20

Performance issues in implementing a portable SMDS server

J.-M. Jézéquel and F. Guerber
I.R.I.S.A. Campus de Beaulieu
F-35042 RENNES CEDEX, *FRANCE*
E-mail: jezequel@irisa.fr
Tel: +33 99847192 Fax: +33 99847171

Abstract

Connectionless servers may be used to provide connectionless data services in ATM wide area networks. However, their performance might be a critical point, since, like with any other server, undersized connectionless servers might become bottlenecks in the network. This paper aims at providing some insight on this issue. We present the design and the implementation of a portable SMDS (Switched Multi-megabits Data Service) server, and discusses performance related aspects of a SMDS server-based connectionless network implemented in our laboratory. We conclude on the interest and perspectives of this kind of network architecture.

Keywords Internetworking, SMDS, High Performance Routing, Implementation and Performance Evaluation

1 INTRODUCTION

In their comprehensive overview on connectionless data services for ATM networks, Le Boudec et al. [LB et al.94] identify two families of methods for routing connectionless messages on ATM networks.

On the one hand, *server-based* methods rely on an overlay network with non-ATM switching to transfer connectionless messages. This network consists in a set of ATM interconnected *connectionless servers*. Clients willing to do connectionless traffic just have to access the nearest connectionless server using any available protocol, e.g. an ATM connection.

On the other hand, *integrated* methods rely entirely on the ATM switches to transfer messages. In that case, the VPI/VCI labels are usually given a new semantics to allow for a *connectionless service*, and the ATM switches are modified accordingly.

Integrated methods provide a cost effective and efficient solution for ATM-based LAN with a limited number of addressable entities, but where connectionless traffic typically makes up for an important part of the overall traffic.

R. Puigjaner (ed.), *High Performance Networking*

The server-based methods are more affordable for WAN because they have the advantage of putting no specific requirement on the ATM switches. The connectionless servers can then be developed independently, using any standard access to the switch. However, their performance might be a critical point, since, like with any other server, undersized connectionless servers might become bottlenecks in the network.

This paper aims at providing some insight on this issue. It discusses performance related aspects of a SMDS (Switched Multi-megabits Data Service [Bellcore92]) server-based connectionless network implemented in our laboratory. In the section 2, we introduce SMDS and the context of our study. We then discuss the design and the implementation of a portable SMDS server (section 3). The section 4 is dedicated to the presentation of single server performances, whereas the section 5 addresses multi-server performances. We conclude on the interest and perspectives of this kind of network architecture.

2 BUILDING A PORTABLE SMDS SERVER

2.1 An overview on SMDS

SMDS is a connectionless, packet-switched data transport service running on top of connected networks such as ATM or DQDB.

SMDS has been designed to provide high throughput and low-delay transmissions, and to be able to maintain them over a large geographic area. As a result, it can be used to interconnect multiple node LANs and WANs, providing them a "any-to-any" service that includes features such as virtual private network facilities.

Being a connectionless service, SMDS eliminates the need for carrier switches to establish a call path between two points of data transmission. Each switch reads the E.164 address included in SMDS packets, and then forwards them one-by-one over any available path to the desired endpoint.

The benefit of this connectionless "any-to-any" service is that it puts an end to the need for precise traffic-flow predictions and connections only between fixed locations. With no need for a pre-defined path between devices, data can travel over the least congested routes in an SMDS network, providing faster transmission, increased reliability and greater flexibility to add or drop network sites.

2.2 SMDS network architecture

An SMDS network is based on a three-tiered architecture: a switching infrastructure made of SMDS switches, a delivery system made of SNIs (Subscriber Network Interface), and an external network access system, ICI (Independent Carrier Interface). So each SMDS server has to switch packets coming from SNIs, ICIs, and ISSI (Inter Switching-System Interface) links (see figure 1).

SMDS has been designed to be supported by various lower-level layers, e.g. ATM (AAL 3/4 or even 5) or DQDB (see figure 2). SMDS thus features a technology independent interface, allowing for fully portable SMDS servers. There also exists a variant in which this technology independence is not respected. In the so-called high performance SMDS,

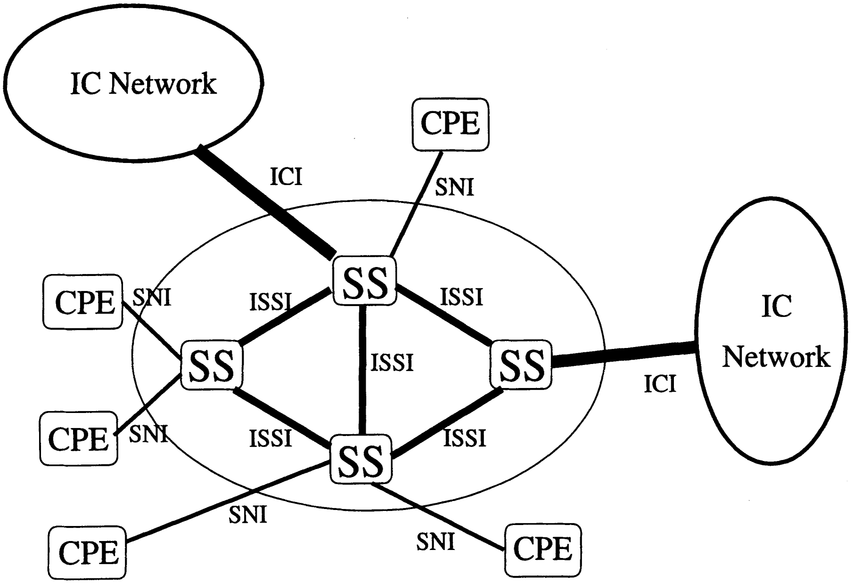

Figure 1 Architecture of an SMDS network

the SAR (Segmentation and Reassembly) of AAL3/4 cells is avoided thanks to a *cut-through* routing, making use of their Message Identifier (MID) field. In this paper, we concentrate on portable SMDS servers: *cut-through* routing might then be considered as yet another hardware dependent optimization.

The idea behind a portable SMDS server is the possibility to implement it on a parallel computer featuring a "general purpose" (Unix like) operating system, while prototyping it on a standard Unix workstation. It also makes it possible to use a high level language allowing to concentrate on algorithmic optimization issues instead of C or assembly language fine tuning and debugging.

Portability is not yet an established idea in the telecommunication world where most systems are of a real time nature, and are finely tuned to get the best performances out of a given architecture. However, this is changing because the versatility of the new value added telecommunication services induces huge software development costs that need to be paid off on more than one hardware generation (itself becoming shorter and shorter). It is thus important to get some insight on the performance costs of portability for this kind of software.

We want to know whether this approach can provide high-throughput and low-delay transmissions without the use of dedicated materials, further that a general purpose parallel computer. Our evaluation criteria is whether such a (parallel) "low-performance" SMDS server is likely to provide gigabit data flow rates (cited as an unlikely reachable limit in [LB et al.94]).

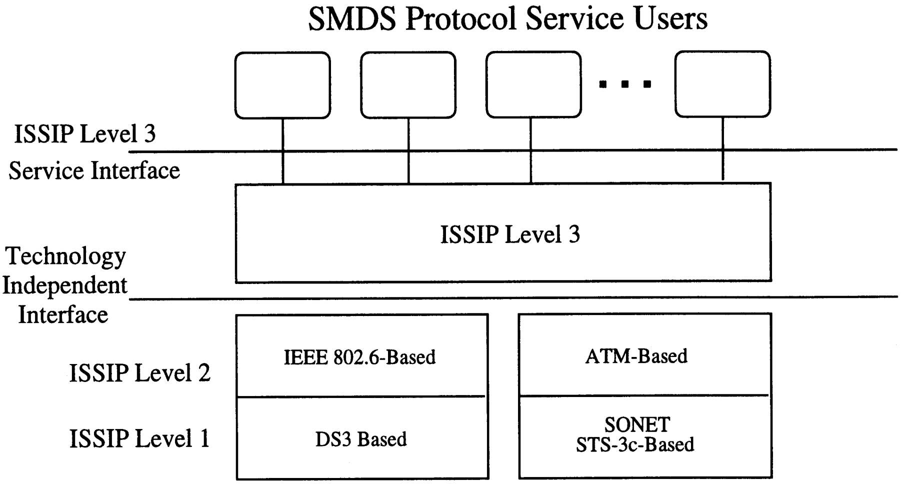

Figure 2 Architecture of an SMDS server

3 DESIGN AND IMPLEMENTATION CONCERNS

3.1 Achieving high performances

They are a number of studies in the literature identifying bottlenecks in high-performance communication systems. Depending on the context in which their authors work, a number of approaches aiming at circumventing these bottlenecks have been proposed.

First, the PDU header processing speed determines an absolute limit on the global performances of the communication system: the system may not have a throughput per I/O board greater than the maximum PDU size divided by the header processing speed. This header processing speed has then to be optimized as much as possible. The introduction of parallelism has been considered at this level, but it does generally not pay off [Diot91, Tantawy93] because of the limited intrinsic provision for concurrency in such kind of processing. This tendency is even enforced with recent "light-weight" protocol (XTP, SMDS, etc.) featuring simplified header processing, actually leaving nearly no room for parallelization.

Another important source of performance loss is linked to data movement inside the system [Zitterbart91, Ito et al.93]. To be efficient, an implementation should avoid to copy PDU from memory to memory because of its time cost (DRAM bandwidth does not follow processing power). Also, for parallel implementations, data transfers between two separate nodes should be minimized because excessive internal communications can lead to link saturations and even to a global system slowdown. A good solution [Braun et al.91] is the use of a shared memory where are stored all the PDU. The processing nodes would only access the headers and trailers of these packets and would never deal with the data they include.

The execution environment has also a great influence on performances. The raw power of each processor determines the throughput of the system [Ito et al.93], and the bandwidth of the channels connecting the different nodes limits the global data flow.

The last bottleneck may be the interface between the processing system and the physical layer [Braun et al.91]: its throughput should be sufficient not to limit the system. If the number of access points to the physical links is insufficient, these are considered as shared resources. They can then be saturated if too many processing units try to use them at once.

All these points were taken into consideration in the design of our SMDS server.

3.2 An Object Oriented Approach

Since we wanted to design a highly evolutive and portable SMDS server, we decided to use an object oriented analysis and design method (the *Object Modeling Technic* [Rumbaugh et al.91]) followed with an implementation with an object oriented language: Eiffel [Meyer92].

The first step towards an object-oriented analysis is concerned with devising a precise, relevant, concise, understandable, and correct model of the real world. The purpose of object-oriented analysis is to model the problem domain so that it can be understood, and serve as a stable basis preparing the design step.

The analysis model extends itself in three dimensions:

- the object model, showing the static structure of the real world system through abstract or physical classes and their relationships.
- the dynamic model, showing the temporal behavior of the objects in the system.
- the functional model, showing the constraints between the objects in the system (and notably between inputs and outputs).

The *design* phase starts with the output of the analysis phase and gradually shifts its emphasis from application domain to computation domain: the implementation strategy is defined, and trade offs are made according to the priorities defined in the previous section. Then the definition of classes is refined by collapsing on the object model the two other analysis dimensions (dynamic and functional). Auxiliary classes may be introduced at this stage to deal with complex relationship or implementation related matters.

The output of the object oriented design phase is a blueprint for the implementation. If the implementation is made with an object oriented language such as Eiffel, it is basically an extension of the design process. The Eiffel object oriented language emphasizes the design and construction of large, high-quality softwares by assembling reusable software components made of classes, coming from either standard libraries (providing arrays, lists, hash tables, etc.), or classes developed by other programmers during previous projects.

Beyond classes (on which modularity is based), Eiffel offers multiple inheritance, polymorphism, static typing and dynamic binding, genericity, garbage collection, a disciplined exception mechanism, and systematic use of assertions to improve software correctness in the context of *programming by contract.*

The output of the design stage gave us the class hierarchy blueprint allowing us to build an actual prototype for each class (with stub routines), and then implement the routine bodies class by class. After a class is actually implemented, it is tested separately to validate its internal consistency (unit testing). It is then added to the system *in lieu* of its stub class, and the system is tested against this new set of functionalities. The testbed is a bunch of Unix workstations communicating through an Ethernet. Since we are not interested in performances at this stage, the low bandwidth of Ethernet is not a problem.

Our SMDS server actually uses a highly portable communication library called POM (Parallel Observable Machine). This library, ported on all the parallel computer OS available in our lab (including the network of workstations) greatly facilitates the porting of communicating applications across widely different architectures. The POM is then used as the underlying layers 1 and 2 for our SMDS server, whose layer 3 is technology-independent.

This way, the SMDS server is built and tested incrementally, module by module, thus minimizing integration problems. The reuse rate of our code is quite high. Including the Eiffel libraries we used, our system comes up to 119 classes with a total of 19660 lines of code. From these, 45% of the classes and 60% of the lines of code composing our system come from the standard Eiffel libraries.

3.3 Implementation remarks

3.3.1 Using a garbage collector

With many languages, programmers must explicitly reclaim heap memory at some point in the program, by using a *free* or a *dispose* statement. Eiffel frees the programmer from this burden, thanks to a garbage collector.

It was once widely believed that garbage collection was quite expensive relative to explicit heap management, but recent advances in garbage collection technology make automatic storage reclamation affordable for use in high-performance systems. Generational techniques reduce the basic costs and disruptiveness of collection by exploiting the empirically-observed tendency of objects to die young. Incremental techniques may even make garbage collection relatively attractive for real-time systems. Most Eiffel implementations come with such an incremental garbage collector, which can be activated and suspended at will.

We measured that the garbage collector's work is shorter when it is called often, and that it globally needs less than 1% of the computation time. So, with frequent iterative collections, its work duration has a Gaussian kind of distribution with an average around 3ms. As a result, our SMDS server launches an iterative collection each time it is otherwise idle (not to penalize packet processing), but if a working period is too long, a collection is forced so the optimal interval between two collections is respected, and the collecting time never exceeds 15 ms and has a probability of 99.9% to be under 10ms.

Further that allowing us to limit the garbage collector monopolization of the processor to short periods, the shifting of memory management processing to idle periods allows time savings during active periods. Our server then has a higher ability to absorb traffic peaks.

Kind of processing	headers processed/s	mean processing time
SNI to ISSI	5025	199 us
ISSI to SNI	8093	124 us
Switching (ISSI to ISSI)	5347	187 us

Table 1 Header processing speed in an SMDS server

3.3.2 SMDS Protocol problems

The Bellcore specifications describing SMDS sometimes introduces hardly achievable constraints. For example, with respect to the updating routing tables, the O6-1 and O6-5 paragraphs specify that it should take no more than 100ms to recompute these tables in case of a modification in the network. However with some tricky network topologies, this computation has a $O(n^2.log_2n)$ complexity since n shortest path spanning trees may need to be calculated (n is the number of SMDS servers in the network), each with the Dijkstra algorithm (complexity $o(n.log_2n)$). As a result, the size of an SMDS network is bounded in some way by this real time limit.

By the way, the SMDS protocol sometimes has strange behaviors. In the case of a particular topology, full adjacency in the network is sometimes really long to be reached: two fully adjacent subnets connected by a unique link A need half an hour (*LSRefreshTime*) to become adjacent if A is *down* (which occurs when getting started or after a failure) . This is due to a problem in the database loading stage: the two servers connected to the link A become adjacent but never give all informations to the other servers of their subnets, so these have to wait for the LSA (Link State Advertisement) automatic re-flood.

4 THE SMDS SERVER PERFORMANCES

4.1 Header processing speed

We first determine the internal performance limits of our SMDS server. The significant figures are the speed of header processing in different contexts: transmission (packet received from an SNI and then injected in the SMDS network), reception (PDU coming from an ISSI link and delivered to a SNI), and switching of traffic (from an ISSI link to an other one).

For these specific measures, we use a specialized SMDS server, where the lower layers are simulated: a transmission only consists in incrementing a counter; and as for receptions, the server is always told that a PDU is ready to be read and the reception is simulated (a fixed set of predefined PDU is used). The measures consist in performing continual operations on the server and compute their mean durations (which is more realistic than exploring the assembly language listing to add up individual times of machine language instructions on a given path of the header processing).

The following measures have been made on Sun SPARC20 workstation. Since these

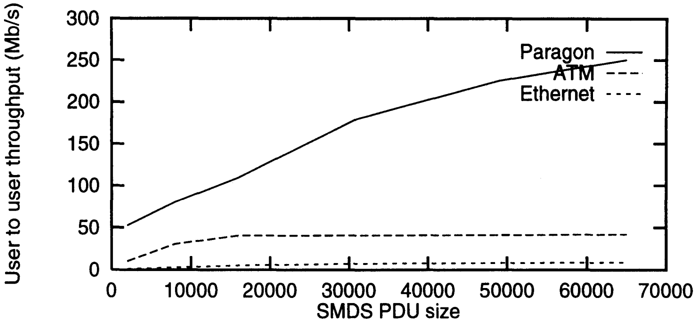

Figure 3 Maximal data flow rate of a SMDS server

tests involve no network connection, the results are directly proportional to the processing power of the processor.

The performance figures exposed in table 1 show that the internal speed of a sequential server is sufficient to reach Gigabit flow rates on a standard processor: the slower operation reaches 3.29 Gb/s with 64k PDU.

Thus the bottleneck in such a system is not likely to be localized in the header processing but in the interface with the network. We have to measure the maximal data flow rate of a real server to confirm that.

4.2 Flow rate measurement

We want to determine the maximal data flow rate of our SMDS server for a set of representative physical network technologies. We measure a one way, user to user, actual data flow rate through two SMDS servers communicating through a unique link. This measure takes into account the header processing, the operating system and the SAR overheads, and the actual data transmission between the two users. This experiment has been done with a range of PDU sizes, allowing us to get information on both the latency and the throughput of the network.

The Ethernet (maximum bandwidth 10 Mbs/s) and ATM experiments were led on Sparc workstations, with Fore System SBA-200 SBus interface boards in the later case (maximum bandwidth 150 Mbs/s). To experiment with an higher bandwidth, we also used the internal network of a parallel computer, the Intel Paragon XP/S. This computer is made of 56 processing nodes, linked by high-speed communication channels (having a maximum bandwidth of 640 Mbs/s for 64K messages) in a 2D grid topology. Each node of this machine has, in addition to the main processor (i860), a co-processor dedicated to communications with the other nodes. As a result, we consider the Paragon XP/S as a really efficient support for an application such as our SMDS server. By the way, its use allowed us to simulate large SMDS networks in a real context of multi-megabit communication lines.

The parameters of these measures are a buffer size (for reception on ISSI links) of 10 PDUs, and a transmission rate different for each PDU size and optimized to fit the

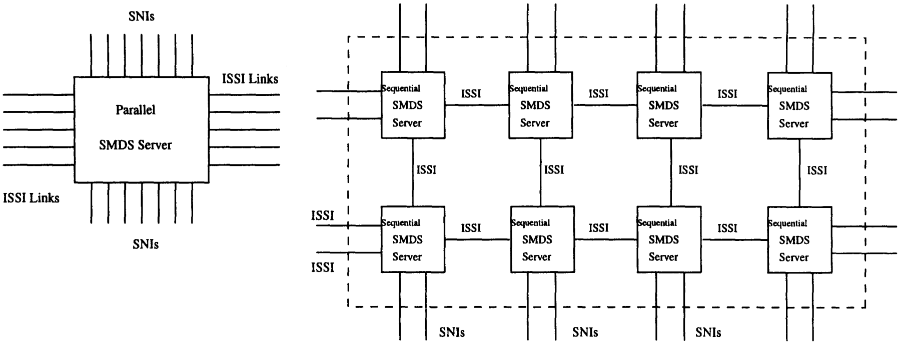

Figure 4 Distributing the network connections to get a Parallel SMDS server

throughput of the receiving server, thus avoiding artificial congestions. The results appear on figure 3.

In any of these cases, the physical network bandwidth and the internal processing speed (since a i860 has approximately the same processing power as a Sparc) of our SMDS server both widely exceed the server maximal unidirectional flow rate. This means that in our system, there exists a bottleneck located in the lower layers of the protocol (probably due to some inefficiencies at the network-server interface, to SAR, and to OS overhead).

5 MULTI-SERVER PERFORMANCES

5.1 Circumventing the bottleneck with parallelism

The classical solution to this problem consists in multiplying access points to the network. For that, we distribute the ISSI and SNI connections among the different nodes of a parallel computer (see figure 4), each one having its own OS and interface(s) with the SMDS network, and collaborating with other nodes to provide the SMDS service.

This "parallelization" technique is related to the SPMD (Single Program, Multiple Data) model. Each node of the parallel SMDS server is actually the sequential SMDS server described above: no new code has to be written, only configuration files have to be modified.

This simple (yet widely used) method results in extending the number of SMDS servers in the network. It has the advantage of being quite scalable: the aggregate bandwidth of such a server should be proportional to the number of supporting nodes, hence making available a range of performances easily adjustable to the user needs (because no new software has to be written).

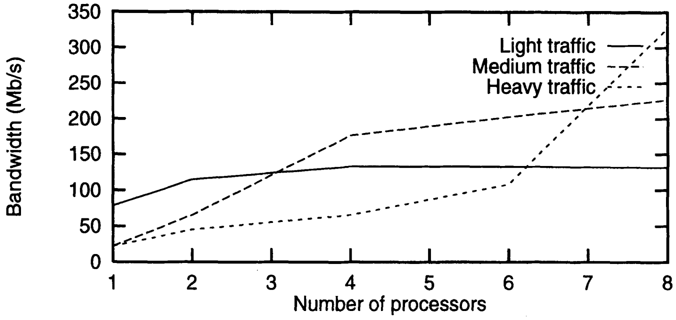

Figure 5 Total user data flow rate for 32k packets, with various traffic densities

5.2 Measuring Aggregate Bandwidth

These performance tests consist in measuring the maximal switching capacity of a parallel server, depending on the number of processor it has. The test architecture includes:

- a parallel server (as defined above) implemented on 1, 2, 4, 6 or 8 nodes of the Paragon XP/S
- an environment (surrounding the parallel server) made of a number of other SMDS servers (e.g. eight), each one implemented on one node of the Paragon XP/S and achieving traffic generation and absorption.

Since we are mainly interested in its switching capacity, our parallel server does not support SNIs, whereas the other servers are normal servers supporting SNIs that send and receive packets.

We are interested in seeing how faster works a parallel server, depending on the number of nodes it has. So we measure data flow rates for various PDU sizes, under different conditions of (random) traffic load. We got the kind of results represented on the figure 5. These aggregate bandwidth results were obtained with 32k PDU under three different traffic densities.

As expected, the aggregate bandwidth of a parallel server grows with the number of its nodes. An interesting point is the behavior of the server when switching heavy traffic: small servers (less than 4 nodes) are congested and deliver less than 50 Mb/s, whereas the larger ones are able to manage it, delivering more then 300 Mb/s. This is because the SMDS policy is to discard incoming user data PDU when the server is overwhelmed.

In case of a light traffic, the discarding rate is low and the total throughput of the server does not benefit from the larger number of processor. Large servers work under their maximal capacity, and are quite overkill in this context.

On the other hand, if the traffic is heavier, the small servers crash down: their throughput falls, and the discarding level may becomes really high (up to 87%!).

Each parallel SMDS server thus has an optimal traffic range, under which it does not use the maximum of its capabilities, and over which it is congested. By the way, an entity called the *Congestion Management Protocol* is included in the SMDS server to react in

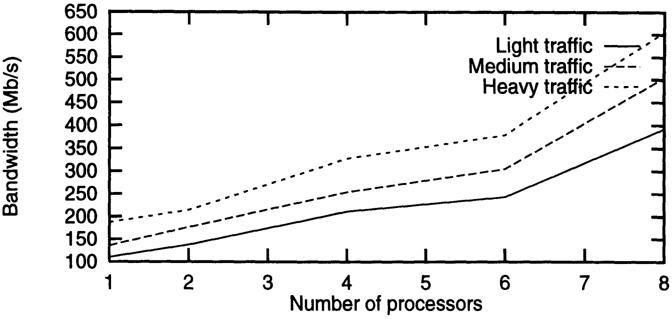

Figure 6 Maximal aggregate bandwidth in optimal conditions

real time to the congestion status of the server. When a server is undergoing a congestion, its CMP entity tell the neighbor servers to slash a part of their traffic directed to it.

5.3 Maximal global performances

From all previous measures (various PDU sizes and traffic densities), we extract the best aggregates bandwidth results for each size of SMDS servers (from one to eight nodes). These results are displayed on figure 6.

A single processor server achieves user data switching at a speed of 130Mb/s, whereas parallel servers reach nearly 430Mb/s for 8 processors. We obtain a quasi-linear speed-up, with a mean slope of 1/2. Extrapolating this result, greater flow rates should be achievable by multiplying the number of nodes of the "parallel" SMDS server. But since the speed-up is only of 4 for a 8-nodes server (1/2 slope), these results are not optimal (a slope closer to unity would be more satisfying).

Note that the slight break noticed on the maximal flow rate graphic for the 6-nodes server has a simple explanation: the topology used to support this parallel server is not as efficient as the others, because it is harder to place six nodes surrounded by eight traffic simulators on the Paragon 2D grid. As a result, the achieved flow rate is a bit lower than optimal.

6 CONCLUSION AND FUTURE PROSPECTS

We have presented the design and the implementation of a technology independent and portable SMDS server consistent with Bellcore's specifications. Despite the use of high level tools in a standard Unix environment, the SMDS real time constraints have been respected, due to our control of the Eiffel garbage collector.

Through experimental performance studies, we have shown that a bottleneck existed at the network-server interface. This bottleneck can be circumvented with a simple but scalable parallelization technique based on the mere duplication of the servers. This approach offers interesting performances, since such a parallel SMDS server has a total aggregate bandwidth increasing linearly with the number of its processors to reach 430 Mb/s for 8 processors on a Paragon XP/S.

However, the slope of the linear speed-up is only 1/2, instead of the ideal unity. Furthermore, our simple parallelization technique induces an increase of the number of addresses in the network, since each node of a global parallel SMDS server is considered as an independent sequential server, with its own ID. As a result, the routing tables are more complex and their computation takes much more time: if all the servers in the network are parallelized and implemented on n nodes, the number of addresses in the network would be multiplied by n and the computation time for routing tables by more than n^2 (cf. section 3.3.2), which poses a hard problem since the SMDS routing tables computation time has a fixed upper limit (100ms).

Our ongoing work is then to find more sophisticated parallelization techniques in order to share some resources among the different nodes composing the parallel server, using for example a Shared Virtual Memory [Li86]. In particular we are considering the use of an unique ID for all nodes, and possibly the sharing of the routing tables, along with their parallel computation.

REFERENCES

[Bellcore92] Bellcore. – *Generic Requirements for SMDS Networking.* – Technical Report TA-TSV-001059, Bell Communication Research, 1992.

[Braun et al.91] Braun (T.) and Zitterbart (M.). – Parallel XTP implementation on transputers. *In : The 1991 Singapore International Conference on Networks.* pp. 91–96. – G.S.Poo, Sep 1991.

[Diot91] Diot (C.). – *Architecture pour l'implantation hautes performances des Protocoles de communication de niveau transport.* – PhD thesis, Institut National Polytechnique de Grenoble, January 1991.

[Ito et al.93] Ito (M.), Takeuchi (L.) and Neufeld (G.). – A multiprocessor approach for meeting the processing requirements for osi. *IEEE Journal on Selected Areas in Communications*, vol. 11 (2), February 1993.

[LB et al.94] Le Boudec (J.-Y.), Meier (A.), Oechsle (R.) and Truong (H. L.). – Connectionless data service in an atm-based customer premises network. *Computer Networks and ISDN Systems*, vol. 0 (26), July 1994, pp. 1409–1424.

[Li86] Li (Kai). – *Shared Virtual Memory on Loosely Coupled Multiprocessors.* – PhD thesis, Yale University, September 1986.

[Meyer92] Meyer (B.). – *Eiffel: The Language.* – Prentice-Hall, 1992.

[Rumbaugh et al.91] Rumbaugh (James), Blaha (Michael), Premerlani (William), Eddy (Frederick) and Lorensen (William). – *Object-Oriented Modeling and Design.* – New Jersey, Prentice Hall, 1991.

[Tantawy93] Tantawy (A.). – Réalisation de protocoles à haute performance. *In : Actes du colloque CFIP'93 sur l'ingénièrie des protocoles, Monreal.* – Hermès, September 1993.

[Zitterbart91] Zitterbart (M.). – High-speed transport components. *IEEE Network Magazine*, January 1991, pp. 54–63.