

PC Based ATM Link Delay Simulator (LDS)

DOE Grant No. DE-EG02-95ER25272

Final Report

DOE Patent Clearance Granted

MP Dvorscak Sept 24 2001
Date
Mark P. Dvorscak
(630) 252-2393
E-mail mark.dvorscak@ch.doe.gov
Office of Intellectual Property Law
DOE Chicago Operations Office

Harvard University
33 Oxford St.
Cambridge, MA 02138

Technical Point of Contact

H.T.Kung
(William Gates Professor of Computer Science)
Ph: (617) 496-6211
Fax: (617) 496-5508
Email: htk@eecs.harvard

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

**Portions of this document may be illegible
in electronic image products. Images are
produced from the best available original
document.**

1 Executive Summary

The increasing popularity of high speed Wide Area Networks (WANs) has created a need for specialized equipment to facilitate tailoring new and modifying existing protocols to better fit the increased bandwidth these networks offer.

Unfortunately, these specialized tools can be very expensive, creating a major impediment to research and development of protocols for better use of the combination of high speed and long propagation delays inherent with modern WANs. This project illustrates that one cost-effective solution to building specialized test equipment is to use the personal computer as an implementation platform.

The ATM Link Delay Simulator (LDS) implemented in this project adds propagation delay to the ATM link on which it is installed, to allow control of link propagation delay in network protocol experiments simulating an adjustable piece of optical fiber. Our LDS simulates a delay of between 1.5 and 500 milliseconds and is built with commodity PC hardware, only the ATM network interface card is not generally available. Our implementation is special in that it preserves the exact spacing of ATM data cells a feature that requires sustained high performance.

Our implementation of the LDS shows that applications demanding sustained high performance are possible on commodity PC hardware. This illustrates the promise that PC hardware has for adaptability to demanding specialized testing of high-speed network. The result suggests that the PC is a cost-effective platform to implement high performance networking tools.

2 Project Summary

We have implemented a low-cost, flexible, and easy-to-use Link Delay Simulator (LDS) for OC-3 (155 Mbps) ATM links using a commodity Pentium PC equipped with a single PCI ATM host adapter. The LDS is a useful tool for studying the effects of varied link delays on protocol behavior. By simulating only the WAN portion of the link, the LDS allows actual protocol implementations to be incorporated into the simulation results. The LDS demands 100% sustained, full-duplex bandwidth from the ATM host interface. This performance requirement is significantly greater than those of typical applications. The LDS cannot use the host interface in whole-packet (AAL5) mode, essential for attaining sustained high-performance on many host adapters. Our LDS implementation meets the bandwidth requirements of a full-duplex OC-3 link (300+ Mbps over the PCI bus) by aggressively using burst-mode DMA, large network buffers, device polling to reduce host CPU interrupts, and null cell buffering to obviate scheduling computations for preserving inter-cell spacing of the delayed data. The contributions of this work are a detailed description of the LDS architecture and implementation, and a discussion of host adapter architecture for high-performance, real-time, network applications. Other real-time systems also present streaming input, and require careful attention to buffer sizing, interrupt load on the host CPU, and bus performance; we offer a detailed study of the LDS as one such system. To show the utility of the LDS, we use it to validate others' analytic predictions of TCP fairness for competing flows in the presence of varying round-trip delay.

3 Main Body

3.1 Introduction to the LDS

The ATM Link Delay Simulator (LDS) described in this paper (based on [Gaynor, Karp, and Kung]) adds propagation delay to the ATM link on which it is installed, to allow control of link propagation delay in network protocol experiments. Implemented on a NetBSD UNIX Pentium PC with an ATM adapter card, the LDS can add a delay between 1.5 and 500 milliseconds to an OC-3 (155 Mbps) ATM link. The device is logically a fixed-delay shift register for ATM cells, in the way shift registers delay data paths in circuit designs. ATM links use SONET, a synchronous line protocol. Each 53-byte cell time at the OC-3 rate, a cell arrives at and departs from the LDS. The LDS passes each arriving cell through this fixed-delay shift register before transmitting the cell further along the link on which it is interposed.

While link delays can be achieved several ways, none is as economical, flexible, or easy-to-manage as an LDS made from commodity hardware. For example, actual links in real-world ATM WANs certainly provide propagation delays, but are expensive, hard to manage, frequently do not exhibit reproducible behavior, and do not have adjustable propagation delays. Other alternatives, such as custom-built devices and spools of fiber, are often not cost-effective, and require resources not widely available.

The purpose of this work is to create a flexible, easy-to-use, and cost-effective research tool for studying varied propagation delays' effects on network protocol performance in a controlled laboratory setting using real protocol implementations. The tool should support a wide range of delays, and should not modify the delayed data in

content, order of interleaving, or inter-cell spacing. Because our LDS design simultaneously sources one OC-3 fiber and sinks another, it demands real-time, high-bandwidth I/O, at over 300 Mbps. We meet this performance requirement by aggressively using host-adapter-mastered, burst-mode DMA, large network buffers, device polling to reduce host CPU interrupts, and null cell buffering to obviate scheduling computations for preserving inter-cell spacing through the delay.

The contributions of this work are a detailed description of our LDS implementation and a discussion of host adapter architecture for high-performance, real-time, network applications such as the LDS. We have also used the LDS to study the effects of varying round-trip delay on competing TCPs' bandwidth sharing.

3.1.1 Using the LDS

One typically uses the LDS in conjunction with an ATM switch, by interposing it between two switch ports and routing circuits to be delayed over this physical path, as depicted in Figure 1.

Multiple instances of the LDS with ATM switches can simulate more complex network configurations. Figure 2(a) illustrates such a configuration. Host A in Massachusetts is connected to Host C in California by a transcontinental ATM/SONET link. Host A is also connected to Host B in Japan via a satellite ATM circuit. Figure 2(b) shows how two instances of the LDS and an ATM switch can simulate this network configuration in a laboratory, with the proper round-trip delays.

We have built several working prototypes of the LDS, and use them to conduct network research experiments at Harvard. These prototypes delay ATM circuits routed through them without changing, reordering, or dropping any data cells.

3.2 LDS Host and Host Adapter Platforms

The prototype LDS runs on a 133 MHz Pentium PC with 32 MB of RAM and a Triton PCI chipset, and Intel's OC-3 IJET PCI ATM host adapter. Triton's support for burst transfers is essential to high PCI throughput; without burst transfers, bus arbitration overhead cannot be amortized over multiple words. The earlier Neptune PCI chipset supported only single-word (32-bit) PCI transfers, and could not sustain sufficient bandwidth to sink one OC-3 link rate TCP flow. (The LDS must maintain twice OC-3 bandwidth across the PCI bus.) The LDS software runs in the NetBSD UNIX 1.0 kernel, and is built on an IJET device driver developed by Carnegie Mellon in collaboration with Intel.

The IJET research prototype ATM host adapter, like many other ATM host adapters, is optimized for processing packets, i.e., AAL5 PDUs. This design choice makes sense for typical host uses of ATM networks; high-bandwidth flows generate and receive large packets (significantly larger than 48-byte ATM cell payloads, with hundreds of cells possible). Because ATM cells arrive very frequently at the OC-3 rate (one arrives every 2.83 microseconds), the host CPU should not be involved in per-cell processing; rather, it must perform packet-level computations only.

The IJET controls AAL5 PDU segmentation and reassembly with an ASIC, to keep per-cell work off the host CPU. In addition, the IJET ASIC contains separate hardware DMA engines to read data to be sent on the link from host memory and write

data received on the link to host memory. There is no cell buffering on the IJET apart from a small amount required to rate-match the OC-3 link with the PCI bus; cell data are transferred to or from host memory via DMA as they arrive or depart, respectively. While network data are transferred to and from host memory directly by DMA, the IJET keeps descriptors (pointers) to these host memory buffers in a local control SRAM.

The IJET hardware demultiplexes the multiple VCs of the inbound cell stream on the basis of cell VCI and VPI values, and requires host applications enqueue data per-VC. In typical operation, cells for different VCs will be received interleaved on the ATM link. The IJET maintains a separate host memory receive/reassembly buffer for each VC, and appends inbound cells to that VC's current buffer via DMA. Host receive buffers are obtained from a host buffer free list, maintained in control memory on the IJET. When a receive buffer becomes full or the end of the received PDU is detected, the IJET optionally interrupts the host CPU to report the buffer's filled status, and obtains a new, empty receive buffer from the free list. Note that buffer full and PDU complete are separately maskable interrupts. This separation allows the host CPU to be notified of receive DMA completion per-buffer, regardless of whether received data correspond to AAL5 packet boundaries. It is the host CPU's responsibility to return receive buffers to the receive free list.

The IJET associates a separate transmit queue with each VC. A scheduling mechanism, not relevant to this work, selects one VC per cell cycle. The IJET DMA's one cell's worth of data (stored in host memory) from the head of this VC's transmit buffer descriptor list (stored in control memory) and transmits it on the ATM link. As with

receive operation, when a transmit buffer has been exhausted (fully transmitted), the IJET optionally interrupts the host CPU so that the host CPU can reuse the transmit buffer.

Other ATM host adapters, such as Digital's Otto [Digital] for the TurboChannel and newly released Meteor [Digital 1996a] for PCI, take somewhat different approaches to reassembly and buffer management than the IJET. The Otto, for example, reassembles PDUs in local adapter memory, holds whole PDUs which await transmission in local adapter memory, and DMA's whole PDUs between host memory and local adapter memory in single TurboChannel transfers. The Otto also organizes receive and transmit completion event notification buffers in a contiguous ring of physical host memory.

It is what these ATM interfaces, others, and the IJET have in common that is more noteworthy: namely, all are optimized for AAL5 operation, and demultiplex the received ATM cell stream into separate, per-VC, host memory buffers. These ATM interfaces also support a raw cell mode, in which a VC can be identified as non-AAL5, so that for that VC, individual cells are sent and received via DMA, and interrupts on receive and transmit completion per-cell are generated. As one would expect, these commodity ATM interfaces cannot sustain high bandwidths in raw cell mode when these interrupts are enabled, because these interrupts violate the "per-packet-only" computation requirement at the host CPU, and thus saturate it. Raw cell mode is a data formatting concern, which dictates that cells be buffered with header and payload both. The IJET decouples raw cell mode from interrupt frequency, and can thus transfer raw cells to and from buffers while only generating interrupts once every buffer completion, or never, if per-buffer interrupts are disabled.

3.3 LDS Architecture

The overall structure of the LDS is quite simple. The IJET receives cells as they arrive on the link (including null cells) at the OC-3 line rate, stores them in host memory without demultiplexing them per-VC, and transmits them in the same order. As the transmit and receive link rates are equal, the buffer usage in host memory remains constant. Spacing of data cells due to idle time on the receive link is precisely preserved, because null cells are stored and delayed. Other approaches to preserving cell spacing are possible, but not as accurate; for example, multi-cell receive buffers could be time-stamped, and these time stamps could be used to schedule full receive buffers for transmission in an attempt to preserve the spacing of received data. Note, however, that such a scheme would be imprecise in that it quantizes receipt times to buffer boundaries (i.e., timings of cell receipts within a buffer are lost). Scheduling on the basis of time stamps would complicate our implementation, and further increase host CPU overhead. While our design consumes a great deal of PCI bus bandwidth, it consumes no more than the timestamp design would in the worst case (all back-to-back data cells received with no null cells), and we meet this PCI bandwidth requirement.

For VCs in raw mode, the IJET does not transfer only cell payloads to and from host memory on receive and transmit (as in AAL5 operation); it transfers whole cells, headers included. Thus, if the IJET is made to do no per-VC demultiplexing, and instead views all arriving cells as logically corresponding to the same, raw-mode VC, it will stream all received cells into a single chain of receive buffers in host memory. Each receive buffer will be full of cells exactly as they arrived on the link. It is then a simple

matter to “turn around” these full receive buffers in FIFO order, by appending them to a queue of transmit buffers for a single, raw-mode transmit VC.

Because the single transmit VC is scheduled for transmit at the OC-3 link rate, and cells are appended to that VC’s transmit queue in complete receive buffers at the OC-3 link rate, the length of the transmit queue should not change, and determines the delay added by the LDS.

To initialize the LDS, we disable the IJET’s cell transmitter, and enqueue as many null cells for transmit on the single, raw-mode VC as required by the desired delay. We then enable cell receipt on the single, non-demultiplexed, raw-mode VC, and immediately enable the IJET’s cell transmitter. All that is required of the CPU during LDS operation is the per-buffer append operation for completed receive and transmit buffers, described above.

After carefully tuning our system, we found that this approach indeed results in a constant queue length for the single, raw-mode VC’s transmit queue when the LDS is run in UNIX’s single-user mode with 177-cell or larger buffers. The overhead of taking interrupts on the host CPU for receive buffer completion, linking these buffers onto the transmit queue, and other (non-LDS) PCI bus activity does not prevent the system from meeting the real-time, OC-3 transmit and receive constraints under these operating parameters. If there were a rate mismatch between transmit and receive, this transmit queue length would not remain constant; if cells were dropped on receive due to resource limitations in the LDS, the transmit queue would shrink, while this queue would lengthen if the IJET failed to transmit cells at the link rate.

In multi-user mode, however, the transmit queue length changes by a single buffer's worth of cells once every several hours even with large, 354-cell buffers. We conjecture that transient resource limitation in the LDS is the cause of such changes. If the PCI bus becomes unavailable long enough to cause overrun in the rate-matching FIFO in the IJET receive path, a cell will be dropped by the LDS. Transmit DMA is strictly lower priority than receive DMA, by design--data queued for transmit are buffered in host memory, and not dropped when transmit DMA is starved on the PCI bus. Thus, if transmit DMA on the IJET cannot transfer a whole cell's data in less than a transmit rate-matching FIFO's worth of time, the transmit queue will grow by a single cell. Over time, multiple instances of these single-cell events can cause a change of a full buffer's length in the transmit queue. In the interest of maintaining the desired delay in the face of such low-frequency fluctuations in multi-user mode operation, the LDS monitors and adjusts the number of buffers on the transmit queue and adds or deletes a buffer full of cells as needed. The LDS adjusts the delay at the buffer level because the system's software can only monitor the transmit queue length efficiently at a buffer granularity; finer monitoring would require cell-level computations, which are intractable. We continue an investigation of the exact cause of the transmit queue length variation in multi-user mode.

There are limits to the delay range and precision, which we explore in detail in Section 9 and Section 10. To summarize, the LDS allows delay adjustment in 500-microsecond increments, and our prototype supports delays between 1.5 and 500 milliseconds.

3.4 Distinguishing Requirements of the LDS

As we noted earlier, the LDS is an atypical ATM host adapter application. The requirements of the LDS which distinguish it as atypical include:

- **100-percent receive and transmit duty cycle:** The real-time nature of the LDS design requires that the host adapter maintain full, OC-3 rate transmit and receive, averaged over a short interval (25 microseconds for transmit and 50 microseconds for receive), beyond which the rate-matching receive and transmit FIFOs on the IJET overflow or run dry, respectively. Typical ATM applications do not approach this 300+ Mbps sustained bandwidth requirement; they send and receive data in either lower bit-rate streams (e.g., video or voice) or in short bursts at (or below) the link rate (e.g., distributed simulation, or web browsing). Many, including all applications which use TCP and real-time voice and video, tolerate data drops, unlike the LDS.
- **Lack of per-VC cell demultiplexing:** The LDS must preserve the precise interleaving of cells it receives by storing them contiguously in host memory. Cells from different VCs must all be transferred to the same receive buffers. In typical ATM applications, separate VCs represent logically distinct connections, each of which has a separate receive buffer in host memory, and each of which is owned by a distinct application (or protocol in the operating system kernel).
- **Receipt of null cells:** The LDS preserves the spacing of cells which arrive on the receive fiber by storing the null cells which arrive on the fiber between non-back-to-back data cells; because SONET is a synchronous physical link

protocol, idle slots on the link are filled with these null cells. Host adapters in normal operation drop null cells without even considering them for reassembly or DMA to the host, because these cells arrive at a potentially great rate and normally contain no useful information.

- **Receipt of cells with headers:** The LDS must store cells with their headers, so that it can forward them to the next switch with their header information (including VCI and VPI values) intact. Because typical ATM applications see data at the PDU (packet) level, cell headers are normally irrelevant to host software, and are therefore discarded without being transferred to host memory.
- **Exact receive/transmit clock synchronization:** The LDS relies very heavily on the equality of the host adapter's receive and transmit rates. Even an extremely small difference in these rates could result in a progressive lengthening or shortening of the delay over time. But the LDS must keep the delay constant over indefinitely long periods. A host adapter in normal operation recovers its receive clock from the receive data stream, but uses a local crystal to generate its transmit clock. Even a tiny difference between the frequency of the crystal used for transmit clock generation upstream of the LDS and the crystal on the LDS' host adapter, within tolerances of crystals, will be a problem.

3.5 Design Features of the LDS

The above properties of the LDS require a substantially different configuration of the ATM host adapter than is typical. Indeed, the required configuration is potentially

outside the normal parameter range which yields high performance for transmit and receive, as adapter designers optimize their designs for the most common application usage patterns.

An examination of how the unusual requirements of the LDS we list above interact with host adapter architecture and system software is instructive. Our prototype, its software, and its IJET hardware meet these LDS requirements as follows:

- **Large DMA burst size:** The Triton PCI chipset and IJET DMA engines make good use of burst-mode DMA operations for transfer of cells between host memory and the ATM link. Burst mode amortizes the latency of PCI bus arbitration over a multi-word transfer, and thus yields significantly greater bus bandwidth than single-word transfers. Burst mode DMA is essential to attaining the LDS' required 300+ Mbps average bus bandwidth.
- **Pipeline and DMA engine designed for single-cell DMAs:** The IJET does not buffer full AAL5 PDUs locally for reassembly; rather, it DMAs each arriving cell into the appropriate VC's reassembly buffer in host memory. This design feature reduces latency of PDU delivery to system software and applications, but it requires aggressive pipelining on the IJET. Consider that each cell time (2.83 microseconds), the IJET must be prepared to identify two different VC data buffers (one for transmit, one for receive) and begin two cell-sized DMAs (again, one for transmit, the other for receive). In a local reassembly and segmentation design, such as Digital's Otto, DMA operations are performed much less frequently, at the PDU level. Because the design of the IJET does not rely on infrequent, PDU-sized DMAs to maintain the link

rate, the LDS can sustain full-duplex OC-3 even without PDU reassembly if large buffers are used to accumulate the raw cells.

- **Raw cell receive and transmit:** The IJET permits each logical transmit and receive VC to be tagged as raw, includes cell headers in DMA operations for VCs so tagged, and does not compute AAL5 CRCs or search for end-of-PDU bits for them. By marking the single transmit and receive VCs used in the LDS as raw, we successfully preserve delayed cells' headers.
- **VCI demultiplexing masking:** For every cell received, the IJET must map the cell's VCI and VPI fields to a receive buffer. Because the space of possible VCIs and VPIs is so large (24 bits), the IJET uses a configurable subset of the VCI and VPI bits to generate a 10-bit index. This smaller index is used to look up VC state for receive processing. We use the mask register which selects the VCI and VPI bits used in index generation to map all cells whose 10 low-order VCI bits are zero to the same 10-bit index, with the result that the IJET does not demultiplex cells which arrive on the remaining 16K VCI/VPI values. Thus, we meet the LDS' requirement that cells for all delayed VCs be kept in order, without being demultiplexed.
- **Null cell preservation:** The SUNI Lite SONET chip on the IJET discards null cells by default, but we configure it to pass null cells to the IJET's reassembly and DMA hardware. Thus, we meet the LDS' requirement that null cells be preserved in the cell stream for conservation of data cell spacing across the delay.

- **Allocation of large blocks of contiguous physical memory:** The virtual memory implementation in NetBSD UNIX only guarantees physical contiguity within pages. Thus, it does not directly support allocation of physically contiguous regions more than 4K (the Pentium virtual memory page size) in size. Because the IJET generates one interrupt per filled receive buffer, and because the maximum delay will be determined by the total amount of data that can be queued for transmit, we need to allocate large receive buffers. But the IJET requires physically contiguous buffers, as it is unaware of virtual-to-physical address translation. To permit allocation of large (20K), physically contiguous receive buffers, we modified NetBSD UNIX's virtual memory system to support allocation of physically contiguous regions larger than the system page size.
- **Host CPU decoupling:** The host CPU in the LDS never touches cell data--a luxury unavailable to most network applications. The decoupling of buffer data and management for the IJET keeps the host CPU out of the high-bandwidth data path.
- **Polling to reduce interrupt count:** The LDS configures the IJET to deliver receive complete interrupts. But because the LDS should retire one transmit buffer for every filled receive buffer (from the equality of the transmit and receive rates, on average), notification of transmit buffer completion by interrupt from the IJET is unnecessary. Instead, our software disables the transmit complete interrupt, and polls the IJET for completely transmitted buffers once per filled receive buffer. The LDS software also implements the

common optimization of polling the receive complete queue before returning from a receive interrupt context, to reduce the number of interrupts taken by the host CPU. The polling done by our software reduces the frequency of interrupts, long-latency operations on the Pentium, and permits the host CPU to keep pace with the required buffer management.

- **Receive-to-transmit SONET clock loopback:** To ensure that the transmit clock and receive clock on the IJET are perfectly synchronized, we configure the IJET's SUNI chip to loop back the receive data clock it recovers by phase-locked loop to drive its own data transmission. Thus, we meet the LDS' requirement of exact transmit and receive clock rate matching.

3.6 Discussion of Host Buffer System Design

While our software and the IJET delivered the performance the LDS required, we note that there are further aspects of host adapter architecture which could yield even further reduced overhead for demanding, real-time network systems such as the LDS. One such area is buffer system design. Recall that the IJET's buffer descriptors (used to link host memory buffers into a free list, a receive complete queue, or a per-VC transmit queue) are stored in SRAM local to the IJET. Both the host CPU and the host adapter hardware must manipulate these buffer descriptors. But either the host CPU or the host adapter must pay a penalty for accessing them, because the CPU or the adapter hardware must be on the "wrong side" of the PCI bus from them. The IJET design makes the buffer descriptors more readily accessible to the adapter hardware, and sentences the host CPU to programmed I/O in order to access them. This is not the optimal choice; the host CPU is a precious resource, whereas the host adapter hardware can poll relatively easily.

Putting the buffer descriptors in host memory would reduce the host CPU overhead incurred by the LDS for moving each full receive buffer to the transmit queue. Digital adopts just this strategy in its designs of Fast Ethernet host adapters [Digital 1996b].

The LDS always processes received buffers in strict FIFO order. This behavior is not the norm; UNIX IP drivers have no knowledge of when and in what order received buffers will be consumed by applications. These drivers must return buffer descriptors to the host interface with a different host buffer pointer, to prevent overwriting of the newly filled physical buffer before its contents are consumed by the destination application. When buffer descriptors are stored in a ring, as they are on the Digital Otto, the standard IP driver must both point the descriptor at a new physical buffer and flip a bit in the descriptor indicating its availability to the host adapter. On the Otto, the LDS would never need to assign a new physical buffer to a receive descriptor, because the same physical buffer's contents will always be transmitted before that descriptor comes up for re-use in the ring. Ensuring this property is a matter of allocating more receive descriptors than the delay duration. Thus, the LDS would only need to flip the available bit on the descriptor in a single programmed I/O to return it to a ring of buffer descriptors like the Otto's. To return a receive buffer descriptor to the IJET, the LDS must read the tail pointer of the receive buffer descriptor list, set the next pointer of that tail descriptor to the appended descriptor's address, and set the next pointer of the newly appended descriptor to NULL. These operations take three programmed I/Os.

3.7 PCI and Interrupt Performance

In this section, we present a description and timing measurements of the work performed by the LDS when receiving a buffer full of raw cells, appending it to the

IJET's transmit list, and recycling a fully transmitted buffer onto the free list. This information will be used in Section 10 to explain the LDS' performance limitations. For each filled receive buffer, the LDS performs the following work:

- **DMA of received and transmitted cells between host memory and IJET over PCI:** Receive DMA occurs as cells arrive, and competes for the PCI bus with ATM transmit DMA. Both DMAs require OC-3 throughput, and compete in turn with all other PCI devices in the system (including a display adapter and disk controller). Bus contention and efficiency of the IJET's DMA implementation determine the throughput of these DMA operations.
- **Change-of-buffer in the IJET:** The IJET must obtain an empty receive buffer descriptor from its free list in time to continue DMA without interruption after filling the last cell of a receive buffer. If the latency of this operation exceeds the time required to fill a buffer (dependent on buffer size), or if the operation is not begun far enough in advance of filling the current receive buffer for any reason, the IJET will drop incoming cells until its ASIC obtains the new buffer descriptor.
- **Interrupt to the host CPU:** The IJET interrupts the host CPU once for every receive buffer fill event. If the kernel is already in the IJET's interrupt handler, the interrupt handler finds newly filled buffers by polling for them before returning. The latency of an interrupt on the Pentium varies widely depending upon the processor context at the time of the interrupt. The load on the host CPU from interrupts is determined by buffer size, as receive buffers fill at the OC-3 rate. Interrupt load on the host CPU is also determined by the

interleaving of interrupts; it is possible to average less than one interrupt per buffer if many buffer fill events occur while the IJET interrupt handler is already active.

- **Reading identity and contents of receive buffer descriptor across PCI bus:** In the receive interrupt handler, the host CPU reads the identity of the completed receive buffer and a few fields from the buffer's descriptor from the IJET's control memory, across the PCI bus. These small reads are rather inefficient programmed I/O across a busy bus.
- **Writing and appending of receive buffer descriptor across PCI bus:** The host CPU must modify a few fields in the buffer descriptor to prepare it for the transmit queue, and link it there. These small writes, too, are programmed I/O across the busy PCI bus.
- **Processing completed transmit buffers across PCI bus:** The host CPU must read the IJET's control memory across the PCI bus to determine whether buffer descriptors for transmitted buffers need to be returned to the IJET's transmit free list. The process for returning a completed transmit buffer descriptor to the transmit free list involves a few writes across the PCI bus.

Noteworthy in the above itemized list is the amount of host CPU programmed I/O required by the placement of buffer descriptors in SRAM on the IJET; we mention the trade-offs of this placement decision in Section 8. A typical interrupt in LDS operation occurs after a single buffer's worth of cells is received and a single buffer completes transmission, and requires a total of 8 32-bit reads and 8 32-bit writes by the host CPU

across the PCI bus. We find that over 99% of all interrupts generated by the IJET in a running LDS fall into this category.

In Table 1 we show the time required for PCI programmed I/O reads and writes, and the total host CPU execution time required for buffer processing, not including hardware interrupt latency and software interrupt dispatch. We made these measurements with the Pentium performance counters [Intel 1995]. We make single read and write timings after warming the instruction cache with the measurement code, and with all interrupts disabled. These single reads and writes are to and from CPU registers, and so are not distorted by main memory access times. The timings of the IJET interrupt routine are not instruction-cache-warmed, and are made under the usual interrupt masks set by the IJET device driver. Thus, our programmed I/O timings reflect only variation caused by PCI bus activity, while our interrupt routine timings measure the whole system's behavior, including the PCI bus, interrupts, and the CPU's caches

	Samples	Mean	Minimum	Maximum	Variance
32-bit PCI PIO Read	5,144	140	59	699	0.06
32-bit PCI PIO Write	5,066	34	8	457	0.13
IJET Interrupt Routine	5,057	3,754	2,831	9,005	3,774

Table 1: PCI access and interrupt timings (in 133MHz cycles) for 354-cell buffers

The data in Table 1 show that programmed I/O does not significantly limit the performance of the LDS; the average total cost per-interrupt of programmed I/O to IJET registers and memory is 1,392 CPU cycles. Buffering of PCI bus writes by the Triton PCI chipset reduces the best-case host CPU PCI write latency to only 8 cycles, whereas

unbuffered, non-cached reads take a minimum of 59 cycles. As the PCI bus runs at 33 MHz, the ratio of clock cycles to PCI cycles in our measurement machine is 4:1.

Histograms of the frequencies of ranges of per-read and per-write times for programmed I/O in Figures 3(a) and (b) reveal that most of these operations' durations are distributed more closely to their minima than their maxima.

On average, the LDS interrupt handler consumes 3,754 cycles per interrupt. With 354-cell buffers, approximately 1,000 IJET interrupts occur per second, and so the handler consumes 4% of the available 133 MHz CPU cycles. Note that these measurements neglect hardware interrupt latency and operating system interrupt dispatch latency, both of which can sum to hundreds of clock cycles [Endo et al. 1996]. Even if we included hardware interrupt latency and kernel interrupt dispatch, the worst-case several hundred additional cycles per interrupt would still not come close to saturating the host CPU. Thus, PCI bus contention or the cost of per-buffer operations on the IJET board itself are the remaining plausible bottlenecks in the LDS.

3.8 LDS Delay Bounds And Granularity

The user-configurable size of the buffers in which the IJET stores incoming ATM cells determines the granularity at which the delay may be specified, and the frequency of interrupts to the host CPU (one per completed receive buffer). Our system reliably sustains the OC-3 rate with 177-cell (about 500-microsecond) to 1,236-cell (maximum length supported by the IJET hardware) buffers. The default LDS buffer size is 354 cells (1 ms of data at OC-3). For shorter buffers, the interrupt rate and bus contention increase. Bus contention can cause cell drops two ways. The IJET receive rate-matching FIFO will drop cells when it becomes overrun after the bus has been unavailable to the IJET receive

DMA engine for 50 microseconds. More subtly, our software must drop entire cell buffers when the IJET transmit rate matching FIFO (lower priority for DMA than the receive DMA engine by design) repeatedly underflows because of PCI bus unavailability--otherwise, the delay would grow as transmit averaged a lesser long-term rate than receive.

The upper delay bound is determined by two constraints. First, host memory size minus operating system overhead limits the quantity of data that can be buffered. This limitation is not very restrictive; our prototype, 32 MB LDS allocates 12 MB of contiguous physical memory at the end of core, sufficient to buffer about 0.75 seconds of data at OC-3. The second constraint, the quantity of SRAM for buffer descriptors, is not very restrictive either; over 30,000 descriptors fit in the IJET's SRAM. While our prototype has large physical memory and buffer descriptor SRAM, there is a trade-off in any design similar to ours: small buffers offer maximal delay precision, but large buffers offer maximal delay length. Maximizing one reduces the other, for fixed physical memory and buffer descriptor SRAM. Our LDS uses one-millisecond buffers, and we've tested it with up to 500 such buffers. Longer delays (with larger and/or more numerous buffers) are possible.

We conjecture that some combination of PCI bus contention and limited processing for change-of-buffer operations on the IJET, both of which would limit the buffer processing rate of the system, determines the minimal buffer size at which the LDS can sustain full-duplex OC-3. The shortest delay the LDS can create depends on the system's minimum transmit queue length in cells. The transmit queue length can be shortened both by making host buffers smaller and by enqueueing fewer of them before

enabling the LJET transmitter. When the delay is long, the worst-case latency between the filling of a receive buffer and its linking onto the transmit queue need only be shorter than the total delay, so long as the average rate of appends remains equal to the OC-3 link rate. However, when the delay is very short, the worst-case latency between receive completion and append becomes a bottleneck, as there is a danger that the transmit queue can run dry. When we run the LDS with 177-cell buffers in UNIX's single-user mode, the smallest delay attainable is 1.5 milliseconds.

3.9 Measurement of LDS Induced Delay

We show that the LDS meets two important correctness criteria: that it does not drop ATM cells, and that it produces a consistent delay.

We have verified that the LDS does not drop cells by sending UDP and TCP packets through it at rates between a few bytes per second and the link rate. The receiving system's kernel counters reveal no bad packet header checksums, so no partial packets were dropped by the LDS. For whole packets, in the case of UDP, we measured application-level, per-packet sequence numbers to ensure all packets were delivered; for TCP, the sender's kernel retransmission counters verified that no packet drops occurred.

To measure the delay introduced by the LDS, we time-stamp UDP packets on a sending system, reflect these time stamps to the sender at the receiver, and compute the difference between the time a reflected time stamp arrives at the sender and the received time stamp's value. This difference represents the round-trip time experienced by the UDP packet, including the LDS, link delay, switch delays, and end-system hardware and software delays. To isolate the delay contributed by the LDS, we simply repeat these measurements both with and without the LDS installed on the path between receiver and

sender. The difference between the average round-trip time with and without the LDS in the loop is the delay introduced by the LDS. The variance in round-trip times with the LDS installed and without it installed hovers at 35 microseconds for delays of 1.5 milliseconds and greater, so the LDS does not add measurable variance to the round-trip time of the path between end systems.

3.10 Example Use for the LDS

To demonstrate the utility of the LDS in protocol behavior studies with real implementations, we present empirical measurements of TCP's fairness behavior for competing TCPs with varying round-trip delays. Because it permits controlled generation of delays, the LDS makes such measurements possible over a wide range of round-trip times, while most such work in the past has been limited to analytical models and simulations. In this case, our measurements closely match those predicted analytically by Floyd [Floyd 1991]. These results demonstrate the usefulness of the LDS for creating different network topologies for experimentation in a laboratory.

3.10.1 Measurement Configuration

Our configuration, shown in Figure 4, consists of 4 133 MHz DEC Alpha 3000/400 hosts, a Fore ASX-200BX ATM switch, a Harvard-Nortel CreditNet ATM switch, and the LDS. Each of two senders opens a single, greedy TCP connection to one of two receivers, and the two connections share a bottleneck link. The CreditNet switch links the LDS into the first TCP connection, permitting adjustment of long-tcp's RTT. The RTT of short-tcp is roughly one millisecond.

We adjust the RTT of long-tcp between one and 20 milliseconds, and compare the bandwidths achieved by long-tcp and short-tcp. Our measurements use 64K TCP windows (the norm in 4.4-derived, Reno TCP) and 9180-byte packets (the IP MTU for ATM). The bottleneck on the Fore switch has a 1,200-cell buffer and applies EPD (Early Packet Discard) [Romanow and Floyd 1995] with a threshold of 1,000 cells. It is conjectured in the network research community that EPD gives ATM switches packet dropping behavior similar to that of drop-tail packet routers. With 64K windows, this switch buffer size is about half the sum of the combined TCP windows.

3.10.2 Measurement Results

Figure 5 shows link goodput of long-tcp and short-tcp as a function of the ratio between their RTTs. Our measurement of goodput is the percentage of link capacity (133 Mbps after allowing for SONET, ATM headers, and IP and TCP header overhead with 9180-byte packets) received at the application level on a receiving host. Our data points represent mean values among at least three repeated measurements, and each of the three or more repetitions varied no more than ten percent from the mean. The two lines together show the fairness of the bandwidth allocation between the two TCPs on the bottleneck link. Note the bias in favor of short-tcp, the connection with the shorter RTT, as the ratio between the two connections' RTTs grows.

Figure 6 depicts the same bandwidth sharing comparison, but as predicted by Floyd's analytical model. Note that Floyd's model predicts throughput, while we measure goodput; because we observed few retransmissions in our measurements, the two are nearly equal in this case. While the overall trend between our empirical results and Floyd's analytical ones is the same--greater bandwidth for the connection with the shorter

RTT--our measurements differ at the boundary cases where the RTT ratio is least. In particular, our measurements suggest similar goodput for the two connections in this region. We conjecture that this disparity may be caused by buffering in the end systems for the shorter RTT connection (typically 1 or 2 packets' worth in the device driver and socket buffer at any instant, where in this case a packet is a full millisecond of data at the link rate). This buffering effectively narrows the RTT ratio between long-tcp and short-tcp, and the narrowing is most significant at the smallest RTT ratios.

3.11 Related Work

The heterogeneity of the individual links, nodes, protocols, and applications currently used in the Internet, combined with its rapid growth, creates a difficult environment to simulate [Paxson and Floyd 97]. Application-level software network simulators like UCB's and LBNL's "ns" (<http://www-mash.cs.berkeley.edu/ns/>) allow high-level simulation of algorithms and protocols for end-systems and routers or switches. For example, ns has been used extensively to measure the effects of different packet dropping strategies at congested IP routers carrying TCP traffic [Floyd 91]. Such high-level simulators allow rapid implementation of protocols and algorithms for measurement purposes, and can simulate topologies with many network nodes on a tractable time scale because they abstract away the low-level details a real, non-simulated implementation requires. In exchange for these two advantages, high-level simulators sacrifice certainty that real implementations behave as the simulated ones do. A study of eleven different TCP implementations [Paxson 97] found that many had unique traits, and several behaved significantly differently than the idealized, simulated TCP does. For

the heterogeneous and rapidly changing Internet environment, high-level simulators and tools for measurement of real implementations, like the LDS, are complementary.

By only simulating the length of fiber connected to the network interface, the LDS allows real implementations of TCP/IP to be tested in the simulation. The host computers must put the packets on and take packets off the network, thus including the interaction between network, the network interface, and the operating system in our simulation results.

BBN's Long-Link Emulator (LLE) [Milliken 1993] works at the SONET link protocol level, rather than the ATM layer. It is built on a SPARC workstation with custom VME cards to implement SONET. The range of delays supported by the LLE is 0.41 microseconds to 826 milliseconds in 0.82-microsecond (128-bit-time) increments. While the LLE's delay is adjustable, the operation of the LLE link must be disabled to change it; the LDS allows delay adjustment during operation. The LLE is made from expensive, custom hardware, and is special-purpose--its ATM interface cannot be used as a conventional host adapter.

Carnegie-Mellon's virtual port card [Kosak et al. 1997] implements delay and switch functions (such as flow control protocols) in commodity PC hardware. However, the virtual port card requires two host adapters (the LDS only requires one), and the virtual port card does not support full OC-3; it runs at a significantly reduced rate. Furthermore, the virtual port card does not preserve the time spacing of ATM data cells as they arrive on the link; because it discards null cells, it can only approximate data cell spacing by transmitting bursts of null cells to rate-match its input and output when it receives data at less than the link rate

4 Conclusion

We have presented an ATM link delay simulator (LDS), built from commodity PC hardware, that supports delays between 1.5 and 500 milliseconds for OC-3 ATM links. The LDS architecture preserves the precise ordering and spacing of delayed data cells by avoiding per-VC demultiplexing and by buffering null cells. This architecture requires 100-percent duty cycle, full-duplex OC-3 from the PCI ATM host adapter, at a total PCI bandwidth of over 300 Mbps. We meet this significant bandwidth requirement in our system by using DMA, tuning DMA buffer sizes, excluding the host CPU from the cell data path, and avoiding all cell or buffer scheduling computations on the host CPU. As a non-standard, high-performance ATM end system application, the LDS identifies host adapter design features important for application flexibility and performance, demonstrates the feasibility of such applications on hardware of moderate cost, and illuminates whole-system (hardware and software) ATM performance tuning. Other real-time systems also present streaming input, and require careful attention to buffer sizing, interrupt load on the host CPU, and bus performance; we have offered a detailed study of the LDS as one such system. We also have shown the value of the LDS for measuring real behavior of protocol implementations in the face of varying propagation delays.

References

- Digital Equipment Corporation, Otto ATM Interface Device Driver Source Code, <ftp://gatekeeper.dec.com/pub/DEC/SRC/AN2/otto.tar.Z>.
- Digital Equipment Corporation, Digital Semiconductor 21140A PCI Fast Ethernet LAN Controller, Hardware Reference Manual, Order Number EC-QN7NE-TE, November, 1996.
- Digital Equipment Corporation, METEOR PCI-ATM 155 Mb/s SAR Chip, Functional Specification, Revision 1.2, February, 1996.
- Endo, Y., Wang, Z., Chen, J.B., and Seltzer, M., Using Latency to Evaluate Interactive System Performance, Second Symposium on Operating Systems Design and Implementation (OSDI'96), October, 1996
- Floyd, S., Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic, ACM Computer Communication Review, 21(5):30-47, October, 1991.
- Gaynor, Mark, Karp, Brad, and Kung, H.T., A PC-Based ATM Link Delay Simulator, Simulation (SCSC'98).
- Intel Corporation, Pentium Processor Family Developer's Manual, Volume 3: Architecture and Programming Manual, Appendix H, 1995.
- Kosak, C., Eckhardt, D., Mummert, T., Steenkiste, P., and Fisher, A., Buffer Management and Flow Control in the Credit Net ATM Host Interface, Proceedings of IEEE INFOCOM '97, Kobe, Japan.
- Milliken, W., The Long-Link Emulator System Description, White Paper -- Advanced Networking Department, BBN Systems and Technologies, September, 1993.
- Paxson, Vern and Floyd, S. Why we Don't Know How To Simulate The Internet. In Proceedings of the 1997 Winter Simulation Conference (Atlanta, GA). IEEE, Piscataway, N.J.
- Paxson, V. Autoated packet trace analysis of TCP implementations, Proceeding of SIGCOMM'97. IEEE Piscataway, N.J.
- Romanow, A. and Floyd, S., Dynamics of TCP Traffic over ATM Networks, IEEE Journal on Selected Areas in Communications, 13(4), 1995.