Experimental ATM Network Interface Performance Evaluation

Apostolos Dollas* Kyprianos Papadimitriou Constantinos Mathioudakis
Department of Electronic and Computer Engineering
Technical University of Crete
73100 Chania, Crete
Greece
Evangelos Markatos† Manolis Katevenis†
Institute for Computer Science
Foundation of Research and Technology - Hellas
71110 Heraklion, Crete
Greece

Abstract

Traditional methods of measuring network interface performance have been based on throughput-sensitive benchmarks. However, the performance of an ever-increasing number of applications depends on the latency*, and not on the throughput of the underlying communication network system. Experimental performance evaluation of latency and its constituent parts for high speed network interfaces of personal computers has not been studied in depth to date. We have developed performance evaluation methods that derive detailed PCI to ATM network interface latency results for personal computers, based on a combined software-hardware cooperation. Our methods address the seamless data collection of events that have six orders of magnitude difference in their duration, and range from user level requests down to segmentation latencies for packets within the network interface. As a result, all steps which contribute to the total transmission latency have been accurately measured for a specific network interface type. Experimental results derived with the new method are presented, and its adaptation to vendor-independent measurements is described. Our experience suggests that measuring network interface latency using our methods is entirely feasible with equipment normally found in digital design laboratories. Furthermore it may lead to a deep understanding of the underlying communication system, which would be difficult (if not impossible) to acquire only with throughput-sensitive benchmarks.*

KEYWORDS: Network Interface, PCI, ATM, Performance Evaluation, Experimental

1 Introduction

High performance networking has entered the mainstream of the workspace computing environment. With the proliferation of high speed networks such as ATM and new services employing them, it is important to evaluate the performance of network interfaces in order to achieve better quality of service, and locate bottlenecks in the transmission process. Some issues that complicate this task are:

- the plethora of factors affecting network performance,
- the multitude of steps that comprise the transmission of even a single byte of data,

*Also with ICS-FORTH.
†Also with the University of Crete.

- the substantially dissimilar results from systems with seemingly minor variations, and,
- the dependence of performance metrics on the user application and needs.

To illustrate these issues, some of the factors affecting the performance of networks include the speed of the CPU and the method with which the operating system translates user requests to network transmissions. It is desirable but not always possible to bypass the operating system, and significant work has been done to accomplish this goal [12, 1, 2, 7, 8, 11]. Two additional factors affecting network performance are the architecture of the system (e.g. whether network operations are mapped to I/O or to memory, the bus on which the network interface is connected), and the network protocol that is employed. Some of the steps involved in the transmission may be a system call from a user's application, invocation via some device driver of the network protocol, access of the network interface, conversion of the data into packets, and transmission thereof with additional information such as error checking codes. Network performance strongly depends on seemingly minor factors. For example, the exact same configuration of a hardware platform performs substantially differently if only different drivers are used in the system. Lastly, application characteristics also influence what is important and what is not: a networked server for large file transfers will behave differently from a system that handles many small interactive transactions, such as a web server.

Two metrics associated with network performance are the *bandwidth* or *throughput* and the *latency.* The bandwidth is associated with the aggregate capacity of the network and corresponds to total information flow rate, whereas the latency is associated with the time to transfer one packet of information. Depending on application, the metric of importance varies: for a multimedia application transferring video over a network, the bandwidth is of primary importance, and the initial delay of a few milliseconds until the stream reaches its destination is tolerable; on the other hand, in multiprocessor synchronization, a single token passed among processors determines the continuation of execution, and a few millisecond delay may mean that the time for over one million instructions is wasted.

In general, bandwidth is easier to measure because a long transmission can amortize startup effects. Bandwidth can be adequately measured with software based methods (e.g. the Netperf tool which can be obtained from `http://www.cup.hp.com`). Latency is more difficult to measure because it requires simultaneous monitoring of multiple subsystems and in general requires hardware based methods (e.g. fast oscilloscope or logic analyzer) [6, 13].

Performance evaluation of computer networks is an important task because it provides valuable data with respect to actual vs. ideal performance, as well as insight into the reasons for any delays. In recent years new architectures ranging from the low level protocol itself (e.g. ATM) to the processor (e.g. Pentium II 266MHz) have emerged. Despite the great number of high performance network products that have emerged on the personal computer market, very little has been reported in the literature on PC systems [1, 3]. Indeed the publications to date are largely on RISC computer architectures with the Unix operating system which dominated until now the high-performance networked computer market [6, 4, 10, 9, 5]. Many of the papers on Unix systems report studies which include switches or routers.

The purpose of this work was to develop performance evaluation methodologies for personal computer network interfaces, and use them to evaluate in depth one ATM network interface for the PCI bus. The work was performed from May, 1996 through September, 1997, and therefore the actual computers involved in our experiments are not high performance by today's standards. However, the main contribution of the paper is not the measurements themselves (which are bound to become obsolete no matter how recent they are), but the methodology used to take the measurements which uses a novel software/hardware cooperation. The development of our methodology, allows for performance evaluation of PCI-to-ATM network interfaces on any kind of personal computer. Furthermore, our use of the logic analyzer allows for a breakdown of the transmission into its constituent parts, which have not been studied in detail to date. In order to isolate the effects of the network interface itself, the measurements were made for point-to-point connections, with no switch in between the observed systems.

The rest of the paper is organized as follows: section 2 describes our experimental setup and the transmission characteristics that we measure. Section 3 presents in detail our methods. Section 4 contains experimental results and is followed by a brief section with conclusions from our work. One forewarning to the reader is that the network community at large has specific names and acronyms. We have maintained these names and acronyms in order to present a realistic picture, but in order to make this paper readable we have included two sidebars, on protocols and on the PCI bus, introducing most of the arcane nomenclature; the remaining terms are introduced within the paper as they appear. **NOTE TO REVIEWERS: This paper can be revised without major loss of its key contributions, such that the nomenclature and acronyms can be substantially reduced. We elected not to do so in order to maintain a realistic picture - your feedback on the readability of this paper is appreciated**.

2 Experimental Setup

The setup for our experiments was comprised of two PC computers, one 133MHz Pentium based system and one 100MHz Intel 486 DX/4 based system. Both computers had 32Mbytes of RAM. The choice of two different computers was deliberate, since we could measure how the processor speed difference affected the performance of the network interface cards (NIC). We used the Integrated Device Technology (IDT) network interface IDT77901, which runs on PCI v.2.16 bus and has 155Mbps ATM outputs on optical fiber transmission medium. The Windows NT 4.0 operating system with the IDT supplied drivers were used. The Windows NT was found to be a robust and reliable operating system, and it was observed to be faster than Windows '95 too, in accordance with previously published results [3]. The organization of the NIC, the PCI bus and the host system are shown in Figure 1.

2.1 The Network Interface Card

The methods and measurements described in this paper were made on the IDT77901 card, but they can be thought of as vendor independent for PCI to ATM network interfaces. Practically all 155Mbps ATM cards use the UTOPIA bus, and their architectural differences are centered on whether they support on-board SRAM or the host's memory is used, how the FIFO's and control registers are structured, and of course how fast the vendor-specific integrated circuits perform the segmentation and reassembly of packets.

The main component of the network interface is the IDT77201 NICStAR integrated circuit which is an ATM Segmentation and Reassembly (SAR) controller. As shown in Figure 1, the SAR controller accesses directly the PCI bus; it is also connected to four 32K×8 15nsec SRAM integrated circuits, and to the ATM physical layer (PHY) integrated circuit IDT77155. The PHY chip is connected to the SAR controller via three separate busses, two UTOPIA busses for transmitted and received data, plus a control bus called the utility bus. In addition, the PHY drives directly the Hewlett Packard HFBR-5103 Optical Data Link chip on which the pair of optical fibers are connected for full duplex transmission. The general organization of the IDT77901 network interface is fairly typical of its kind and therefore its study can be fairly readily adapted to other network interfaces.

During operation, the data originally exists in a buffer up to 64Kbytes, located in the host's main memory. This buffer corresponds to the largest data transmission supported by AAL5 (see sidebar on protocols) and is called CS-PDU (Convergence Sublayer-Protocol Data Unit). During the initialization of a transfer, i.e. when the VPI and VCI are determined, the corresponding information is placed on the SRAM of the NIC. Subsequently, data is transferred from the CS-PDU to the SAR via the PCI bus. Within the SAR, the data gets assembled into ATM cells which are then transmitted by the SAR via the TxUTOPIA bus to the PHY physical layer integrated circuit. The PHY chip serializes the data transmission from the 8-bit TxUTOPIA bus to the physical link (which can be SONET or SDH), adds the 5-byte ATM header, and with a 19.44MHz clock produces the 155.52Mbps serial data transmission ($155.52 = 8 \times 19.44$).

The reverse process is also supported, i.e. reception of serial data by the PHY chip, transfer to the SAR via the RxUTOPIA bus to the SAR and buffering of the data into the SAR on-chip FIFO memory, stripping of the header, entering the data at the appropriate SRAM buffer, and passing the data via the PCI bus to the host. In terms of control signals (which are used to determine status of the cell transmission), the control signal TSOC indicates when a new cell is transmitted from the SAR to the PHY; TWRENB# indicates the entry of a new cell into the FIFO of the PHY. In the opposite data transfer direction, the RSOC signal from the PHY to the SAR indicates that a received cell is passed to the SAR, and the RRDENB# signal from the SAR to the PHY indicates that the SAR can read data from the incoming FIFO of the PHY chip.

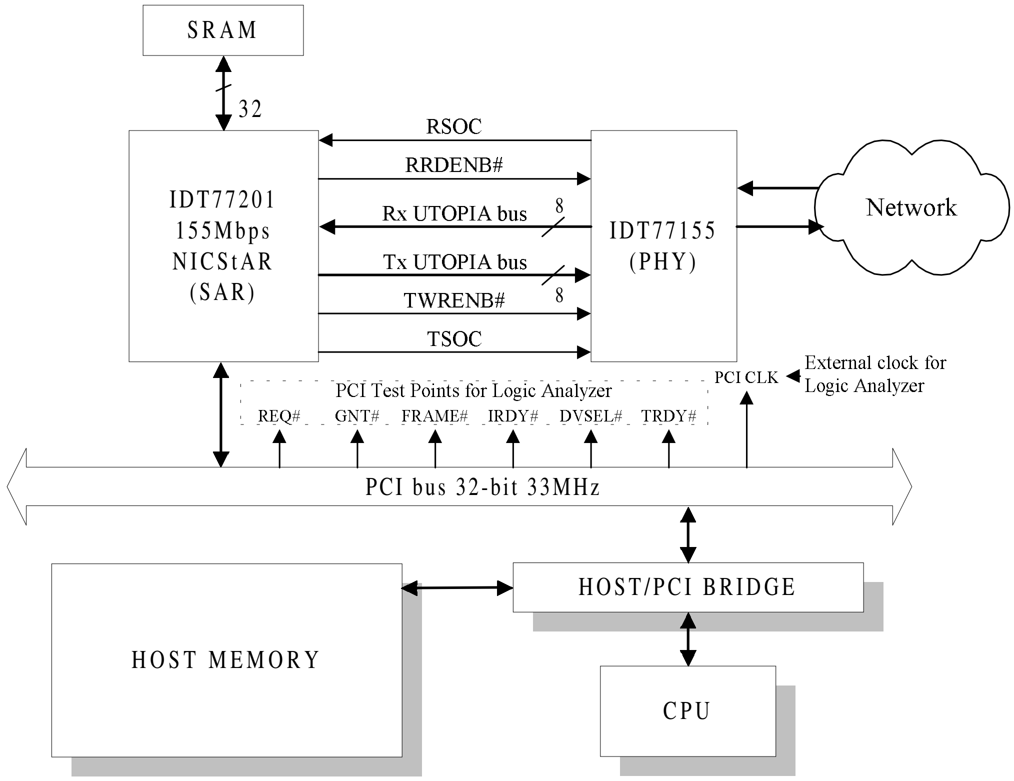

Figure 1: Organization of the IDT77901 Network Interface

2.2 Latency Breakdown

In order to gain complete understanding of the delay within a network interface, we need to define what are the latencies that add up to it. Figure 2 shows a simple client-server model, in our case two personal computers. The arrows with names in the figure correspond to the processes that the network interface performs. For one-way data transmission, these are the following:

- PCI Bus Access Latency (NIC Initiator): is the time to transfer data from the host to the NIC card, and can be determined by observing the PCI bus signals *TDRDY#* and *REQ#*. This latency is affected by the processor speed, the overall load of the PCI bus, and the design of the NIC (if it inserts wait states).

- SAR Segmentation Latency: is the time to perform segmentation of the packet transferred via the PCI bus from the CS-PDU into ATM cells. It can be observed by monitoring the delay from the PCI bus access to the TxUTOPIA bus *for the same packet*.

- Serialization and Transmission Latency: is determined from the time a packet enters the TxUTOPIA bus of the sender (e.g. PHY1) until it emerges at the RxUTOPIA bus of the receiver (e.g. PHY2). A latency breakdown for the actual physical transmission would require optical probing and its synchronization with the logic analyzer, but since the speed of light is known for optical fibers this latency can be calculated rather than measured. In our experiments it was less than 20nsec and could be safely ignored.

- SAR Reassembly Latency: is determined from the time data crosses the RxUTOPIA bus until the SAR is ready to place it on the PCI bus by asserting the *REQ#* signal. This time includes the latency to enter a cell in the "receive cell FIFO" (RCFIFO), as well as the update of the receive connection table on the SRAM of the SAR. In case the received cell is not the first in a packet, the receive connection table is checked rather than updated.

- PCI Bus Access Latency (NIC Target): is determined similarly to the PCI Bus Access Latency of the NIC Initiator.

In our experiments, the server echoes the data back to the client, with a Software Echo Latency (SEL) which largely depends on the processing power of the server, the protocol used, and the load of the processor. The method to measure the SEL time will be presented in Section 3.

The remaining components in Figure 2 are shown as clouds, to represent the cloudy indeed situation that corresponds to the Software Transmission Latency (STL) and Software Reception Latency (SRL) from/to the user's application to/from the PCI bus. In many respects these times are more interesting than the corresponding times of the network interfaces, because they come from complex behavior of many components, and are much higher than the times we measured (hence, the true bottleneck in transmissions). The factors affecting STL and SRL are the user level system calls, the relatively slow software execution of protocols such as TCP (which often and for good reason introduce new idle delays and hence latencies of their own), the architecture of the processor, and its load.

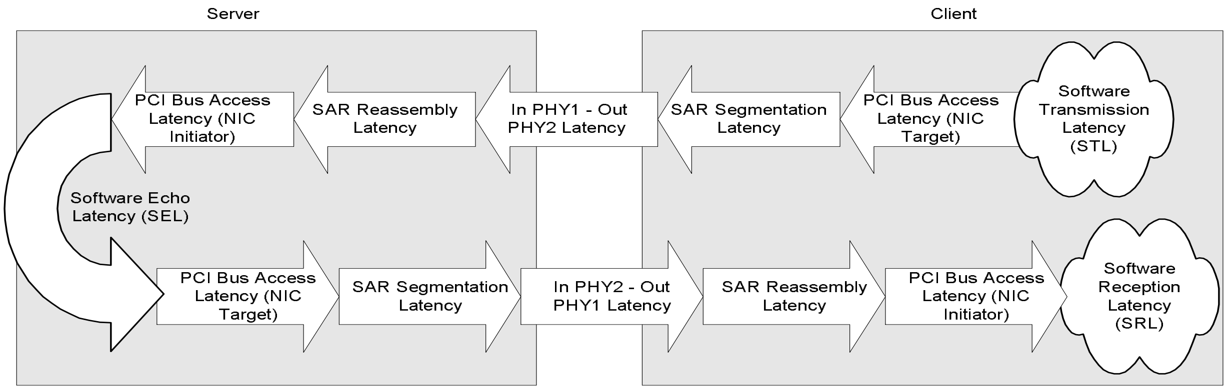

Figure 2: Network Access Latency and its Components

The reverse process has corresponding times, as shown in Figure 2.

2.3 The Logic Analyzer

The usefulness of a logic analyzer is its ability to accurately resolve time down to a few nanoseconds, and to monitor simultaneously dozens of binary signals. We used the Hewlett-Packard 1663A logic analyzer, which has 34 channels, it can resolve time (depending on the mode used) down to 4 nsec, and can store up to 8K words of data. The above features, however, are not sufficient to perform the task at hand, otherwise the data collection would be a trivial exercise. The main challenges come from the limitations in the number of channels and in the memory depth of the instrument. To illustrate, in order to capture all major signals (including data values) on the PCI and the UTOPIA busses of the server and the client, over 130 signals would need to be monitored. Even a large number of channels and a deep memory, however, would not solve all our problems, because in that case huge amounts of data would be produced during each run. With a total of over 2,000 runs performed for various experiments, including multiple runs for each set of parameters, it is evident that a better methodology was needed, rather than brute force.

The continuous data collection mode of the logic analyzer, called "timing analysis" was invaluable in providing detailed snapshots of various aspects of the operation. These were in turn used to determine the "big picture" which was captured in the "state analysis" mode. State analysis allows for up to 8K states to be stored in the logic analyzer's memory, even though a great number of cycles might take place between successive states. A finite state machine (FSM) controls the transition of the logic analyzer from state to state. Our methodology is largely based on the mapping of various events, some of which take a few nanoseconds and some of which take a few milliseconds into a FSM which captures the entire process. The economy of event representation into states was also of importance, as several multi-step capture sequences exceeded the ability of the instrument to store them.

With the 34 channels of the logic analyzer and a appropriate structure of the data to be transferred, only a small number of data lines needed to be monitored, whereas six control signals for each side of the PCI and the UTOPIA bus control signals adequately produce the piece-wise latencies. Of great value was our ability to initialize and arm the logic analyzer from software events. Any kind of scheme can be used, but in our case the serial port of the PC's was used.

2.4 Experiment Parameters

Experimental work is tricky in general. It is possible to have irreproducible results because they depend on factors that cannot be controlled. To illustrate, experimental performance evaluation of network interfaces would be most interesting if it were performed on systems having a normal load of applications. In this case, however, two systems with *the same* CPU load could exhibit substantially different network behavior because one might have few PCI bus transactions whereas the other may be I/O intensive and thus allow for little bandwidth for the network interface. For this reason there were no other PCI cards present in the bus, other than the network interface, and there were no user applications other than the ones developed for the actual performance evaluation.

Few results are presented in detail in this paper, despite a broad number of results that were derived. Results were derived for one way data transmission of varying lengths which were very useful in showing some protocol-related performance data, echoed back data transmission also of varying lengths, with either the 486/100 or the Pentium/133 serving as the host, and for different protocols (mostly TCP/IP and UDP/IP). The methods are of more general usefulness than the specific numbers which could fill pages upon pages of data regarding obsolete by now machines.

The last parameter affecting the experiments was the statistical collection of data. For every set of the experiment parameter values, a minimum of ten runs (and as appropriate, many more) were performed.

3 Hardware-Based PCI-to-ATM Performance Evaluation

3.1 Experiment Methodology

The previous sections largely indicate which signals are involved in the piece-wise latencies. Based on the above, the patient reader would be able to reproduce our results after a few months of experimentation. This section aims at the presentation of the specific methods used and how they were derived. The logic analyzer control FSM in Figure 3 is not the only one we used, but it encompasses the full complexity of measurements from the time the user application initiates an experiment up to the time that received data reaches a user application. Before this experiment was derived, a large number of experiments with the logic analyzer in timing mode was performed in order to characterize the piece-wise latencies. Figure 3 shows the FSM which controls the logic analyzer, and Figure 4 shows a corresponding trace. It is interesting to note that the relative times of signals in Figure 4 do not correspond to the physical time scale of these signals due to the sampling in state mode. Each state sampled takes one logic analyzer memory word, and if two states are sampled successively after a long wait period the corresponding signals will appear adjacent to each-other. The annotations in Figure 4 demonstrate which state of the FSM corresponds to each transition, and the associated logic analyzer produced times. To illustrate the

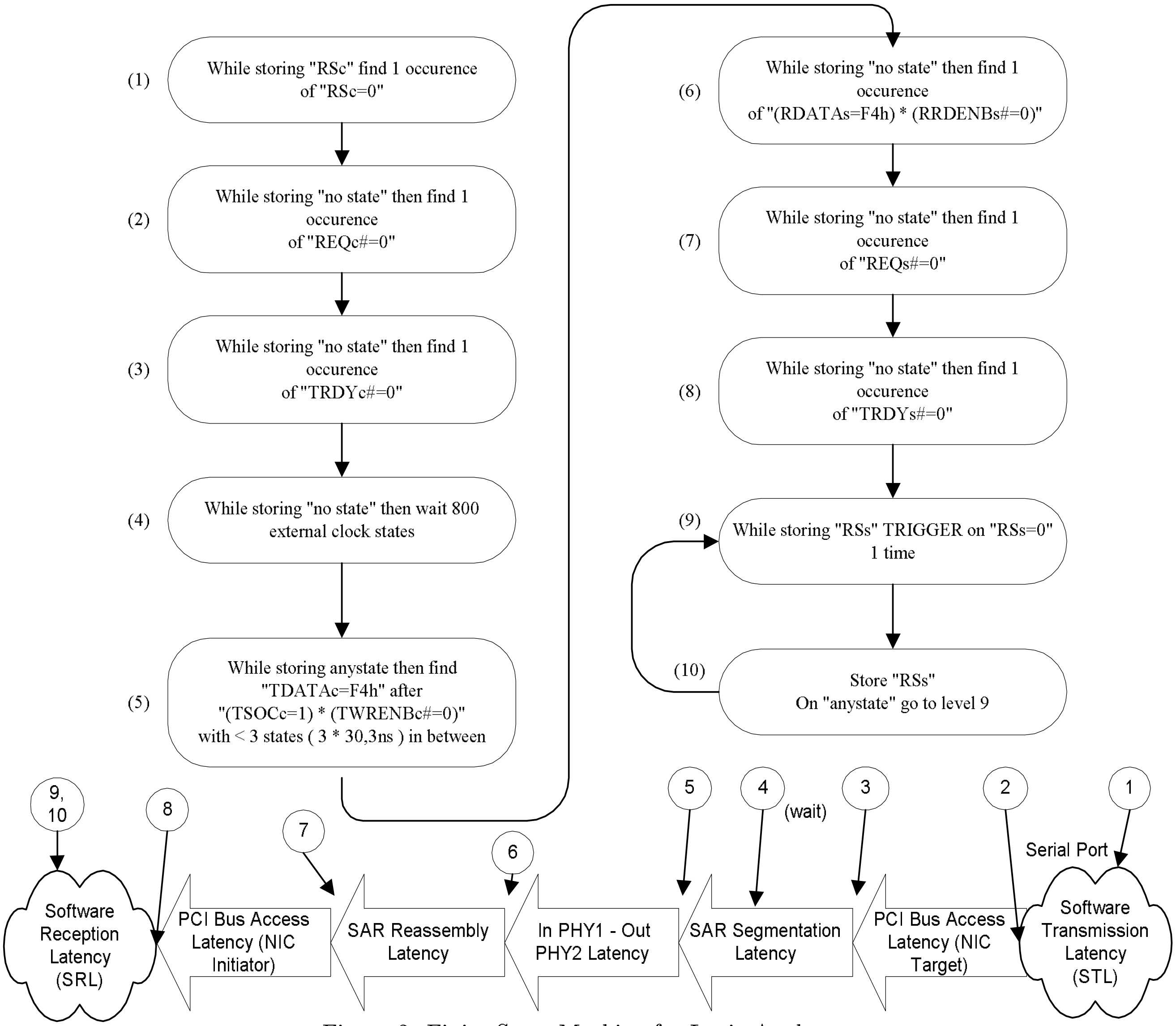

Figure 3: Finite State Machine for Logic Analyzer

The methodology of the trace capture will be presented by examining the FSM state-by-state. Every wait period will be explained in detail so that there are no aspects of the methodology that appear as black magic. The suffixes S and C in control signals denote the server and the client machine, respectively, and the * denotes an "active-low" signal (i.e. corresponds to signals denoted with # in the PCI nomenclature). The logic analyzer is connected to the PCI clock, and therefore each sampling period is 30.3 nseconds.

The best way to initialize the FSM from a user application without taking too many signals of the logic analyzer was the usage of the serial port of each machine. This way, and with experimentally characterized system call and serial port latencies, the cloudy parts in Figure 2 can be determined too, albeit with less precision than the rest of the measurements because they correspond to aggregates of many factors. The indication of what state is stored *during each sampling* of the logic analyzer enables us to have a sequence of events which do not correspond to memory usage (indicated as 'While storing "no state"') until one single event occurs ('find 1 occurrence of'); other states sample inputs on every sampling period ('While storing "anystate"').

The code, below, is the actual sockets code for the PC which sends one byte at the serial port. The data value does not matter for the experiment because the logic analyzer triggers on the start bit bringing

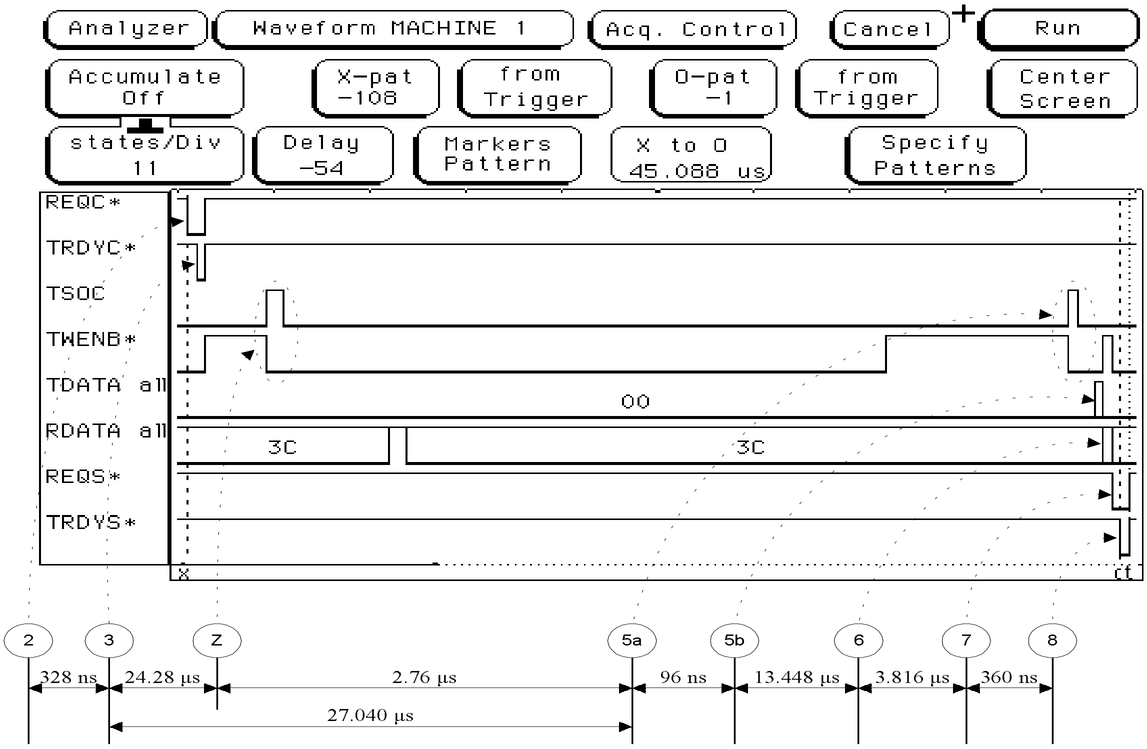

Figure 4: State Mode Trace of Network Access

the normally "1" line to "0". The code is for the transmission process (i.e. it triggers the logic analyzer during the client STL) for the TCP protocol. Similarly, upon reception of data from the network, a character is output in the serial port. Similar codes exist for the UDP protocol. codes exist for the reception, as well as for the UDP protocol.

```
/*              PC code associated with STL                                 */

/*              Write to serial port the character "?"                      */
WriteFile( hCom,              /* hCom determines COM1/COM2                  */
           "?",               /* character to write                         */
            1,                /* number of bytes in the buffer              */
            &WrittenBytes,    /* number of bytes written                    */
            &Overlpd) ;       /* the call results in overlapped operation   */

/*              Commence network transmission                               */
WSASend ( socket,             /* opened socket                              */
          (LPWSABUF)&WsaBufOut,/* pointer to array of buffers containing
                                 data to be transmitted                     */
          1,                  /* number of buffers in the array             */
          &BytesSent,         /* number of correctly transmitted bytes      */
          SentFlags,          /* flags                                      */
          NULL,NULL);         /* Non overllaped socket                      */
```

Each step of the logic analyzer FSM is analyzed below, with the numbers corresponding to the state numbers in Figure 3:

1. Wait for a '0' at the RS line (which is connected to the serial port of the client in Figure 2). This is the first step, associated with STL.

2. Wait until the REQ# signal of the PCI bus of the client is asserted, denoting that the application is requesting to become bus master (see sidebar). At this point STL has been completed and the time measured is the actual STL plus the cost of the serial port access.

3. The logic analyzer waits until TRDY# is asserted, which concludes the PCI access, upon which data will be transferred. This time between states 2 and 3 is the PCI bus access latency.

4. Wait for 800 clock cycles without storing data, so that logic analyzer memory is not excessively used up.

5. Sample on every clock period until the actual first byte of data of our application (in this case the value F4H) appears on the RxUTOPIA bus of the client. The latency from state 3 to state 5 represents the SAR latency. We note that the condition which is evaluated is quite complex and requires the comparison of variables which appear within three cycles. In order for the logic analyzer to evaluate this condition, it must sample on every cycle (and use its memory up accordingly).

6. Determine when the server's RxUTOPIA bus starts to send the data received from the network to the SAR of the server. The latency from state 5 to state 6 is the PHY1 IN to PHY2 OUT latency. The same data (F4H) is expected to be found in the RxUTOPIA bus so that the measurement is accurate.

7. Wait until the server's REQ# signal is asserted, to determine the reassembly latency of the SAR.

8. Wait until the server's PCI arbitration process is complete, when the TRDY# signal is asserted, signifying that the server's NIC is now the bus master.

9. To determine the time until the data reaches the user process, wait until anything appears on the serial port of the server.

10. The loop between states 9 and 10 forms the sink state to end the run.

Figure 4 shows an actual logic analyzer trace captured with the above state machine. It is annotated with the numbers 2-8 which correspond to the same-numbered states of the FSM. In the case of the complex state 5, the annotations 5a and 5b show the three cycles during which the condition gets evaluated. The non-uniformity of time in this capture is clearly demonstrated in many cases, e.g. when the 96 nsec between 5a and 5b correspond to more samples and appear longer than the 3.816μsec between states 6 and 7.

Although the mapping of the states corresponds directly to the arrows in one-way data transmission as shown in Figure 2, the actual choice of conditions came from substantial study and experimentation. Let us consider the 800 cycle delay in state 4: although the manual of the NIC indicated that the control signals in the UTOPIA bus are asserted during packet transmission, we found that they were asserted at times not associated only with our packets. If these signals were used, we would obtain erroneous results (too fast). The actual value of the data (F4H in this example) allows us to confirm that the appropriate cell crosses the UTOPIA bus but we also want the control signals associated with the transaction so that there would not be false readings for the payload of some unrelated cell. The wait period in state 4 was introduced to save state memory on the logic analyzer, since state 5 has to sample on every period. The actual value of 800 cycles corresponds to 24.24μsec. It was chosen to be safely smaller than the 26.6μsec minimum cell assembly latency by the SAR which we determined *experimentally* with the logic analyzer in timing mode. The 800 clock cycles are shown in Figure 4 as period Z. State 6 need not have such an elaborate setup because the only data transmitted after state 5 is the correct data. Finally, the loop between states 9 and 10 is used to end the capture. The serial port user level call of state 9 corresponds to the *server* receiving the data, which is not shown in Figure 2.

Figure 5 shows in more detail the transmission of the beginning of an ATM cell through the Tx-UTOPIA bus. Although it was captured with a different FSM on the logic analyzer and on different runs than those described above, it corresponds to the same user cell transmission and contains the characteristic F4 pattern. In actuality the F4 pattern is not part of the payload of the cell, but the third

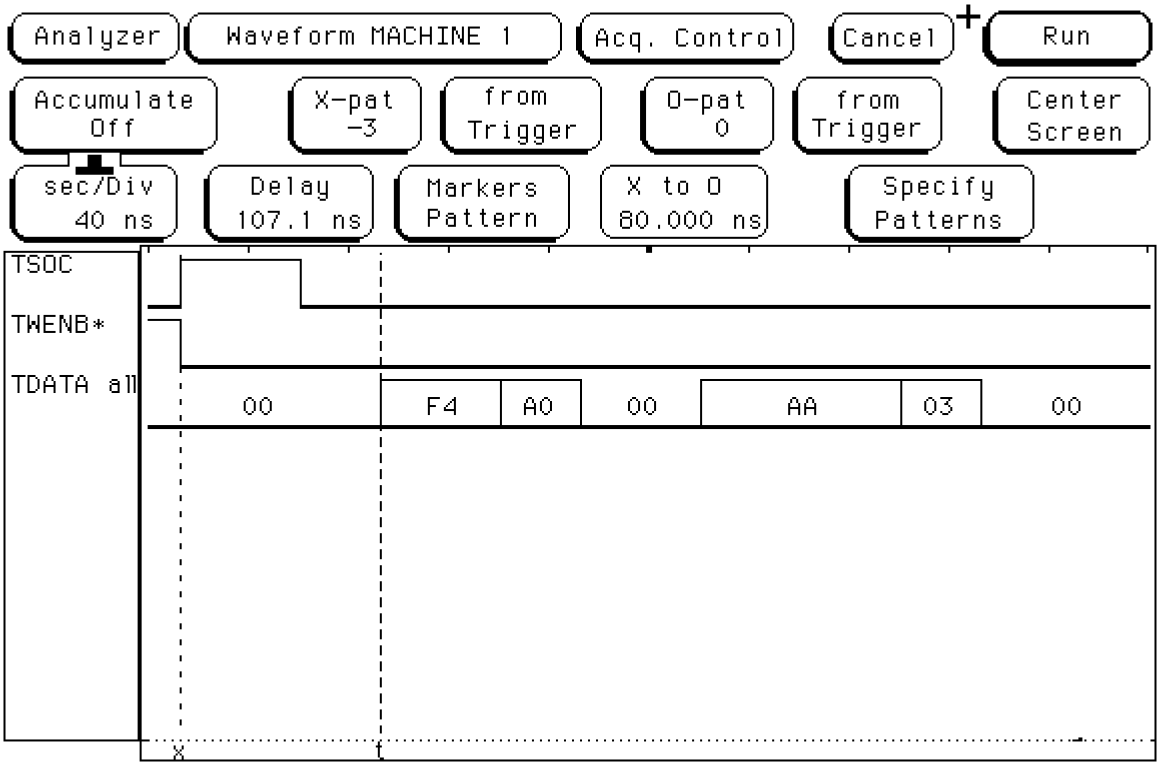

Figure 5: Timing Mode Trace of Single ATM Cell Transmission

byte of the header and belongs to the VCI (bits 11:4). It was chosen because it was the first byte that could be modified during the setting of the network interface. The usage of the TSOC control line status and the F4 pattern within a three cycle window in the state mode logic analyzer FSM allowed for proper capture of the beginning of an ATM cell transmission even if some cell contained the F4 pattern as part of its payload. The usefulness of the timing mode of the logic analyzer in order to determine how to set it up in state mode is evident in this case, which resulted in two FSM states combining a wait period and capture based on appropriate control and data patters within a three cycle window.

3.2 Conducting the Experiments

The methods shown in the previous section were used repeatedly to measure all aspects of the latency, as shown in Figure 2. The first set of experiments recorded in detail the latency from the client to the server, and was described in the previous subsection. We note, however, that in order to terminate the acquisition, the server needs to access its serial port once it has received data from the network. Such an action would add system call delays to the measurements of the Software Echo Latency. In effect, the previous experiment records the server's SRL in addition to the transmission latency. A second experiment was conducted in a very similar way, to record the latency of the return path of the data. This experiment includes the (ultimately unwanted) STL of the *server*. To complete the picture, a third experiment was conducted such that the server did not output anything in its serial port, but rather performed the return of the data at the fastest possible rate. During the third experiment the piece-wise latencies of the data transmission could not be captured for lack of channels and FSM states on the logic analyzer. Rather, the system was probed at the PCI bus *of the client*, and at the RxUTOPIA and TxUTOPIA busses of the server. With the usage of the client's serial port, these measurements accounted for these times:

- User's process to client PCI (i.e. the STL plus one access to the client's serial port)
- Client PCI to RxUTOPIA of server NIC

- RxUTOPIA to TxUTOPIA of server NIC (i.e. the SEL plus two SAR accesses plus two PCI bus accesses)
- Server TxUTOPIA to client PCI bus
- Client PCI bus to client application (i.e. the SRL plus one access to the client's serial port)

The reason that SEL was not captured on the server's PCI is that the bus might be active with additional received data while it returns the first packet, and thus there would be confusion as to what should be captured. Furthermore, in systems with several PCI cards the bus may be used for other purposes too. By capturing SEL at the UTOPIA busses and knowing the PCI bus access latency and the SAR latencies, the SEL can be determined. The aggregate latencies PCI to RxUTOPIA and TxUTOPIA to PCI in the third experiment were verified against the detailed measurements from the previous experiments and found to be correct. This way, a complete breakdown of the delays that affect network latency was possible. Additional experiments were performed to fully characterize the PCI access latency and the user application's access to the serial port.

4 Experimental Results

Many experiments were conducted with the setup and methodology described in this paper. Given that for each case many runs (usually ten to fifteen) were conducted with the same parameters to obtain minimum, average, and maximum times, and each class of experiments was conducted with a multitude of transmission size parameters, we could include here tables upon tables. Rather, we chose to present three tables: Table 1 shows the breakdown of the latencies as described in this paper. Table 2 shows the measured STL and SRL latencies for the Pentium as the client. Finally, Table 3 shows the measured SEL for the two machines.

	486 to Pentium			Pentium to 486		
	Min	Avg	Max	Min	Avg	Max
PCI Bus Access Latency(μsec) (NIC is PCI Target)	0.45	0.67	1.11	0.33	0.39	0.84
SAR Segmentation Latency(μsec) (for first cell)	26.65	27.07	27.55	26.65	27.07	27.55
IN PHY(1/2)-OUT PHY(2/1) Latency (μsec)	13.02	13.54	13.63	13.02	13.54	13.63
SAR Reassembly Latency(μsec) (for first cell)	3.61	3.66	3.82	3.61	3.66	3.81
PCI Bus Access Latency(μsec) (NIC is PCI Initiator)	0.12	0.15	0.18	0.34	0.49	1.06

Table 1: Network Interface Latency Breakdown

In Table 1 we observe that the latencies associated with the NIC do not vary with respect to whether the NIC is the server or the client, but there is a significant difference between the segmentation and reassembly times. On the other hand, there is a substantial variation on the PCI Bus Access Latency depending on the machine and whether it was used as the server or the client, but the PCI bus access times are very small with respect to the NIC times. There is a 1μsec worst case latency for the PCI bus whereas assuming that the PHY OUT and PHY IN times are symmetrical the best case is when a cell is received which would have a NIC associated time of $13.020/2 + 3.608 \approx 10\mu$sec. To demonstrate though how latency and throughput are often related in strange ways, after the initial latencies shown in Table 1, the NIC can send and receive an ATM cell every 2.76μsec. This rate can be derived from ATM specifications, and it was indeed observed by us experimentally. In this case, the PCI bus of a loaded system *receiving* ATM packets *can* be the bottleneck during long transmissions and as a result cells can be lost. Of course, the culprit in these cases is not the PCI bus itself, but the speed of the CPU which determines how quickly a transfer will take place, as demonstrated by comparing the 486 to the Pentium times when the NIC is the target. The converse does not hold, because during the transmission of ATM cells the drivers use DMA from the host memory CS-PDU buffer to the NIC, which is accomplished with the burst mode of the PCI.

	STL		SRL	
Data	TCP μsec	UDP μsec	TCP μsec	UDP μsec
1	202.33	205.78	332.17	283.50
10	207.93	209.90	326.23	293.90
100	209.52	215.36	342.86	296.02
500	235.33	234.90	403.06	337.86
1000	266.73	263.78	469.93	396.32
1460	294.65	-	571.34	-
1472	-	273.26	-	374.43

Table 2: Software Transmission Latency and Software Reception Latency for Pentium

The Table 2 has the STL and SRL for the Pentium processor only (the Pentium is the client). Actually, these are the experimentally produced times and therefore each entry includes a 88.2μsec serial port access latency. We determined that on the Pentium machine, the 88.2μsec did not vary at all between runs, and as a result we could accurately take into account the effect of the serial port. In the case of the underpowered 486 the serial port access latency not only was higher but also it varied considerably among runs. We did not make the adjustments in order to present the actual experimental data. The SRL times are for *unloaded* server, as experiments with the server sending large amounts of data at the same time it received data would skew the results. We notice that the SRL times are much higher than the STL times, (almost twice as high in the TCP case for 1460 bytes transmission). Since all entries correspond to latencies it should not be assumed that the transmission times for the longer transmissions are practically the same as for short ones, these are the latencies for the *first* cell of each transmission. In effect, these times show protocol and buffering induced delays.

The entries for transmission of 1460 bytes in the TCP case and 1472 bytes in the UDP case correspond to the Maximum Segment Size (MSS) of the TCP and the corresponding quantity of UDP.

We observe that the UDP latencies are higher than the associated TCP ones for small transmissions, and lower for large transmissions. These latencies are the *release latencies* and do not correspond to the throughput that the user will observe with the respective protocols. Indeed, the lack of establishment of connection by UDP makes the protocol appear slow with respect to the TCP which establishes connection prior to sending data. In different experiments, with software-only methods, UDP throughput was found to be close to two times that of TCP, but also UDP had as much as 35% lost datagrams as compared to 0.2% retransmissions of TCP. These results too show how different the latency can be from the inverse of the throughput.

Table 3 shows the measured SEL for both the Pentium and the 486. We included data for both processors because SEL significantly depends on processing power, and because these measurements do not include system calls to the serial port. The measured times, of course, do include two PCI bus accesses, one SAR reassembly and one SAR segmentation latency, the sum of which in the worst case is less than 35μsec (as derived from Table 1). This table was largely included to complete the experimental data associated with the Figure 2 experiments.

5 Conclusions

In this paper we presented hardware methods to characterize the piece-wise latencies that form the final network transmission latency for ATM networks. Our methods are based on a cooperation of software programs and hardware monitors (like the logic analyzer). Our software uses the computer's

	Pentium		486	
Data	TCP μsec	UDP μsec	TCP μsec	UDP μsec
1	403.9	407.1	1,288.1	1,266.1
10	407.9	411.9	1,149.3	1,272.5
100	413.0	423.0	1,214.0	1,252.6
500	481.5	482.4	1,398.3	1,421.0
1000	571.0	554.1	1,571.0	1,602.3
1460	640.8	-	1,788.8	-
1472	-	615.7	-	1,852.0

Table 3: Software Echo Latency for Pentium and 486

serial port to efficiently communicate with (and control) the logic analyzer, which in turn tracks the route of ATM cells through the network. Effectively, our methodology allows user-level software to efficiently monitor the ATM traffic with a granularity at the μsec range. Using our approach we have performed extensive latency measurements in an ATM testbed and found:

- Latencies are asymmetric with respect to the transmitter and the receiver
- More often than not the latency in high speed networks *cannot* be derived from the inverse of the throughput.

As a result, hardware based performance evaluation methods are effectively mandatory for complete evaluation of high performance network interfaces, and networks in general.

Acknowledgements

We wish to express our appreciation to Mr. George Kalokerinos for his valuable assistance regarding the fine aspects of the PCI bus, and to numerous members of the technical community at large who responded via Internet to many technical questions ranging from device driver issues to Windows NT subtleties. This work was partially supported by the "ASICCOM" ACTS project.

References

[1] R. Ahuja, S. Keshav, and H. Saran. Design, Implementation, and Performance Measurement of a Native-Mode ATM Transport Layer (Extended Version). *IEEE/ACM Transactions on Networking*, 4(4):502–515, August 1996.

[2] Matt Buchanan and Andrew A. Chien. Coordinated Thread Scheduling for Workstation Clusters Under Windows NT. In *Proceedings of the USENIX Windows NT Workshop*, pages 47–54, August 1997.

[3] J. Bradley Chen, Yasuhiro Endo, Kee Chan, David Mazieres, Antonio Dias, Margo Seltzer, and Michael Smith. The Measured Performance of Personal Computer Operating Systems. In *Proceedings of the 15th ACM Symposium on Operating System Principles*, volume 29, pages 299–313. ACM SIGOPS Operating System Review, December 1995.

[4] Aloke Guha, Allalaghatta Pavan, Jonathan Liu, Ajay Rastogi, and Todd Steeves. Supporting Real-Time and Multimedia Applications on the Mercuri Testbed. *IEEE Journal on Selected Areas in Communications*, 13(4):749–763, May 1995.

[5] Dilip D. Kandlur, Debanjan Saha, and Marc Willebeek-LeMair. Protocol Architecture for Multimedia Applications Over ATM Networks. *IEEE Journal on Selected Areas in Communications*, 14(7):1349–1359, September 1996.

[6] Jonathan Kay and Joseph Pasquale. Profiling and Reducing Processing Overheads in TCP/IP. *IEEE/ACM Transactions on Networking*, 4(6):817–828, December 1996.

[7] Evangelos Markatos and Manolis Katevenis. User-Level DMA without Operating System Kernel Modification. In *3rd International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE Computer Society, February 1997.

[8] Shubhendu S. Mukherjee and Mark D. Hill. The Impact of Data Transfer and Buffering Alternatives on Network Interface Design. In *4th International Symposium on High-Performance Computer Architecture (HPCA)*, pages 207–218. IEEE Computer Society, January 1998.

[9] John H. Naegle, Steven A. Gossage, Nickolas Testi, Michael O. Vahle, and Joseph H. Maestas. Building Network for the Wide and Local Areas Using Asynchronous Transfer Mode switches and Synchronous Optical Network Technology. *IEEE Journal on Selected Areas in Communications*, 13(4):662–671, May 1995.

[10] Allyn Romanow and Sally Floyd. Dynamics of TCP Traffic over ATM Networks. *IEEE Journal on Selected Areas in Communications, vol.13, no.4*, pages 633–641, May 1995.

[11] Ioannis Schoinas and Mark D. Hill. Address Translation Mechanisms in Network Interfaces. In *4th International Symposium on High-Performance Computer Architecture (HPCA)*, pages 219–230. IEEE Computer Society, January 1998.

[12] Thorsten von Eicken, Anindya Basu, Vineet Buch, and Werner Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*. IEEE Computer Society, December 1995.

[13] Matt Welsh, Anindya Basu, and Thorsten von Eicken. Incorporating Memory Management into User-Level Network Interfaces. In *Proceedings, Hot Interconnects Symposium V 1997*. IEEE Computer Society, August 1997.

SIDEBAR: PCI BUS

The Peripheral Component Interconnect (PCI) bus is the most common bus used in personal computers today. Although the PCI bus is actually a collection of standards which correspond to busses with different datapath width (32 or 64 bit) and speed, a commonality of control signals allows for upward compatibility and interoperability of PCI cards. The bus used in this work follows version 2.16 of the standard, and supports 32-bit datapath. With a clock of 33MHz and a minimum of one clock cycle for a bus transaction in burst mode, the maximum theoretical bandwidth of this bus is $4 \times 33\ Mbytes/sec = 132\ Mbytes/sec = 1,056\ Gbit/sec$, or, roughly seven times the theoretical maximum speed of the ATM network we examined. We will introduce a minimum of concepts regarding the PCI bus, and mostly in the context of signals that this paper refers to.

In each PCI bus transfer there is an *initiator* and a *target*. The initiator (or bus master) is the device that originates the transaction and configures the target (or slave) for the data transfer. Data can be transferred in either direction, i.e. it is not necessary that data is transferred from the initiator to the target. The PCI bus supports *burst mode* transfers, which is the transmission of a single address (address phase) followed by two or more data phases. The initiator controls how long the bus is occupied during burst transfers. The starting address and transaction type are determined at the address phase. The target device latches the start address in an address counter and increments it automatically between successive data phases.

PCI Control Signals

Every device that can be a PCI bus master has one pair of control signals which can be directly connected to the PCI arbiter, *REQ#* and *GNT#* (The # sign signifies an active low signal). The *REQ#* signal is asserted to request control of the bus, and the arbiter asserts the *GNT#* signal to enable the corresponding device to become bus master. This way there can be different bus masters at different times. The following (small) subset of control signals is important in determining PCI bus transactions:

The table below shows in summary the operation of the PCI bus as it is determined by the above signals being asserted (A) or deasserted (D). The $*$ entries indicate "don't care" situations.

FRAME#	*IRDY#*	*DEVSEL#*	*TRDY#*	*REQ#*	*GNT#*	
D	*	*	D	*	A	Bus is given from the arbiter to the bus master
A	D	D	D	*	*	Initiator transmits initial address phase, to be decoded by the target
A	A	*	*	*	*	Initiator is ready to send/receive data
A	A	A	*	*	*	Target has decoded the address from the initiator
A	*	A	A	*	*	Target is ready to send/receive data
D	A	*	*	*	*	Last data phase transmission

Table 4: PCI Control Signals and Bus States

- *FRAME#:* is asserted by the initiator for the duration of the transaction.
- *TRDY#* (target ready): is asserted by the target device and indicates that the target can send or receive data (as appropriate) during the data phase.
- *IRDY#* (initiator ready): is asserted by the initiator and indicates that the initiator can send or receive (as appropriate) data to or from the target.
- *DEVSEL#* (device select): is asserted by the target after it has decoded the initiator requested address and has determined that belongs to itself.

PCI Bus Access latency

The total time from the assertion of *REQ#* by the initiator until the assertion of the *TRDY#* by the target is called the "bus access latency". The times that constitute the bus access latency are summarized below. It can be clearly seen that the theoretical bus transfer rate of 132Mbytes/sec is attainable *after* the initial bus access latency, and only for burst transfers.

- Arbitration Latency: The time from the assertion of *REQ#* by the initiator unitl the reception of the asserted *GNT#* by the initiator.
- Bus Acquisition Latency: The time from the reception of the asserted *GNT#* by the initiator until the assertion of the *FRAME#* by the initiator.
- Target Latency: The time from the assertion of the *FRAME#* by the initiator until the assertion of *TRDY#* by the target.

These latencies are shown graphically below.

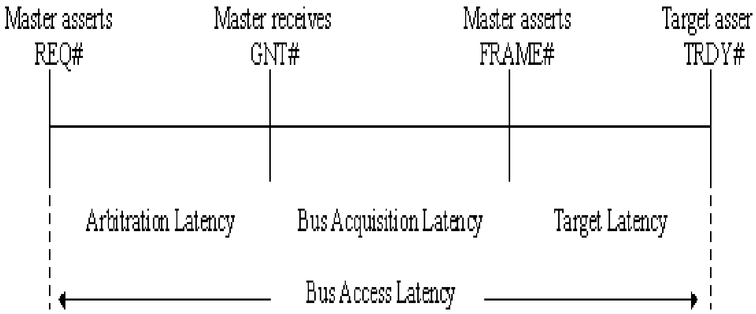

Figure 6: PCI Latencies