January 1996

A New Look at Bandwidth Latency Tradeoffs

John H. Shaffer
University of Pennsylvania

Jonathan M. Smith
University of Pennsylvania, jms@cis.upenn.edu

Follow this and additional works at: http://repository.upenn.edu/cis_reports

Recommended Citation

This paper is posted at ScholarlyCommons. http://repository.upenn.edu/cis_reports/189
For more information, please contact libraryrepository@pobox.upenn.edu.
A New Look at Bandwidth Latency Tradeoffs

Abstract
Concerns about propagation delay have dominated the discussion of latency, bandwidth and their effect on distributed applications. In this paper, we argue that the relevant latency measure for applications is the Application Data Unit (ADU) Latency, defined as the time between the sending of an ADU and its receipt. Since ADUs are often large, ADU latency is influenced by throughput as well as propagation delay. We investigated the effects of ADU latency with an experimental study of several applications. The applications used Distributed Shared Memory as an interprocess communications mechanism, constraining the ADUs to page sized units. The applications were run on an Ethernet, an experimental ATM LAN, and using ATM on an experimental high-speed WAN. The measured results were used to normalize results gathered by inserting an experimental ATM switch output port controller in the network to create tunable delays. The results conclusively demonstrate the effect of ADU latency on distributed application response time. The experiments give a precise characterization of the effect of varying bandwidth and propagation delay on a real system, and suggest promising directions for further improving application performance in future networks.

Comments

This technical report is available at ScholarlyCommons: http://repository.upenn.edu/cis_reports/189
A New Look at Bandwidth/Latency Tradeoffs

John H. Shaffer and Jonathan M. Smith
Distributed System Laboratory
University of Pennsylvania
Philadelphia, Pennsylvania
January 24, 1996

Abstract

Concerns about propagation delay have dominated the discussions of latency, bandwidth and their effect on distributed applications. In this paper, we argue that the relevant latency measure for applications is the Application Data Unit (ADU) Latency, defined as the time between the sending of an ADU and its receipt. Since ADUs are often large, ADU latency is influenced by throughput as well as propagation delay.

We investigated the effects of ADU latency with an experimental study of several applications. The applications used Distributed Shared Memory as an interprocess communications mechanism, constraining the ADUs to page-sized units. The applications were run on an Ethernet, an experimental ATM LAN, and using ATM on an experimental high-speed WAN. The measured results were used to normalize results gathered by inserting an experimental ATM switch output port controller in the network to create tunable delays.

The results conclusively demonstrate the effect of ADU latency on distributed application response time. The experiments give a precise characterization of the effect of varying bandwidth and propagation delay on a real system, and suggest promising directions for further improving application performance in future networks.

1 Introduction

"There is an old network saying: Bandwidth problems can be cured with money. Latency problems are harder because the speed of light is fixed - you can't bribe God." - Dave Clark, quoted in [Hennessy 96].
As various technological advances have provided us with greater bandwidths, latency remains as a fundamental barrier limiting performance of distributed applications. Since latency is a function of propagation delay, they are often confused. In a networking architecture, latency at any layer is the time required to send a correctly received data unit to a logical peer at the same layer. Correctly discussing latency, then, requires specifying the architectural layer at which the latency is measured as well as the size of the data unit. The “Application Data Unit” (ADU) of [Clark 90] is a convenient unit, as when its latency is measured between applications, we get the closest correlation between a definition of “latency” and the observed response time of the distributed application. Propagation delay, in contrast, measures only the physical layer latency.

The latency between peers at Layer n in a protocol stack can be correctly represented by a recurrence on the layer number, \( n > 1 \), with \( n = 1 \) the physical layer:

\[
latency(n) = \frac{1}{throughput(n-1)} \times size(n-1) + latency(n-1)
\]

So, in this description, the latency between peers would be computed as the time between sending the first bit and the time that the last bit of the data unit is received. The product term represents the difference in time between the first bit to arrive and the last bit to arrive, which is added to the time for the first bit to arrive. This accurately captures the latency costs at each step in the motion of data through a protocol stack, and while ILP[Clark 90] and ALF were focused on improving throughput, the consequences of such techniques may include reductions in latency as well.

It is clear that propagation delay will dominate latency as bandwidths increase. It is also clear that there is a space of \( bandwidth \times delay \) tradeoffs, in which there are regions where increasing bandwidth causes reductions in the latency observed by ADUs, and consequently improves distributed application performance. Thus, there are in fact opportunities (albeit limited) where latency problems can be solved with money.

We investigated those opportunities. The experimental apparatus is described in the next section. The experimental methodology is then described in Section 3. We present the results of the experiments in Section 4, and confirm that bandwidth can effect an ADU latency reduction, leading to improved performance for a diverse set of experimental applications. Section 5 presents our conclusions and suggests how performance of distributed applications can be further improved by some changes in software architecture suggested by our performance study.
2 Experimental Configuration

In this section we describe the setting for the experiments.

2.1 Hardware and Network Infrastructure

The AURORA gigabit testbed[Clark 93] interconnected our site, Bellcore, IBM and MIT. The switched ATM infrastructure, based on the Sunshine[Giacopelli 91] ATM switch prototype, interconnected our site, Bellcore and MIT. The testbed hardware consisted of SONET OC-48 connected 4x1 multiplexors which provided multiple OC-12 paths between the sites. One set of paths interconnected the Sunshine ATM infrastructure, which required further demultiplexing[Lalk 92] to OC-3c rates. Experimental ATM host interfaces[Traw 93] connected IBM RS/6000 workstations to this network. At the time the WAN experiments in this paper were made, the Philadelphia to Morristown link, a geographic distance of slightly over 130 miles as the packet flies, was the only operational portion of the network.

The configuration used for the measurements is shown in Figure 1. It consists of three RS/6000 workstations connected to the testbed with an ATM host interface.
The host interface is connected to a pair of GLINK adapters which serialize and reparallelize signals from/to the host interface. This was needed because the parallel connection to the host interface is too susceptible to electrical noise to have a length of more than a few inches. Therefore one GLINK is near the workstation and the other is secured in the Bellcore Sunshine switch. The data is serialized and sent to the GLINK in the Sunshine switch where it is then reparallelized and fed into one of four the I/O ports on the switch. These ports may be configured via monitor commands; any input port may be connected to any output port. This versatility is important for troubleshooting as it enables switched loopback in addition to a hardware loopback at the GLINK adapter or a direct loopback on the host interface itself. An additional benefit of this versatility is that it allows a number of experimental test configurations which are discussed later in this section.

This experimental platform allows the collection of data and performance statistics from several different configurations:

1. Between hosts over a local 10Mb/s ethernet.
2. Between two hosts over 155Mb/s ATM LAN. (Loopback at our Sunshine switch.)
3. Between two hosts over a 272 mile (544 mile round-trip) 155Mb/s ATM WAN. (Using both of our hosts with the Bellcore Sunshine switch in loopback mode.)
4. Among several local ethernet participants with the addition of an ATM connected host which may be 1) local, or 2) 272 miles distant.

2.2 Software

AIX loadable device-drivers were used both for raw ATM support[Smith 93] and for supporting a TCP/IP overlay[Alexander 94]. The software subsystem was limited to about 70 Mbps of measured ttcp throughput over the OC-3c fabric due to extensive copying in the TCP/IP fragmentation and reassembly operations. As this software was experimental, there were several operational limitations, such as an inability to sustain multi-packet bursts over 32 Kbytes at full (130+ Mbps) throughput rates. This precluded using the most aggressive prefetching strategies in the higher levels of the implementation.

The application programming interface for our experiments was distributed shared memory (DSM)[Delp 94], which provides a simple and familiar interface for applications programmers while
Figure 2: "C" fragment to create and access a region of MNFS shared memory

```c
char * make_shmem(name) /* name: an MNFS mounted filesystem filename */
char *name;
{
  int flags = O_CREAT|O_RDWR;
  int fd, mode = 0777;
  int size = 1048576; /* 1 Megabyte ==> 256 4K pages */
  int *region_pointer;

  fd = open(name, flags, mode);
  return( mmap(0, size, PROT_READ| PROT_WRITE, MAP_SHARED, fd, 0) );
}

/* Get a pointer to a memory region using make_shmem() */

region_pointer = (int *) make_shmem( MNFS_FILENAME );
```

2.3 Measured properties of this environment

The actual round trip distance this connection must traverse is about 272 miles and has a measured ATM cell round-trip delay of about 2.1ms. Light travels at about 70% of its speed in a vacuum over this transmission system, and the delay was measured using a logic analyzer triggered by cell transmission and arrival events.

An IP packet obtains burst throughput of about 130 Mbps once it reaches the adapter. The ATM switch adds about 3 cell times of delay, and we see a small but noticeable rate of lost cells due
to CRC checksum failure. Since these are associated with switch cooling, we shut down the switch if the error rate became unacceptable and resumed measurement at a later time.

2.4 OPC-V2

In an effort to more fully explore the effects of latency on application performance it is useful to vary the latency on a network channel. This enables the experimental determination of the point at which the effects of latency or propagation delay overcome those of transmission delay. Marcus at Bellcore has designed and built an output controller for the Sunshine switch, version 2 (OPC-V2), a programmable, eight input, four output statistical multiplexer for ATM. The OPC-V2’s Intel 960 was programmed to delay ATM cells by a predetermined amount of time, emulating an ATM channel with known fixed propagation delay.

The delay emulation code for the OPC-V2 maintains a linked list of cells in its substantial cell buffer. The OPC-V2 keeps a global timer running with a resolution of one cell time; at 155Mb/s this is 2.8μs. When a cell arrives it is added to the tail of the linked list with an associated departure time and the first cell’s departure time is compared to the current timer value. On timer expiry, the cell is removed from the list and transmitted. Buffering is traded directly for delay.
3 Experimental Methodology

The Ethernet was used as a partitioned LAN, and it was idle except for the experimental MNFS applications. The Sunshine-based ATM LAN was about 20 feet in diameter, operating over the GLINK SONET emulation hardware and using UTP5 cabling. The Sunshine-based ATM WAN spanned 136 miles of distance for a 272 mile/2.1 ms round-trip delay. The OPC-V2 was used to vary the delay over a wide range calibrated by the measurements taken from the testbed.

There are a variety of ways to measure ADU latency. For example, remote procedure call[Birrell 84] (RPC) is a relatively straightforward way to obtain latency estimates. RPC latency has been aggressively reduced[Bershad 89], particularly for null RPCs. Data transport has been less of an issue in these designs, yet since larger application data units will be more affected by throughput, data transport is a major issue in investigating bandwidth x delay tradeoffs.

We chose to use distributed shared memory[Delp 94] (DSM) as our interprocess communication paradigm for the experiments. The advantages are as follows:

- Existing parallel applications can be easily ported to a networked setting.
- DSM exhibits low operating system effects on latency. This is partially due to heavy optimizations of VM operations by vendors.
- DSM allows straightforward measurement of page fault times, giving the latency for a fixed-size ADU.

Several applications were used to measure performance. The first measures the time for a series of MNFS page fault resolutions. This provides a good elementary metric for comparative performance. As applications vary in their page-fault frequency, page-fault access pattern, locality of reference, and ratio of computation to memory reference, two unlike applications were chosen to characterize real applications. This makes a more convincing statement about applications gains from reduced transmission time than reduced page fault satisfaction times. A heat equation solver was chosen to represent mathematical and scientific applications and a video-on-demand application was chosen to represent data intensive applications. Each of these is discussed below.
3.1 Page-Fault Exerciser

This application causes page faults and immediately invalidates the page after installation. This enables the precise measure of time required to resolve the fault. The application runs as a user process; data gathering is done via tracehooks. AIX[IBM 90] provides an extremely flexible fine-granularity tracing feature. Trace macros are called with zero to five word-length arguments, and are identified by a hookword. Tracing for hookword-specified events is selectively enabled and disabled. The trace macros test if tracing is active for the hookword, and if so, a record is written to a kernel trace buffer. When tracing is stopped, the buffer contents are dumped to a file, which is formatted by interpreting the hookword and printing any arguments to the macro found in the trace dump. When tracing is not active, the trace facility costs a conditional branch.

Page-fault statistics were gathered, including the time when a fault occurred and when it was resolved, and whether it was a read or write fault.

The test program loops, repeatedly touching a page in shared memory space and then invalidating it via msync(MS_INVALIDATE). The shared memory in this case consists of a total of 512 pages. Each of the pages is 4096 bytes giving a total of 2MB. The average page fault time once the page cache is warm is 6.9\text{ms}.

3.2 Heat Equation Solver

A parallelized heat equation solver for the Poisson equation is partitioned evenly among hosts in the computation. The number of machines thus selects the problem size allocated to each machine once the global problem size is set. The solution is based on iterative refinement of a linear system using a numerical “mesh”. The important parameters for problem size are the boundaries and the mesh size, which give rise to a matrix, solved using the Gauss-Seidel method.

It was originally coded using the PVM package[Geist 92]. Modifying the application for MNFS required, in addition to some minor restructuring, the implementation of the PVM functions \texttt{pvm\_send()} and \texttt{pvm\_recv()} to use MNFS as a communication platform.

The problem is extremely memory usage and computation intensive, since the computational complexity of matrix solution is at least quadratic in terms of the problem size and we used large problem instances. We also control the amount of communication by precise choice of problem dimensions, since messages are passed at boundary-crossings in the iterative solution.
3.3 Video On Demand

3.3.1 Basic Operation

The mpeg-play application processes a video stream stored in an MPEG encoded file. A local buffer is maintained which is filled with incoming MPEG data, which must be uncompressed and displayed. A low-water mark is used to refill the buffer from the file.

Because of the buffer mechanism employed, skipping around in the data stream becomes a non-trivial data management problem since the local buffer is merely a 64KB window into the entire data segment. Moving the window to randomly selected points becomes much more problematic as the window must be kept consistent with its position in the data segment. By accessing the data as memory, O.S. memory management transparently handles these issues.

3.3.2 MNFS Implementation

This application was easy to port to MNFS, and the memory model allowed further simplification of its behavior; data access was done using the shared memory space instead of a local buffer. The original code was modified so that once the shared memory space, containing the MPEG data is accessed a pointer to that space replaces the pointer to the local buffer. The low water mark, is irrelevant since the memory segment is the size of the MPEG data itself. This eliminated explicit file reads and low-water mark checking. Random access to the data stream is achieved by modifying a memory pointer.

Since video processing is highly data intensive as well as time sensitive, this application is very different in nature from the parallel heat equation solver.

4 Results and Analysis

We used the tools and applications we developed to measure the behavior of the system in various bandwidth and delay scenarios, and to gain some insight into causes and possible opportunities.

4.1 Throughput versus ADU latency

The applications were run repeatedly using the configuration of networks and machines shown in Figure 1. A key goal was reproducibility. The AURORA network subsegment was dedicated to this
application, and the Ethernet was lightly loaded, carrying mainly periodic interworkstation traffic generated by daemons. A most important hypothesis to test is the correlation between ADU latency and distributed applications performance. We tested this in a straightforward and convincing manner by measuring minimal page fault times as the OPC-V2 was used to vary the emulated propagation delay of the underlying network. Selecting minima ensures that we are measuring latency rather than scheduler or workload-induced contributions; the minima were extracted from runs with several thousand page fault events.

Figure 4 shows the minimal page times measured for 4KB pages; the delay refers to a point-to-point delay rather than the total round-trip. The delay was varied from 0ms to 5ms for this experiment, which would be 0 to 650 miles of distance. Each millisecond is equivalent to 130 miles of fiber distance; thus a 1 millisecond delay, where the ATM configured system outperforms the Ethernet LAN, represents the distance to Bell Communications Research from our site. The delay measured to Bellcore and back through the AURORA WAN was 2.1 milliseconds, or equivalent to 1.05 milliseconds on this plot. These measurements suggest that a significant opportunity for applications speedup exists in the lower left-hand region of the plot, where the ADU latency on the ATM network is less than that measured for our Ethernet LAN.

We address the question of the correlation between ADU latency and distributed applications response time for the heat equation solution application next.

4.2 Application Latency versus bandwidth

Figure 5 plots completion time for a large problem instance (1200 by 1000 elements) against the delay induced by the OPC-V2 for the ATM-connected host, showing the relationship between application performance and geographic distance.

The key observations to make are that the ATM solution outperforms the Ethernet solution, with the same problem on the same software on the same machines, for a variety of emulated distances. The measured completion times for the computation are consistent between the real and emulated environments. The equation governing this problem instance for this experimental configuration of an ATM-connected station is:

Completion time in seconds = 44.8 + 0.6 × (ATM delay in ms)

This compares to the unloaded Ethernet case of 45.7 seconds.
Figure 4: Minimum Page Fault times vs. propagation delay, Ethernet & ATM

Figure 5: Performance of distributed heat equation solution, DSM/ATM.
What is clear from a visual comparison of Figures 4 and 5 is that the shape of the plots are quite similar, and the application's response time demonstrates the realization of application speedup in the bandwidth x delay region we identified in Figure 4.

A final concern is the broadness of the applications domains in which the opportunity of speedup can be exploited. We took measurements using the mpeg_play tool and measured performance using the metric of displayed frames per second over video. The results are plotted in Figure 6. As can be seen, the region for performance improvement, that is, less than about 1 millisecond delay, again shows improvement relative to the Ethernet LAN. The somewhat more gradual reduction in performance as latency is increased in this application is due to the pipelined nature of data stream access, which is less directly affected by ADU latency than the heat equation solver.

4.3 Components of application data unit latency

Isolating the factors contributing to ADU latency was done via kernel tracing. Table 1 shows the times in μsec for various steps in the page fault resolution process on an MNFS system. The percentages of total page fault time are indicated in the rightmost column.

The Ethernet case shows that the software overhead is significant for a 4KByte page fault. The average page fault time we have measured in this configuration is 6765 μsec. The page is transmitted
over the Ethernet as IP packets encapsulated in Ethernet frames: these are 1514, 1514 and 1278 bytes in length and represent (4306-4096) or 210 bytes of overhead. The transmission delay for the 4306 bytes was 3445 μsec, representing 50.9 % of the total time measured. (It is interesting to note that any software latency increases would benefit distributed applications operating over an Ethernet as well – there appears to be plenty of opportunity for improvement!).

<table>
<thead>
<tr>
<th>Description</th>
<th>μsec</th>
<th>%-age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Traversal from Page Fault till page request arrives at network interface driver:</td>
<td>317</td>
<td>4.7</td>
</tr>
<tr>
<td>2 Output Request (194 bytes)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Output Interface Overhead:</td>
<td>102</td>
<td>1.5</td>
</tr>
<tr>
<td>- Transmission Delay</td>
<td>155</td>
<td>2.3</td>
</tr>
<tr>
<td>3 Server Overhead:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Interface &amp; driver overhead:</td>
<td>320</td>
<td>4.7</td>
</tr>
<tr>
<td>- MNFS acquisition of page &amp; Delivery to server network interface:</td>
<td>1820</td>
<td>26.9</td>
</tr>
<tr>
<td>4 Transmission Delay of 4306 bytes:</td>
<td>3445</td>
<td>50.9</td>
</tr>
<tr>
<td>5 Input Network Interface &amp; driver Overhead:</td>
<td>218</td>
<td>3.2</td>
</tr>
<tr>
<td>6 Give page to VMM and Install Page in segment:</td>
<td>319</td>
<td>4.7</td>
</tr>
<tr>
<td>7 Return from fault</td>
<td>69</td>
<td>1.0</td>
</tr>
<tr>
<td>TOTAL</td>
<td>6765</td>
<td>99.9</td>
</tr>
</tbody>
</table>

Table 1: Breakdown of Page Fault

The ATM trace was for page faults averaging 2977 μsec, and there is a clear reason for the performance increase: throughput. On the OC-3c fabric, 3445 μsec versus 252 μsec gives a ratio of 13.7, or within 5% of the throughput ratio between Ethernet and ATM. While the ratio between software overhead and throughput on the Ethernet is about 1-1, the 2725 μsec versus 252 μsec ratio is about 11-1, indicating significant room for improvement.
5 Conclusions and Opportunities for improved performance

While it remains true that propagation delay is a fundamental limit on latency, we demonstrated that ADU latency, more relevant to applications performance, can be significantly improved with greater throughput. Our measurements of page-fault times showed how closely they can reflect the effect of ADU latency on distributed applications.

Our experimental apparatus suffered from a number of performance limitations; in spite of these we were able to exhibit comparable performance between applications running on a campus Ethernet LAN and the same applications operating on an OC3c-rate ATM WAN spanning over 100 miles. This performance was exhibited across diverse applications; the heat equation solver is a computation-intensive scientific application, and the data-intensive MPEG player uses the DSM to support data streaming.

The consequence is that high bandwidth networks can reduce delay in many environments simply by reducing the latency component associated with throughput. This is especially true of data objects of a large enough size to be affected by throughput considerations - for example virtual memory page sizes are typically 4096 or 8192 bytes. Our performance study showed that software overhead contributed more than half of the total ADU latency. Recent research[vonEicken 95], while providing restrictive demultiplexing capabilities, has demonstrated methods for reducing this latency, and thus the benefits of throughput on distributed applications can be more pronounced. Since much of the software overhead is paid on a per-packet basis[Clark 89], throughput will have more of an effect on applications with larger ADUs than our VM pages, and with increasing use of image data in systems such as the WWW, there will be an increasing opportunity.

6 Acknowledgments

W. S. Marcus of Bellcore designed, implemented and installed the OPC v2 in our laboratory, as well as keeping the AURORA network alive for these experiments. Brendan Traw designed, implemented and maintained the experimental ATM interfaces used for this research, and D. Scott Alexander developed the IP/ATM implementation used by MNFS; Ron Minnich helped greatly with the porting of his MNFS Distributed Shared Memory System from SunOS to AIX.
References


