32

High Performance ATM Terminals: Design and Evaluation

George E. Konstantoulakis and George I. Stassinopoulos

Division of Computer Science, National Technical University of Athens
Zographou, GR-157 73 Athens, Greece

This paper intends to show that even low end terminals can, under given implementation guidelines, become hosts of multimedia applications over ATM based B-ISDN. The main line of thought is that the terminal has to be seen as an Interworking Unit between its peripheral devices (disks, etc.) and the high bandwidth network. This view enables to jointly address and process in a "centralized" manner, layered protocol stack(s) used for accessing the network and the peripheral devices. In particular we give directions to eliminate unnecessary block transfers in memory and explain why extensive hardware implementation of the ATM protocol stack is not required to achieve high performance.

Key Words

Interworking Unit, ATM Networks, Broadband Terminal, Multimedia Terminal, Layered Protocol Stack, Cell and Packet Hardware Interface, Protocol Control Information, PC Adapter, File Transfer, Video Transfer.

Acknowledgment

The authors would like to acknowledge the support received within the framework of RACE projects R1022 "Technology for ATD" and R2061 "Exploit" by the European Union, while performing this research.

1. INTRODUCTION

Broadband integrated networks are by now based on the ATM (Asynchronous Transfer Mode) principle [1, 2] and irrespective of controversies about the eventual large scale applicability of this concept, two main lines of products are currently available. On one hand, network administrations promote ATM based crossconnects covering the need of remote LANs (mainly Ethernet) interconnection. This requires newly designed bridges but does not concern new terminals. On the other hand we have the emergence of several ATM based LANs. These are ATM switches with appropriate terminals attached in a star architecture. Here we have the need of high performance terminals, being able to support a communication related stack for up to 155 Mbit/sec ATM rate. At the same time these must provide the processing power needed for commensurably fast and complicated multimedia applications.

D. D. Kouvatsos (ed.), *Performance Modelling and Evaluation of ATM Networks*

It appears at fist sight that a future proof way to address the above terminal performance requirements is to provide a wide array of specialized hardware components and peripheral subsystems in order to cover separately each distinct communication and application related task. Extensive loosely coupled parallelism will thus solve complexity and speed related requirements. One cannot overlook this approach, which might prevail in the future. However, this paper addresses the terminal area in another fashion. We consider the problem of how to upgrade the vast population of installed PC-based terminals, so as to enable access not only to local, but also to upcoming wide area ATM networks. Minimal additional hardware components, essentially a single ATM adapter card, and a centralized software realization is the answer which we present below. In this direction we are encouraged by recent emphasis on the need of appropriate application/network service interfaces and proper hardware/software partitions [3, 4, 5 and 6].

Section 2 gives an overall view of our approach, assumptions and options. In this section we describe an efficient terminal architecture with minimum interprocess and interlayer overhead, considering a simple cell or packet level hardware interface. Furthermore, we present and briefly analyze the architecture of the hardware component of a high speed broadband terminal. In Section 3 we describe a high performance interlayer communication software architecture which is key factor for achieving adequate performance and show that this can be satisfactorily addressed by the terminal's CPU. In Section 4 we present a kernel architecture able to support the multiprocessing requirements of broadband architectures. Queue based event handling can, even under DOS, give the multitasking characteristics required. Section 5 describes a realized case for which both hardware and software developments are mentioned. Section 6 presents the achieved performance and its dependency on some critical protocol parameters. Section 7 takes two typical components of an overall multimedia application (e.g. data and video transfer) and, in a descriptive manner, highlights specific terminal resource management issues. Finally, Section 8 concludes the paper.

2. ARCHITECTURE DESCRIPTION

In this section we present and analyze the requirements and dependencies of a broadband terminal. The system is examined under three perspectives. Firstly, its position and role in an end-to-end communication application is examined with special attention to the overall flow harmonization. Then the terminal internal architecture is analyzed and its functional partition in hardware and software is presented with respect to the executed applications. Finally we focus in a terminal hardware architecture able to give a throughput appropriate to the ATM User Network Interface (UNI) capacity and present our implementation.

2.1. General Issues

The transient network offers resources in terms of capacity and buffering. The capacity of ATM links is the prime parameter for Connection Admission Control (CAC). Since capacity allocation follows different schemes and policies (maximal, mean or "effective" bandwidth) this resource is also the cause of ATM-impairments. Buffering can

accommodate more effective use of bandwidth, or equivalently to prevent impairments which would be caused by temporary lack of it. High transmission rate and segmentation into small sized PDUs (cells) bring burstiness to the emitted traffic characteristics. It is then the terminal's responsibility to control this effect, so that it complies to policing mechanisms and to adapt to the network's buffering resources. Buffering is associated with node memory and tendency exists to resort to large buffering solutions. The emerging concept of the Available Bit Rate (ABR), [15], is a manifestation of this trend.

Between terminal and network, at the UNI, the role of the policing function is essential. This takes a decoupling role between bandwidth related performance and QoS. Policing protects network resources (bandwidth and buffers) from excessive and unauthorized use by terminals. Conversely, adherence by the network to the guaranteed performance parameters, ensures that the terminal is solely responsible for providing the required QoS. One can thus view the network from the terminal side, as only a port characterized by the policing function. According to this simplified view, one can assume that the generated traffic, once it has passed the policing mechanism, is ideally transferred through the network.

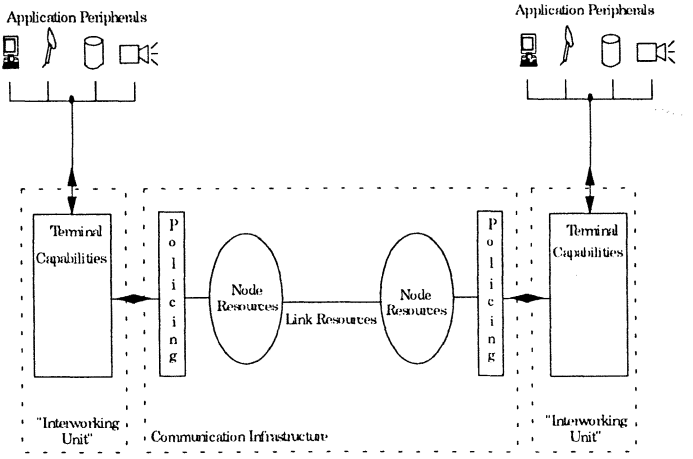

Figure 1: The terminal as Interworking Unit between Peripherals and Network

The policing function and resources at either side of the UNI are shown in Figure 1. Information issued or consumed by the application, involves either humans or peripheral devices. We can see the application part transferring traffic from some peripheral device to the network and vice versa as an Interworking Unit case. Characteristic examples are file and video/sound transfer. In the file transfer case the terminal, as an Interworking Unit, segments (transmitter) and reassembles (receiver) large bulks of data (large PDUs) to and from an ATM cell stream, so that the whole set-up serves a disk-to-disk transfer. In the case of video the ATM cell flow transfers a different PDU flow, e.g. that required by the video protocol. We again have transmission from one peripheral (camera) to another (display).

Performance measures and their mapping to QoS, involve in this case a continuous real time flow.

Throughput is the main parameter. It depends on ATM network performance, since the Adaptation Layer protocol resorts, upon detection of lower layer impairments, in many cases to retransmissions, dropping throughput to lower values. Thus the frequently used term "goodput" is employed. Delay and delay jitter of the ATM cell flow is either accommodated by the Adaptation Layer or converted to cell loss.

Non-conformance of the source (terminal transmitter) to the characteristics of the policing mechanism, can lead to massive cell rejections, perceived as cell losses at the other end. In order to conform, shaping and flow control functions, undertaken by the terminal (transmitter), are essential. The terminal has an inherent flow pattern as given by the delivery of higher layer PDUs from the "peripheral" side. Added to that, the terminal imposes its smaller scale, shorter term shaping to conform to policing. Thus we see that the terminal undertakes an interworking role not only in terms of protocol conversion but also in terms of flow control adaptation between peripheral and ATM access link. The same happens at the receiver and can be easier or more difficult according to the final destination. In the case of file transfer this is straightforward, in the case of audio or video exact playback timing has to be preserved by adaptive, buffer occupancy or time stamping procedures, which pose stringent limits on the information flow to avoid buffer over/underflow.

The terminal internal hardware and software architecture is critical for achieving throughput values comparable to the ATM network's capabilities [3, 4, 5 and 6]. Other performance parameters are considered as of secondary importance and to a certain degree are masked away within the lower layers of the terminal's protocol stack. The terminal must support at the same time the processing tasks associated with the resident broadband service. We thus take the broad view that throughput is essentially determined by appropriate internal design of the terminal, while other QoS parameters by the exogenous performance parameters of the ATM network. In the remainder of this section we address the internal terminal architecture both from a generic and from an implementation specific viewpoint.

2.2. Technical Design Guidelines

Data copies and checksum calculation are shown to be the most consuming operations [4, 7]. Dedicated hardware components are best suited to perform checksumming on the fly, that is following the receive or transmit rates. The result is attached to or compared with the trailer of higher layer PDUs, since checksumming might extend over the payload of a sequence of lower layer units. Data copying for the sake of PDU transfer between adjacent protocol layers occupies processor time as well as the system bus, a critical resource. The same can be said for the exchange of information between memory and network, and memory and peripheral device. It is therefore desirable to minimize data copying and derive an efficient method for directly accessing the 'payload'. The remaining

protocol functions are simple counting, logical and arithmetic operations, that are easier to accommodate in whatever solution effectively addresses the above.

Our approach is therefore to efficiently solve in software the key problem of data copying between layers, use software solutions for straightforward protocol functions influenced by the PCI bits and isolate in hardware only key and well demarcated hardware functions like CRC, independent on their particular placement within the layered stack. Hence, complex and high cost hardware modules are avoided. Figure 2 shows a general terminal architecture designed for efficient resource usage. Three main blocks are foreseen for the configuration of the entire terminal system: the *peripheral equipment* (PE), *the microprocessor system* running the application software (APS) and the *terminating equipment* (TE).

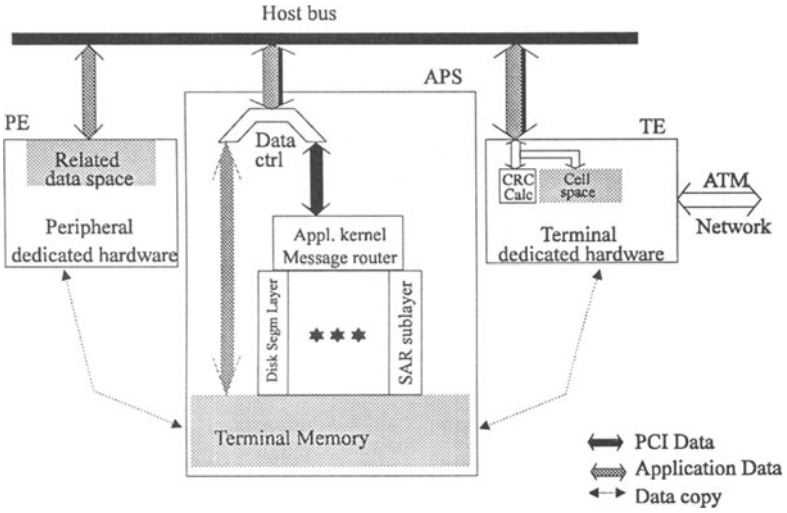

Figure 2: ATM Terminal Components.

The setup of Figure 2 is suited for a microprocessor based computer system using a common bus (i.e. a Personal Computer, PC). The common bus supports the communication among the components, while a usually large memory is needed for storing the exchanged data and messages. A centralized kernel is used to control the entire operation and execute functions equivalent to the routing of messages from one protocol layer to another. The leftmost component (PE) shows the peripheral device that is required by each application. This could be a hard disk for file transfer, a display monitor for image/video transfer, a microphone for sound or a group of these servicing a multimedia application. The rightmost segment (TE) implements the required functions to access the network (at least ATM and Physical layers in our case), while some storage space is used in order to buffer the

incoming cell stream. We also see the CRC calculation function to be executed on-the-fly by a dedicated module within the TE hardware block.

Obviously, some of the data, exchanged through the network are protocol control information (PCI) and will be consumed by the intermediate layers, while the application is executed. The remaining part of the cell stream payloads is the final application data (user data). Thus a special component laying between the hardware and the software is required to distinguish the PCIs from the user data. The data controller component of Figure 2 is a small process, dedicated to recognize the destination of the incoming cell contents. This decision can be taken deterministically for every cell according to the protocol's state and can be efficiently implemented with a look-up table. Then, the data controller initializes DMA operations to transfer the data to the right destination (protocol layer or application buffer). A prominent role is played by the data controller, which undertakes for the transmit and receive flows, the separation of different VP/VC connections and manages the emulation of interlayer communication. Section 4 is devoted to the description of the implementation of the data controller.

In Figure 2 we also show the data flow between the three segments (dotted lines). All data are carried through the host bus. These data are composed by the application payload (APL) and the PCIs. Both segments are transferred from TE to APS while only the APL is transferred from APS to PE. The entire operation as far as the data are concerned is harmonized by the data controller. The PCIs are very small in comparison to the final data used (APL). The application payload in traditional implementations of OSI systems is replicated from the lowest to the upper layer several times as the operation evolves. In this architecture we early separate the application data from the protocol control fields and directly place it at the final destination, achieving only two copies as indicated in Figure 2.

The above description is based on the receiver side of an application. For the transmitter the proposed architecture is also applicable and moreover easier to implement. The data controller at this side multiplexes the application payload with the PCIs forming the final stream. Again only two data copies are needed. This architecture is quite generic and applies either for a packet or a cell level hardware interface. The architecture described above, when performing only cell level operations in hardware, provides the high flexibility required for the evaluation of ATM principles in the early phase of its application.

2.3. Hardware Implementation

When executing protocols in software a centralized architecture is employed where all software modules (protocols) are served by the PC processor and share the system bus. On the other hand for hardware designs a pipelined architecture is appropriate, where data movement is not considered as a problem. Any additional stage in the pipeline causes a negligible delay overhead time delay caused by the additional buffer.

Currently, there is a trend by terminal adapter implementors to provide not a cell but rather a packet level interface. The main idea will be presented by refering to Figure 3. The

processor is relieved from the execution of the SAR sublayer and the rate of the interlayer exchanges of PDUs is one or two orders of magnitude smaller. However major design problems are emerging. There is no flexibility to selectively employ or switch between different SAR types (1/2, 3/4 and 5) unless high complexity can be afforded. Buffer management for the segmentation and reassembly operations becomes a very difficult task, very often causing inefficient memory use. Finally, the resulting cell flow characteristics can neither be controlled nor monitored directly at cell level. The severity of this deficiency is currently overlooked, in the light of new requirements for ABR and charging control.

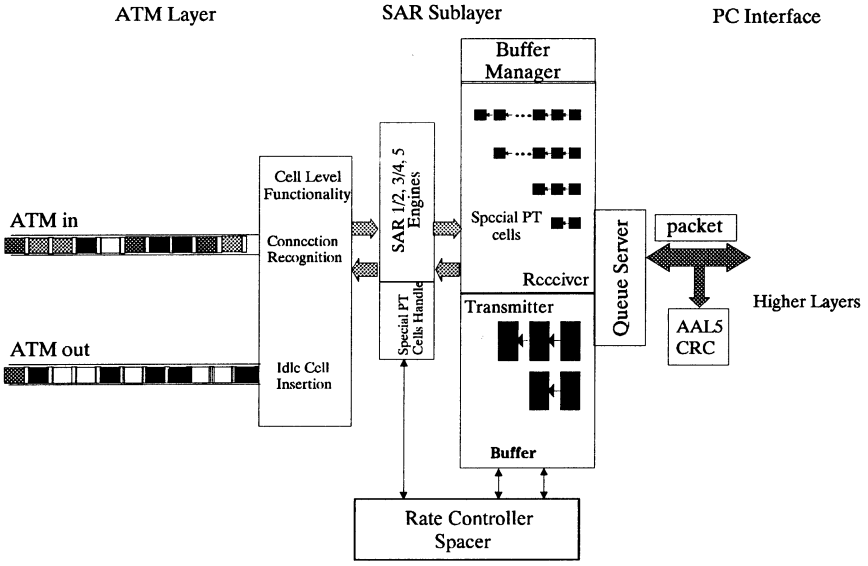

Figure 3: ATM Terminal Hardware Component Architecture.

Considerable hardware is needed for the implementation of all SAR sublayers. Among the standardized SARs, SAR type 3/4 is the most difficult utilizing a complex Finite State Machine (FSM), while SAR 1/2 used by isochronous services is delay critical. In the early ATM adapters the SAR sublayer was executed on-board by a dedicated microprocessor. This approach, although flexible, was neither efficient enough nor able to support a high throughput close to the ATM link capacity. This was not because of the μPs' speed but due to the non real time nature and the impairments in the low level control system. On the other hand pure hardwired logic based on reconfigurable components (i.e. FPGAs) can be used, thus providing the required flexibility along with high bandwidth. These components operate at frequencies comparable to the ATM parallel stream parallelized with 8 bits

(running at 20MHz). Furthermore, these components are reconfigurable and easy to reprogram, thus meeting the requirements of prototype systems.

Cells, arriving at the UNI, are multiplexed and belong to different connections. In addition cells of the same VP/VC channel may have different payload type identifiers (PTI) field thus requiring extraction from the main stream. The system must be able to distinguish the data from the other payload type cells and handle these by a different engine. An intermediate memory space is required to store the incoming SAR payloads for the formation of an integral packet (SAR-SDU). In the literature there are a few algorithms on how to manage the buffer in broadband components [8, 9 and 10]. On the transmitter side, we can use lists of pages instead of cell payloads. The main reason is that the flow control performed by the host is at packet level giving a simpler hardware implementation.

Considering the above implementation we show how a pipelined system can provide a throughput comparable to the ATM bandwidth at the point after the reassembly (packet level interface). It is envisaged that the PC interface must be fast enough to support the communication between the PC and the terminal hardware. Here a fast DMA transfer technique is realized in order to minimize the data copy overhead. A critical parameter is the data bus width and the interface standard utilized. The very first bus considered for peripherals was the 8-bit wide ISA interface that gives a performance of 8 x 8 Mbps=64Mbps (DMA performs at 8MHz), while for the AT 16-bit wide data bus the performance is doubled. Other buses like EISA or Local bus give higher performance and operate at 8MHz or more. The main drawback of these interfaces is that the utilization of the common bus decreases due to several idle periods, appearing while accessing the computer peripherals.

A major development in this area is the new Peripheral Component Interconnection (PCI) standard. This protocol provides a very high throughput in the Gbps range (1.056 Gbps or 2.112 Gbps for 32 or 64-bit wide bus respectively) performing at 33MHz. This high frequency poses new constrains in the implementation of the adapter. The DMA burst transfer requires a memory component able to perform at this speed. Since RAMs are difficult to find at these speeds, fast FIFO type memories are the next choice to support PCI burst transfers.

A PCI device can also act as a bus master for a burst transfer, and this transparently for the PC microprocessor. The design of Figure 3 can then be simplified using a segment of the main memory as the segmentation and reassembly buffer. Each time the ATM adapter has to write or read from memory a cell payload, it becomes bus master and makes a burst transfer in only a few bus cycles. For a 32bit (64) wide PCI device a cell payload can be transferred in 48/4=12 (6) cycles without the PC microprocessor realizing it.

3. EFFICIENT INTERLAYER COMMUNICATION

In this section we present the data structure and an associated conceptual procedure for a centralized software based implementation model of a protocol stack. This stack is supposed to involve successive segmenting (at the transmitter) and reassembling (at the receiver) functions. The ideas presented here are incorporated in the data controller specified in section 2.2, see also Figure 2.

3.1. Direct Access to the Payload

A general goal is to enable communication between adjacent layers through their common boundary in a way which avoids commonly encountered drawbacks. The first arises with physical copying in memory for each interlayer communication. Trying to remedy this with the exchange of pointers brings a second drawback. Information units relevant for each layer are placed at non contiguous memory locations. Either drawback leads to considerable performance loss. It is also evident that segmenting and reassembling are major functions contributing to the need of passing segments of higher (lower) layer SDUs to (from) a lower layer. The implementation model presented below avoids both drawbacks. Data units exchanged by the lowest layer are copied to memory only once, while the data unit part relevant for each layer is always to be found in contiguous memory. In particular, for the uppermost layer we obtain the full SDU of the entire stack at one contiguous memory block. Thus interchange with a peripheral device can be performed efficiently by a single DMA-like operation. Later on, Section 5 will show the considerable increase in performance due to these features.

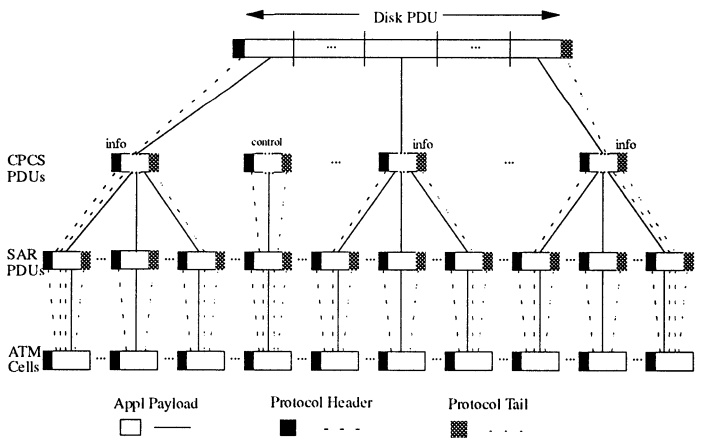

Figure 4: Header - Tail Composition Tree of Cell Train.

The above ideas can be presented in a general setting, but for our purpose, we investigate a layered stack consisting of ATM [11], SAR 3/4 (Segmentation and

Reassembly) [12], CPCS (Common Part Convergence Sublayer) [12] (sub)layers and a proprietary layer concerning the segmentation/reassembly of the disk file or in general the user data. For this particular case, Figure 4 shows the incorporation of upper (sub)layer (headers) tails in lower layer payloads. It is evident on Figure 4, which lower three layer protocols require header or tail according to I.363 [12]. The uppermost, fourth layer, is a proprietary, peripheral device related 'disk layer'. Thus the 'disk PDU' is the information unit supposed to be exchanged with the peripheral storage space through DMA. According to the specifications of the utilized protocols, the leftmost cell of the cell train is able to accommodate all headers of the higher protocols as shown in Figure 4.

In Figure 4 the decomposition of the user data into the final cell stream is shown. This cell stream is the result of the "multiplexing" of the PCIs and consecutive small segments from the application data. The "multiplexing" (on the transmitter) and "demultiplexing" (on the receiver) operations are performed by the data controller component (see Figure 2). The solid lines of the tree represent the contribution of the user data to the lower PDUs down to cell payload. The dashed lines show the header segment of the respective PCIs, while the dotted lines the tails. Going towards the lower layers, we see that every cell payload includes at least the PCI of the SAR sublayer. There are cells containing PCIs from more than one protocol layers (BOM or EOM SAR PDUs) while most of them contain segments of the user data and PCI only from SAR (COM type). This is evident since the Convergence Sublayer PDU is much longer than the SAR PDU, so there are much more COM than BOM, EOM cells. The SSM messages are not much interesting since they do not carry user data.

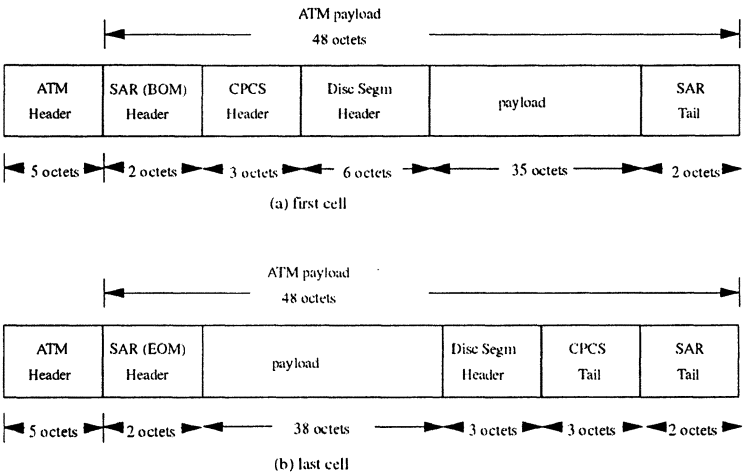

Figure 5. Most complicated ATM cells (first and last ones of the cell train).

The control subtree shown in Figure 4 represents the protocol control information exchanged in the inter layer communication (layer-2 communication in this case). The worst

cases of the cell stream (the leftmost and the rightmost cells) are shown in Figure 5. Here PCIs from the entire stack can be hosted within a cell. Moreover this is the most complicated and time consuming case for the data controller of Figure 2. When the terminal hardware component provides a packet level interface the same principles apply, modified for the remaining protocol stack above the SAR sublayer. The header-tail composition tree for the layered stack becomes much more complicated in cases where some higher layer header or tail extends over more than one lowest layer PDU. In our case the ATM cell size is large enough to prevent this from happening.

3.2 Operation on Common Buffer

In Figure 6 we give the necessary buffers and required pointers enabling the implementation of the stack protocols according to the guidelines previously outlined. Four buffers B(i) with length l(i) (i=1,2,3,4) are needed. Buffer B(4) is the only one of appreciable length at least for this case. Considering the receiver, the payload of each correctly received cell is successively stripped of parts belonging to the four layers. Buffers B(1), B(2), B(3) are reserved for the relevant PCI, while B(4) for the final application payload. Each PCI, for i=1, ... , 4, is designated as $H^k(i)$. This contains, if needed, header and tail. Notice that for B(2) (CPCS) and B(3) (disk-PDU), the header is received first and the tail follows later on a separate cell. Thus $h^k(i)$ denotes a part of $H^k(i)$ expected to be completed at some future cell arrival. SAR-PDUs correctly received have both parts of their PCI in the same cell.

The protocol state machines execute the relevant operation after the reception of the complete PCI of the appropriate message from their lower layer and internally store the required prehistory. Thus for the SAR-sublayer a very small, single PCI buffer is enough. The situation is different with B(2) (CPCS-sublayer). A CPCS PDU is translated over many cells. We need a write pointer w(2) which proceeds upon reception of a CPCS-PCI part through an arriving SAR-PDU. The entire CPCS-PCI is acknowledged by a separate pointer a(2) according to the verification of the window mechanism based protocol. The first CPCS-PDU is received and the corresponding PCI designated $H^1(2)$ is stored and acknowledged as valid in the buffer. The second has been received but not yet been acknowledged. The third CPCS-PDU is partly received (only header) and the write pointer w(2) points to the buffer space waiting for its tail. Buffer B(2) is shown for simplicity as a list. In reality it is a cyclic buffer with its length l(2) determined by the maximally allowed window size.

The acknowledgment pointer a(2) in effect designates that the space above its pointed address is occupied by some correct PCI which has been taken into account (see next layer below) and is free for reuse when w(2) reaches the bottom and rolls back on the B(2)'s top. The disk-PDU PCI buffer B(3) is again simple, since at the beginning (after the reception of the first ATM-cell) it hosts only the header, while the last ATM-cell conveys its tail. Pointer w(3) is then used to complete the storage of the entire disk-PDU PCI. At this point in time the whole disk-PDU will have been received, but notice that up to now we have only

described the storage of PCIs. The actual disk-PDU payload is in B(4). It sits at continuous memory ready to be transferred to the disk using DMA.

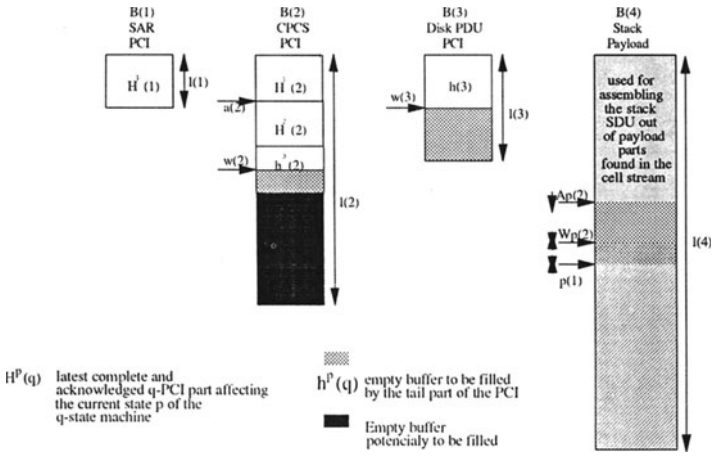

Figure 6: Memory Arrangement for Receiver or Transmitter.

A payload write pointer p(1) is used to point to the next write address throughout the whole disk-PDU reception procedure. Successive arriving, disk-PDU payload segments are stripped away from the SAR payload (found in its arriving ATM cell) and directly written in B(4), through use of the downward proceeding pointer p(1). Pointer Ap(2) keeps track of which part of the already written B(4) buffer space has been acknowledged by the CPCS-sublayer while the Wp(2) indicates the currently written layer 2 PDU. At incorrect receptions leading to retransmissions, the Wp(2) has to be retracked upward to Ap(2) depending on the retransmission mechanism. Pointer Ap(2) is always proceeding downward but never overtaking Wp(2). Similarly, Wp(2) never overtakes p(1) but in the case of retransmissions both Wp(2) and p(1) go back. The go back distance is determined by the length indicators of the contents of the moving window mechanism. Upon reception of the last cell p(2) and p(1) will point one address below the bottom of B(4). Upon configuration of the "disk" layer protocol on the basis of the disk-PDU PCI H(3) written in B(3) the whole B(4) content will be transferred through DMA to the disk.

We emphasize again here that the actual payload has been written *only once* into the system memory, directly placed to its destination upon its reception by the terminal board and through the host bus.

The transmission procedure works out in a similar manner and we thus have achieved the goals stated in the introduction of this section. The main memory is used only once for the information transfer between network and peripheral and any further unnecessary internal

transfers are avoided. Figure 2 shows this schematically, pointing out the bottleneck presented by the computer bus bandwidth resource. This bottleneck will be experimentally examined in Section 6.

4. PROTOCOL STACK MULTITASKING

Regarding the protocol stack the software has to support the hardware functions and implement all remaining functions across all layers. These layers are: Segmentation and Reassembly (if not performed by hardware), Common and Specific Part Convergence Sublayer (CPCS, SSCS). Signaling, application specific protocols, as well as management functions are conceptually provided in a separate plane. The software has also to support multiple connections and emulate parallel and independent execution of the protocol layers. In this section we describe mainly the kernel and data controller, whose role was identified in Figure 2 and explained in Section 3.

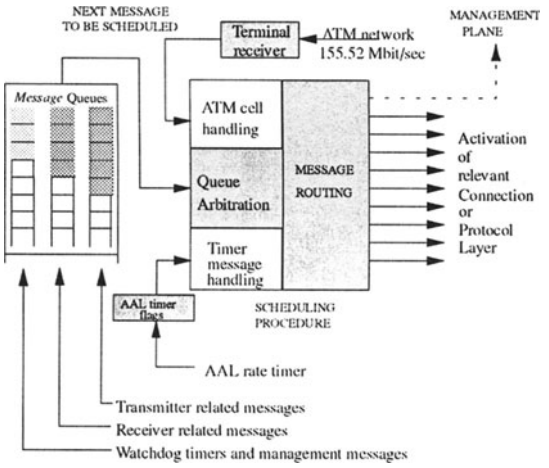

Figure 7: Scheduler and Message Router.

To support the multitasking operation, a scheduling entity has been realized. This scheduler provides execution independence among layers of the protocol stack, by arbitrating and handling the generated messages and events. Figure 7 shows a realization of the scheduling entity and routing of messages. A priority based queuing scheme separates the upstream, downstream, management and time-out events and messages. Both interrupt or polling driven methods are used. The figure shows how the scheduler handles the system queues (with three priorities) and polls the most frequent events (AAL rate timers and terminal receiver). The software is designed as a set of generic modules. Each module implements either a different layer of a connection (Adaptation, Transport layer, etc.) or different types of the same layer (the Adaptation layer type 3/4, 5). Combinations of these

generic modules can be used for the efficient implementation of various communication applications, as well as for testing purposes.

We thus work with a centralized application kernel without using the resources of the operating system. This is advantageous for a DOS application where the operating system does not provide any kind of event handling. In other systems (like OS/2, MS Windows, etc.) for PCs or native multitasking WorkStations, we can use functions provided by the operating system. Here special attention has to be paid for the matching of the operating system resources as these are generally provided to the specific application needs. Interprocess communication and synchronization, real time performance and multiprocessing are a few difficult issues that apply in the implementation of multimedia and multiprocess terminals [6].

5. AN IMPLEMENTATION EXAMPLE

We exemplify the above described ideas with an already implemented ATM PC Terminal Adapter known as PC Adapter (PCA) [13, 14]. The hardware module of the PCA (realized as a PC board) performs the functions of the Physical and the ATM layers, as well as CRC calculation and timer base provision for rate control. It is used mainly for ATM cell stream generation and termination, thus providing the host computer with a direct hardware interface towards the cell based network. The PCA hardware, realized as a PC board, consists of three functional blocks: a block for the Physical layer functions, a block for the ATM layer functions and a block which implements part of the SAR sublayer functions and the interface towards the PC data bus.

The first block utilizes the Line Terminator (LT) generic component. This block's functions involve physical medium adaptation, clock recovery, serial to parallel conversion, cell delineation and header error control. The second block is related to the ATM layer functions and has been built around the Generic Microprocessor Adapter (GMA) component. It performs ATM functions such as connection identification, rate decoupling and limited cell assembly - disassembly.

The third block implements SAR type 3/4 sublayer functions, which are the payload CRC calculation (both upstream and downstream) and rate control. Further, this block implements the necessary functions for the PC to interface - through the PC bus - with the generic components of the PCA. It utilizes the PC Interface (PCIF) component. Thus, well demarcated, delay critical and/or computationally intensive functions have been assigned to hardware, independent of their position in the communication stack. Efficient low level software integrates the board with the software. This module runs on the host (PC) and it has been realized as a set of low level functions in assembly language allowing direct control over the hardware features.

Both the per-byte and per-packet [7, 14] operations have been optimized as follows.

- *Optimization of per-byte operations*

These operations include the methods used for exchanging data messages among the protocol layers. The per-byte operations are implemented using pointers instead of data copies (according to Section 2) and performed by the data controller (Figure 2) explained in Section 3.

- *Optimization of per-cell operations*

These operations include the methods used by the software module for interaction with the hardware. Polling was used to determine the appearance of very frequent events (like the reception of a cell). This allows fast service time because delays associated with interrupt handling are avoided. Infrequent and asynchronous events (i.e. receiver buffer overflow) are signaled using a common hardware interrupt.

The software component of our implementation addresses two alternative protocol stacks involving several communication protocols. The protocols used were specified within the RACE 1022 project, following the ITU-T recommendations for the layers up to Layer 3 [12, 14]. In order to efficiently support different Applications (APL), all based on file transfers, a File Transfer (FT) layer has been internally designed with minimal complexity. An alternative stack is also available. It employs de-facto standard solutions based on a packet driver, built according to the FTP specifications. The same applications can be run on top of TCP and moreover well-known DOS or Windows based software can be supported (i.e. WWW).

The efficient and modular implementation of the lower level software enables one to host ATM particular features, which are indispensable for traffic and QoS management and monitoring. These low level, terminal based functions are increasingly required as standards and access modalities are specified. Such functions currently address

- burstiness - the number of cells transmitted consecutively (burst size) as well as the interburst duration.
- rate - the time distance between two consecutive occupied slots.
- packet size - the size of the ATM Adaptation layer or the TCP PDU.
- the impact of various bit error rates to the performance has been measured using an error injection function.

In addition to the above, new requirements emerge and should be supported.

- charging - operations based on occupied/unoccupied cell counting to be used as terminal internal estimation or verification of the cost incurred.

- support of ABR - automatic and periodic rate decrease and then restoration of the initial cell rate upon arrival of an FRM cell (terminal end-to-end flow control scenarios [15]). Recognition of congestion and issuing of FRM cells on the receiver side .

6. MEASUREMENTS

During the implementation of the PC Terminal Adapter, several approaches in the hardware and software configurations have been tested to achieve throughput optimization. These different implementations are presented along with measurements and conclusions. The scenarios used in the performance tests, as described below, involved measurements in different transfer rates and in some cases selective incorporation of several low level features as described at the end of the last section. In particular, in order to efficiently emulate network errors, the intentional injection of errors at the transmitter side has been involved.

The following subsections address individual issues of interest either to the internal terminal design options, or to its performance depending on exogenous, network dependent parameters.

6.1. Efficient Interlayer Data Transfer

One of the main issues addressed in this paper is the data movement optimization through the protocol stack. In order to transfer the protocol data between the protocol stack layers (per-byte operations [14] as described in Section 2), two implementations have been used. The first is transfer by using data copies. In this implementation each time a PDU is transferred between two protocol layers, the information contained in the PDU of the source layer is copied in the PDU of the destination layer. Thus, each protocol layer has access only to the information that is related to it, without interfering with the information of other layers. The second follows the ideas on efficient interlayer communication as given in Section 3.

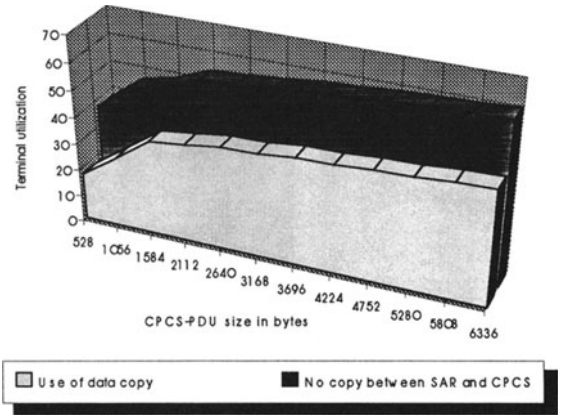

Figure 8: Data Copies vs. Pointers between two Adjacent Layers.

In Figure 9 we show the effect on the terminal utilization. This is the terminal hardware utilization with respect to the packet (CPCS-PDU) size.

As it is shown with data copies the terminal software is able to provide only 35% utilization of the hardware while without data copies between the first two layers (SAR and CPCS) the terminal operates at 60% of the hardware bandwidth. It is evident that the throughput has been approximately doubled.

6.2. Use of Multiple Cell Transmission on Each Timer Rate Tick

According to the ITU-T Recommendations [12] the SAR type 3/4 generates constant bit rate traffic. The PCA software uses a real time rate controller to make sure that the traffic generated by the SAR layer follows the traffic profile. The software also uses another special parameter, which defines the number of ATM cells that will be transmitted each time the rate timer expires. This parameter defined as "cellburst", represents the number of SAR PDUs to be transmitted at each transfer-rate-timer expiration. The product "cell burst" x " duration between rate timer expiration" remains constant. Measurements have been taken for two different CS PDU sizes. We can notice in Figure 9 that, as the cellburst value increases, the throughput reaches an upper bound and then drops in a repeated fashion.

The reason for this behavior is that by increasing the cellburst, the time between timer ticks, during which the SAR layer is waiting to send the next cell, decreases. This can lead to situations where the SAR layer is activated, while the segmentation procedure is still incomplete and no cell is ready for transmission. This causes the decrease in the effective rate that is depicted in Figure 9. In other words we have to harmonize the internal protocol process with the statistical behavior of the cell traffic load offered to the network.

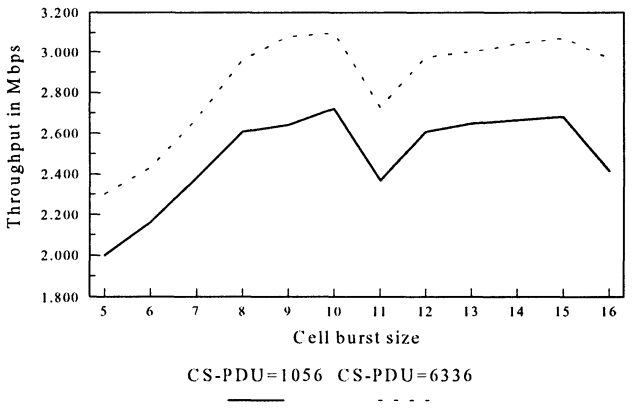

Figure 9: Multiple Cell Transmission per Time Slot.

6.3. Hardware CRC vs. Software CRC calculation

As mentioned is Section 2 the CRC calculation (in software) causes a considerable throughout decrease since it requires an additional data copy in addition to the burden of arithmetic operations. Software CRC calculation is more flexible for future modifications. With hardware CRC calculation we found a throughput increase equal to 30% of the overall performance. The operation is performed on the fly, as the cells are copied into the transmitter buffer.

6.4. Event notification: Polling or Interrupts

Upon arrival of an ATM cell, the insertion of this cell in the protocol stack (per-packet operations [14]) can be performed following two modes. In the interrupt mode, the cell is inserted in the queue as a queue message by using an interrupt service routine. The message which contains the cell's information is forwarded from the queue to the SAR sublayer. In this implementation each cell is inserted in the queue as soon as it arrives in the PC Terminal Adapter. During high rate periods the large number of interrupts lowers the software throughput due to the delay imposed by the interrupt service routine. In the polling mode, instead of inserting each cell in the queue, a software flag notifies the scheduler that the PCA board buffer contains at least one ATM cell. This flag is periodically polled by the scheduler and in the case of activated flag, the contents of the PCA board buffer will be directly forwarded to the SAR sublayer without the use of the queue. This mode has been proved to be much faster than the previous. Several measurements that have been performed to test the difference in speed between the two modes indicate a difference of about 80 Kbytes/sec at a maximum rate of approximately 330 Kbytes/sec.

Similar behavior can be also found in the transmitter. The mechanism that notifies the scheduler that a rate timer has expired, involves a timer expire message which is inserted in the message queue. This message is inserted by using an interrupt service routine similar to the one that has been used for the insertion of an ATM cell in the message queue. Note that, the problems of the implementation of the previous paragraph exist also in this case. Therefore, a second implementation which uses flags to notify the scheduler regarding the expire of a rate timer has been realized. In this implementation each time a rate timer expires, a timer flag which is periodically polled by the scheduler is set. The time for setting and resetting these flags is less than the time required for inserting and extracting a message from the queues, leading to a significant throughput improvement especially in high rates.

6.5. Emulation of ATM bit error rate

Due to the fact that the transmission line used in order to develop and test the PCA software is error-free, the bit error rate inserted by an ATM network is emulated. For this reason erroneous cells have been created randomly, in a rate equivalent to the bit error rate of ATM networks.

The measurements in Figure 10, show the performance of the software under the conditions described above for various values of the CS-PDU. The measurements have been taken for error-free transmission and for transmission using bit error rates equal to 10^{-9} and 10^{-8}. All measurements have been taken for various CS-PDU sizes. Figure 10 indicates that

for transmission that is not error free, the throughput reaches a peek value for a certain CS-PDU size and then starts to decrease. This is due to the fact that larger CS-PDU size values reflect to larger units of information that are corrupted by an error. The smaller relative PCI overhead of larger CS-PDU is offset by the bandwidth waste caused by the size of the retransmitted unit. For higher values of the bit error rate the peak transfer rate is achieved for smaller values of the CS-PDU size.

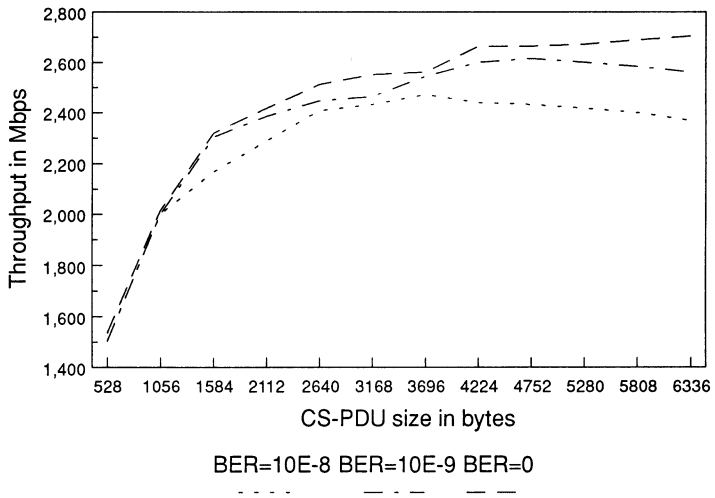

Figure 10: Bit Error Emulation.

6.5. Variable packet size

Measurements in this section utilize various CS-PDU sizes, to examine the throughput of the module for each value. In Figure 11 the throughput is almost linearly related to the CS-PDU length until this reaches a certain length. This upper CS-PDU limit value depends on the processing power of the PC that hosts the PCA. The reason for this behavior, is that in the case of a low processing power PC, the time spent for segmenting and handling the data of a big CS-PDU puts an excessive burden on the scheduler's operation and saturates the efficiency improvement in the file transfer procedure. In the left part of the Figure 11 we see that the bottleneck is the communication component (protocol stack and the network), while on the right side it is the processing power of the host computer. On the same curve we see the performance of the TCP transport protocol over ATM. It is evident that the more broadband oriented internal 'file transfer' protocol directly on top of the AAL is much more efficient than the insertion of the full TCP protocol.

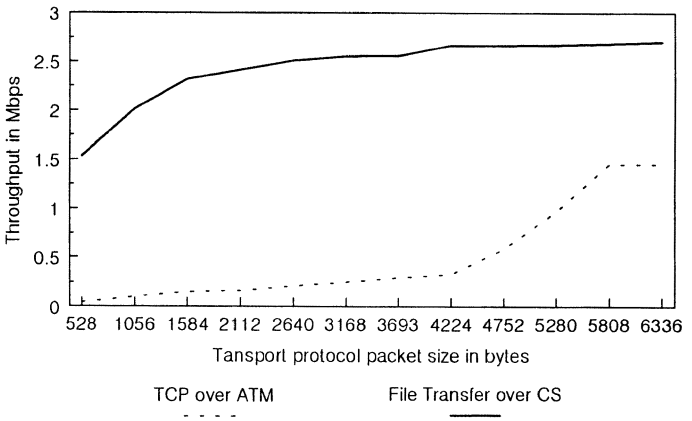

Figure 11: Variable Packet Size in CS and TCP.

6.6. Protocol Stack Alternatives

It is interesting to compare different alternative protocol stacks in the example of file transfer. The protocol stack internally specified in RACE 1022 project for minimal complexity, achieves throughput after interprocess optimization of up to 410 Kbytes/sec while the throughput of the TCP/IP based file transfer is up to 220 Kbytes/sec end-to-end. It is evident that the first is broadband oriented and that the second more data network oriented (Ethernet).

A third alternative, represented by a broadband transport protocol, XTP described in [16] has been tested. The results were disappointing, giving only 90 Kbytes/sec. This shows in our view that, as experienced with TCP/IP's long history, implementation dependend optimization is a long and difficult task, depending on long term acquaintance and on hand experience. Thus efficient XTP implementations have not appeared yet and proven solutions like TCP/IP are preferable, despite the fact that their design objectives had initially not covered broadband networks.

7. GENERAL TERMINAL REQUIREMENTS

Figure 12 intends to show a macroscopic comparison between packet and streaming application. At the right the ATM communication subsystem offers its resources in the form of bandwidth. We show schematically the policed bandwidth as well as the unpoliced one allowing the full UNI rate of, say, 155 Mbps. However an unpoliced VC is unthinkable for a real network. In both cases terminal internal resources are dynamically varying due to other processing needs. In the average, these always have to provide a "bandwidth" pertinent for communication layered stack processing below that of the ATM subnetwork and above that imposed by the application (goodput).

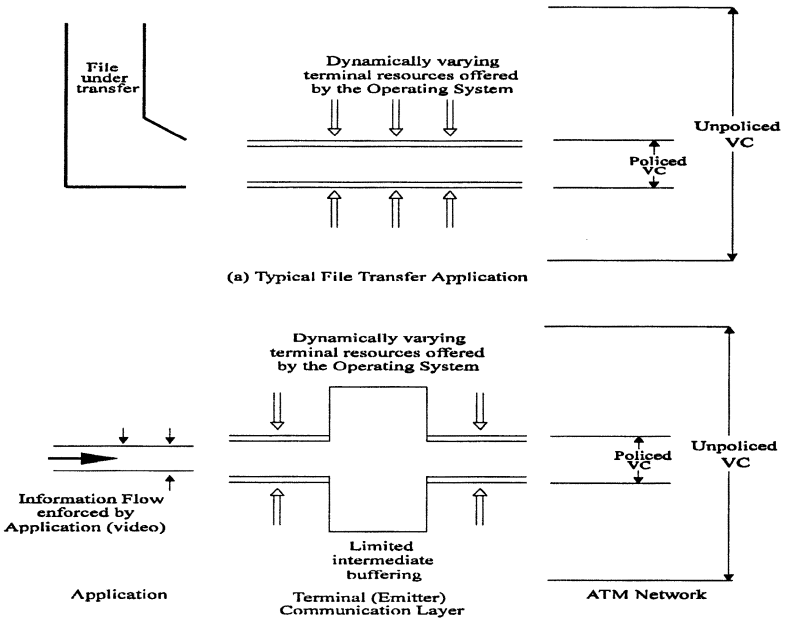

Figure 12: Alternative Terminal Requirements.

However only in the video application (Figure 12.b) is there such an imposition. With file transfer (Figure 12.a) infinite buffering within the application is an appropriate conceptual idealization, so that the effective overall throughput is determined by the bottleneck, that is the terminal resources offered for the communication layered stack. With video the information flow from above towards that stack is imposed and fluctuations either in this flow (of very limited magnitude) or in the communication stack resource availability have to be smoothed out by internal (e.g. interlayer) buffering. It is at least qualitatively evident that the matching between resources at the upper and lower boundary of the communication stack (Figure 12.b) is critical for the video application.

8. CONCLUSION

In this paper we have considered in general terms the important, and sometime overlooked, role that terminals will play in terms of QoS and throughput in broadband services over ATM based B-ISDN. Based both on general considerations and a specific realization, we have shown that high end terminals can be achieved out of ordinary computers (PCs) with minimal hardware add-ons. For this purpose we have given in some

detail a general concept on how to handle interlayer communication, as well as an internal software architecture appropriate for the task.

Furthermore, the architecture proposed in Section 2 is able to support real time broadband applications without expensive and dedicated hardware developments. Software implementation of the utilized protocols is very flexible since we can change without much effort the protocol layers involved in a specific application. The latter is very useful for the optimization of the protocol parameters in order to achieve maximal throughput. In addition this can aid at the implementation and the evaluation of new high speed broadband protocols by using a very flexible and high performance environment able to offer the required load.

From the tests and measurements that have been studied and presented in this paper we can conclude the following: The implementation of selected functions of the lower layers of the protocol stack in hardware (i.e. implementation of the CRC calculation over the cell's payload in SAR type 3/4) results in significant improvement on the aggregate throughput. An increase of the size of PDUs of the layers above SAR (i.e. CS layer, Layer 4) as well as an increase of the memory buffers also leads to a throughput improvement. However, in the case of errors in transmission, this improvement reaches an upper bound for certain PDU sizes which depend on the error rate. Further increase of the PDU's size above this upper bound will result in a reduction of the effective transfer rate. This is due to the increase of the retransmitted information, which is large enough to overcome the benefit of a lighter load for PCI processing.

The processing power of the host PC affects the transfer rate. In the case of the use of hard disk in applications such as bulk data transfer, the disk access time is a critical parameter of the throughput. Improved performance can be achieved by changing the traffic parameters such as the cellburst length. Transfer rates over 300 Kbytes/sec can be achieved with low cost hardware - such as a 386 PC - and no special requirements on its configuration.

REFERENCES

1. Jean-Yves Le Boudec, "The Asynchronous Transfer Mode: a tutorial" Computer Networks and ISDN Systems, Vol 24, Pages 279-309, 1992.
2. ITU-T Study Group XVIII - REPORT R 34, "General B-ISDN Aspects", Geneva, May 1990.
3. IEEE Communication Magazine, Special Issue on "High speed network protocols", June 1989.
4. IEEE Network Magazine, Special Issue on "End-System Support for High Speed Networks", July 1993.
5. IEEE JSAC, Special Issue: "High Speed Computer/Network Interfaces", February 1993.
6. V. Trecordi, "Inter-Process Communication Performance in High Speed Networks", IEEE Int. Conf. on Communications, May '94, New Orleans, pp. 57-63.

7. P.A. Steenkiste, B.D. Zill, H.T. Kung, S.G. Schlick, "A host Interface Architecture for high speed networks", Proc. of 4th IFIP Conf. on "High Performance Networking", Sept. 92, Belgium.
8. G.E. Konstantoulakis, D.I. Reisis, "Real Time Buffer Management for High Speed Broadband Networks" to be presented in IEEE COM CON 5, June 1995.
9. C.R. Kalmanek, S.P. Morgan, and C. Restrick, "A high-performance queuing engine for ATM networks", ISS '92, October '92, Vol. 1, A4.1.
10. E. Wallmeier and T. Worster, "The spacing polisher, an algorithm for efficient peak rate control in ATM networks", ISS '92, October '92, Vol. 2, A5.5.
11. ITU-T SG XVIII - Rec. I.3621, "B-ISDN ATM layer specification", June 1992.
12. ITU-T SG XVIII - Rec. I.362,3, "B-ISDN Adaptation Layer specification", March 1992.
13. Gr.A. Doumenis, G.E. Konstantoulakis, Ch.Z. Patrikakis, V.J. Tzerpos, D.I. Reisis, "Protocol Measurements using Terminal Adapter for the ATM B-ISDN". in proc. of IEEE COMCON 4, June 1993.
14. Gr.A. Doumenis, G.E. Konstantoulakis, D.I. Reisis, G.I. Stassinopoulos, "A Personal Computer Hosted Terminal Adapter for the Broadband Integrated Services Digital Network and Applications". in proc. of IEE, 2nd Int. Conf. on "Broadband Services Systems and Networks", Brighton, UK, Nov. '93.
15. ATM Forum, "B-ICI" Specification, Version 1.0 September 1993.
16. XTP Protocol Definition, Revision 3.6, Janualy 1992, Santa Barbara: Protocol Engines Inc, 1992.