtual Private Networks. In *Proceedings of the European Conference on Multimedia Applications, Services and Techniques (ECMAST'96)*, pages 209–221, 1996.

[CJRS89] D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen. An Analysis of TCP Processing Overhead. *IEEE Communications Magazine*, 27(6):23–29, June 1989.

[CLZ87] D. D. Clark, M. Lambert, and L. Zhang. NETBLT: A Bulk Data Transfer Protocol. In *Proceedings of the SIGCOMM'87 Conference*, 1987.

[CS94] R. Cohen and A. Segall. Connection Management and Rerouting in ATM Networks. In *Proceedings of IEEE INFOCOM*, pages 184–191, 1994.

[CW89] D. R. Cheriton and C. L. Williamson. VMTP as the Transport Layer for High-Performance Distributed Systems. *IEEE Communications Magazine*, 27(6):37–44, June 1989.

[Dav93] B. S. Davie. The Architecture and Implementation of a High-Speed Host Interface. *IEEE Journal on Selected Areas in Communications*, 11(2):228–239, February 1993.

[DDK+90] W. A. Doeringer, D. Dykeman, M. Kaiserswerth, B. W. Meister, H. Rudin, and R. Williamson. A Survey of Light-Weight Transport Protocols for High-Speed Networks. *IEEE Transactions on Communications*, 38(11):2025–2039, November 1990.

[DDP94] P. Druschel, B. S. Davie, and L. L. Peterson. Experiences with a High-Speed Network Adaptor: A Software Perspective. Technical Report TR 94-5, Department of Computer Science, The University of Arizona, 1994.

[DLW94] B. J. Dempsey, J. Liebeherr, and A. C. Weaver. On Retransmission-Based Error Control for Continuous Media Traffic in Packet-Switching Networks. Technical report, Computer Science Department, University of Virginia, 1994.

[dM95] M. d'Ambrossio and R. Melen. Evaluating the Limit Behavior of the ATM Traffic Within a Network. *IEEE/ACM Transactions on Networking*, 3(6):832–841, December 1995.

[DOSW96] J. J. Dongarra, S. W. Otto, M. Snir, and D. Walker. A Message Passing Standard for MPP and Workstations. *Communications of the ACM*, 39(7):84–90, July 1996.

[DSBC95] P. W. Dowd, S. M. Srinidhi, E. Blade, and R. Claus. Issues in ATM Support of High Performance Geographically Distributed Computing. In *Proceedings of IPPS'95 Workshop on High Speed Networks*, pages 352–358, 1995.

[ECD+95] H. Esaki, G. Carle, T. Dwight, A. Guha, K. Tsunoda, and K. Kanai. *Necessity of an FEC Scheme for ATM Networks.* ATM Forum, Contribution ATM_Forum/95-0325R2, October 1995.

[ECG+95] H. Esaki, G. Carle, A. Guha, K. Tsunoda, and K. Kanai. *Draft Proposal for Specification of FEC-SSCS for AAL Type 5*. ATM Forum, Contribution ATM_Forum/95-0326R2, October 1995.

[Fel93] D. C. Feldmeier. A Framework of Architectural Concepts for High-Speed Communication Systems. *IEEE Journal on Selected Areas in Communications*, 11(4):480–488, May 1993.

[FGCF95] S. Fotedar, M. Gerla, P. Crocetti, and L. Fratta. ATM Virtual Private Networks. *Communications of the ACM*, 38(2):101–109, February 1995.

[FHW96] V. J. Friesen, J. J. Harms, and J. W. Wong. Resource Management with Virtual Paths in ATM Networks. *IEEE Network*, 10(5):10–19, September/October 1996.

[FM89] A. Fraser and W. Marshall. Data Transport in a Byte-Stream Network. *IEEE Journal on Selected Areas in Communications*, SAC-7:1020–1033, September 1989.

[For94] Fore Systems. *Devices and Network Interfaces (man pages)*, 1994.

[For96] Fore Systems, Inc. *ForeThought Bandwidth Management, Version 1.0*, 1996.

[Fos95] I. Foster. *Designing and Building Parallel Programs.* Addison-Wesley, Inc., Reading, 1995.

[FW94] R. Fatoohi and S. Weeratunga. Performance Evaluation of Three Distributed Computing Environments for Scientific Applications. In *Proceedings of Supercomputing'94*, pages 400–409, 1994.

[G+94] A. Geist et al. *PVM 3 Users' Guide and Reference Manual*. Oak Ridge National Laboratory, 1994.

[Gar96] M. W. Garret. A Service Architecture for ATM: From Applications to Scheduling. *IEEE Network*, 10(3):6–14, May/June 1996.

[Gil96] R. B. Gillett. Memory Cannel Network for PCI. *IEEE Micro*, 16(1):12–18, February 1996.

[GVH96] X. Garcia Adanez, O. Verscheure, and J.-P. Hubaux. New Network and ATM Adaptation Layers for Real-Time Multimedia Applications: A Performance Study Based on Psychophysics. In *Proceedings of the 3rd International COST 237 Workshop*, pages 216–231, 1996.

[Hen95] T. R. Henderson. Design principles and performance analysis of SSCOP: a new ATM Adaptation Layer protocol. *ACM Computer Communication Review*, 25(2):47–59, April 1995.

[HHM95] C. Huang, Y. Huang, and P. K. McKinley. A Thread-Based Interface for Collective Communication on ATM Networks. In *Proceedings of the 1995 International Conference on Distributed Computing Systems*, 1995.

[HP96] J. L. Hennessy and D. A. Patterson. *Computer Architecture. A Quantitative Approach*. Morgan Kaufmann, 2nd edition, 1996.

[HPPF94] S. Hariri, JB Park, M. Parashar, and G. C. Fox. A Communication System for High-Performance Distributed Computing. *Concurrency, Practice and Experience.* Special Issue: High Performance Distributed Computing, June 1994.

[Int96] Intel corp. and other companies. *Windows Sockets 2 Protocol Specific Annex.* Revision 2.0.2, January 1996.

[ITU92] ITU-T, Recommendation X.213. *Information Technology — Network Service Definition for Open Systems Interconnection*. Geneva, November 1992.

[ITU93a] ITU-T, Draft Recommendation Q.2100. *B-ISDN Signalling ATM Adaptation Layer Overview Description*. Geneva, December 1993.

[ITU93b] ITU-T, Recommendation I.311. *B-ISDN General Network Aspects.* Helsinki, March 1993.

[ITU93c] ITU-T, Recommendation I.361. *B-ISDN ATM Layer Specification*. Helsinki, March 1993.

[ITU93d] ITU-T, Recommendation I.362. *B-ISDN ATM Adaptation Layer (AAL) Functional Description*. Helsinki, March 1993.

[ITU93e] ITU-T, Recommendation I.363. *B-ISDN ATM Adaptation Layer (AAL) Specification.* Helsinki, March 1993.

[ITU93f] ITU-T, Recommendation I.364. *Support of Broadband Connectionless Data Service on B-ISDN*. Helsinki, March 1993.

[ITU93g] ITU-T, Recommendation I.365.1. *Frame Relaying Service Specific Convergence Sublayer (FR-SSCS)*. Geneva, November 1993.

[ITU93h] ITU-T, Recommendation X.214. *Information Technology — Open Systems Interconnection — Transport Service Definition*. Geneva, November 1993.

[ITU94a] ITU-T, Draft Recommendation I.365.2. *Service Specific Coordination Function to Provide CONS (SSCF-CONS)*. Geneva, September 1994.

[ITU94b] ITU-T, Draft Recommendation Q.2110. *B-ISDN ATM Adaptation Layer - Service Specific Connection Oriented Protocol (SSCOP).* Geneva, March 1994.

[ITU95a] ITU-T, Draft Recommendation I.371. *Traffic Control and Congestion Control in B-ISDN*. Geneva, July 1995.

[ITU95b] ITU-T, Recommendation I.365.3. *B-ISDN ATM Adaptation Layer Service Specific Coordination Function to Provide COTS (SSCF-COTS)*. Geneva, November 1995.

[ITU96] ITU-T, Draft new Recommendation I.363.2. *B-ISDN ATM Adaptation Layer Type 2 Specification*. Madrid, November 1996.

[Jai91] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc., New York, 1991.

[Jai95] R. Jain. Congestion Control and Traffic Management in ATM Networks: Recent Advances and A Survey. *Submitted to Computer Networks and ISDN Systems*, February 1995.

[JKG+96] R. Jain, S. Kalyanaraman, R. Goyal, S. Fahmy, and R. Viswanathan. *The ERICA Switch Algorithm: A Complete Description*. ATM Forum, Contribution ATM_Forum/96-1172, August 1996.

[JKGF96] R. Jain, S. Kalyanaraman, R. Goyal, and S. Fahmy. Source Behavior for ATM ABR Traffic Management: An Explanation. *Submitted to IEEE Communications Magazine*, July 1996.

[JKVG95] R. Jain, S. Kalyanaraman, R. Viswanathan, and R. Goyal. *A Sample Switch Algorithm*. ATM Forum, Contribution ATM_Forum/95-0178R1, February 1995.

[KAP95] K. K. Keeton, T. E. Anderson, and D. A. Patterson. LogP Quantified: The Case for Low-Overhead Local Area Networks. In *Proceedings of Hot Interconnects III*, 1995.

[LDTM95] M. Lin, D. H. C. Du, J. P. Thomas, and J. A. MacDonald. Distributed Network Computing over Local ATM Networks. *IEEE Journal on Selected Areas in Communications*, 13(4):733–748, May 1995.

[LST+95] H. Li, K.-Y. Siu, H.-Y. Tzeng, C. Ikeda, and H. Suzuki. TCP Performance over ABR and UBR Services in ATM. In *Proceedings of the International Phoenix Conference on Computers and Communications (IPCCC'96)*, 1995.

[LT94] G. M. Lundy and H. A. Tipici. Specification and Analysis of the SNR High-Speed Transport Protocol. *IEEE/ACM Transactions on Networking*, 2(5):483–496, October 1994.

[MABG96] J. Miguel, A. Arruabarrena, R. Beivide, and J. A. Gregorio. Assessing the Performance of the New IBM SP2 Communication Subsystem. *IEEE Parallel and Distributed Technology*, 4(4):12–22, Winter 1996.

[MPBO96] D. Mosberger, L. L. Peterson, P. G. Bridges, and S. O'Malley. Analysis of Techniques to Improve Protocol Processing Latency. In *Proceedings of the SIGCOMM'96 Conference*, 1996.

[MSD94] M. Medin, S. M. Srinidhi, and P. W. Dowd. *Issues in ATM API Support for Distributed Computing*. ATM Forum, Contribution ATM_Forum/94-1153R1, November 1994.

[MYW94] A. Mainwaring, C. Yoshikawa, and K. Wright. *NOW White Paper: Network RAM Prototype*. U.C. Berkeley, White Paper, November 1994.

[New94a] P. Newman. ATM Local Area Networks. *IEEE Communications Magazine*, 32(3):86–98, March 1994.

[New94b] P. Newman. Traffic Management for ATM Local Area Networks. *IEEE Communications Magazine*, 32(8):44–50, August 1994.

[NRS90] A. N. Netravali, W. D. Roome, and K. Sabnani. Design and Implementation of a High-Speed Transport Protocol. *IEEE Transactions on Communications*, 38(11):2010–2024, November 1990.

[PHK+96] S.-Y. Park, S. Hariri, Y. Kim, J. S. Harris, and R. Yadav. NYNET Communication System (NCS): A Multithreaded Message Passing Tool over ATM Network. In *Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing (HPDC-5)*, pages 460–469, 1996.

[RF94] A. Romanow and S. Floyd. Dynamics of TCP Traffic over ATM Networks. In *Proceedings of the SIGCOMM'94 Conference*, pages 79–88, 1994.

[RFC93] RFC 1483. *Multiprotocol Encapsulation over ATM Adaptation Layer 5*. IETF, July 1993.

[RFC94] RFC 1577. *Classical IP and ARP over ATM*. IETF, January 1994.

[Ros95] T. L. Ross. ATM APIs: The Missing Links. *Data Communications*, pages 119–124, September 1995.

[RV91] J. W. Roberts and J. T. Virtamo. The Supoerposition of Periodic Cell Arrival Streams in an ATM Multiplexer. *IEEE Transactions on Communications*, 39(2):298–303, February 1991.

[S+95] C. B. Stunkel et al. The SP2 High-Performance Switch. *IBM Systems Journal*, 34(2):185–204, 1995.

[SBT96] D. J. Scales, M. Burrows, and C. A. Thekkath. Experience with Parallel Computing on the AN2 Network. In *Proceedings of the 10th International Parallel Processing Symposium (IPPS'96)*, 1996.

[SDW92] W. T. Strayer, B. J. Dempsey, and A. C. Weaver. *XTP: The Xpress Transfer Protocol*. Addison-Wesley, 1992.

[SGI96] SGI Supercomputing Team. *Tuning and Optimization Seminar*. Seminar given by Silicon Graphics, February 1996.

[SPSLA95] J. Solé-Pareta, D. Sarkar, J. Liebeherr, and I. F. Akyildiz. An Adaptive Multipath Routing Scheme for Connectionless Traffic in an ATM Network. In *Proceedings of ICC'95*, pages 1626–1630, 1995.

[SPVS96] J. Solé-Pareta and J. Vila-Sallent. Network-Based Parallel Computing over ATM Using Improved SSCOP Protocol. *Computer Communications*, 19(11):915–926, September 1996.

[SV96] M. W. Sachs and A. Varma. Fibre Channel and Related Standards. *IEEE Communications Magazine*, 34(8):40–50, August 1996.

[TL93] C. A. Thekkath and H. M. Levy. Limits to Low-Latency Communication on High-Speed Networks. *ACM Transactions on Computer Systems*, 11(2):179–203, May 1993.

[TS93] C. B. S. Traw and J. M. Smith. Hardware/Software Organization of a High Performance ATM Host Interface. *IEEE Journal on Selected Areas in Communications*, 11(2):240–253, February 1993.

[vEBB94] T. von Eicken, A. Basu, and V. Buch. Low-Latency Communication Over ATM Networks Using Active Messages. *Proceedings of Hot Interconnects II*, 1994.

[vEBBV95] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, 1995.

[vECGS92] T. von Eicken, D. Culler, S. Goldstein, and K. Schauser. Active Messages: A Mechanism for Integrated Communication and Computation. In *Proceedings of the 19th ACM International Symposium on Computer Architecture*, pages 256–266, 1992.

[VSSP96a] J. Vila-Sallent and J. Solé-Pareta. High Performance Distributed Computing over ATM Networks: A Survey of Strategies. In *Proceedings of the 2nd International Conference on Massively Parallel Computing Systems (MPCS'96)*, pages 153–160, 1996.

[VSSP96b] J. Vila-Sallent and J. Solé-Pareta. Supporting HPDC Applications over ATM Networks with Cell-Based Transport Mechanisms. In *Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing (HPDC-5)*, pages 595–604, 1996.

[VSSP97] J. Vila-Sallent and J. Solé-Pareta. Providing Low-Latency Communications for ATM-Based Parallel Computing Environments. Submitted to Infocom'98, 1997.

[VSSPJT97] J. Vila-Sallent, J. Solé-Pareta, T. Jové, and J. Torres. Potential Capability of ATM to Support Network-Based Parallel Computing. Submitted to Globecom'97, 1997.

[VSSPTJ97] J. Vila-Sallent, J. Solé-Pareta, J. Torres, and T. Jové. Performance Comparison of Parallel Algorithms on Different Communication Architectures. In *Proceedings of the 9th International Conference on Parallel and Distributed Computing and Systems (PDCS'97)*, 1997. Accepted to appear.

[WÅS95] S. White, A. Ålund, and V. S. Sunderam. Performance of the NAS Parallel Benchmarks on PVM-Based Networks. *Journal of Parallel and Distributed Computing*, 26:61–71, 1995.

[WHJ+95] D. A. Wallach, W. C. Hsieh, K. L. Johnson, M. F. Kaashoek, and W. E. Weihl. Optimistic Active Messages: A Mechanism for Scheduling Communication with Computation. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'95)*, 1995.

[YRHF95] R. Yadav, R. Reddy, S. Hariri, and G. C. Fox. A Multithreaded Message Passing Environment for ATM LAN/WAN. In *Proceedings of the 4th IEEE International Symposium on High Performance Distributed Computing (HPDC-4)*, 1995.

[ZG95] H. Zhou and A. Geist. Faster Message Passing in PVM. In *Proceedings of IPPS'95 Workshop on High Speed Networks*, 1995.