SurfBoard – A Hardware Performance Monitor for SHRIMP

Princeton University Technical Report TR–596–99

Scott C. Karlin, Douglas W. Clark, Margaret Martonosi†

Princeton University
Department of Computer Science
†Department of Electrical Engineering
Princeton, NJ 08544

{scott, doug}@cs.princeton.edu, mrm@ee.princeton.edu

March 2, 1999

Abstract

Growing complexity in many current computers makes performance evaluation and characterization both increasingly difficult and increasingly important. For parallel systems, performance characterizations can be especially difficult to obtain, since the hardware is more complex, and the simulation time can be prohibitive.

This technical report describes the design, implementation, and case studies of a performance monitoring system for the SHRIMP multicomputer. This system is based on a hardware performance monitor which combines several features including multi-dimensional histogram generation, trace generation, and sophisticated triggering and interrupt capabilities.

Demonstrated in the case studies is the direct measurement of an implicit form of interprocessor communication implemented on the SHRIMP multicomputer.

1 Introduction

This technical report describes a performance monitoring tool we have developed for the SHRIMP system and reports how we have used this tool to gain new insights in to the performance of several benchmark applications.

The SHRIMP (Scalable High-performance Really Inexpensive Multi-Processor) project at Princeton studies how to provide high-performance communication mechanisms in order to integrate commodity desktop computers such as PCs and workstations into inexpensive, high-performance multicomputers [4, 5].

While there have been several incarnations of SHRIMP, this technical report applies only to the SHRIMP-II (hereafter referred to simply as "Shrimp") system. A key feature of the Shrimp system is an implicit communication mechanism known as *automatic update* which makes software based monitoring methods ineffective.

The SurfBoard (Shrimp Usage Reporting Facility) is designed to measure all communication with minimal impact to the system. Additionally, the SurfBoard provides several interesting features which can provide real-time feedback to specialized applications designed to take advantage of performance information about the system as it operates.

After giving an overview of the Shrimp hardware in Section 2 and the SurfBoard design in Section 3, we describe several case studies (both "raw" data and analysis) in Section 4. A hardware retrospective appears in Section 5. We discuss related work in Section 6 and give conclusions in Section 7. Appendix A gives details of the hardware and Appendix B gives details of the software.

2 The Shrimp Multicomputer

The Shrimp multicomputer nodes are unmodified Pentium PC systems, each configured with standard I/O devices such as disk drives, monitors, keyboards and LAN adaptors. The primary interconnection network is the Intel Paragon mesh routing backplane [19]. The connection between a network interface and the routing backplane is via a simple signal-conditioning card and a cable. As each node contains a commodity Ethernet card, there is a secondary interconnection via standard IP protocols.

A main goal of Shrimp is to provide a low-latency, high-bandwidth communication mechanism whose performance is competitive with or better than those used in specially designed multicomputers. The Shrimp Network Interface (SNI) board implements virtual memory-mapped communication to support protected, user-level message passing, and fine-grained remote updates and synchronization for shared virtual memory systems. Shrimp supports a variety of programming styles by supporting communication through either *deliberate update* or *automatic update*. With deliberate update, a processor sends out data using an explicit, user-level message-send command. With automatic update, a sending process can map memory within its address space as a send buffer; any time the sending process writes to one of these mapped (outgoing) memory regions, the writes are propagated automatically to the virtual memory of the destination process to which it is mapped.

Figure 1 shows a Shrimp multicomputer prototype system. The highlighted components in the figure correspond to the experimental system components being designed and implemented at Princeton. The right hand side of the figure focuses in on a single Shrimp compute node, a standard PC system. The nodes interface to the Paragon backplane with a custom designed SNI board. Packet data received by an SNI board is forwarded to a SurfBoard (SURF) for data measurement. The direct connection from the Xpress memory bus to the SNI board allows the SNI to snoop memory references; it is this snooping which enables the SNI board to perform the automatic update communication mechanism.

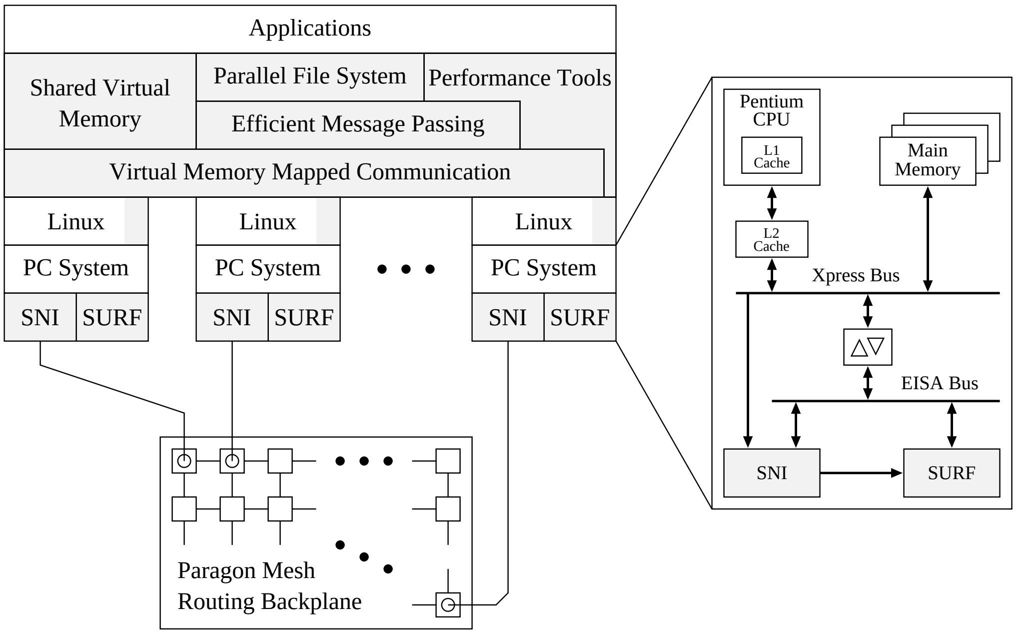

Figure 1: An overview of the components of the Shrimp system.

2.1 VMMC Communication Model

Virtual memory-mapped communication (VMMC) is a communication model providing direct data transfer between the sender's and receiver's virtual address spaces [10]. The idea is to separate the mapping setup from the data transfer. The Shrimp system uses the LAN to setup the mappings and the Paragon backplane for data transfer. A key feature of VMMC is that the there is no explicit receive function. Data is transparently delivered into the memory space of the destination node. This means that the receiver of the data does not know when the data has arrived.

2.2 Automatic Update

Automatic update allows pages in the sender to be mapped to pages in the receiver so that ordinary memory writes on the sender are propagated through the network to the receiver. Under automatic update, neither the sender nor the receiver explicitly transfer data. By its very nature, instrumenting an automatic update based system using only software would likely be very intrusive. One could imagine periodically comparing the contents of receive buffers with backup copies of the data. However, even this approach could not easily identify the packet sender.

2.2.1 Combining

With automatic update there needs to be a policy decision as to when to send a packet. One could eagerly send a separate packet for every memory access, or one could wait and combine multiple writes into a larger packet before sending the packet. In the later case, one must also decide when to stop combining.

The SNI board supports combining of automatic update memory writes (which are snooped from the Xpress bus). Combining stops and the packet is completed when any of the following occur:

- Combining is disabled,
- A read memory access is snooped,
- A non-sequential write memory access is snooped (i.e., the address is not one more than the previous address),
- A user-defined address boundary is crossed (either 256, 512, 1024, 2048, or 4096 bytes), or
- An optional timeout is exceeded.

3 SurfBoard

In a fully configured Shrimp system, a SurfBoard is located at each Shrimp node, and captures information at the arrival of incoming packets. Figure 2 shows a block diagram of the SurfBoard. (A description of the block diagram is in Appendix A.1 on page 49.) The board responds to user commands (e.g., start, stop, etc.) encoded as standard EISA bus writes. Once the monitoring has begun, the SNI board sends the SurfBoard a copy of each raw packet as it is received, and the monitor parses the raw packet data to extract the fields of interest. It then updates its statistics memories appropriately, and waits for the next packet. Control circuitry arbitrates access requests for the statistics memories among incoming packets, EISA cycles, and local DRAM refresh. Like some previous performance monitors (e.g., [12]) we have a flexible hardware design based on FPGAs, but we have designed mechanisms into the monitor for *runtime* flexibility as well. The subsections below discuss some of the key features in more detail.

3.1 Data Collection Modes

The SurfBoard has three independently data collection modes: *word-count mode*, *histogram mode*, and *trace mode*. The three modes are independently controlled and can operate simultaneously.

In word-count mode, the SurfBoard keeps a running count of the number of 32-bit payload words received by the node. The count is kept in a 32-bit register. When the count rolls over to zero, the node CPU can be notified via a maskable interrupt.

In histogram mode, the SurfBoard increments a count in a 32-bit histogram bin associated with a selected set of packet characteristics. Section 3.2 describes which packet characteristics can be combined to form a histogram address. When the count in a bin rolls over to zero, the bin address is placed in a FIFO, and

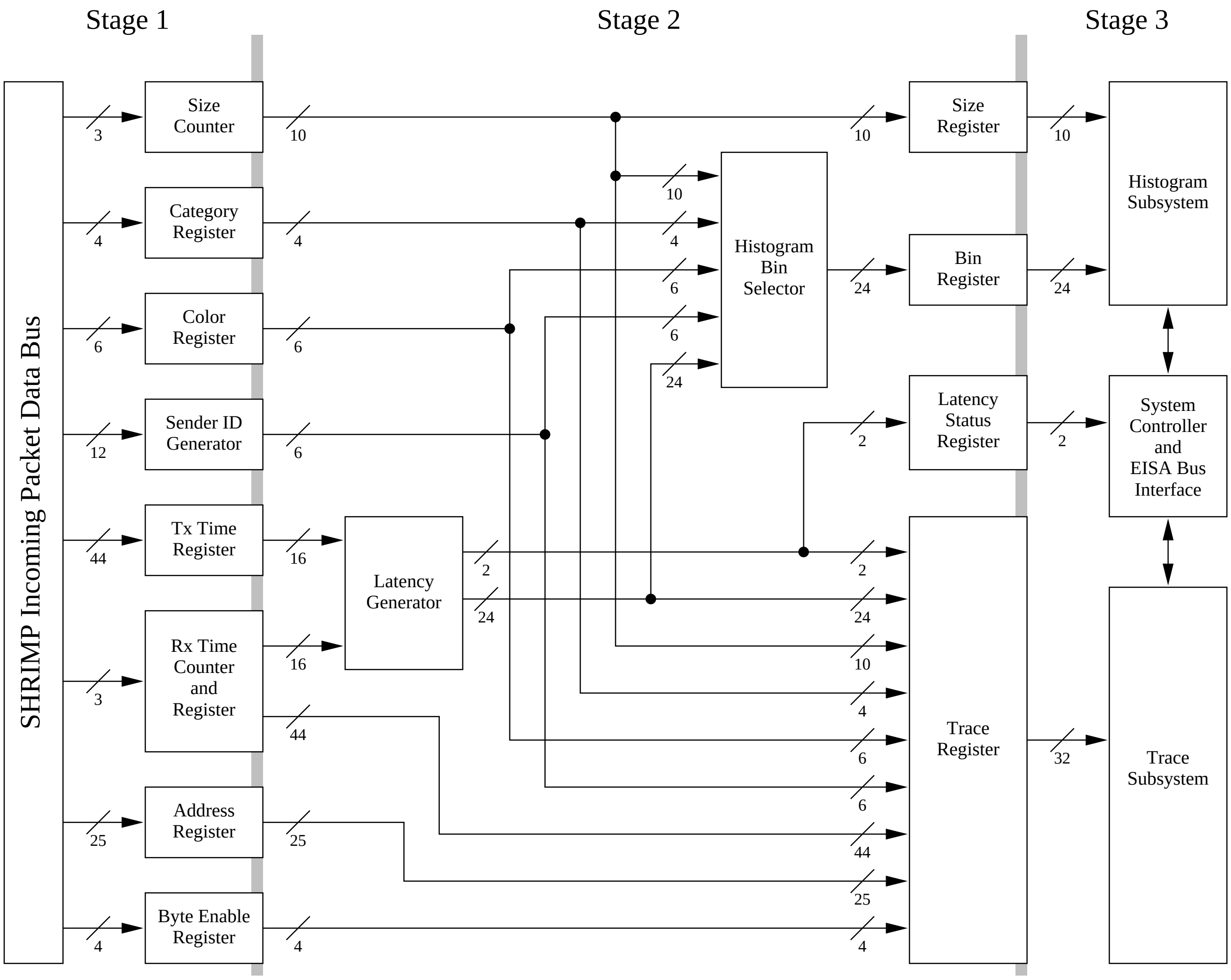

Figure 2: SurfBoard Block Diagram.

the node CPU can be notified via a maskable interrupt. Additionally, the node CPU can be notified via a maskable interrupt based on various FIFO conditions including “not empty”, “almost full”, and “full”.

At the end of an experiment the memory contains a histogram of the various combinations of values taken on by the packet characteristics. By using more than one characteristic to form the address, the result is a “multi-dimensional” histogram showing the joint distribution of the packet characteristics. The Histogram Bin Selector in Figure 2 is implemented as a simple interconnect using a single user configurable field programmable gate array (FPGA). Because the FPGA simply maps input wires to output wires with no intervening logic, a new design can be compiled in under 60 seconds and then downloaded in a few milliseconds without changing any other board state.

In trace mode, the SurfBoard records all collected information about each packet in the 128-bit wide trace memory. As described in Section 3.6, we also provide a mechanism to determine when tracing starts and stops.

For all three modes, the SurfBoard’s design is facilitated by its relatively modest speed requirements: in the Shrimp system the minimum packet rate (start word to start word) is 1080 nanoseconds and the minimum inter-packet time (end word to start word) is 240 nanoseconds, so that inexpensive dense DRAMs are used for the histogram and trace memories.

3.2 Measurable Variables

The FPGA-based selection scheme lets experimenters select arbitrary bits from five variables for histogram address selection. These variables are:

- **Size:** As the packet data arrives, a 10-bit counter tracks size of the payload in units of 32-bit words.
- **Category:** User-level software can set a 4-bit category register on the monitor board. This allows users to attribute statistics gathered by the hardware back to the responsible software constructs.
- **Color:** The SurfBoard has also been designed to provide support for coarse-grained data-oriented statistics, by allowing different pages in memory to be assigned different "colors" (in the form of a 6-bit page tag), which can then be used in histogram statistics. These 6-bit tags[1] are stored in incoming page table entries on the SNI board; they are read out at the same time as the page-mapping information [5]. Updating page tags requires (protected) memory-mapped I/O writes of the appropriate locations in the page tables.
- **Sender ID:** The 12-bit identity of the sending node from the packet header is mapped to a 6-bit value based on a SurfBoard configuration register.
- **Latency:** Each SNI board and each SurfBoard maintains a 44-bit global clock register.[2] These are controlled by system-wide 10 MHz clock signals distributed by the Paragon backplane. At the sending node, the SNI board inserts its copy of the global clock into each outgoing packet as a timestamp. Note that the sending timestamp value is determined when the first word of the packet is written. With combining enabled, it may be some time before the packet is finally sent. At the receiving node, the performance monitor reads its copy of the global clock as the packet arrives, and subtracts the packet's timestamp from it; this yields the packet's end-to-end hardware latency in 100 nanosecond cycles. The Latency Generator in Figure 2 performs this subtraction, scales the value, and limits the range of values to fit into a 24-bit value. Values greater than the range or less than the range, force the latency to all 1's or all 0's, respectively.

In addition to the above metrics, the trace memory also records the following additional variables:

- **Latency Flags:** Overflow and underflow.
- **Receive Time:** 44-bit receive timestamp.
- **Interrupt Bit:** Indicates if the interrupt bit in the incoming page table was set.
- **Packet Address:** Bits 26..2 of the packet address. (Bits 31..27 are zero for the Shrimp system.)
- **Byte Enables:** The 4 byte enables for this packet. These only have meaning if the packet size is a single word.

3.3 External Interface

The SurfBoard's external I/O connector supports a single output bit and a single input bit. The output can be configured to follow the external input value, a bit in a control register, or the histogram bin overflow condition. The input can be read from a status register; additionally, an event can trigger the trace function. By interconnecting the external interfaces among several SurfBoards, all boards can trigger on the same event. This combined with a global timestamp allows system wide traces to be interleaved and correlated.

[1] The current version of the SNI board has the top two bits hardwired to "01" limiting the color to 4 bits.

[2] Actually, the SNI board maintains a 45-bit register; the SurfBoard ignores the highest bit. This does not cause a problem as the SurfBoards will interrupt their host CPUs when their 44-bit count rolls over to zero.

3.4 Multiplexing Variables for Flexible Monitoring

Multi-dimensional histograms allow the experimenter to build up a two, three, four or even five-dimensional array of counters in the histogram memory. For example, one might want to collect histogram statistics using both packet size and packet sender, to yield the joint frequency distribution of these two variables. A correlation between the two might indicate that a particular Shrimp node tends to send larger packets on average.

3.5 Trace Modes

The trace subsystem supports three trace modes. These are defined by the relative location of the trigger packet *after* the trace completes. Trigger events are described in the next section. The modes also determine what actually starts and stops the tracing of packet data. The modes are:

- **Beginning**: In this mode, the trigger starts the trace which runs until the trace memory is full. This has the effect of capturing packets which follow the trigger event.
- **End**: In this mode, the trace runs continuously (treating the trace memory as a circular buffer). When a trigger occurs, the trace immediately stops. This has the effect of capturing packets which lead up to the trigger event.
- **Middle**: In this mode, the trace runs continuously (treating the trace memory as a circular buffer). When a trigger occurs, the trace continues until a count of packets equal to half the buffer size are received. This has the effect of capturing packets both before and after the trigger event.

3.6 Trigger Events

For performance monitors to be useful in on-the-fly monitoring, they must be able to take action and/or interrupt the CPU on certain "interesting" events. Without this ability, higher-level software would have to query the monitor periodically to determine its status. In a histogram-based monitor, a particular "event" may involve a large memory region; for instance, detecting packets from a particular sender might mean checking thousands of bins with that sender's address. Thus, reading the histogram locations intermittently throughout the run of a parallel application can often be both time-consuming and disruptive.

The SurfBoard provides efficient support for selective notification. With histogram mode enabled, the monitor has a interrupt feature that notifies the CPU when a count rolls over to zero. By "preloading" a selected bin with a count, an arbitrary threshold is set. The SurfBoard also saves the histogram address that caused the interrupt in a FIFO memory. The software interrupt handler can read the overflow address FIFO to see which bin(s) caused the interrupt, in order to decide how to react. In extreme situations, threshold overflow interrupts can occur faster than they can be handled. We limit this problem by allowing individual threshold interrupts to be masked (the address is still stored in the FIFO) and then interrupting the host when the FIFO has reached a programmable high-water mark. This allows the software to quickly read several overflow addresses during a single interrupt. These low-level mechanisms can be used by higher-level application or operating system policies to take software action in response to monitored behavior.

The monitor can also be configured to use *triggered event tracing*. That is, when the monitor detects a threshold event, it not only signals an interrupt, but also can trigger the trace mode automatically. This feature allows for long periods of histogram-based monitoring, followed by detailed tracing once a particular condition is detected. Triggered tracing and threshold-based interrupts can both be used in several interesting ways, including providing on-the-fly information to running software, or capturing error traces once an error condition is detected.

3.7 Performance Monitor Control Software

Finally, the SurfBoard responds to a set of low-level commands that come in via memory-mapped I/O writes on the EISA bus. These commands provide a basic library of efficient access routines on top of which higher-level tools can be built. The command set includes operations to start and stop monitoring, access histogram or trace memory, adjust the resolution of the packet latency, access the category and threshold registers, and configure the histogram bin selector FPGA.

The SurfBoard's EISA interface sees the commands, accepts the data, and acts on that data to update its control registers and read or write DRAM as needed. When commands are generated by the local node, they incur the overhead of only a single EISA bus write. For many monitoring uses (such as initialization and post-experiment reading of the histogram memory) this overhead is quite tolerable.

To perform a monitoring experiment, user-level software running on the PC initializes the histogram memory (often to all zeroes, but not always) by supplying addresses and data via the monitor's EISA bus interface. The *category* registers are also initialized, as are the bits that enable the word-count, histogram, and trace modes, as well as the trigger configuration. After this initialization, user software can start the monitor by issuing a special EISA write, and can similarly stop the monitor and read DRAM locations using different EISA operations.

Because we expect the SurfBoard to support experiments which run for long periods of time, the complete SurfBoard state (with the exception of the FPGA bitstream) can be saved and restored. This is a key feature when checkpointing long lived runs.

3.8 Effects of Measurement on Shrimp

The memory impact of the SurfBoard software is small. The kernel module which implements the device driver currently uses 3 pages (4096 bytes each) which is small compared to the Shrimp driver module which uses 311 pages.

An application which needs to the control the SurfBoard (to turn on and off measurements) must link with the surf library which is currently 2138 bytes.

The SurfBoard design requires that the optional Shrimp timestamps are always enabled. According to [3], enabling timestamps has very little impact on latency (at most a 1% increase for the smallest packets). However, it does occupy an additional 64-bit position in the SNI board's outgoing FIFO memory. Since the minimum sized packet is two 64-bit words (consisting of a 64-bit header and a payload containing 64-bit trailer), adding a timestamp represents a 50% increase in the packet size. If an application sends minimum sized packets fast enough to fill (or nearly fill) the outgoing FIFO, the additional overhead of the timestamp might have a noticeable impact.

Assuming an application did not want to lose any information, a lower bound for the maximum interrupt rate is due to the roll over of the word count register. At the theoretical peak EISA bus rate of 8.3 Mword/sec, the word count register will roll over and generate an interrupt every 9 minutes. In practice, this rate will be much lower.

4 Case Studies

This section describes several experiments which we performed using the SurfBoard.

4.1 Experimental Setup

Our setup consists of four nodes with one instrumented with a SurfBoard.

4.2 Benchmarks

Each of the four parallel benchmark programs use a shared virtual memory (SVM) model. This SVM implementation is known as Automatic Update Release Consistency (AURC) and is described in [13]. The programs were instrumented to trace all packets in the parallel phase of the program after an initial startup phase.

In addition, each of the programs were compiled to create both "combining" and a "no combining" versions. When combining is enabled, the maximum combining is allowed (up to 1024 words with no timeout) for shared writes. Note that at release time (e.g., barriers), AURC uses deliberate updates to propagate page updates to other nodes. These deliberate update packets are, in general, larger than 1 word.

Detailed descriptions of the LU, OCEAN, and RADIX programs can be found in [20].

4.2.1 LU

The *LU* kernel factors a dense matrix into the product of a lower triangular and an upper triangular matrix. To enhance data locality and to avoid false sharing, we used a version with an optimized data structure for the matrix, namely a 4-d array, in which each 32×32 element block is allocated contiguously in virtual memory and the blocks assigned to a processor are also allocated contiguously. Barriers are used to ensure synchronization between the processor producing the pivot row blocks and the processors consuming them, as well as between outermost loop iterations. This kernel exhibits coarse-grain sharing and low synchronization to computation frequency, but the computation is inherently unbalanced.

4.2.2 OCEAN

The *OCEAN* fluid dynamics application simulates large-scale ocean movements. Spatial partial differential equations are solved at each time-step of the program using a restricted Red-Black Gauss-Seidel Multigrid solver. Ocean is representative of the class of problems that are solved using regular grid near-neighbor computations. At each iteration the computation performed on each element of the grid requires the values of its four neighbors. Work is assigned to processors by statically splitting the grid and assigning a partition to each processor. Nearest-neighbor communication occurs between processors assigned adjacent blocks of the grid.

4.2.3 RADIX

The *RADIX* program sorts a series of integer keys into ascending order. The dominant phase of RADIX is the key permutation phase. In RADIX a processor reads its locally-allocated n/p contiguous keys from a source array and writes them to a destination array using a highly scattered and irregular permutation. For a uniform distribution of key values, a processor writes contiguous sets of $n/(rp)$ keys in the destination array where r is the radix used; the r sets a processor writes are themselves separated by $p-1$ other such sets, and a processor's writes to its different sets are temporally interleaved in an unpredictable way.

4.2.4 SIMPLE

The *SIMPLE* program is a micro-benchmark which performs writes to a single page mapped to another processor (the "next" processor in a ring). The writes are addressed and paced so that different sized packets will be generated by the underlying hardware. The implementation is an outer loop which counts the number of groups. The inner loop writes to consecutive addresses (giving a chance for combining) up to a user specified group size. The pacing is maintained by computing a target "time-to-wait". (Time is based on the Pentium cycle counter.) After a write to an automatic update mapped location, SIMPLE enters a tight loop waiting until the target time. It then calculates the new target time (the previous target time plus the user specified wait time) and continues.

If interrupt servicing causes the pace to fall behind schedule, writes will occur at the maximum rate (limited by software overhead) until it gets back on schedule.

For the experiments performed for this technical report, the outer loop count was 512, the inner loop group size was either 1 word or 1024 words, and the wait time was either 1 or 100 microseconds.

4.3 Raw Data

This section describes the experiments in detail and presents various graphs derived from trace data involving packet latency, packet size, and receive time (relative to the receive time for the first packet in a given trace).

4.3.1 LU, Combining

This section shows the results of 5 runs of the LU benchmark with combining enabled.

Figures 3–7 show the packet latency distributions.

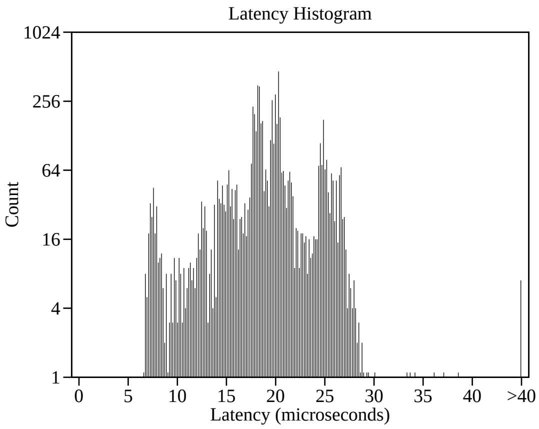

Figure 3: LU, Combining, Run 1

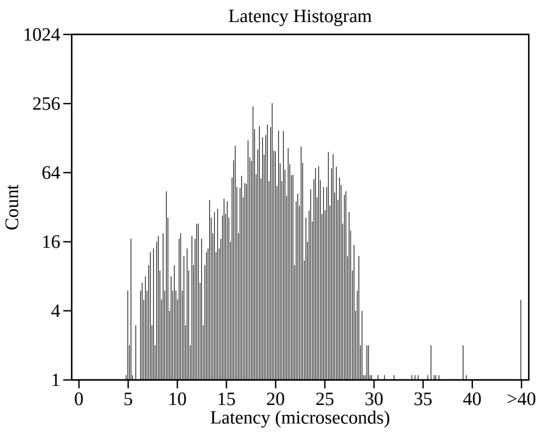

Figure 4: LU, Combining, Run 2

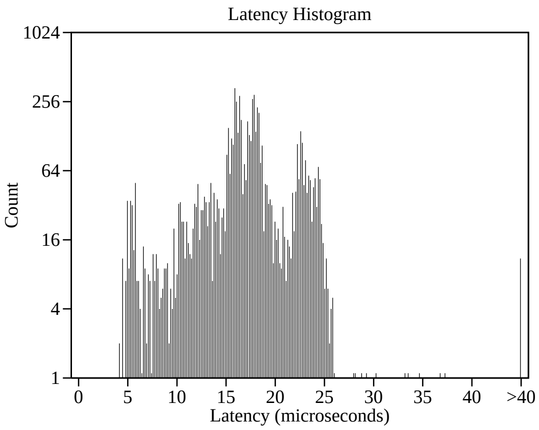

Figure 5: LU, Combining, Run 3

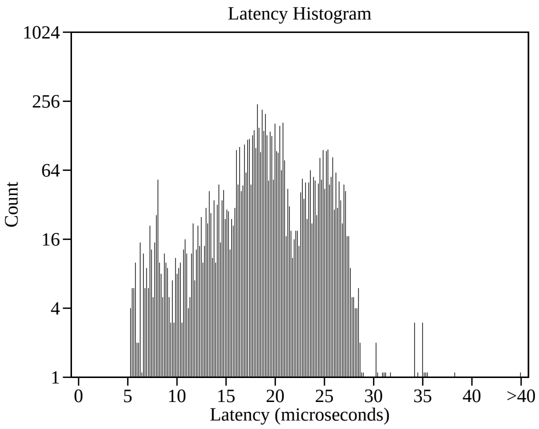

Figure 6: LU, Combining, Run 4

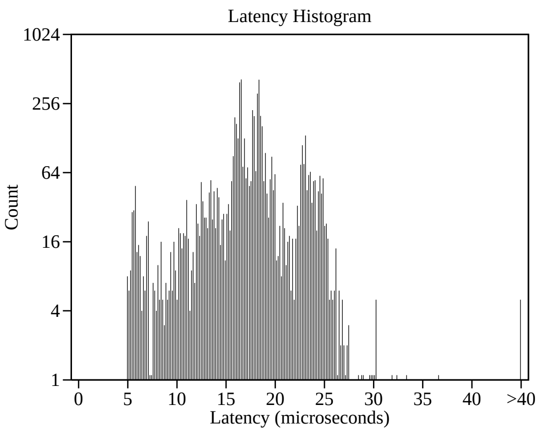

Figure 7: LU, Combining, Run 5

Figures 8–12 show the packet size distributions.

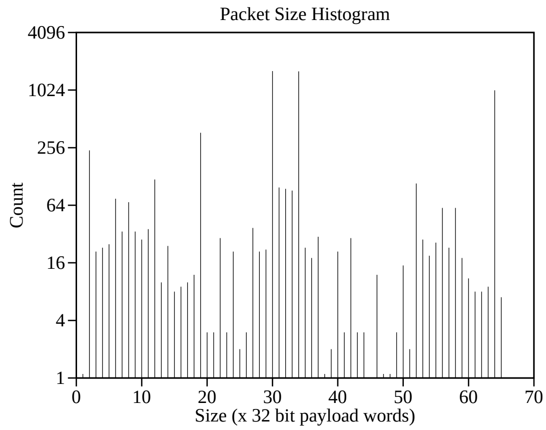

Figure 10: LU, Combining, Run 3

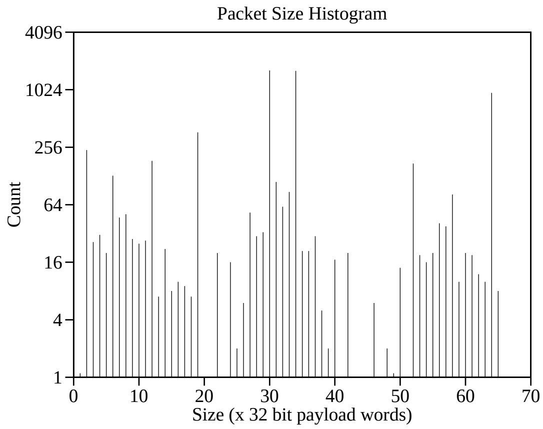

Figure 8: LU, Combining, Run 1

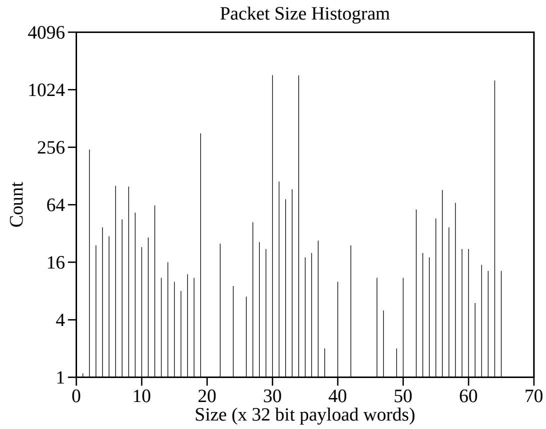

Figure 11: LU, Combining, Run 4

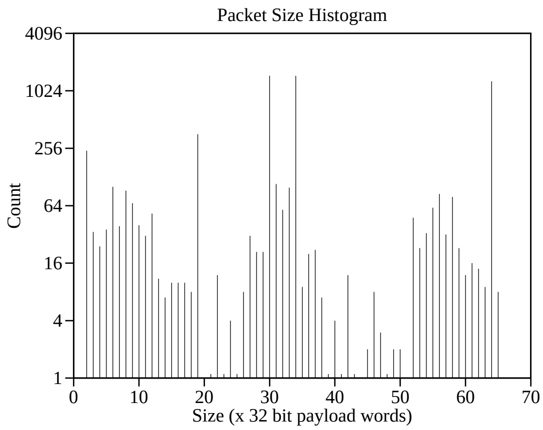

Figure 9: LU, Combining, Run 2

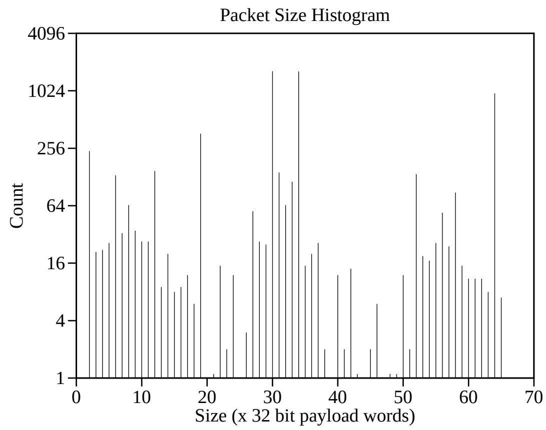

Figure 12: LU, Combining, Run 5

Figures 13–17 show graphs of latency versus time.

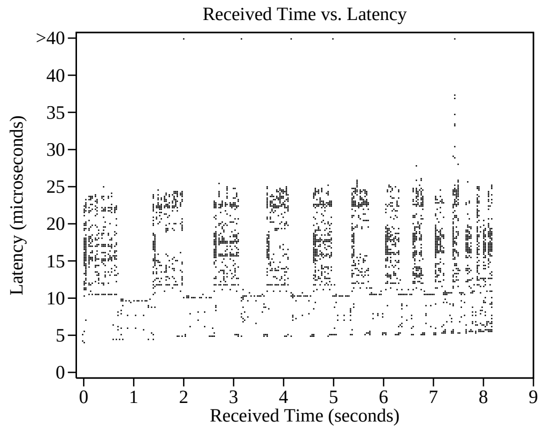

Figure 15: LU, Combining, Run 3

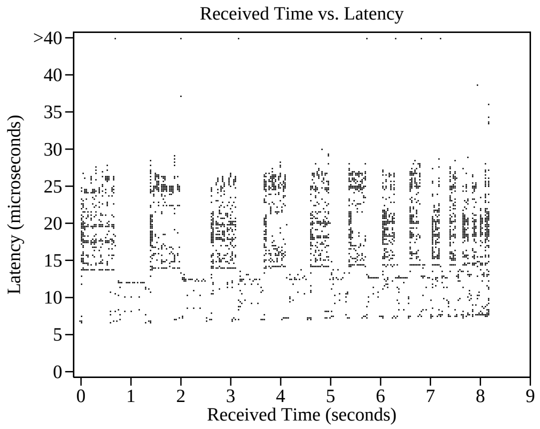

Figure 13: LU, Combining, Run 1

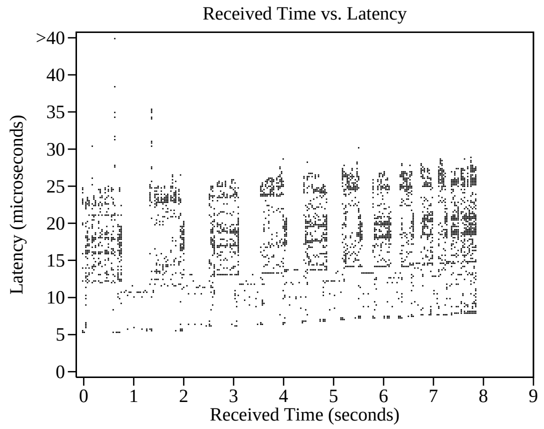

Figure 16: LU, Combining, Run 4

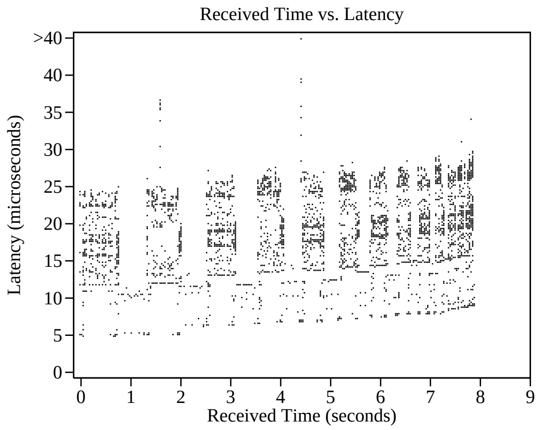

Figure 14: LU, Combining, Run 2

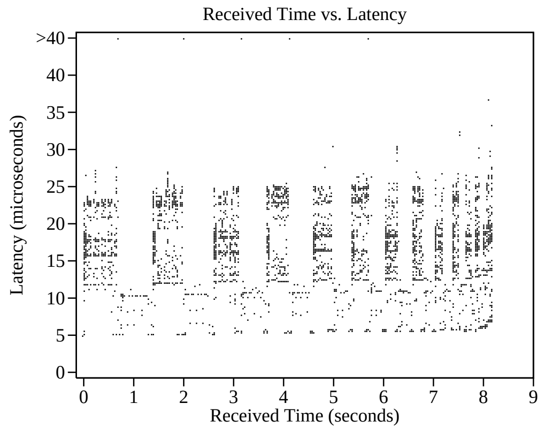

Figure 17: LU, Combining, Run 5

4.3.2 LU, No Combining

This section shows the results of 5 runs of the LU benchmark with combining disabled.

Figures 18–22 show the packet latency distributions.

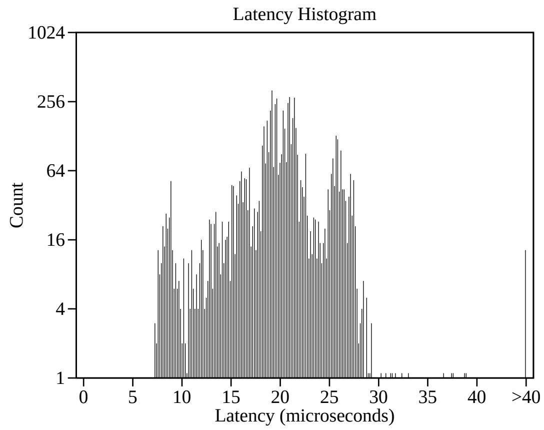

Figure 20: LU, No Combining, Run 3

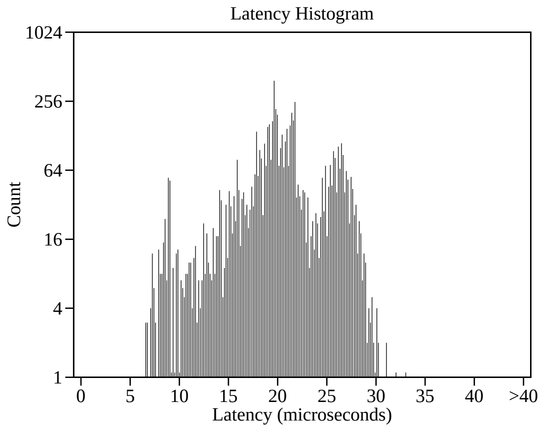

Figure 18: LU, No Combining, Run 1

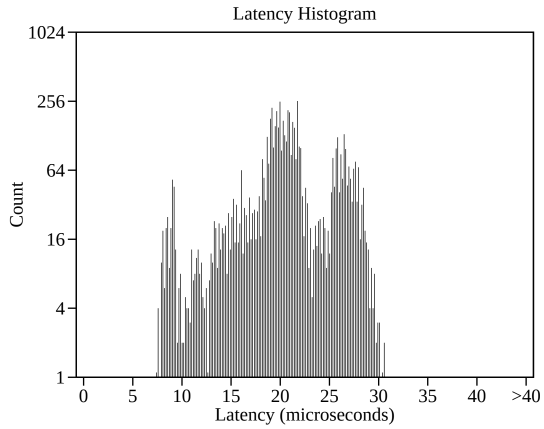

Figure 21: LU, No Combining, Run 4

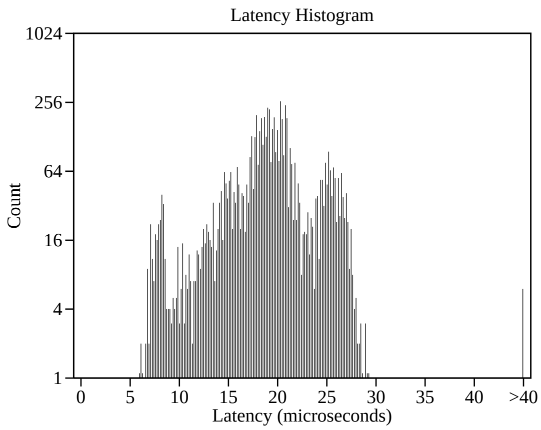

Figure 19: LU, No Combining, Run 2

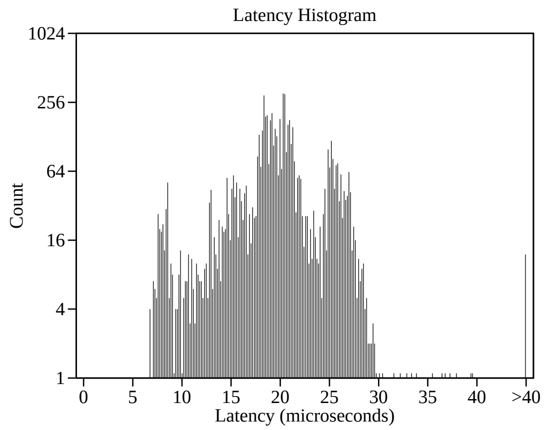

Figure 22: LU, No Combining, Run 5

Figures 23–27 show the packet size distributions.

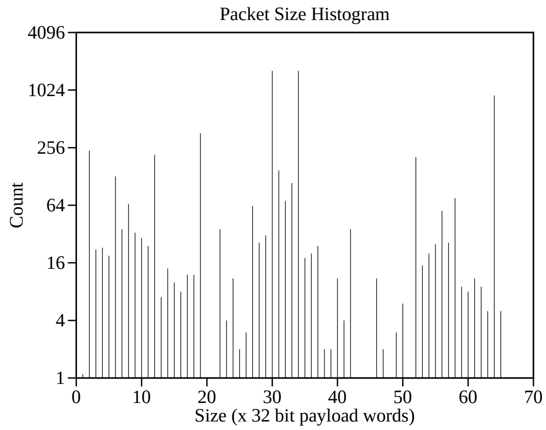

Figure 25: LU, No Combining, Run 3

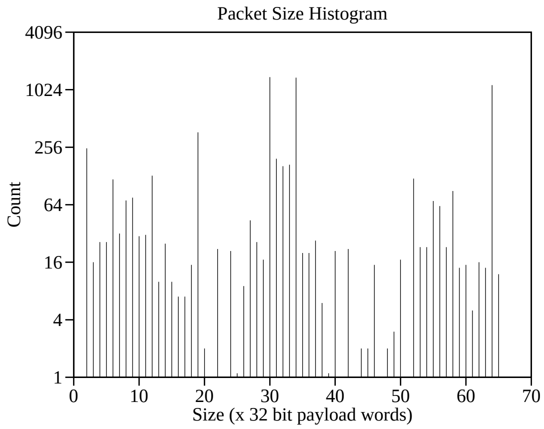

Figure 23: LU, No Combining, Run 1

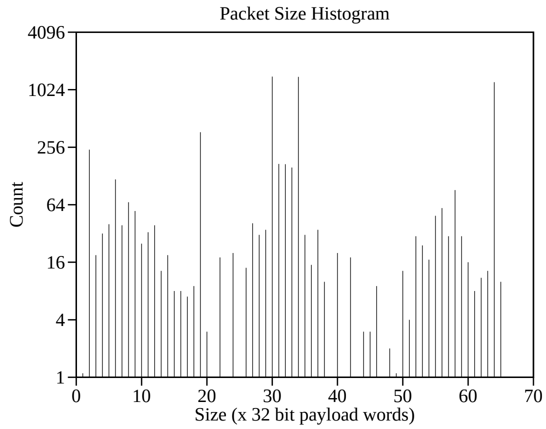

Figure 26: LU, No Combining, Run 4

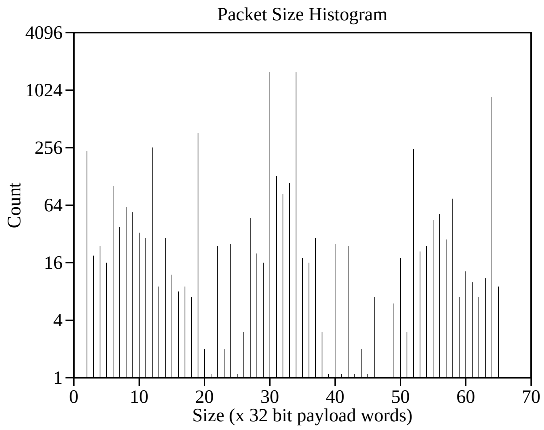

Figure 24: LU, No Combining, Run 2

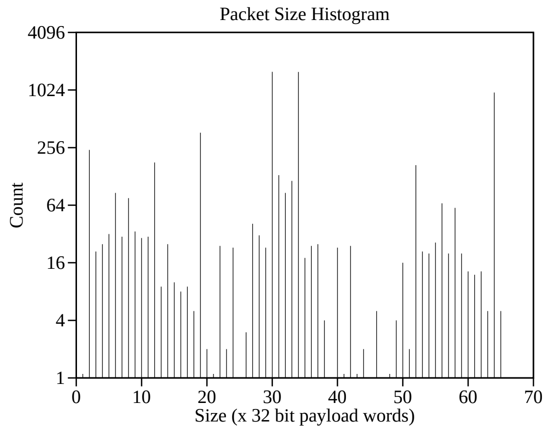

Figure 27: LU, No Combining, Run 5

Figures 28–32 show graphs of latency versus time.

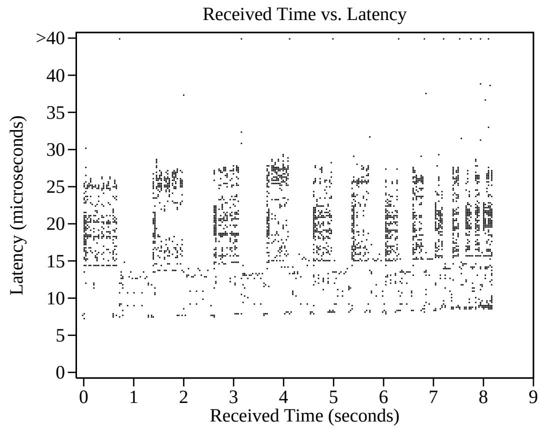

Figure 30: LU, No Combining, Run 3

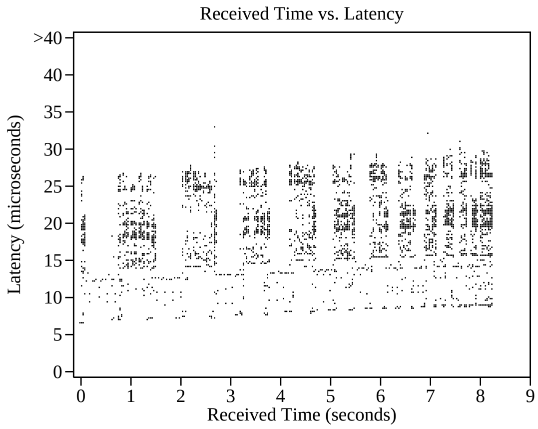

Figure 28: LU, No Combining, Run 1

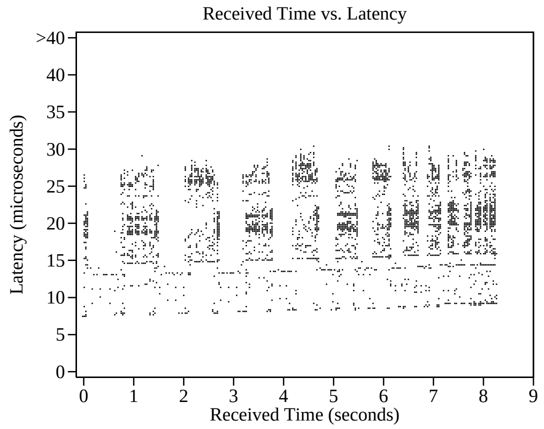

Figure 31: LU, No Combining, Run 4

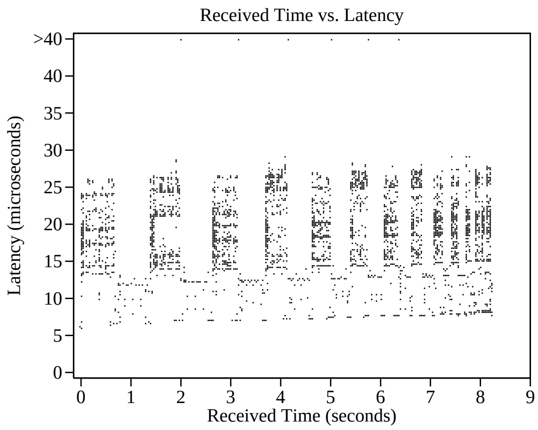

Figure 29: LU, No Combining, Run 2

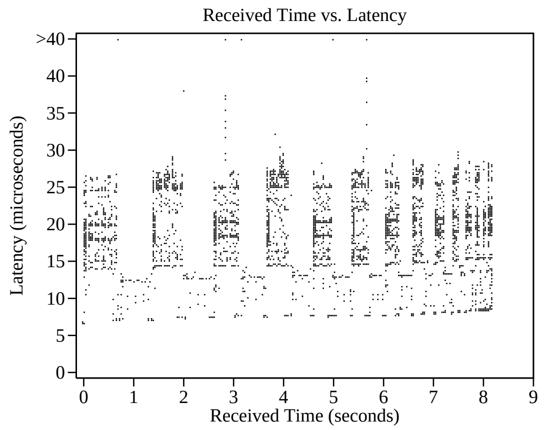

Figure 32: LU, No Combining, Run 5

4.3.3 OCEAN, Combining

This section shows the results of 2 runs of the OCEAN benchmark with combining enabled.

Figures 33–34 show the packet latency distributions.

Figures 35–36 show the packet size distributions.

Comparing these figures with those for no combining (Figures 41–42 on page 17) show that the two spikes near 130 words are due to combining. These, in turn, correspond to the spikes near 1000 μsec in the latency distribution (Figures 33–34).

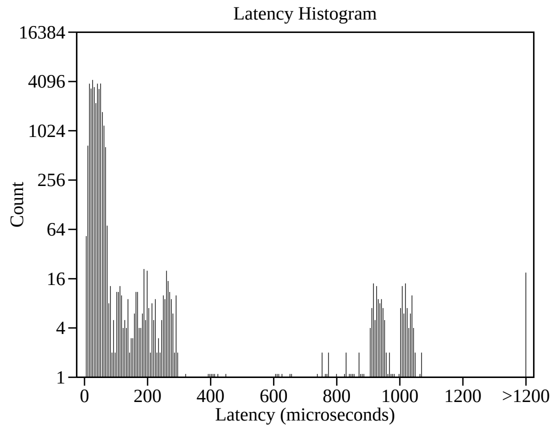

Figure 33: OCEAN, Combining, Run 1

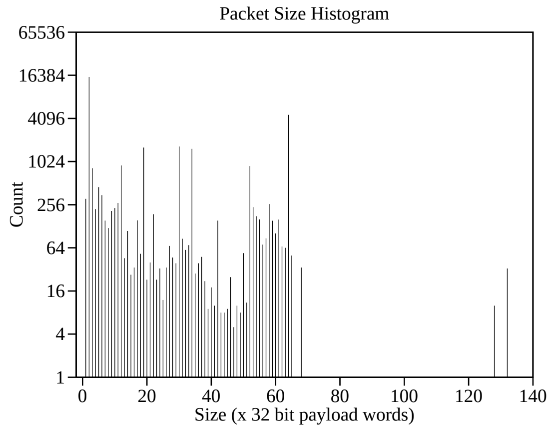

Figure 35: OCEAN, Combining, Run 1

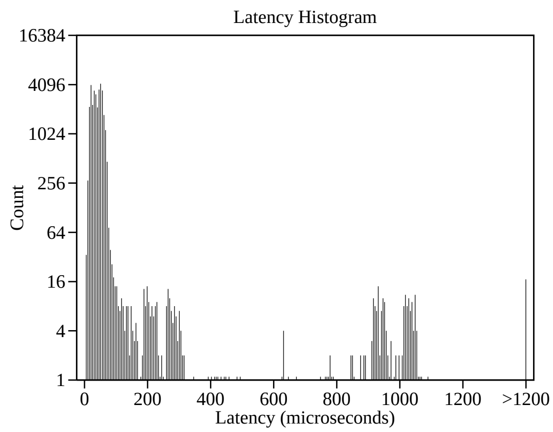

Figure 34: OCEAN, Combining, Run 2

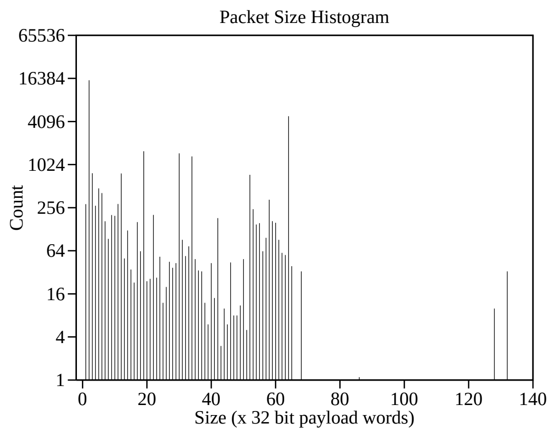

Figure 36: OCEAN, Combining, Run 2

Figures 37–38 show graphs of latency versus time.

4.3.4 OCEAN, No Combining

This section shows the results of 2 runs of the OCEAN benchmark with combining disabled.

Figures 39–40 show the packet latency distributions.

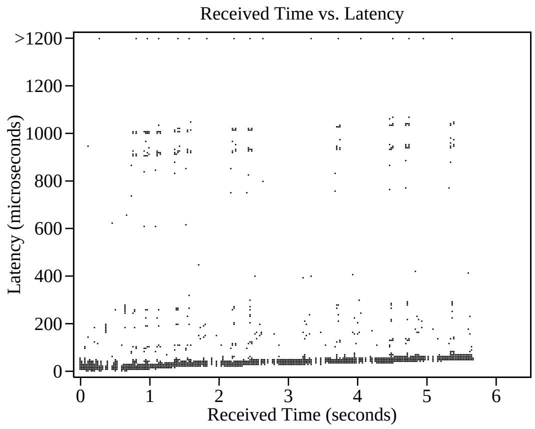

Figure 37: OCEAN, Combining, Run 1

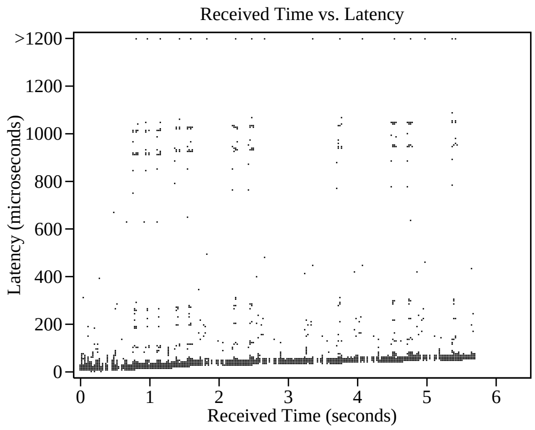

Figure 38: OCEAN, Combining, Run 2

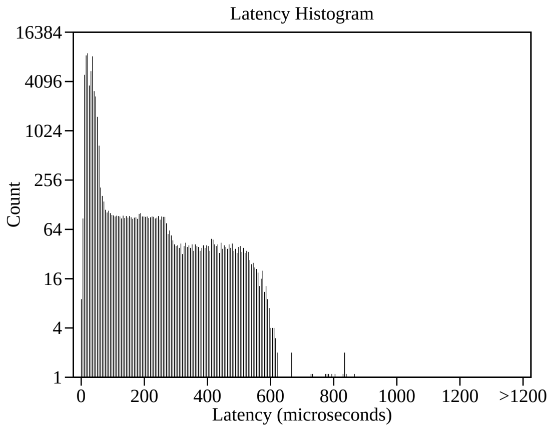

Figure 39: OCEAN, No Combining, Run 1

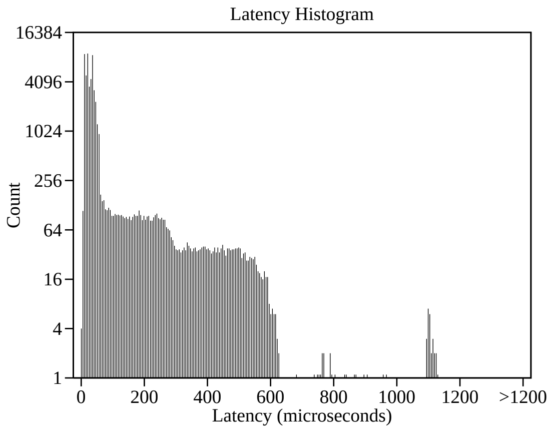

Figure 40: OCEAN, No Combining, Run 2

Figures 41–42 show the packet size distributions.

Figures 43–44 show graphs of latency versus time. The steep ramps in these graphs (which appear as spikes) represent congestion in the SNI board outgoing FIFOs on the sender side.

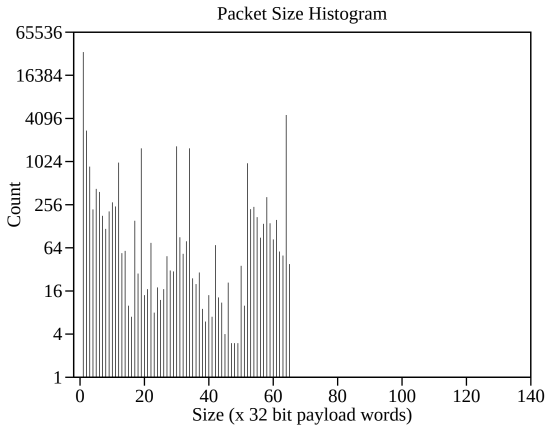

Figure 41: OCEAN, No Combining, Run 1

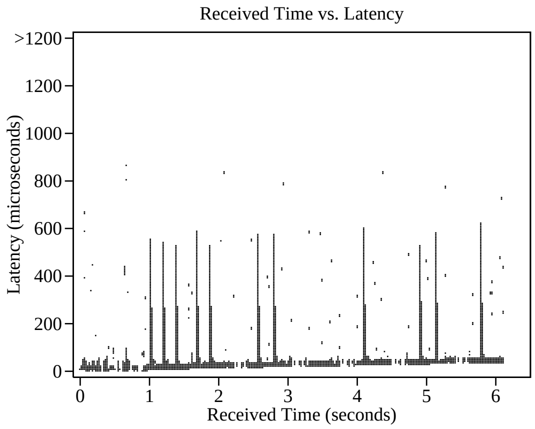

Figure 43: OCEAN, No Combining, Run 1

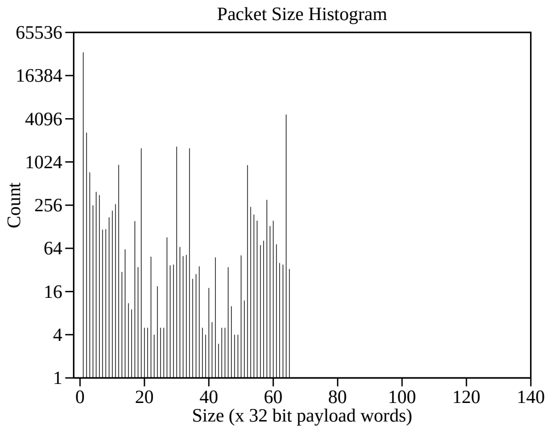

Figure 42: OCEAN, No Combining, Run 2

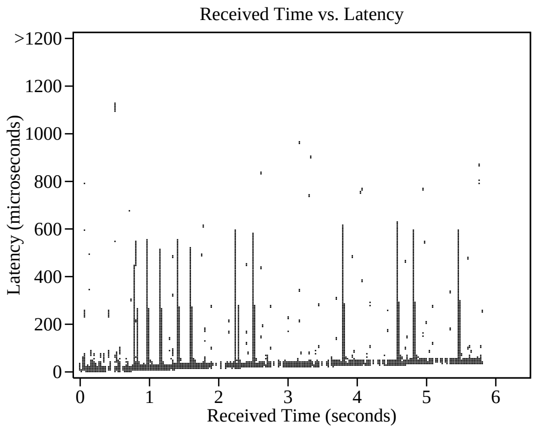

Figure 44: OCEAN, No Combining, Run 2

4.3.5 RADIX, Combining

This section shows the results of 5 runs of the RADIX benchmark with combining enabled.

Figures 45–49 show the packet latency distributions.

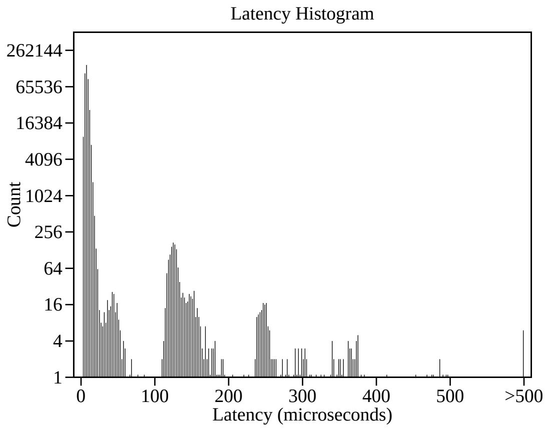

Figure 45: RADIX, Combining, Run 1

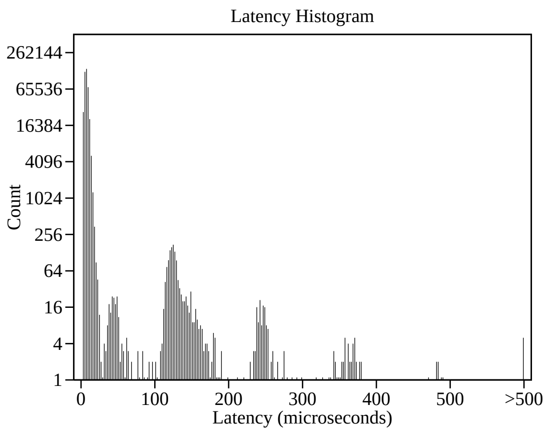

Figure 46: RADIX, Combining, Run 2

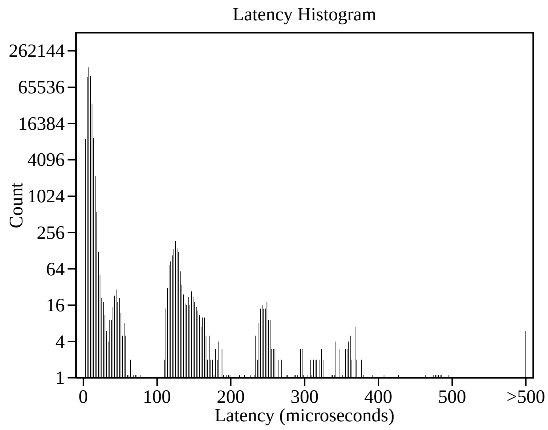

Figure 47: RADIX, Combining, Run 3

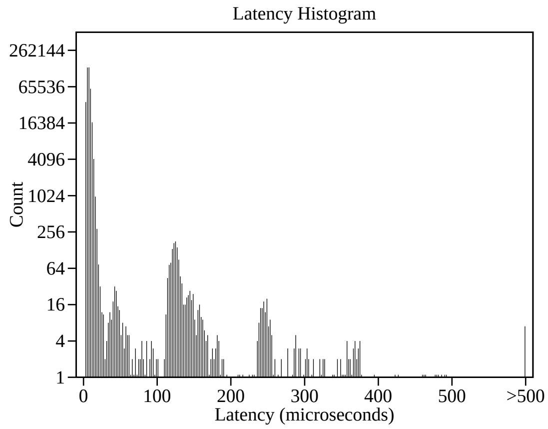

Figure 48: RADIX, Combining, Run 4

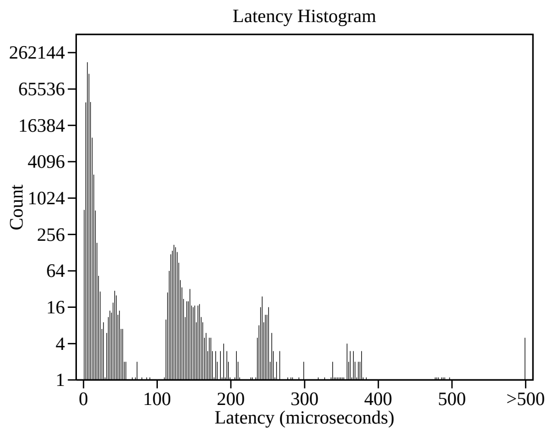

Figure 49: RADIX, Combining, Run 5

Figures 50–54 show the packet size distributions. Comparing these graphs with the no combining case (Figures 65–69 on page 22) shows that the spikes near 250 words are due to combining.

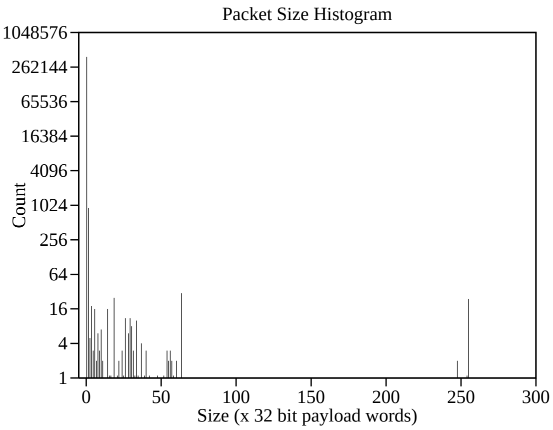

Figure 52: RADIX, Combining, Run 3

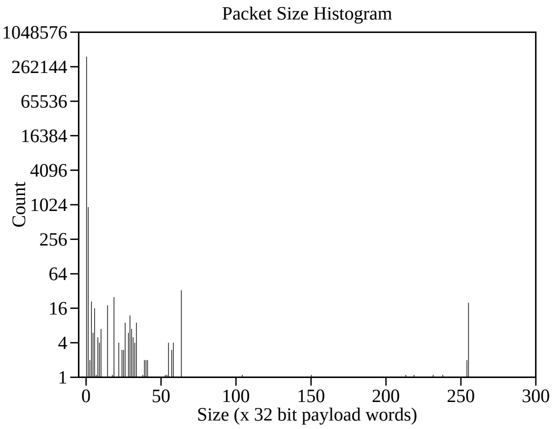

Figure 50: RADIX, Combining, Run 1

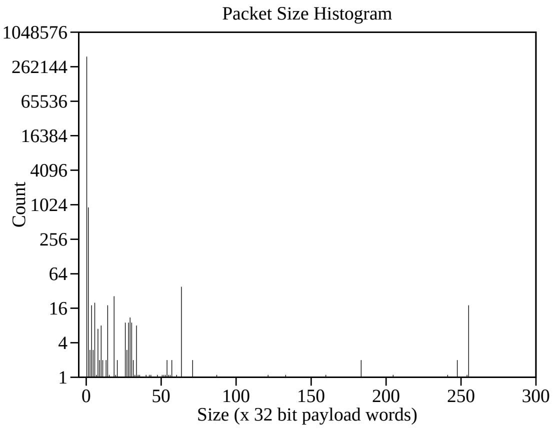

Figure 53: RADIX, Combining, Run 4

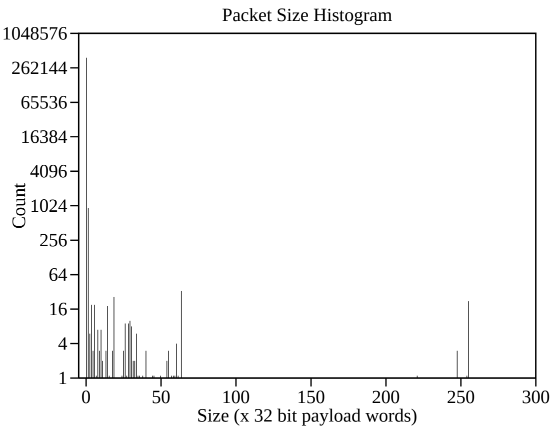

Figure 51: RADIX, Combining, Run 2

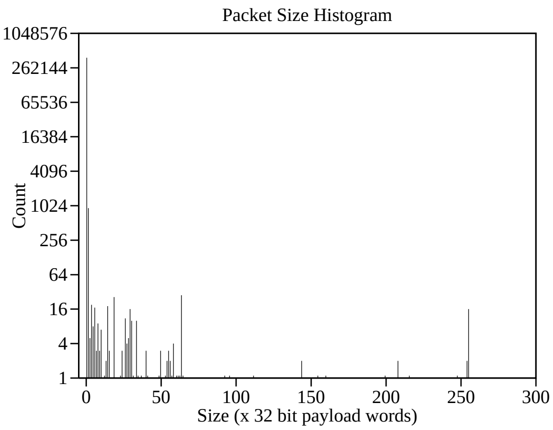

Figure 54: RADIX, Combining, Run 5

Figures 55–59 show graphs of latency versus time. Note the clearly defined pattern. For the problem size used, we expect there to be two cycles of the main loop consisting of a key permutation (communication) phase and an internal sort (computation) phase.

Comparing these graphs with the no combining case (Figures70–74 on page 23) shows that the latencies over 100 μsec are due to combining.

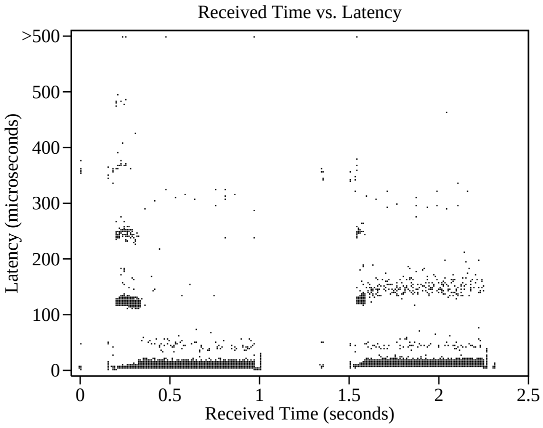

Figure 57: RADIX, Combining, Run 3

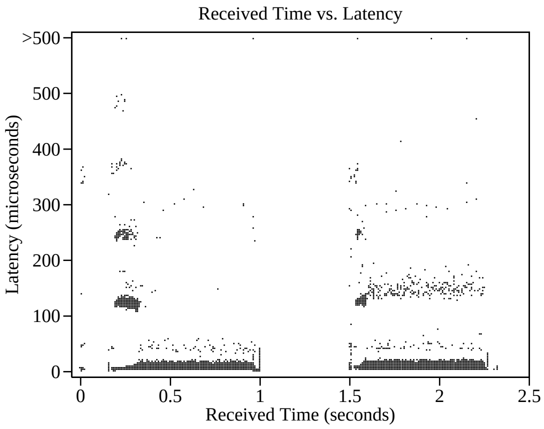

Figure 55: RADIX, Combining, Run 1

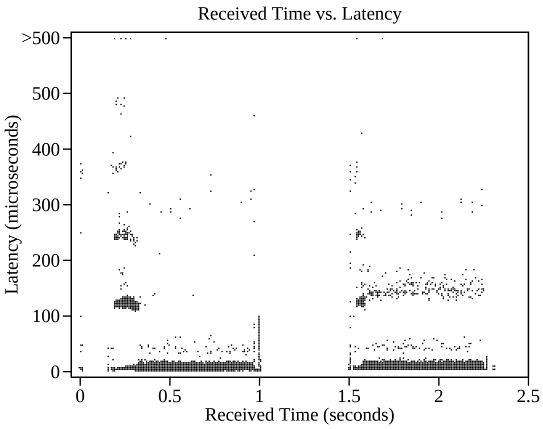

Figure 58: RADIX, Combining, Run 4

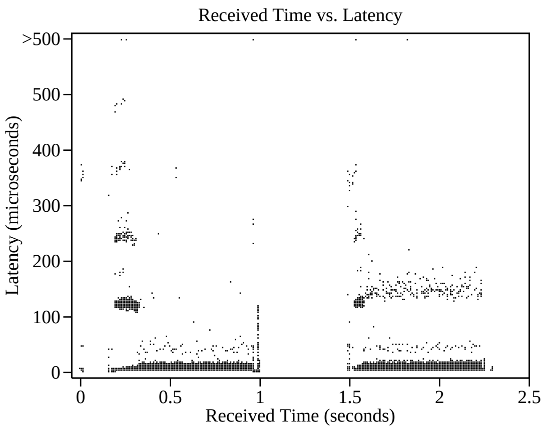

Figure 56: RADIX, Combining, Run 2

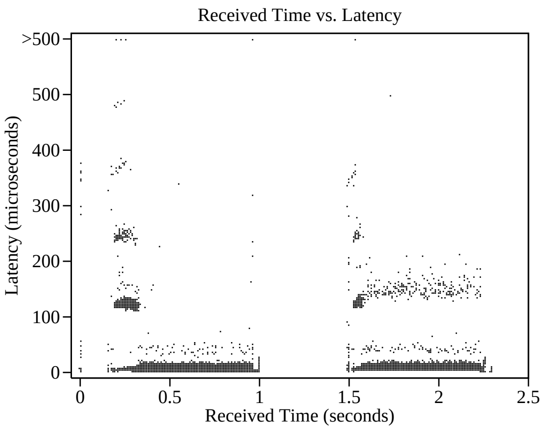

Figure 59: RADIX, Combining, Run 5

4.3.6 RADIX, No Combining

This section shows the results of 5 runs of the RADIX benchmark with combining disabled.

Figures 60–64 show the packet latency distributions.

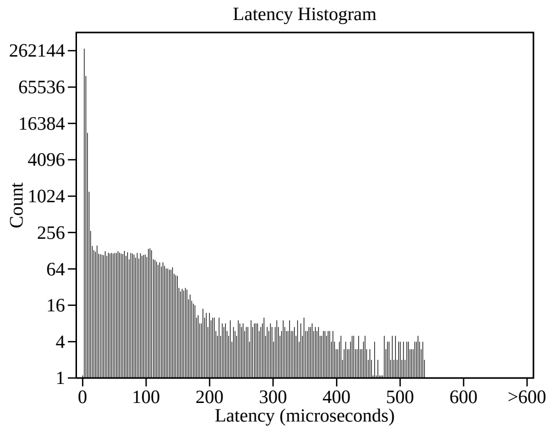

Figure 62: RADIX, No Combining, Run 3

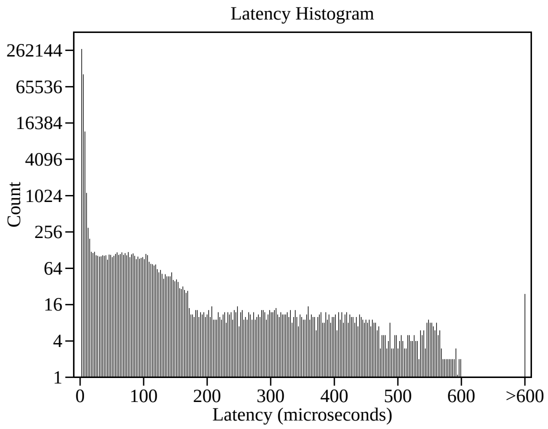

Figure 60: RADIX, No Combining, Run 1

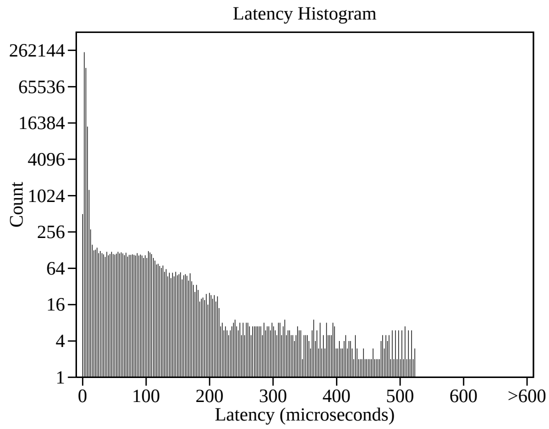

Figure 63: RADIX, No Combining, Run 4

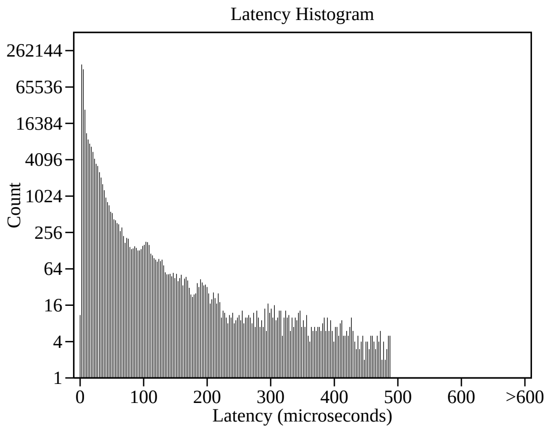

Figure 61: RADIX, No Combining, Run 2

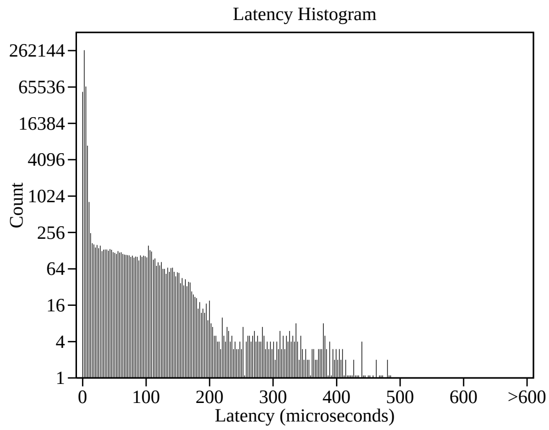

Figure 64: RADIX, No Combining, Run 5

Figures 65–69 show the packet size distributions.

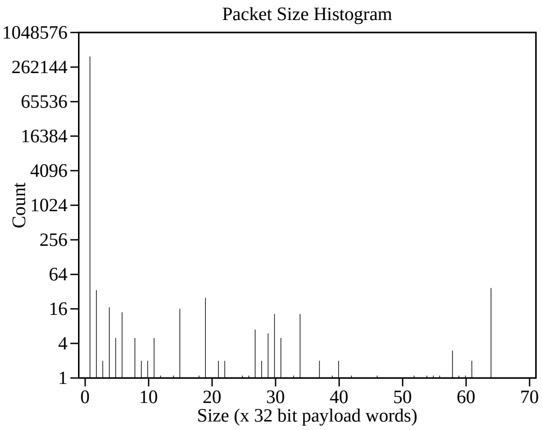

Figure 67: RADIX, No Combining, Run 3

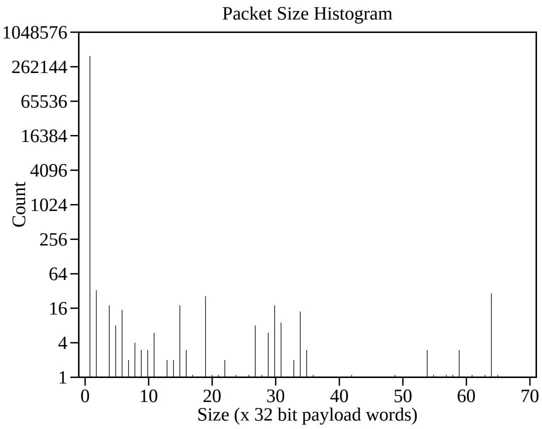

Figure 65: RADIX, No Combining, Run 1

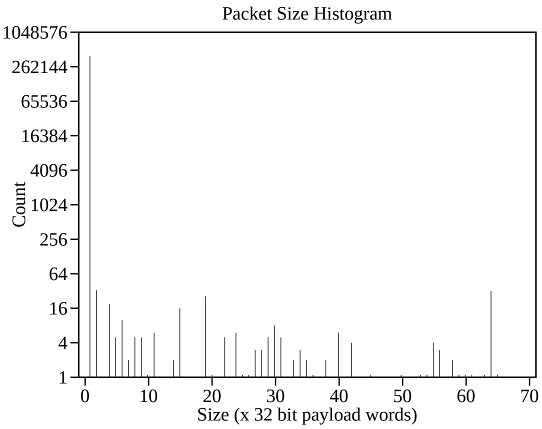

Figure 68: RADIX, No Combining, Run 4

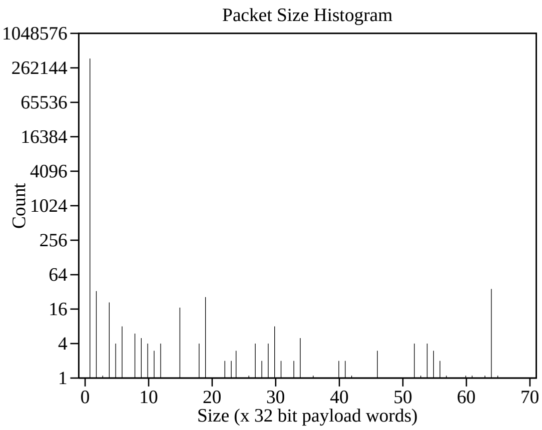

Figure 66: RADIX, No Combining, Run 2

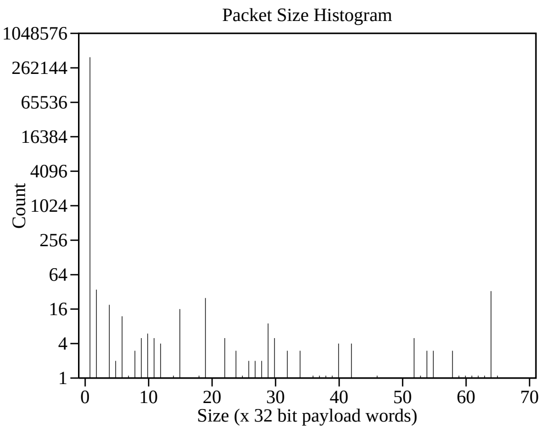

Figure 69: RADIX, No Combining, Run 5

Figures 70–74 show graphs of latency versus time. These graphs show that there are a few times during the run which exhibit intense communication resulting in large latencies due to back-ups in the outgoing FIFO of the sending SNI boards.

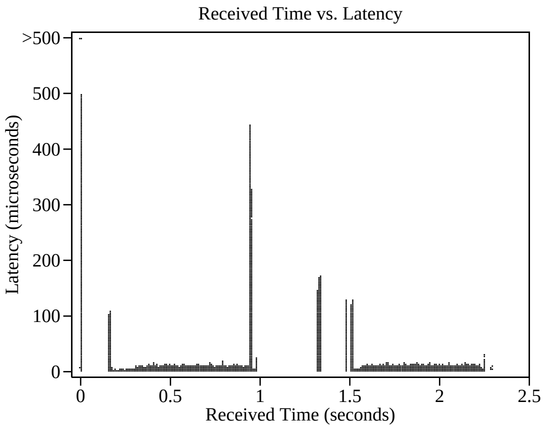

Figure 70: RADIX, No Combining, Run 1

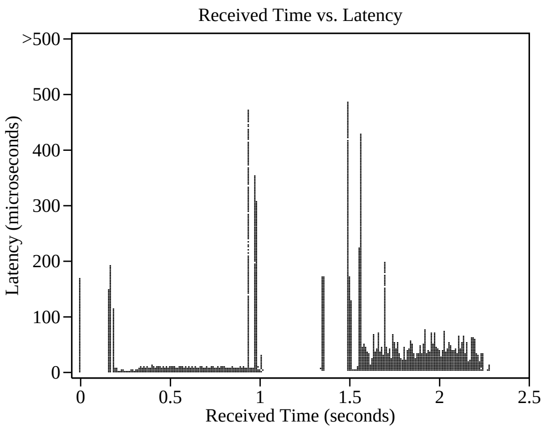

Figure 71: RADIX, No Combining, Run 2

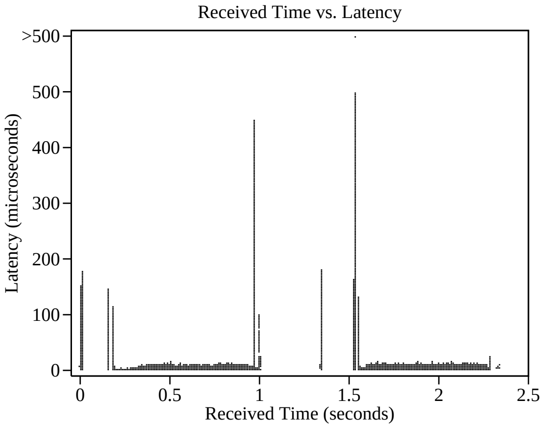

Figure 72: RADIX, No Combining, Run 3

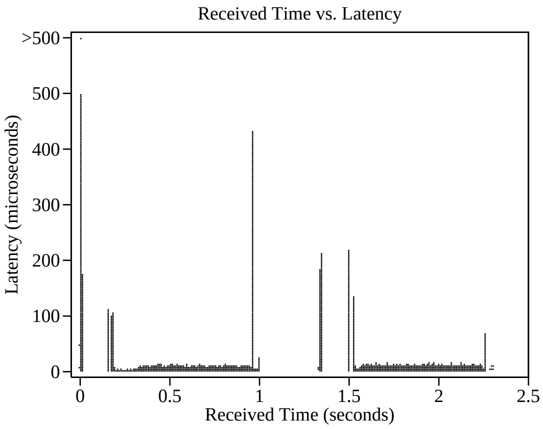

Figure 73: RADIX, No Combining, Run 4

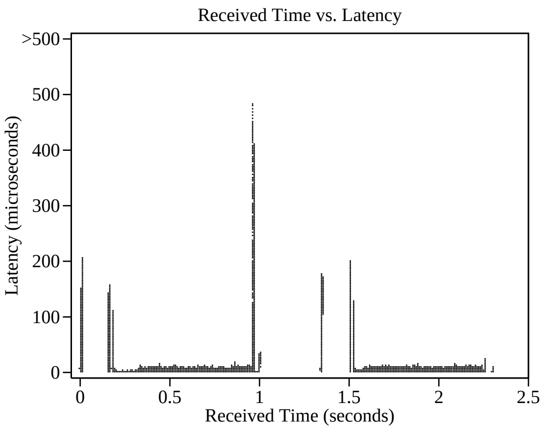

Figure 74: RADIX, No Combining, Run 5

4.3.7 SIMPLE, 1 Word, 1 μsec, Combining

This section shows the results of 5 runs of the SIMPLE benchmark (set to send 512 1 word packets at a 1 μsec word pace) with combining enabled. Only packets associated with the benchmark's data page are shown. The small number of packets (less than 10) associated with barriers at the beginning and end of the run have been removed.

Because all packets are of size 1, the size distributions are not shown.

Figures 75–79 show the packet latency distributions.

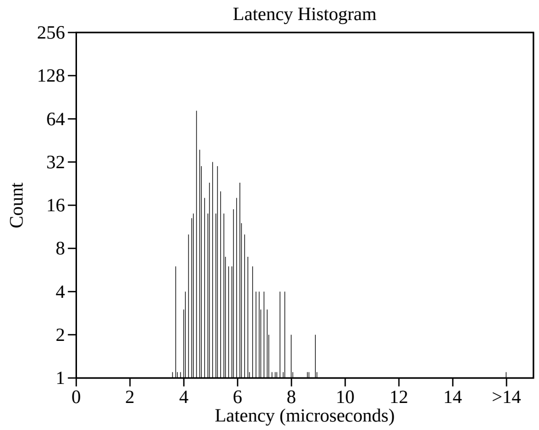

Figure 77: SIMPLE, Combining, Run 3

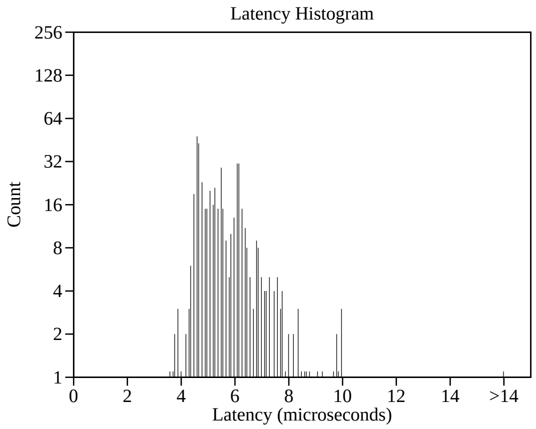

Figure 75: SIMPLE, Combining, Run 1

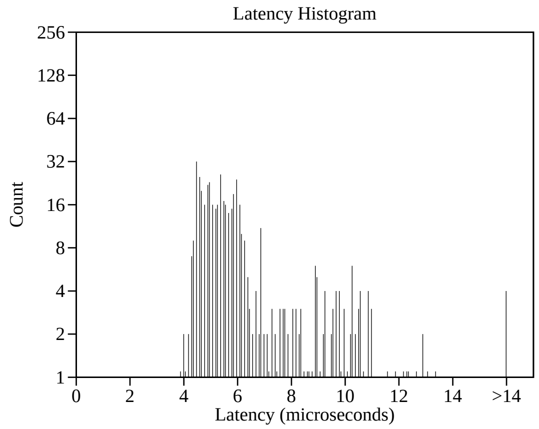

Figure 78: SIMPLE, Combining, Run 4

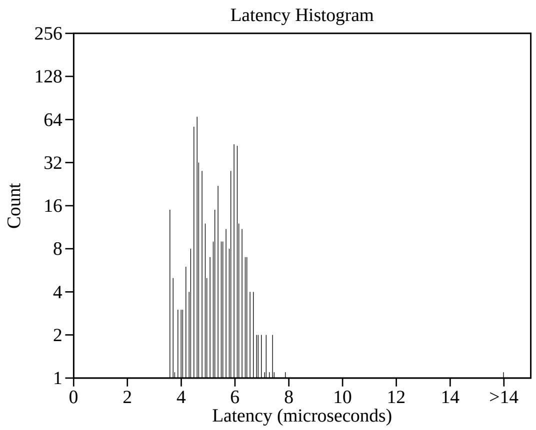

Figure 76: SIMPLE, Combining, Run 2

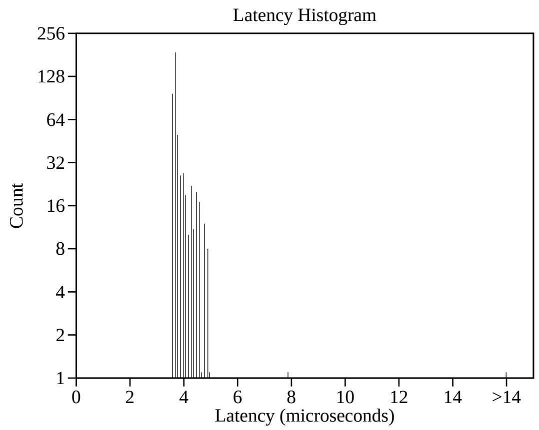

Figure 79: SIMPLE, Combining, Run 5

Figures 80–84 show graphs of latency versus time. These graphs show that the latency for 1 word packets is generally in the neighborhood of 5 μsec. Note that some system event (perhaps an interrupt) was captured in Figure 83.

The large latency of the last packet (the lone point in the upper right corner of the graphs) is due to a packet waiting to be combined with another memory reference, if possible. Eventually, it gets shipped out with a resulting long latency.

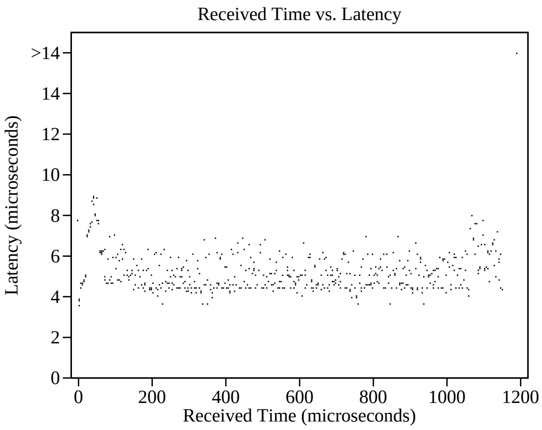

Figure 82: SIMPLE, Combining, Run 3

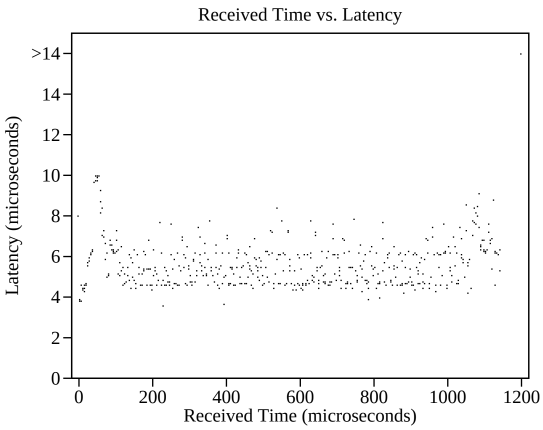

Figure 80: SIMPLE, Combining, Run 1

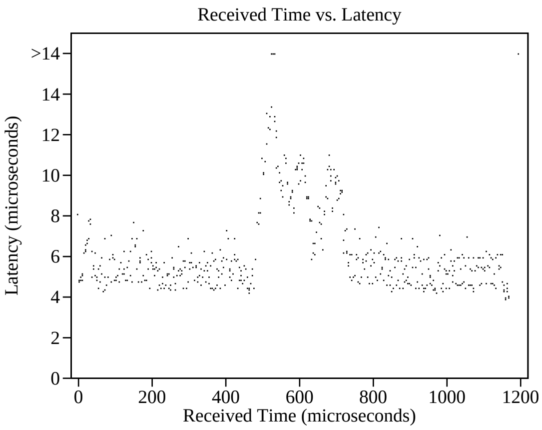

Figure 83: SIMPLE, Combining, Run 4

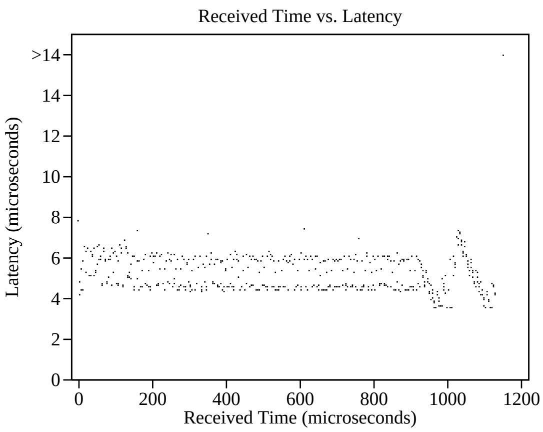

Figure 81: SIMPLE, Combining, Run 2

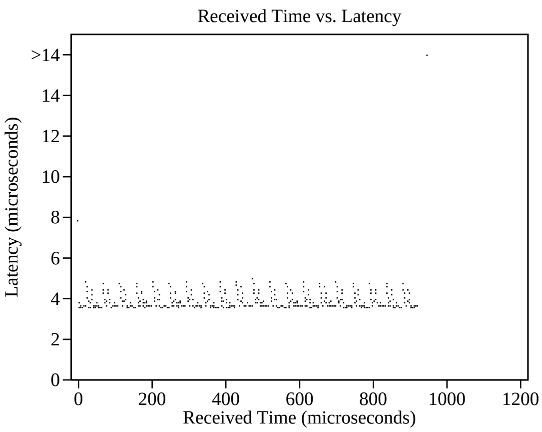

Figure 84: SIMPLE, Combining, Run 5

4.3.8 SIMPLE, 1 Word, 1 μsec, No Combining

This section shows the results of 4 runs of the SIMPLE benchmark (set to send 512 1 word packets at a 1 μsec word pace) with combining disabled. Only packets associated with the benchmark's data page are shown. The small number of packets (less than 10) associated with barriers at the beginning and end of the run have been removed.

Because all packets are of size 1, the size distributions are not shown.

Figures 85–88 show the packet latency distributions.

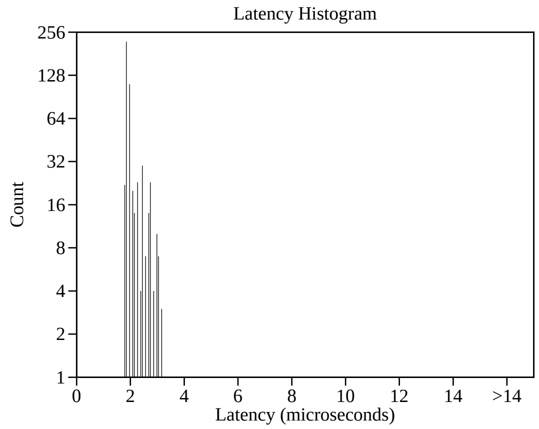

Figure 85: SIMPLE, No Combining, Run 1

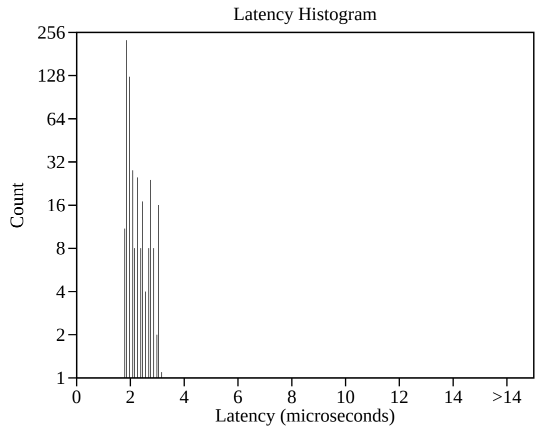

Figure 86: SIMPLE, No Combining, Run 2

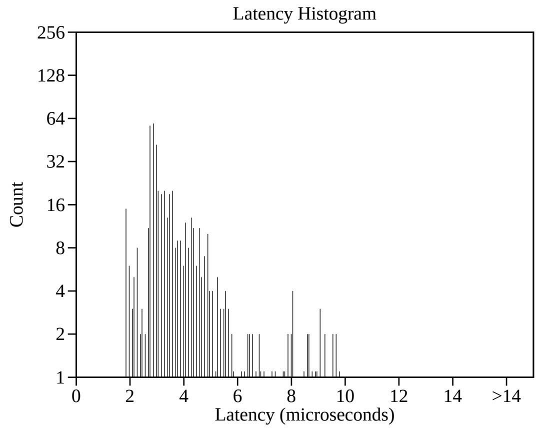

Figure 87: SIMPLE, No Combining, Run 3

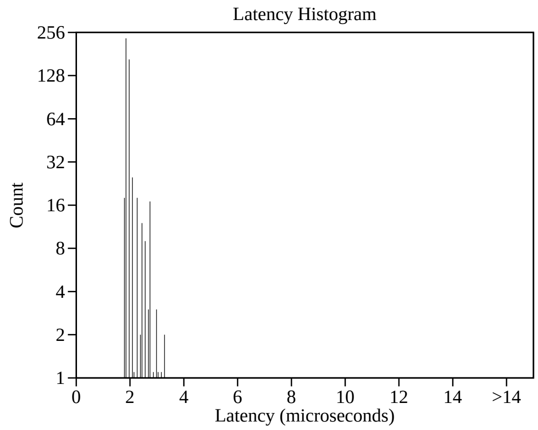

Figure 88: SIMPLE, No Combining, Run 4

Figures 89–92 show graphs of latency versus time. Note that these graphs have a similar structure to the combining case (Figures 80–83 on page 25). The difference is that the latency is slightly smaller because the packets are aggressively shipped out. As a result, notice that there is no lone point in the upper right corner as in the combining case.

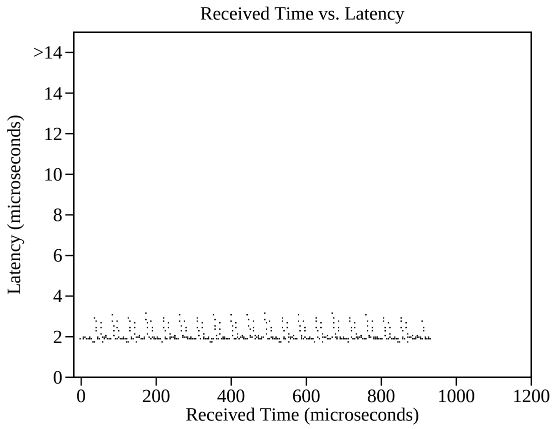

Figure 89: SIMPLE, No Combining, Run 1

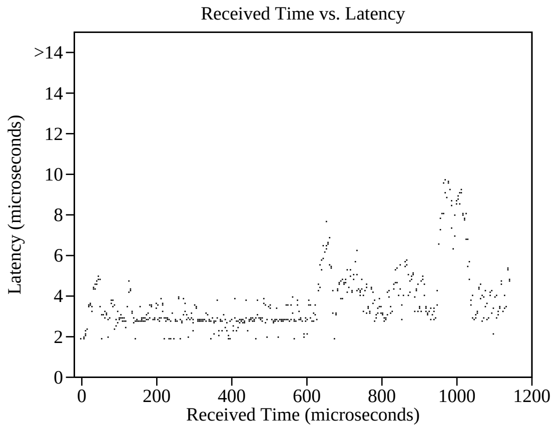

Figure 91: SIMPLE, No Combining, Run 3

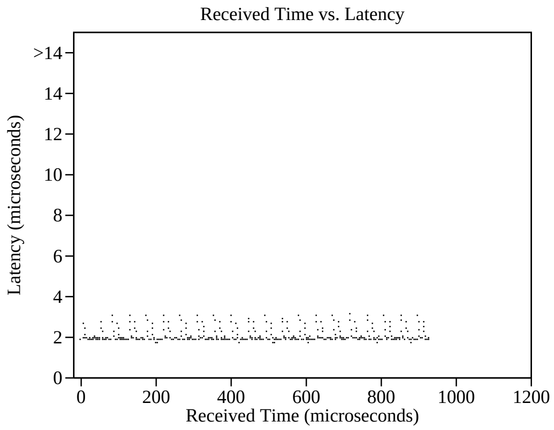

Figure 90: SIMPLE, No Combining, Run 2

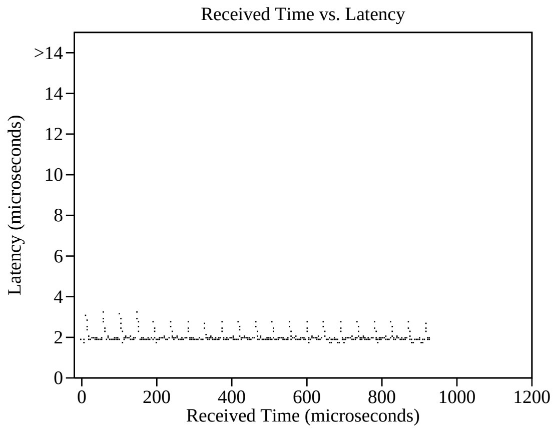

Figure 92: SIMPLE, No Combining, Run 4

4.3.9 SIMPLE, 1 Word, $100\,\mu$sec, Combining

This section shows the results of 5 runs of the SIMPLE benchmark (set to send 512 1 word packets at a $100\,\mu$sec word pace) with combining enabled. Only packets associated with the benchmark's data page are shown. The small number of packets (less than 10) associated with barriers at the beginning and end of the run have been removed.

Because all packets are of size 1, the size distributions are not shown.

Figures 93–97 show the packet latency distributions.

Notice that the scale is different in Figure 96.

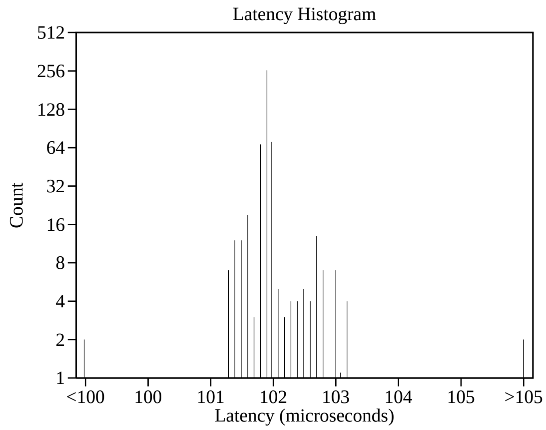

Figure 95: SIMPLE, Combining, Run 3

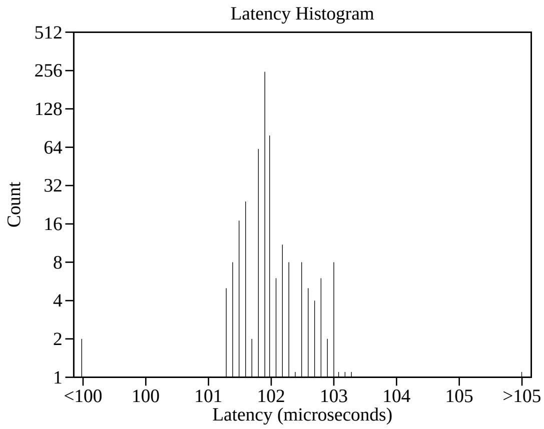

Figure 93: SIMPLE, Combining, Run 1

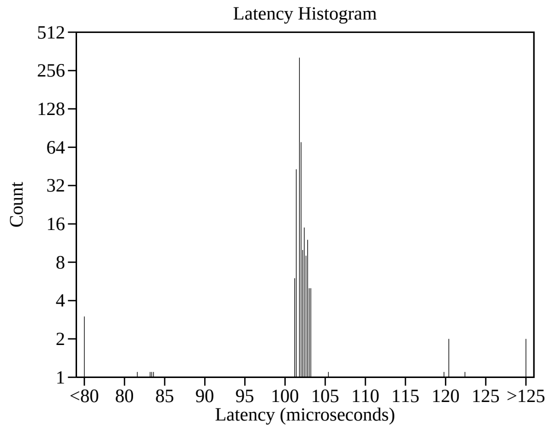

Figure 96: SIMPLE, Combining, Run 4

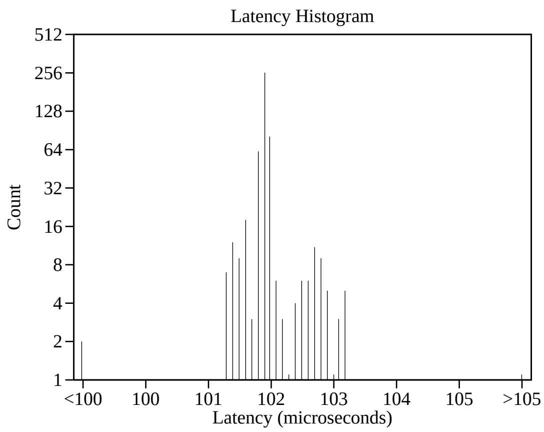

Figure 94: SIMPLE, Combining, Run 2

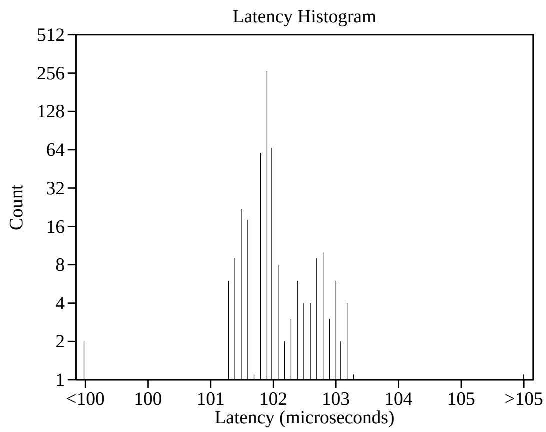

Figure 97: SIMPLE, Combining, Run 5

Figures 98–102 show graphs of latency versus time. The expectation would be for the latencies depicted here to be simply those of the 1 μsec case (Figures 80–84 on page 25) offset by 100 μsec. The fact that they are lower than expected is probably due to the overall program (running on four nodes) finding a “better place”. That is, the larger gaps allow the ensemble to synchronize with one another.

The large latency of the last packet (the lone point in the upper right corner of the graphs) is due to a packet waiting to be combined with another memory reference, if possible. Eventually, it gets shipped out with a resulting long latency.

Notice that the scale is different in Figure 101. This was done to better capture the periodic event spaced every 10 ms (the Linux timer interrupt period) in this run.

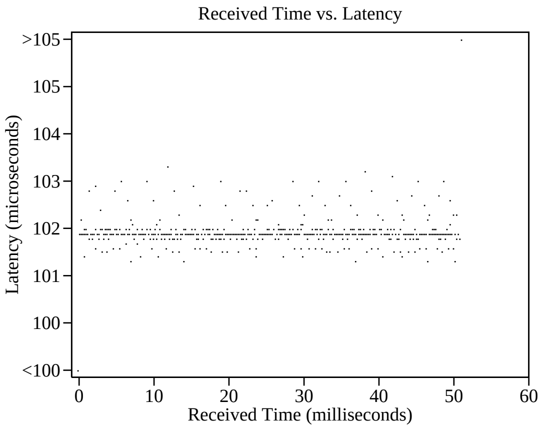

Figure 98: SIMPLE, Combining, Run 1

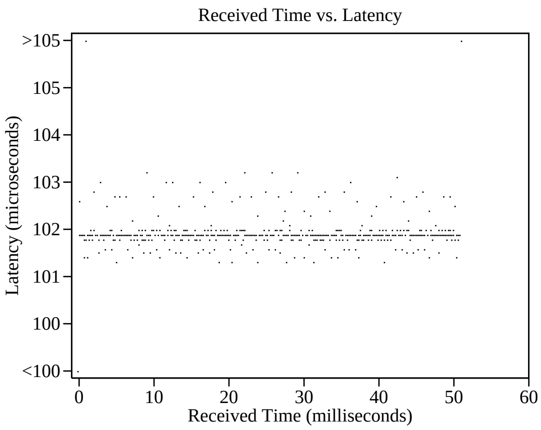

Figure 99: SIMPLE, Combining, Run 3

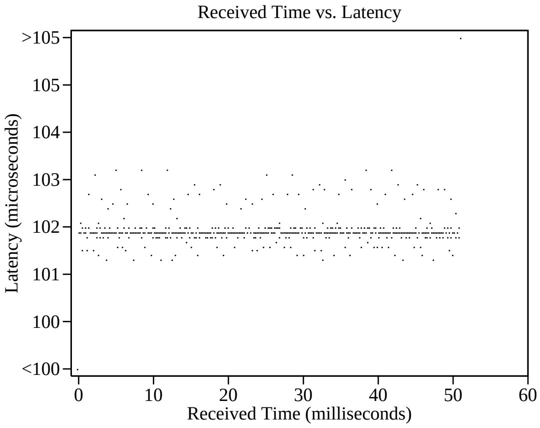

Figure 100: SIMPLE, Combining, Run 2

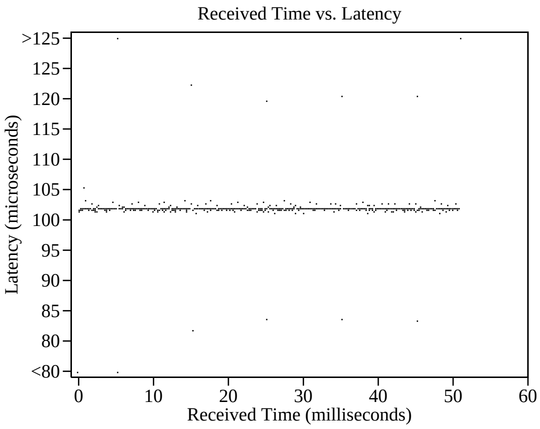

Figure 101: SIMPLE, Combining, Run 4

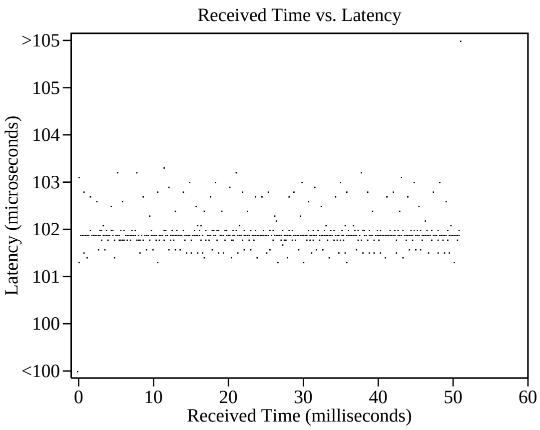

Figure 102: SIMPLE, Combining, Run 5

4.3.10 SIMPLE, 1 Word, 100 μsec, No Combining

This section shows the results of 4 runs of the SIMPLE benchmark (set to send 512 1 word packets at a 100 μsec word pace) with combining disabled. Only packets associated with the benchmark's data page are shown. The small number of packets (less than 10) associated with barriers at the beginning and end of the run have been removed.

Because all packets are of size 1, the size distributions are not shown.

Figures 103–106 show the packet latency distributions.

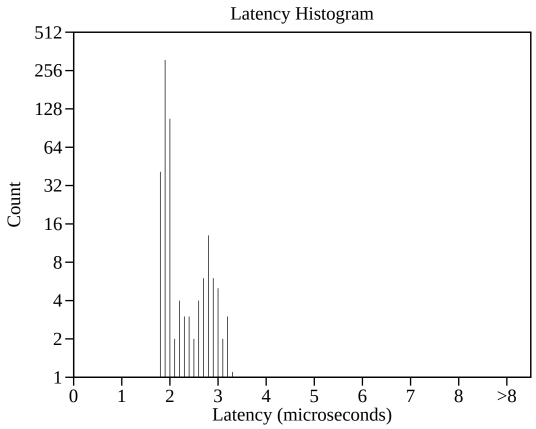

Figure 103: SIMPLE, No Combining, Run 1

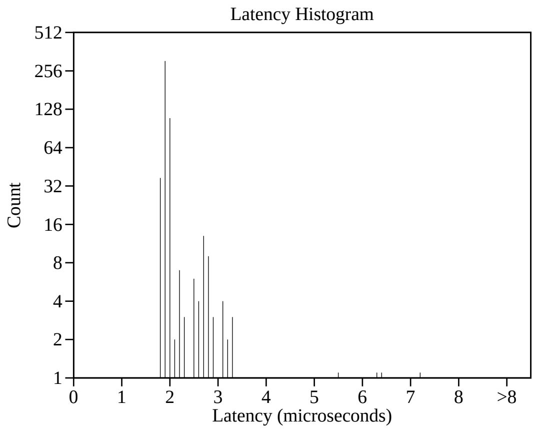

Figure 104: SIMPLE, No Combining, Run 2

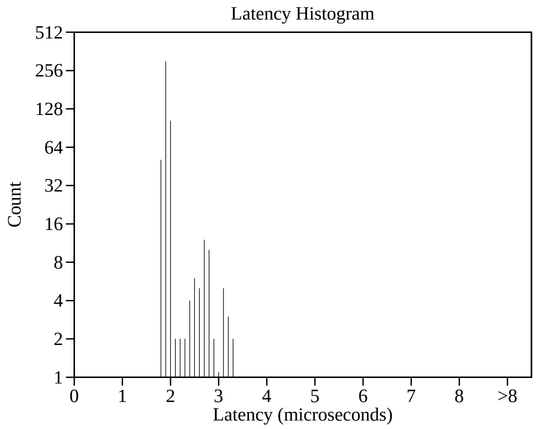

Figure 105: SIMPLE, No Combining, Run 3

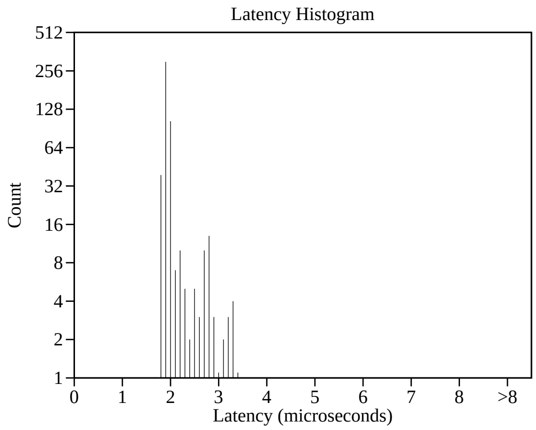

Figure 106: SIMPLE, No Combining, Run 4

Figures 107–110 show graphs of latency versus time. These graphs are similar to those for the 1 μsec case (Figures 89–92). Again, since there is more time between packets, the system runs synchronously across the four nodes leading to lower latencies and tighter latency distributions.

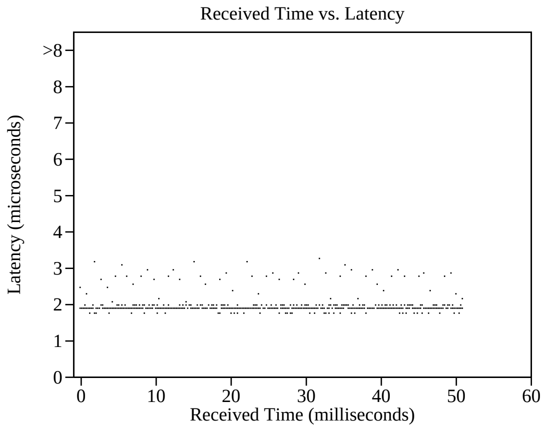

Figure 107: SIMPLE, No Combining, Run 1

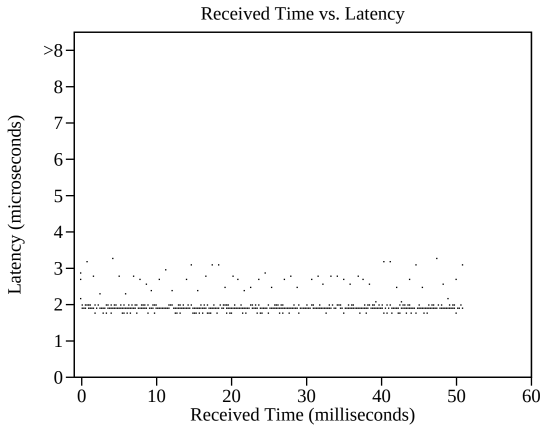

Figure 109: SIMPLE, No Combining, Run 3

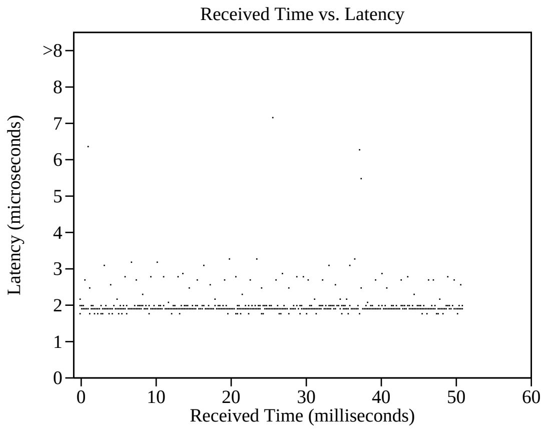

Figure 108: SIMPLE, No Combining, Run 2

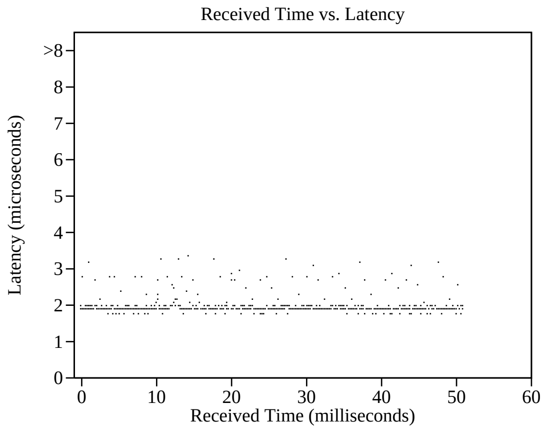

Figure 110: SIMPLE, No Combining, Run 4

4.3.11 SIMPLE, 1 Kword, 1 μsec, Combining

This section shows the results of 5 runs of the SIMPLE benchmark (set to send 512 1 Kword packets at a 1 μsec word pace) with combining enabled. Only packets associated with the benchmark's data page are shown. The small number of packets (less than 10) associated with barriers at the beginning and end of the run have been removed.

Figures 111–115 show the packet latency distributions.

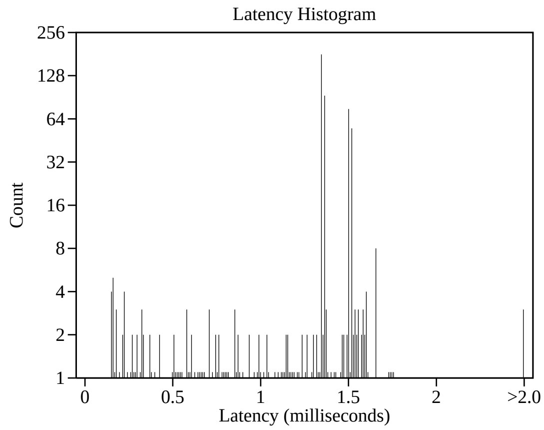

Figure 113: SIMPLE, Combining, Run 3

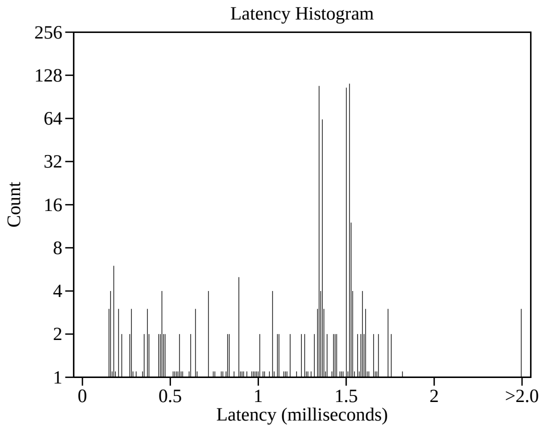

Figure 111: SIMPLE, Combining, Run 1

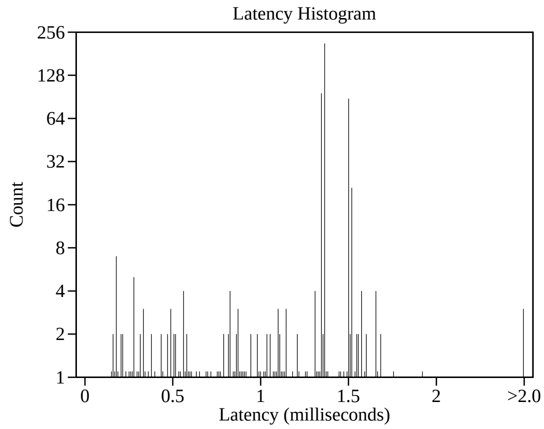

Figure 114: SIMPLE, Combining, Run 4

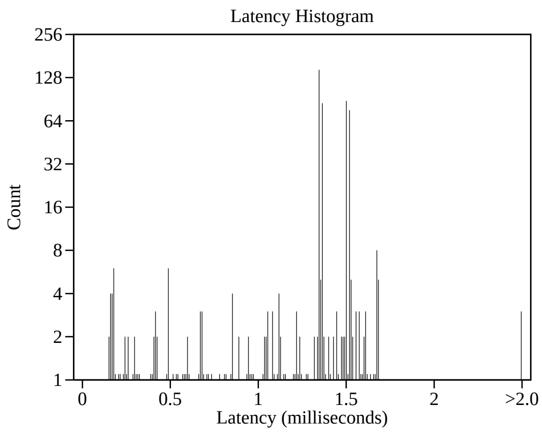

Figure 112: SIMPLE, Combining, Run 2

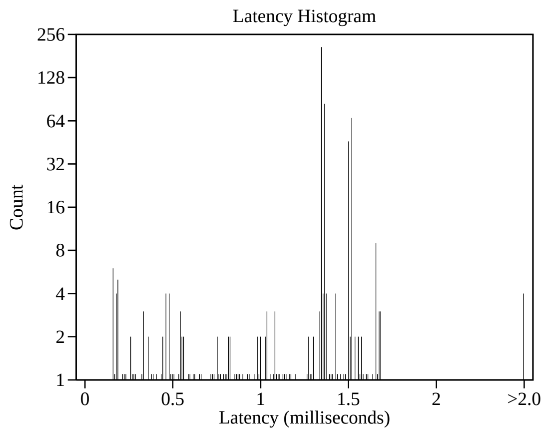

Figure 115: SIMPLE, Combining, Run 5

Figures 116–120 show the packet size distributions. Note that in the absence of interfering EISA bus traffic, all packets would be 1024 words.

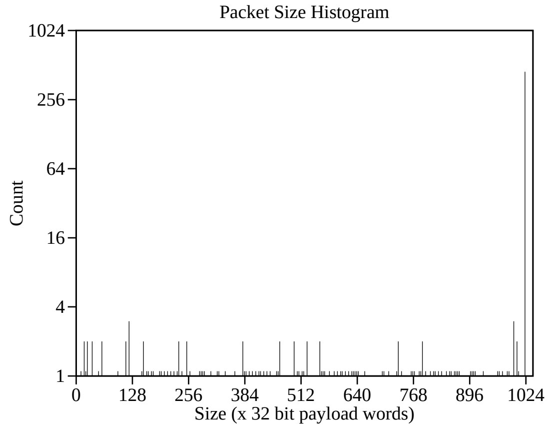

Figure 118: SIMPLE, Combining, Run 3

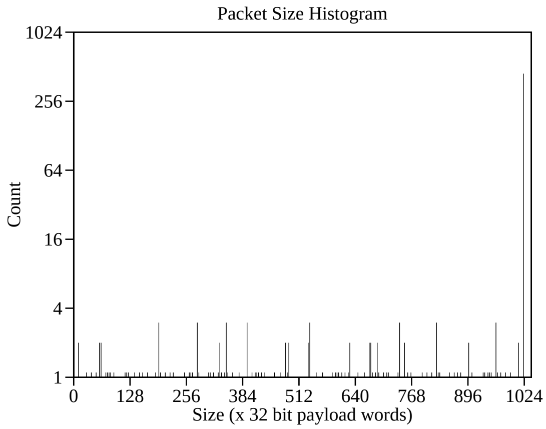

Figure 116: SIMPLE, Combining, Run 1

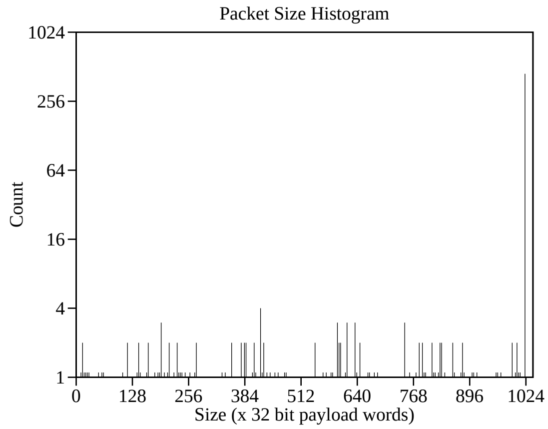

Figure 119: SIMPLE, Combining, Run 4

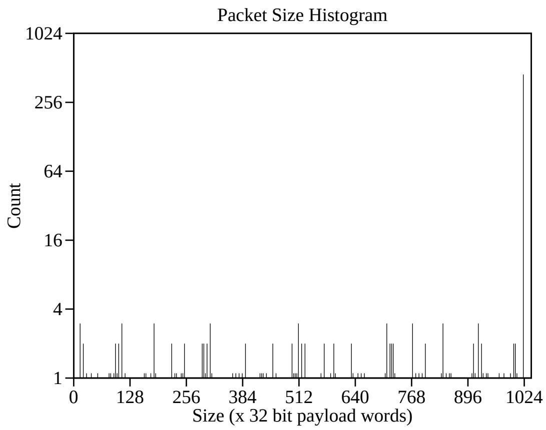

Figure 117: SIMPLE, Combining, Run 2

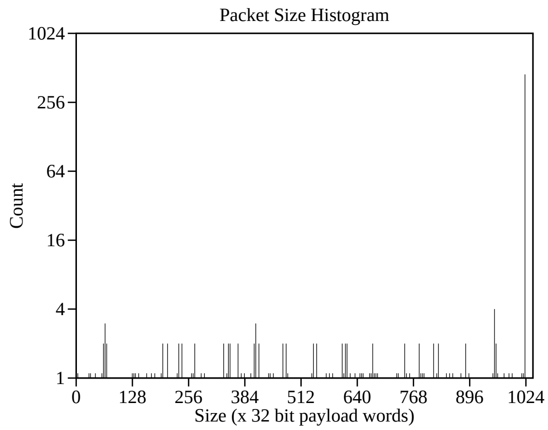

Figure 120: SIMPLE, Combining, Run 5

Figures 121–125 show graphs of latency versus time.

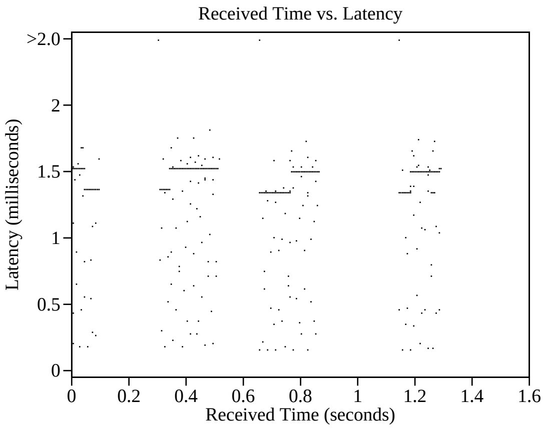

Figure 121: SIMPLE, Combining, Run 1

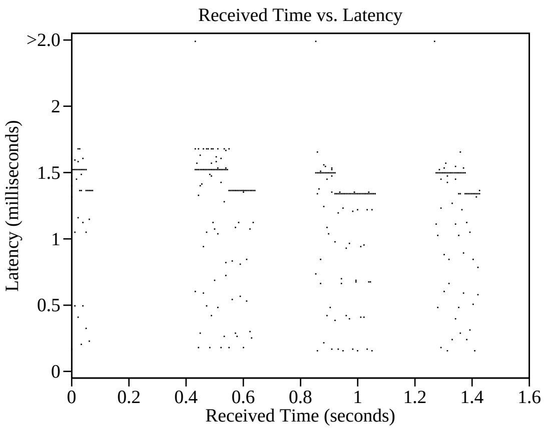

Figure 122: SIMPLE, Combining, Run 2

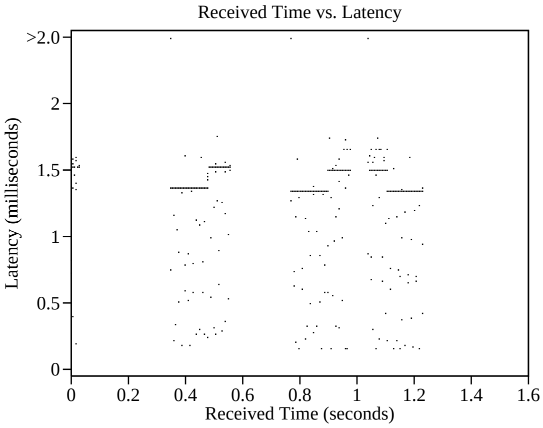

Figure 123: SIMPLE, Combining, Run 3

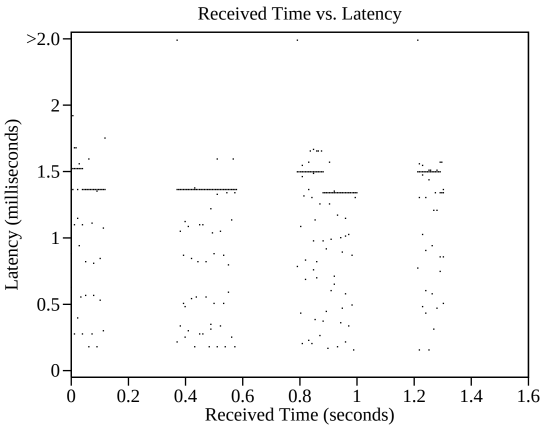

Figure 124: SIMPLE, Combining, Run 4

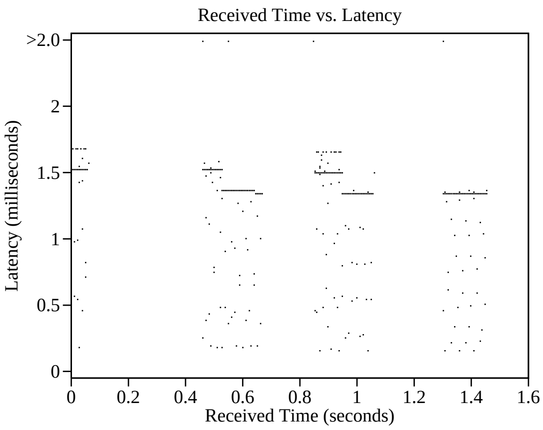

Figure 125: SIMPLE, Combining, Run 5

4.3.12 SIMPLE, 1 Kword, 1 μsec, No Combining

This section shows the results of 5 runs of the SIMPLE benchmark (set to send 512 1 Kword packets at a 1 μsec word pace) with combining disabled. Only packets associated with the benchmark's data page are shown. The small number of packets (less than 10) associated with barriers at the beginning and end of the run have been removed.

Because all packets are of size 1, the size distributions are not shown.

Figures 126–130 show the packet latency distributions.

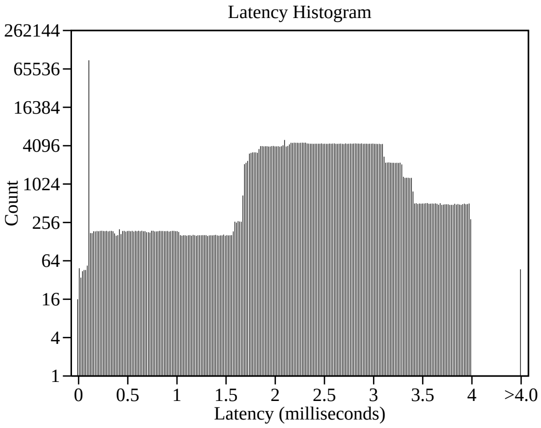

Figure 128: SIMPLE, No Combining, Run 3

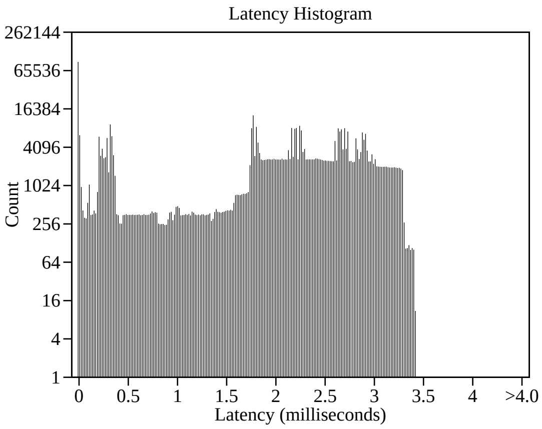

Figure 126: SIMPLE, No Combining, Run 1

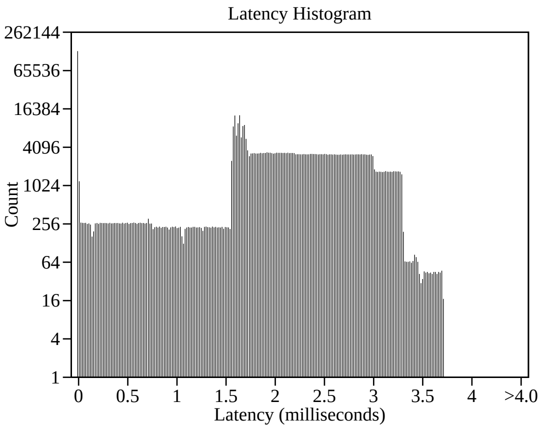

Figure 129: SIMPLE, No Combining, Run 4

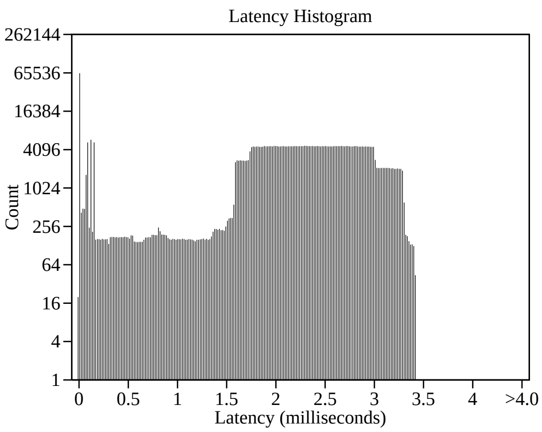

Figure 127: SIMPLE, No Combining, Run 2

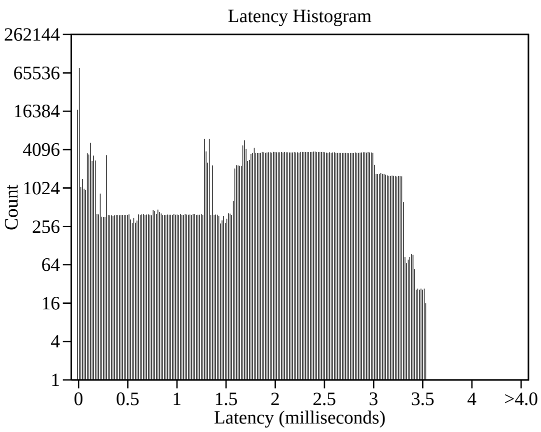

Figure 130: SIMPLE, No Combining, Run 5

Figures 131–135 show graphs of latency versus time. Notice that there are periods of no traffic followed by periods of high latency traffic. Since combining is disabled, all packets are 1 word in size and have maximum overhead relative to their size. The increase in overhead causes the outgoing FIFOs in the sending SNI boards more often than the combining case (with lower overhead). As a result, the overall run times are longer than for the combining case (See Figures 121–125 on page 34).

Figures 163–167 on page 45 explore some of the detail near the 89 msec point in run 1 (Figure 131).

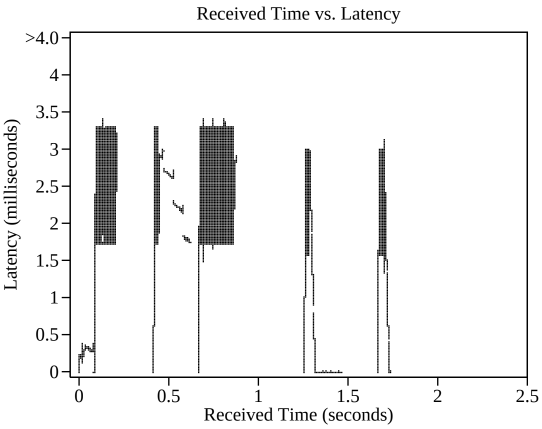

Figure 131: SIMPLE, No Combining, Run 1

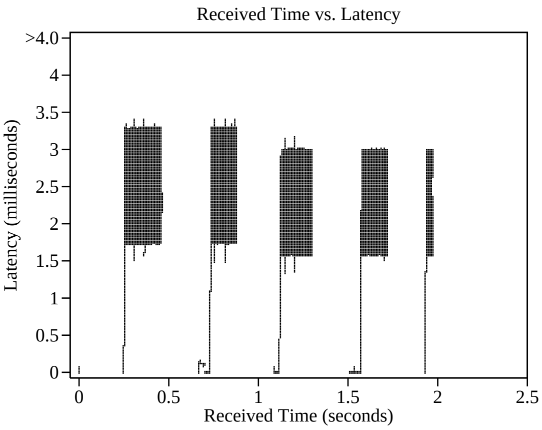

Figure 132: SIMPLE, No Combining, Run 2

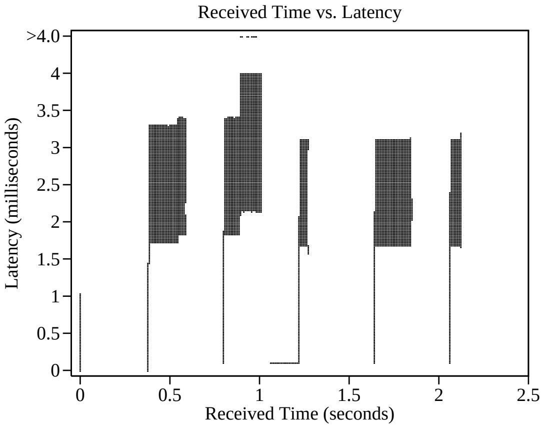

Figure 133: SIMPLE, No Combining, Run 3

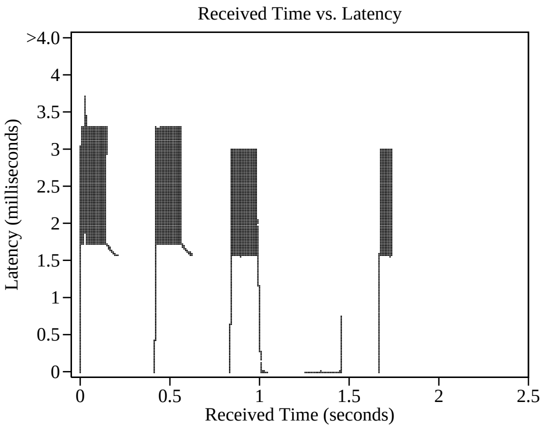

Figure 134: SIMPLE, No Combining, Run 4

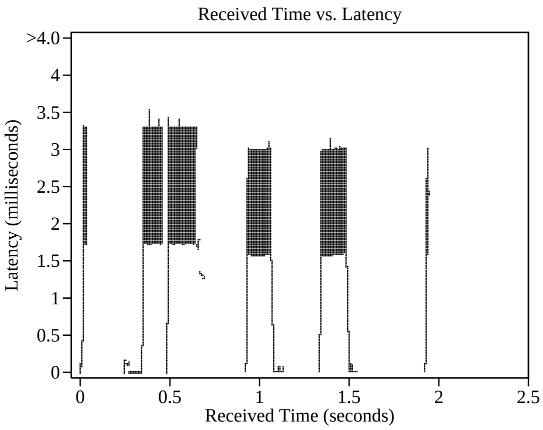

Figure 135: SIMPLE, No Combining, Run 5

4.3.13 SIMPLE, 1 Kword, 100 μsec, Combining

This section shows the results of 5 runs of the SIMPLE benchmark (set to send 512 1 Kword packets at a 100 μsec word pace) with combining enabled. Only packets associated with the benchmark's data page are shown. The small number of packets (less than 10) associated with barriers at the beginning and end of the run have been removed.

Figures 136–140 show the packet latency distributions.

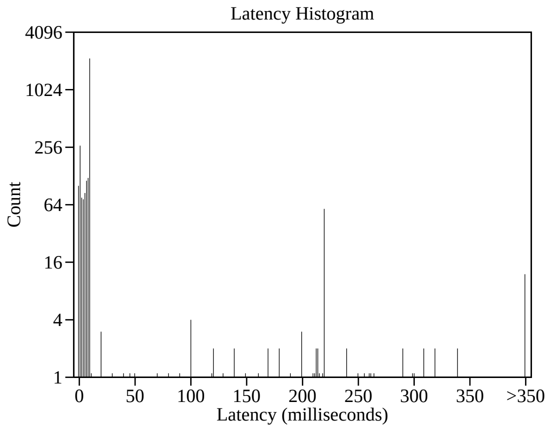

Figure 136: SIMPLE, Combining, Run 1

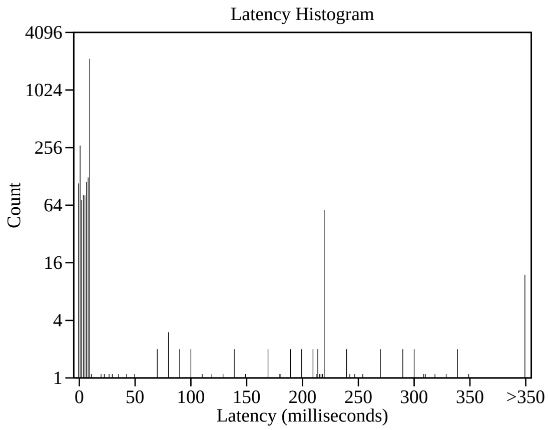

Figure 137: SIMPLE, Combining, Run 2

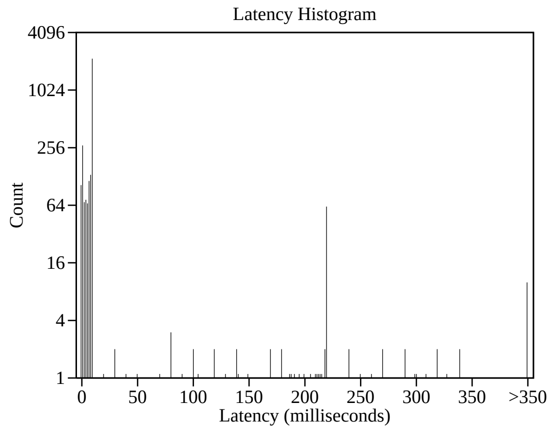

Figure 138: SIMPLE, Combining, Run 3

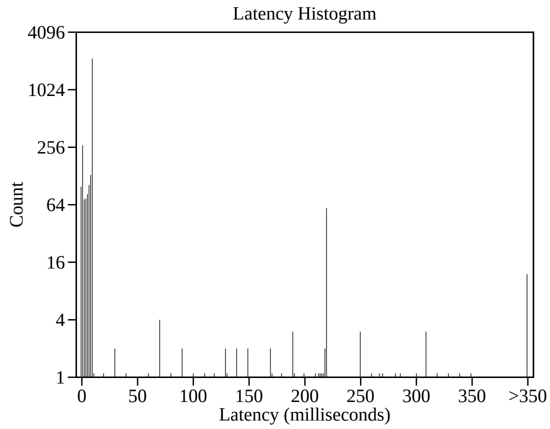

Figure 139: SIMPLE, Combining, Run 4

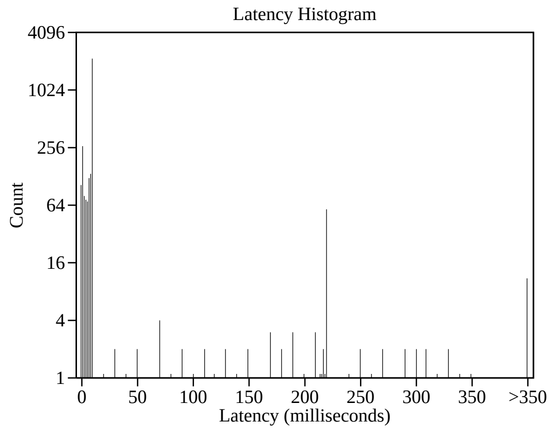

Figure 140: SIMPLE, Combining, Run 5

Figures 141–145 show the packet size distributions. Note that in the absence of interfering EISA bus traffic, all packets would be 1024 words. The distribution shows that approximately 25% of the 512 packets are full 1024 word packets. All other sizes in the distribution are due to packet fragmentation.

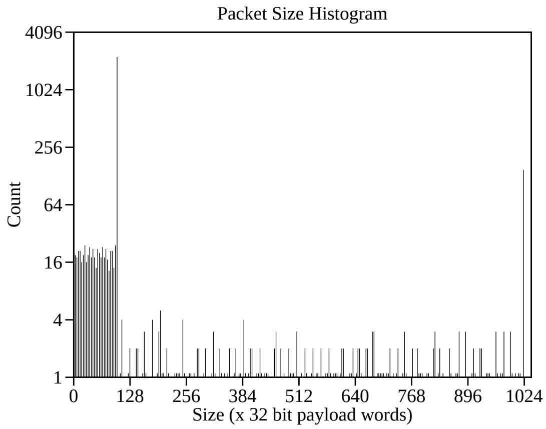

Figure 141: SIMPLE, Combining, Run 1

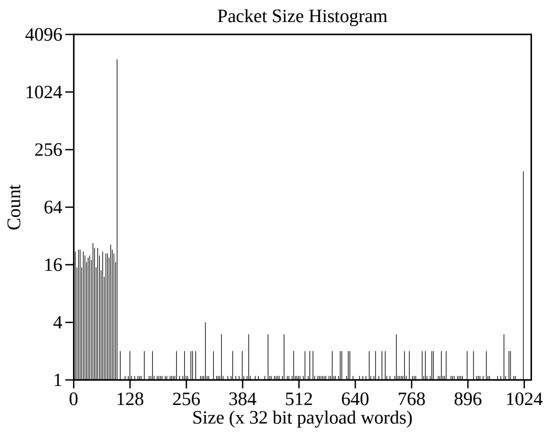

Figure 142: SIMPLE, Combining, Run 2

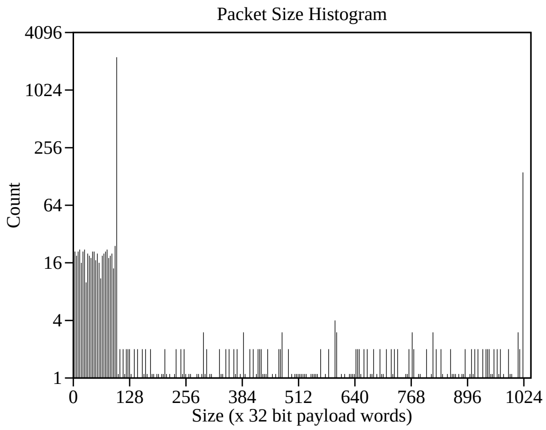

Figure 143: SIMPLE, Combining, Run 3

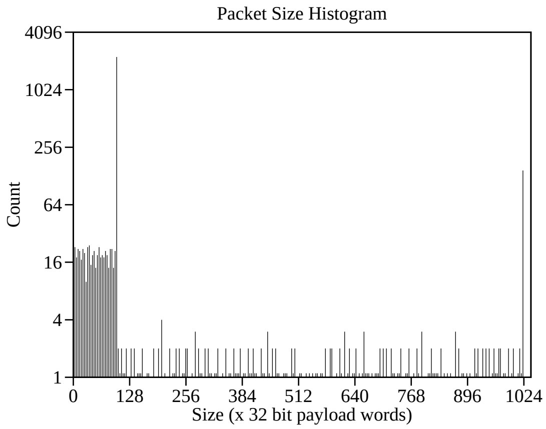

Figure 144: SIMPLE, Combining, Run 4

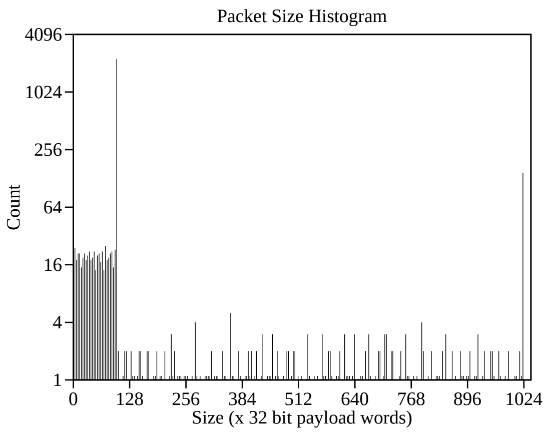

Figure 145: SIMPLE, Combining, Run 5

Figures 146–150 show graphs of latency versus time.

An interesting feature of these graphs is the thin line of points with a latency of approximately 200 msec. This is probably due to the Linux scheduler which uses a quantum of 200 msec.

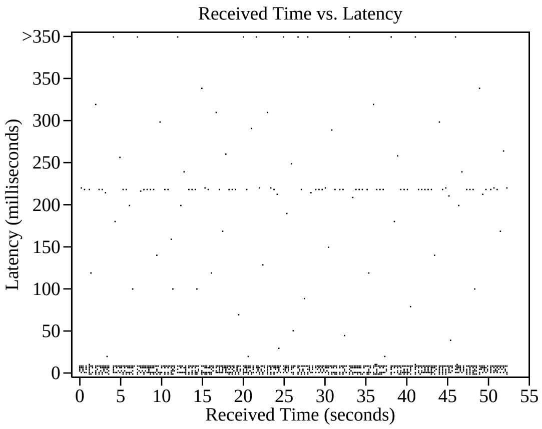

Figure 146: SIMPLE, Combining, Run 1

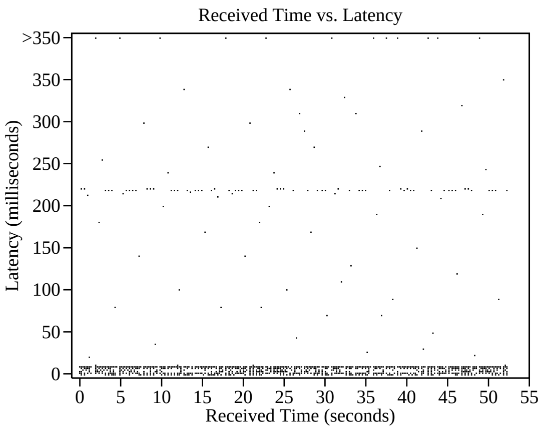

Figure 147: SIMPLE, Combining, Run 2

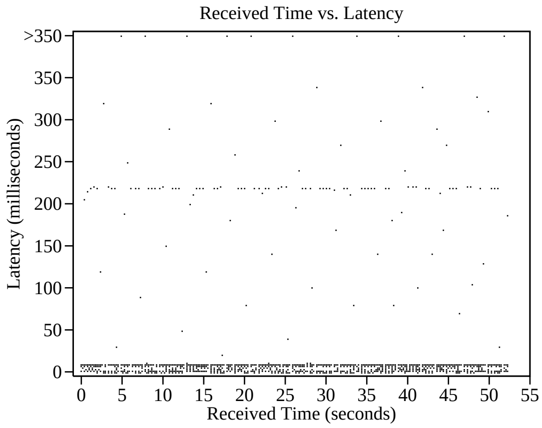

Figure 148: SIMPLE, Combining, Run 3

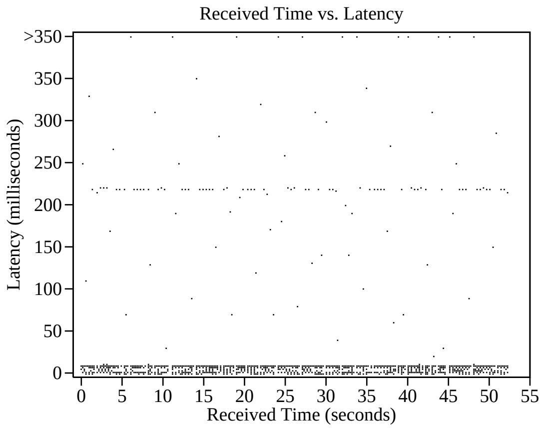

Figure 149: SIMPLE, Combining, Run 4

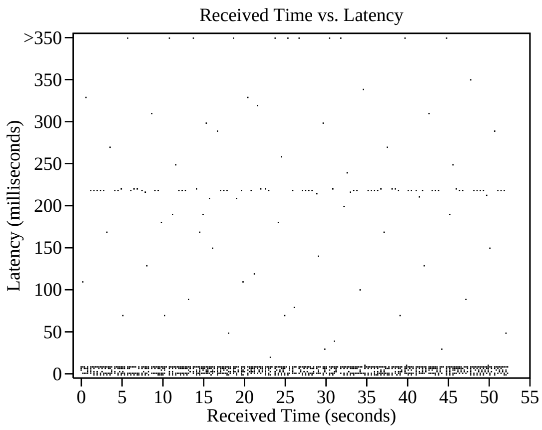

Figure 150: SIMPLE, Combining, Run 5

4.3.14 SIMPLE, 1 Kword, 100 μsec, No Combining

This section shows the results of 5 runs of the SIMPLE benchmark (set to send 512 1 Kword packets at a 100 μsec word pace) with combining disabled. Only packets associated with the benchmark's data page are shown. The small number of packets (less than 10) associated with barriers at the beginning and end of the run have been removed.

Because all packets are of size 1, the size distributions are not shown.

Figures 151–155 show the packet latency distributions.

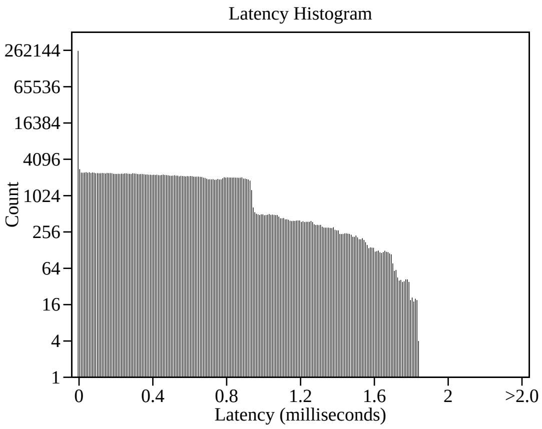

Figure 151: SIMPLE, No Combining, Run 1

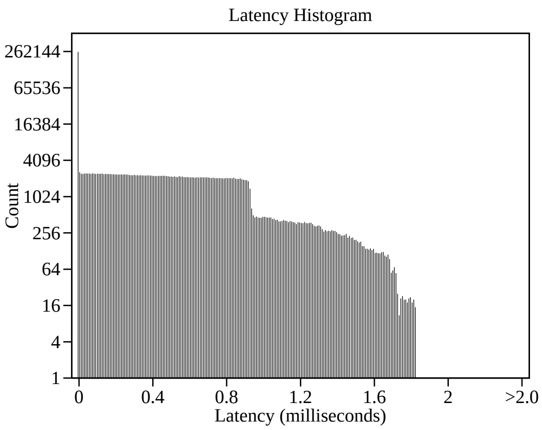

Figure 152: SIMPLE, No Combining, Run 2

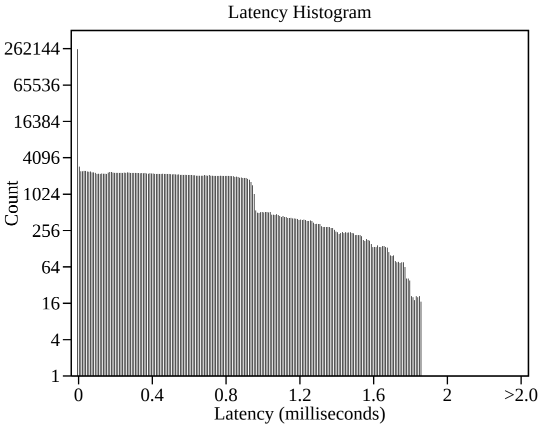

Figure 153: SIMPLE, No Combining, Run 3

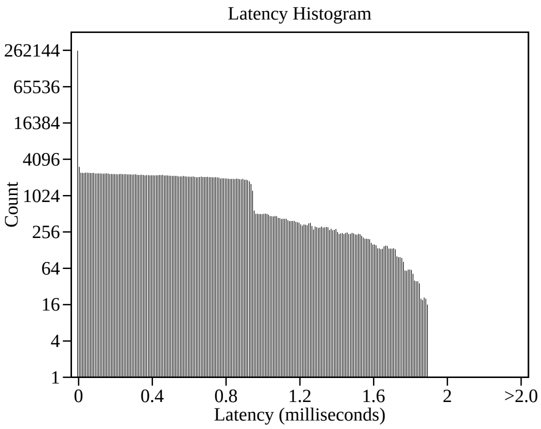

Figure 154: SIMPLE, No Combining, Run 4

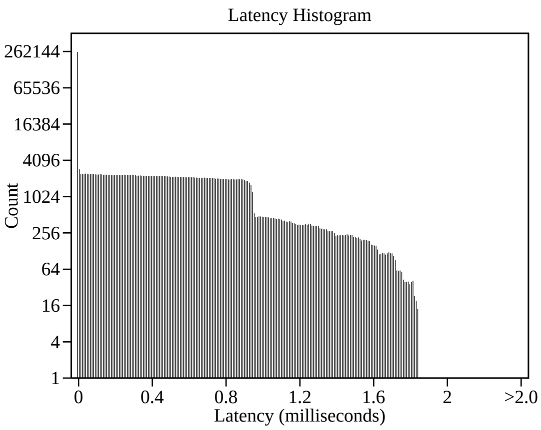

Figure 155: SIMPLE, No Combining, Run 5

Figures 156–160 show graphs of latency versus time. With combining disabled, the outgoing FIFOs repeatedly fill (indicated by the steep ramps); however, the 100 μsec pace allows the FIFOs to empty more often than the 1 μsec case resulting in lower maximum latencies (see Figures 131–135 on page 36).

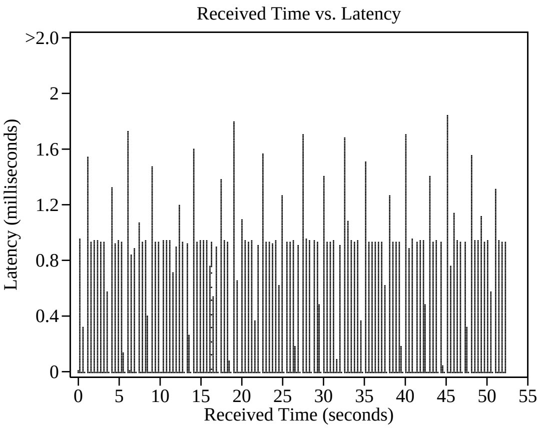

Figure 156: SIMPLE, No Combining, Run 1

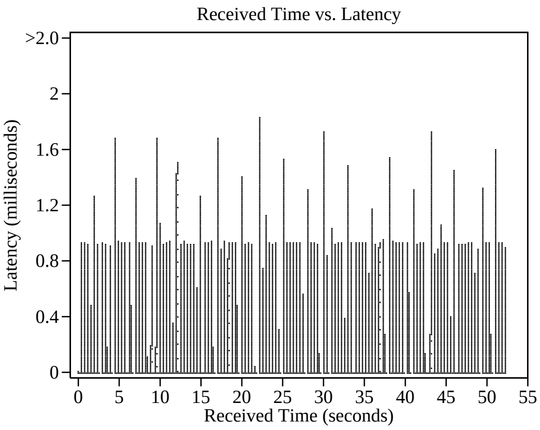

Figure 157: SIMPLE, No Combining, Run 2

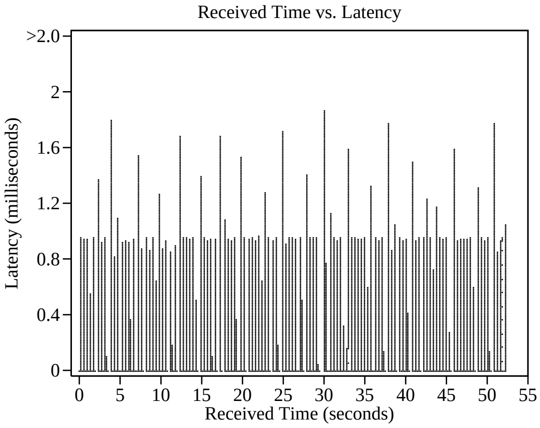

Figure 158: SIMPLE, No Combining, Run 3

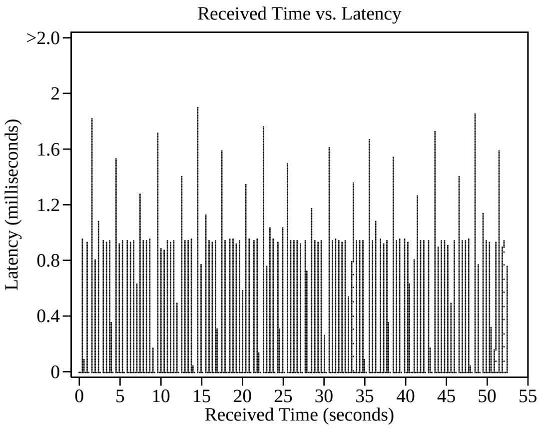

Figure 159: SIMPLE, No Combining, Run 4

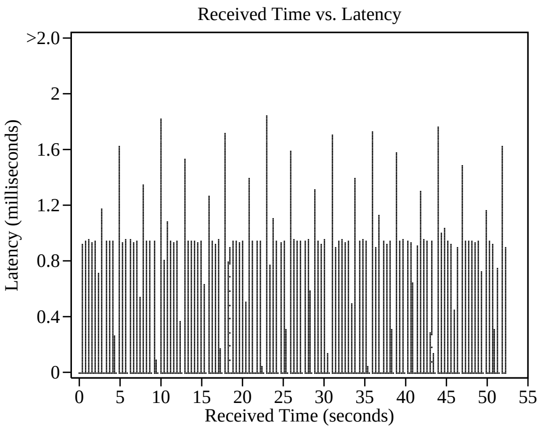

Figure 160: SIMPLE, No Combining, Run 5

4.4 Analysis

In this section we provide some analysis of the raw data presented in Section 4.3.

4.4.1 Backplane Effects

Previous analysis[3] suggests that the bandwidth of the Paragon backplane is fast enough that it will not be the bottleneck in application benchmarks (e.g., RADIX). To confirm this, we look at the joint distributions of run 1 of RADIX with combining enabled. (Other views of the same dataset can be found in Figures 45, 50, and 55.)

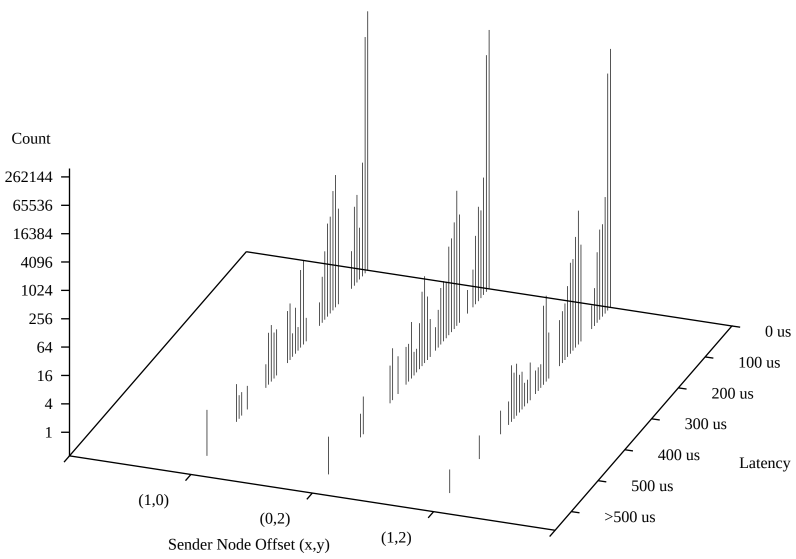

Figure 161: RADIX, Combining, Run 1 – Joint Latency and Sender Distribution

Figure 161 shows the joint latency and sender distribution. (There are 3 possible senders in a 4 node system.) From this we can see that the sender location has little effect on the final packet latency.

Figure 162 shows the joint latency and size distribution. Looking along the latency axis where the packet size is 1, we see that a size 1 packet can have many different latency values. This is due to the fact that there is a varying amount of time after a shared page write and before another non-consecutive access which forces the SNI to send the packet. Looking along the size axis where the latency is small, we see that for packets which are sent immediately (presumably deliberate update packets at release time), the size doesn't have a strong correlation with latency.

4.4.2 Combining Effects

Table 1 shows the effect combining has on packet counts for Run 1 of each benchmark. Table 2 shows the effect combining has on average packet sizes for Run 1 of each benchmark. In both tables, the SIMPLE benchmarks shown are only the ones in which combining was possible (combining enabled and group sizes of 1024 words).

For our benchmarks, combining did not have a significant effect on the overall runtime. It did, however, have a significant effect on the SNI board's outgoing FIFO (especially in the case of the SIMPLE benchmark). With combining disabled, the additional packet overhead causes the outgoing FIFO to fill up. When this occurs, the SNI board interrupts the CPU and enters a busy loop in the device driver waiting for the FIFO to empty to a low water mark.

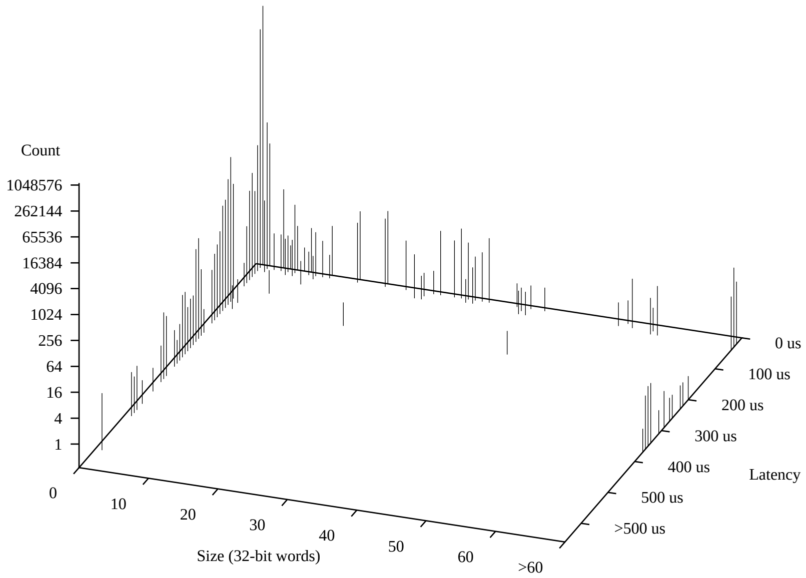

Figure 162: RADIX, Combining, Run 1 – Joint Latency and Size Distribution

Benchmark	Packet Count No Combining	Packet Count Combining	Packet Count Drop	Packet Count % drop due to Combining
LU	6458	6452	6	0.1
OCEAN	54477	33148	21329	39.2
RADIX	401789	394011	7778	1.9
SIMPLE 1 μs	524288	579	523709	99.9
SIMPLE 100 μs	524288	3146	521142	99.4

Table 1: Combining Effects on Packet Counts

Benchmark	Words/Packet No Combining	Words/Packet Combining	Words/Packet Increase	Words/Packet % increase due to Combining
LU	35.4	34.4	-1.1	-3.1
OCEAN	12.4	20.4	8.0	64.5
RADIX	1.0	1.0	0.0	2.0
SIMPLE 1 μs	1.0	905.5	904.5	90550.6
SIMPLE 100 μs	1.0	166.7	165.7	16565.2

Table 2: Combining Effects on Packet Size

Combining also interacts with the EISA bus occupancy. With combining disabled, each automatic update write becomes a separate packet. At the receiving node, each packet requires the SNI board to become the EISA bus master so that it can perform the DMA operation to move the packet data to the final location. When the CPU is actively writing data to memory (which is more common under automatic update as the caches are set to write-through), there can be significant contention for the EISA bus.

To see this effect, we take a closer look at Figure 131 on page 36: the latency vs. time graph of the SIMPLE benchmark (no combining, $1\,rmKword$ groups, $1\,\mu$sec pace).

Figures 163 through 167 show increasing detail as we zoom in at the activity 90 msec into the run (as measured from the receipt of the first data packet).

At the highest resolution, Figure 167, we see that packets are arriving approximately every $2\,\mu$sec with latencies that increase by approximately 500 nsec. By the nature of the SIMPLE program, all these packets are from the same sender and, in particular, come from the same outgoing FIFO on the SNI board.

An EISA bus acquisition time of 3–4 bus cycles corresponds to 500 nsec. It seems that the CPU on the receiving node is in contention with the receiving SNI board for the EISA bus. At the $1\,\mu$sec pace, the node is not able to arbitrate back and forth between these two masters fast enough to keep up with the data. As a result there are long stretches of time characterized by packets with latencies increasing in increments of the EISA bus acquisition time.

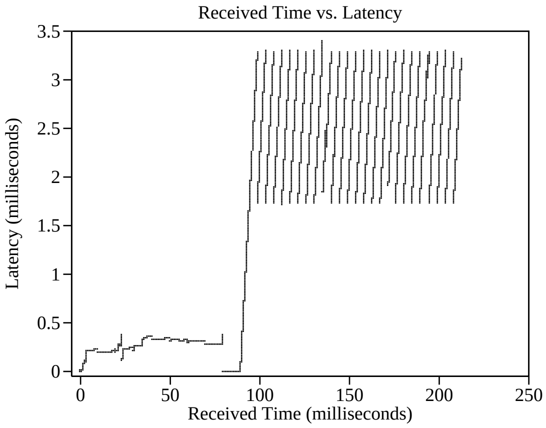

Figure 163: SIMPLE, No Combining, Run 1

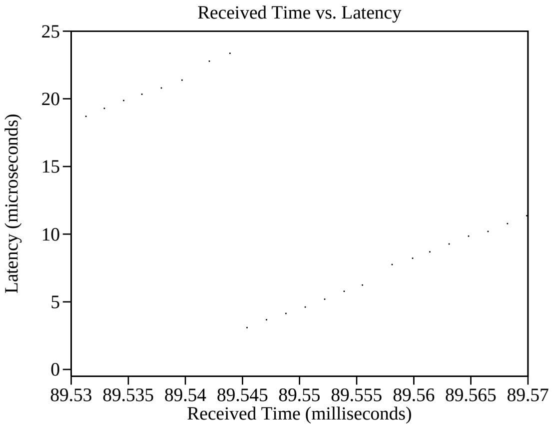

Figure 166: SIMPLE, No Combining, Run 1

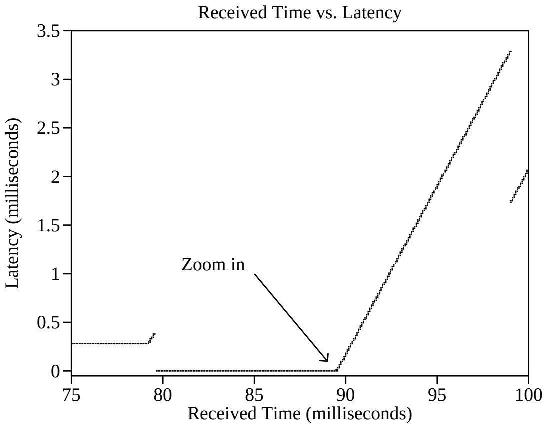

Figure 164: SIMPLE, No Combining, Run 1

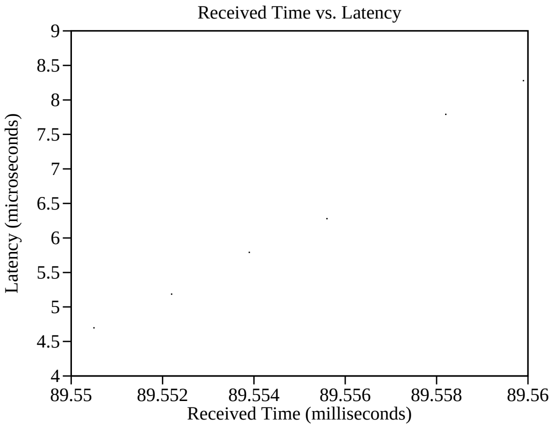

Figure 167: SIMPLE, No Combining, Run 1

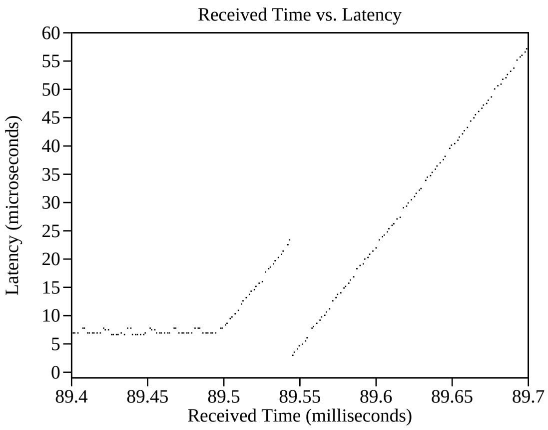

Figure 165: SIMPLE, No Combining, Run 1

5 Hardware Retrospective

This section describes the differences between this design and the previously proposed design [16] as well as some lessons learned.

5.1 Design Differences

Here is a list of some of the major design differences from the original performance monitor and the SurfBoard:

- Histogram address selection. Originally designed using an elaborate multiplexing scheme to allow immediate reconfiguration across a limited number of configurations. The new design uses an FPGA to implement the multiplexing. While a new configuration cannot be set in real-time, the simple design space allows for compilation and down load in approximately one minute. The new approach also allows for arbitrary selection of the bits from the input variables.
- Histogram and Trace modes are now independent and can operate simultaneously.
- Histogram overflow can trigger concurrent trace.
- Histogram memory width reduced from 40 to 32 bits, increase FIFO from a small size (implemented in an FPGA) to 1365 entries. Matching the memory width to the bus width simplified the design.
- No global threshold register, event occurs on overflow. Preset allows each bin to effectively have its own threshold.
- External triggering.

5.2 Lessons Learned

This was an ambitious hardware and software undertaking. Because of its large size and relatively late start in the development of the Shrimp system, the SurfBoard did not come "on line" until near the end of the useful life of the project. By this point most of the measurements that the SurfBoard can make directly were already made by instrumenting the software or by direct measurements using commercially available high-speed logic analyzers.

One of the factors which slowed the testing and integration of the SurfBoard and the SNI board was the interface specification between the two boards. Given the academic nature of the project and the natural "turn over" in personnel (i.e., graduate students) it is not uncommon for specification problems to arise.

The independent design of the word count, histogram, and trace modes simplified the hardware debugging as the modes could "cross check" each other's results.

Lessons learned:

- Reduce the scope of the design.
- For components which must work together (e.g., the SNI board board and the SurfBoard), design, build, and test both components in parallel.
- Overlapping functionality (i.e., word count, histogram, and trace modes) simplified hardware debugging.

6 Related Work

The design of the SurfBoard was originally based on the Shrimp performance monitor design described in [16].

This section discusses the relationship of the SurfBoard to several previously developed projects. For example, the Stanford DASH multiprocessor [15] included a per-cluster histogram-based performance monitor

[12]. In the DASH monitor, histogramming is fixed at the time the FPGA was compiled. While the SurfBoard's histogramming is also fixed at the time the FPGA is compiled, the FPGA implements only the interconnections (there are no storage elements) of the histogram. As a result, we can recompile a new histogram configuration in about one minute. The histograms allow statistics to be categorized into two user and two operating system categories, or by subsets of the data address bits. A later DASH performance monitor configuration was designed specifically to allow CPU interrupts and OS responses based on observed per-page statistics, but did not allow for general interrupts based on any observed statistic. In contrast, our hardware monitor supports both such specific studies as well as more general monitoring.

The performance monitoring system for Cedar used simple histograms [14], while IBM RP3 used a small set of hardware event counters [6]. The Intel Paragon includes rudimentary per-node counters [18], but cannot measure message latency. Histogram-based hardware monitors were also used to measure uniprocessor performance in the VAX models 11/780 and 8800 [8, 11]. These monitors offered less flexible histogramming, and could not categorize statistics based on data regions or interrupt the processor based on a user-set threshold.

On-chip performance monitors are becoming more common for CPU chips. For example, Intel's Pentium line of CPUs incorporate extensive on-chip monitoring [17]. The Pentium performance counters include information on the number of reads and writes, the number of read misses and write misses, pipeline stalls, TLB misses, etc. Here also, there is no support for categorization of statistics or for selective CPU notification. In contrast, the Alpha 21064 does provide some base level of performance monitoring with selective CPU notification [9]. Its on-chip cache performance counter is used by initializing it to a particular value, and then decrementing it whenever a cache miss occurs; when the counter value reaches zero, the CPU is interrupted.

Some researchers have examined using monitoring information to guide operating system policy decisions. For example, Bershad *et al.* proposed a special-purpose hardware monitor (Cache Miss Lookaside Buffer) that would keep per-page statistics on memory behavior in order to guide operating system decisions about virtual-to-physical page mappings [2]. Chandra *et al.* investigated the potential of dynamically using measured data from a more general purpose hardware performance monitor to guide operating system scheduling and page migration decisions [7]. This approach is closer to ours, but they used an existing performance monitor [12] and focused their attention mainly on determining appropriate operating system policies.

7 Conclusions

This technical report has described the design of the hardware and software that make up Shrimp Usage Reporting Facility (SURF) system and reported the results from benchmark studies on an SVM architecture using the SurfBoard.

The SurfBoard design provides flexible instrumentation mechanisms such as multi-dimensional packet-based histograms, page tags, histogram categories, and threshold-driven interrupts.

The Shrimp SVM implementation (AURC) uses an implicit form of interprocessor communication which is very difficult to measure without hardware monitoring. We have shown that the SurfBoard can easily measure this implicit traffic and can determine to what extent packet combining is taking place for a given application.

Acknowledgments

We extend our thanks to Richard Alpert, Angelos Bilas, Yuqun Chen, Liviu Iftode, Rob Shillner, and Yuanyuan Zhou of the Shrimp team who offered us their generous assistance during this research. Matthias Blumrich designed the SNI board.

This project is sponsored in part by ARPA under grant N00014–95–1–1144, by NSF under grant MIP–9420653, and by Intel Corporation. Margaret Martonosi is supported in part by an NSF Career award.

References

[1] BCPR Services Inc. *EISA Specification, Version 3.12*, 1992.

[2] B. N. Bershad, D. Lee, T. H. Romer, and J. B. Chen. Avoiding Conflict Misses Dynamically in Large Direct-Mapped Caches. In *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 158–170, October 1994.

[3] M. A. Blumrich. *Network Interface for Protected, User-Level Communication*. PhD thesis, Princeton University, 1996.

[4] M. A. Blumrich, C. Dubnicki, E. Felten, K. Li, and M. Mesarina. Virtual Memory Mapped Network Interfaces. *IEEE MICRO*, pages 21–28, February 1995.

[5] M. A. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. W. Felten, and J. Sandberg. Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 142–153, April 1994.

[6] W. C. Brantley, K. P. McAuliffe, and T. A. Ngo. RP3 Performance Monitoring Hardware. In Simmons, Koskela, and Bucher, editors, *Instrumentation for Future Parallel Computing Systems*, pages 35–43. ACM Press, 1989.

[7] R. Chandra, S. Devine, B. Verghese, A. Gupta, and M. Rosenblum. Scheduling and Page Migration for Multiprocessor Compute Servers. In *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 12–24, October 1994.

[8] D. W. Clark, P. J. Bannon, and J. B. Keller. Measuring VAX 8800 Performance with a Histogram Hardware Monitor. In *Proceedings of the 15th Annual International Symposium on Computer Architecture*, pages 176–185, May 1988.

[9] Digital Equipment Corporation. DECChip 21064 RISC Microprocessor Preliminary Data Sheet. Technical report, 1992.

[10] C. Dubnicki, L. Iftode, E. W. Felten, and K. Li. Software Support for Virtual Memory-Mapped Communication. In *Proceedings of the IEEE 10th International Parallel Processing Symposium*, April 1996.

[11] J. S. Emer and D. W. Clark. A Characterization of Processor Performance in the VAX–11/780. In *Proceedings of the 11th Annual International Symposium on Computer Architecture*, pages 301–310, June 1984.

[12] M. A. Heinrich. DASH Performance Monitor Hardware Documentation. Stanford University, Unpublished Memo, 1993.

[13] L. Iftode, C. Dubnicki, E. W. Felten, and K. Li. Improving Release-Consistent Shared Virtual Memory using Automatic Update. In *Proceedings of the 2nd International Symposium on High-Performance Computer Architecture*, February 1996.

[14] D. Kuck, E. Davidson, D. Lawrie, et al. The Cedar System and an Initial Performance Study. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 213–223, May 1993.

[15] D. Lenoski, J. Laudon, et al. The DASH Prototype: Logic Overhead and Performance. *IEEE Transactions on Parallel and Distributed Systems*, pages 41–61, January 1993.

[16] M. Martonosi, D. Clark, and M. Mesarina. The SHRIMP Performance Monitor: Design and Applications. In *ACM SIGMETRICS Symposium on Parallel and Distributed Tools*, 1996.

[17] T. Mathisen. Pentium Secrets. *Byte*, pages 191–192, July 1994.

[18] J. Rattner. Paragon System. Presentation: *DARPA High Performance Software Conference*, January 1992.

[19] R. Traylor and D. Dunning. Routing Chip Set for Intel Paragon Parallel Supercomputer. In *Proceedings Hot Chips 1992 Symposium*, August 1992.

[20] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. Methodological Considerations and Characterization of the SPLASH–2 Parallel Application Suite. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, May 1995.

A Hardware Specification

This section describes the SurfBoard hardware in detail.

A.1 Architecture

The design of the SurfBoard is a three stage pipeline (see Figure 2 on page 4).

Stage 1 runs at twice the EISA bus `BCLK` rate, or approximately 16.7 MHz. The EISA specification allows the PC motherboard to occasionally stretch cycles; we have observed this in our systems. The SNI board sends packet data which is synchronized to this clock. The exception is the block labeled “Rx Time Counter and Register” which implements the 44-bit timestamp counter and runs off the Paragon global 10 MHz clock supplied by the SNI board. A framing signal (`PMEN_L`) from the SNI board synchronizes a local state machine which causes the various registers in Stage 1 to sample the packet data at the correct time. Additionally, there is a 10-bit size counter which counts the number of words in a packet as it arrives.

After a complete packet arrives and Stage 1 has sampled data from the packet, Stage 2 is signaled to continue processing the data. Stage 2 runs at 25 MHz. This higher clock rate is used so that the 44-bit subtraction performed by the Latency Generator can be implemented using a single 16-bit ALU (the ALU performs the subtraction in three stages). After the Latency Generator, the Histogram Bin Selector (also known as the Histogram Address Selector) reduces 50 bits of packet information to a 24 bit address. The Trace Register samples all 126 bits of collected packet data. At the end of Stage 2, the histogram address, trace data, packet size, and latency flags are presented to Stage 3.

Stage 3 consists of the Histogram Subsystem, the Trace Subsystem, and the System Controller. This stage also runs at 25 MHz (with the exception of the EISA bus interface section of the System Controller which runs at the `BCLK` rate of approximately 8.3 MHz. The Histogram Subsystem contains the histogram memory and word counter. These share a 32-bit ALU which performs both the histogram bin increment and the word count accumulation. The Trace Subsystem contains the trace memory and counters which track the current index into the memory. The System Controller controls the operation of the SurfBoard including memory arbitration between the incoming packet data, EISA bus requests, and DRAM refresh.

A.2 SNI board to SurfBoard Interface

The external connectors of the SurfBoard use the pin numbering scheme which gives the wire numbers in the ribbon cable the same number as the connected pin. The pin numbers are marked on the SurfBoard. Note that this is *different* than the convention used by the SNI board.

J3 SurfBoard Pin Out (Connects to J7 on the SNI board.)

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	S_DATA(31)	11	S_DATA(21)	21	S_DATA(11)	31	nc
2	S_DATA(30)	12	S_DATA(20)	22	nc	32	nc
3	S_DATA(29)	13	S_DATA(19)	23	nc	33	S_COLOR(5)
4	S_DATA(28)	14	S_DATA(18)	24	nc	34	S_COLOR(4)
5	S_DATA(27)	15	S_DATA(17)	25	nc	35	S_COLOR(3)
6	S_DATA(26)	16	S_DATA(16)	26	nc	36	S_COLOR(2)
7	S_DATA(25)	17	S_DATA(15)	27	nc	37	S_COLOR(1)
8	S_DATA(24)	18	S_DATA(14)	28	nc	38	S_COLOR(0)
9	S_DATA(23)	19	S_DATA(13)	29	nc	39	nc
10	S_DATA(22)	20	S_DATA(12)	30	nc	40	nc

J2 SurfBoard Pin Out (Connects to J8 on the SNI board.)

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	S_DATA(63)	11	S_DATA(53)	21	S_DATA(43)	31	S_DATA(33)
2	S_DATA(62)	12	S_DATA(52)	22	S_DATA(42)	32	S_DATA(32)
3	S_DATA(61)	13	S_DATA(51)	23	S_DATA(41)	33	PMEN_L
4	S_DATA(60)	14	S_DATA(50)	24	S_DATA(40)	34	nc
5	nc	15	S_DATA(49)	25	S_DATA(39)	35	nc
6	S_DATA(58)	16	S_DATA(48)	26	S_DATA(38)	36	S_EOD_L
7	S_DATA(57)	17	S_DATA(47)	27	S_DATA(37)	37	S_GCLK
8	S_DATA(56)	18	S_DATA(46)	28	S_DATA(36)	38	S_GRST_L
9	S_DATA(55)	19	S_DATA(45)	29	S_DATA(35)	39	nc
10	S_DATA(54)	20	S_DATA(44)	30	S_DATA(34)	40	nc

A.3 LED Assignments

LED #	Color	Signal	Description
1	Red	X_IN_L	External Input (at 0 volts)
2	Red	ANYRST_L	Any Reset (Power-on or EISA)
3	Red	TRG_L	Trigger Occurred (HTCR.TRG)
4	Yellow	FNE_L	FIFO Not Empty
5	Yellow	TEN_L	Trace Enabled (HTCR.TEN)
6	Yellow	E_ACCESS_L	EISA Access
7	Yellow	E_IRQPEND_L	EISA IRQ Pending (ISCR.PEND)
8	Green	T_RUN2_L	Packet in Stage 2
9	Green	T_RUN1_L	Packet in Stage 1
10	Green	ULED_L	User Defined (PCLR.LED)

A.4 Software Interface Description

The SurfBoard is a 32-bit EISA slave board [1]. Except where noted, all I/O space registers, memory mapped registers, the histogram, and the trace memory can be accessed (read or write) using 8-bit, 16-bit, or 32-bit transfers. The EISA bus is little endian.

A.5 I/O Space Registers / Bit Descriptions

The I/O registers are slot specific per the EISA specification (denoted by a "z" in the port address). The following is a list of I/O space registers. They are described in detail on the referenced page.

Port	R/W	Name	Description	Page Reference
0zC80	R	EPIR	EISA Product Identification Register	51
0zC84	R/W	EBCR	EISA Board Configuration Register	51

EPIR EISA Product Identifier Register (read only) 0zC80

Bits	Field	Description
31..28	PNUM	Product Number (low 4 bits)
27..24	RLEV	Revision Level
23..16	PNUM	Product Number (high 8 bits)
15..13	MC2	Manufacturer Code 2nd Letter (low 3 bits)
12..8	MC3	Manufacturer Code 3rd Letter
7	ZERO	This bit must be zero. It is used to check for the existence of a board in this slot.
6..2	MC2	Manufacturer Code 2nd Letter (high 2 bits)
5..0	MC1	Manufacturer Code 1st Letter

MCn – The EISA manufacturer code is encoded using 3 letters each consisting of 5 bits. The letters A to Z map to values from 0x01 to 0x1A respectively. In the case of the SurfBoard, the letters are "SRF".

PNUM – The 12-bit product number. In the case of the SurfBoard, the product number is 0x001.

The concatenation of the manufacturer code, the three digit product number, and the one digit revision level is known as the product ID string.

The SurfBoard EISA configuration revision level history is as follows:

Date	Rev	Produce ID String	EPIR Value (32-bit read)
11–Jan–1998	0	SRF0010	0x1000464E

The EISA configuration file name consists of an exclamation point followed by the product ID string with the file extension of "CFG". The name of the revision 0 SurfBoard EISA configuration file is therefore, `!SRF0010.CFG`.

EBCR EISA Board Configuration Register 0zC84

Bits	Field	Description
31..16	RBASE	Memory Mapped Register Base Address
15..11	MBASE	Memory Base Address
10..8	IRLEV	Interrupt Request Level
7..1		Unused. Reserved by EISA specification.
0	ENABLE	Board Enable. (0 = disable; 1 = enable)

PNUM – The 12-bit product number. In the case of the SurfBoard, the product number is 0x001.

RBASE – The top 16 bits of the 32-bit base address of the 64Kbyte memory mapped register address space.

MBASE – The top 5 bits of the 32-bit base address of the trace memory and histogram memory 128Mbyte address space. The trace memory begins at offset 0 and the histogram memory begins at offset 0x04000000.

IRLEV – A three bit value indicating the interrupt level used by the board. The SurfBoard supports interrupts on levels 5, 9, 10, 12, 14, and 15 based on IRLEV as follows:

000	–	Disabled	100	–	IRQ 10
001	–	Disabled	101	–	IRQ 12
010	–	IRQ 5	110	–	IRQ 14
011	–	IRQ 9	111	–	IRQ 15

ENABLE – This bit is the EISA required enable bit. When disabled (the default after power-up), the board only responds to EISA slot-specific I/O operations on the EPIR and EBCR registers. All interrupts are disabled, packet monitoring is disabled, memory mapped register access is disabled, and the external open-collector output signal is disabled. Refresh cycles continue on the histogram and trace DRAM to preserve data. Monitoring registers and the histogram index FIFO retain their values (but are inaccessible).

A.6 Memory Mapped Registers / Bit Descriptions

The register offsets are relative to RBASE in the EBCR. The address space for the memory mapped registers is 64 Kbytes. The following registers are redundantly mapped 1024 times in the address space. They are described in detail on the referenced page.

Port	R/W	Name	Description	Page Reference
0x0000	R/W	FCCR	FPGA Configuration Control Register	52
0x0004	R/W	FCDR	FPGA Configuration Data Register	53
0x0008	R/W	ACDR	Application Configuration Data Register	54
0x000C	R/W	TTCR	Trace Trigger Configuration Register	55
0x0010	R/W	ECDR	External Connector Data Register	57
0x0014	R/W	PCLR	Packet Category / LED Register	57
0x0018	R/W	ISCR	Interrupt Status / Control Register	57
0x001C	R/W	IEMR	Interrupt Event Mask Register	59
0x0020	R/W	HTCR	Histogram / Trace Command Register	59
0x0024	R/W	WCDR	Word Count Data Register	60
0x0028	R/W	HFCR	Histogram FIFO Control Register	60
0x002C	R/W	HFDR	Histogram FIFO Data Register	61
0x0030	R/W	TICR	Trace Index Count Register	61
0x0034	R/W	TECR	Trace Event Count Register	61
0x0038	R/W	TPIR	Trigger Packet Index Register	62
0x003C			Unused. Writes ignored. Reads undefined.	

FCCR	FPGA Configuration Control Register	0x0000

Bits	Field	Description
31..12		Unused. Reads as 0 bits. Writes are ignored.
11..8	TSPD	Trace SIMM Presence Detect (read only)
7..4	HSPD	Histogram SIMM Presence Detect (read only)
3	DONE	Done (read only)
2	ERR	Error (read only)
1	CFG	Configuration (read only)
0	PROG	Program

TSPD – Trace SIMM Presence Detect. This is the JEDEC defined 4-bit value from the Trace DRAM SIMM. It can be used to determine the type and speed of the SIMM. The MSB of this field corresponds to PRD4 and the LSB corresponds to PRD1. (See table below.)

HSPD – Histogram SIMM Presence Detect. This is the JEDEC defined 4-bit value from the Histogram DRAM SIMM. It can be used to determine the type and speed of the SIMM. The MSB of this field corresponds to PRD4 and the LSB corresponds to PRD1.

	Presence Detect			
Memory Module (SIMM) Type	PRD4	PRD3	PRD2	PRD1
Micron 1 M $\times$ 32 60 ns MT8D132	1	1	0	0
Micron 4 M $\times$ 32 60 ns MT8D432	1	1	1	0
Empty SIMM Socket	1	1	1	1

For the SIMMs we are currently using (listed in the table above), the presence detect will unambiguously detect the size of the device. However, this may not be the case with other SIMMs we may use in the future. In the case where the presence detect does not unambiguously determine the memory size, an algorithm must be implemented to determine the memory size by writing and reading to various memory locations to determine the memory extent.

DONE – This bit indicates that the FPGA has completed configuration. This status bit follows the `DONE` pin on the FPGA.

ERR – This bit indicates that an error occurred during the configuration of the FPGA. This status bit is the logical inversion of the `INIT*` pin on the FPGA.

CFG – This bit indicates that the FPGA is in the configuration mode. This status bit follows the `HDC` pin on the FPGA.

PROG – Writing a 1 forces the FPGA into the configuration mode. This signal is inverted and drives the `PROG*` pin on the FPGA. After power-on reset, this bit will be set to a 1 keeping the FPGA in configuration mode until the initialization software is ready to program the device.

The FPGA uses the “Slave Serial Mode” of configuration as described in the device data sheet.

FCDR	FPGA Configuration Data Register	0x0004

Bits	Field	Description
31..1		Unused. Reads as 0 bits. Writes are ignored.
0	DATA	Data

DATA – Each time this register is written, the value of DATA is clocked (using `CCLK`) into the `DIN` pin of the FPGA. Reading this register returns the value of the last stage in the shift register and does not move any bits.

ACDR	Application Configuration Data Register	0x0008

Bits	Field	Description
31..1		Unused. Reads as 0 bits. Writes are ignored.
0	DATA	Data

DATA – Each time this register is written, the value of DATA is clocked into the 28-bit shift register used to configure the SurfBoard's packet monitoring. Reading this register returns the value of the last stage in the shift register and does not move any bits. By reading and then writing this location a total of 28 times, the entire configuration will be recirculated one time.

The shift register contents are in the following table. Items which are listed first are shifted in last. In other words, bits are shifted "in" at the top of the list and bits are shifted "out" at the bottom of the list. Items with more than one bit are shifted in LSB first.

Name	Description	Bits
SENDID	Sender Identification	3
RDBK_A	Readback A (unused/reserved)	1
CATMUX	Category Multiplexer	1
RDBK_B	Readback B (unused/reserved)	3
LAT_S	Latency Configuration (S)	5
LAT_N	Latency Configuration (N)	4
RDBK_C	Readback C (unused/reserved)	3
TSIZE	Trace DRAM size	2
RDBK_D	Readback D (unused/reserved)	2
RDBK_F	Readback F (unused/reserved)	4

SENDID – Sender Identification (3 bits). This setting selects how the 6-bit Sender ID field is generated based on the sender's X and Y coordinates as follows:

000 – 32×2 (1 LSB of Y concatenated with 5 LSB of X)
001 – 16×4 (2 LSB of Y concatenated with 4 LSB of X)
010 – 8×8 (3 LSB of Y concatenated with 3 LSB of X)
011 – 4×16 (4 LSB of Y concatenated with 2 LSB of X)
100 – 2×32 (5 LSB of Y concatenated with 1 LSB of X)
101 – undefined
110 – undefined
111 – undefined

RDBK_A – Readback A (1 bit). Readback bits occupy positions in the shift register. They are unused and are reserved for future use.

CATMUX – Category Multiplexer (1 bit). This bit selects which value will be used as the category for the packet. To use the value of the PCLR.CAT register as the packet category, set this bit to a 0. To use the upper four bits of the first 32-bit word of user data as the packet category, set this bit to a 1.

RDBK_B – Readback B (3 bits). Readback bits occupy positions in the shift register. They are unused and are reserved for future use.

Latency Configuration: The next two fields in the shift register configure the operation of the latency generator. The latency generator subtracts the 44-bit transmit time from the 44-bit receive time to obtain a 44-bit latency. The resolution of the latency is 100 ns. A 24-bit "window" is chosen from the 44-bit latency based on two settings:

NUM: the number of bits from the 44-bit latency to put in the 24-bit window. Allowed values of NUM are even integers from 2 to 24.

START: the starting bit position of interest in the 44-bit latency. Allowed values of START are even integers from 0 to 40.

As an example, if we choose NUM=12 and START=6, the resolution of our window is $2^6 \times 100\,\text{ns} = 6.4\,\mu\text{s}$ and the largest value we can represent is $(2^{12} - 1) \times 6.4\,\mu\text{s} = 26.2\,\text{ms}$.

Since both NUM and START must be even integers, we use N and S to represent NUM/2 and START/2 respectively.

LAT_S – Latency Configuration S (5 bits). START/2. S may be any of 0 to 20 inclusive.

LAT_N – Latency Configuration N (4 bits). NUM/2. N may be any of 1 to 12 inclusive.

RDBK_C – Readback C (3 bits). Readback bits occupy positions in the shift register. They are unused and are reserved for future use.

TSIZE – Trace DRAM size (2 bits). At board initialization time, software should determine the actual amount of trace memory and select the corresponding value for the trace DRAM size as follows:

00	–	256 Bytes (for testing)
01	–	4 MBytes
10	–	16 MBytes
11	–	64 MBytes

RDBK_D – Readback D (2 bits). Readback bits occupy positions in the shift register. They are unused and are reserved for future use.

RDBK_E – Readback E (4 bits). Readback bits occupy positions in the shift register. They are unused and are reserved for future use.

RDBK_F – Readback F (4 bits). Readback bits occupy positions in the shift register. They are unused and are reserved for future use.

TTCR Trace Trigger Configuration Register 0x000C

Bits	Field	Description
31..9		Unused. Reads as 0 bits. Writes are ignored.
8	EIP	External Input Polarity.
7	EOP	External Output Polarity.
6	MEOD	Mask External Output Data.
5	MEI	Mask External Input.
4	MBV	Mask Bin Overflow.
3..2	MODE	Trace Mode.
1..0	QUAL	Trigger Packet Qualifier.

The external connector uses open-collector active-low signaling. The normal polarity for the output is that a TRUE will cause the line to go low. Any other devices connected to this signal may also pull the line low using wired-OR logic. The normal polarity for the input is that a low level will be interpreted as TRUE.

EIP – External Input Polarity. (1 = Invert; 0 = No-invert). When inverted, interpret a low signal on the input as a FALSE.

EOP – External Output Polarity. (1 = Invert; 0 = No-invert). When inverted, the output is driven low when the output should be considered FALSE.

The logical value for the external connector is determined by the following equation:

$$\text{Output} = (\text{MEOD} \cdot \text{ECDR.DATA}) + (\text{MEI} \cdot \text{Input}) + (\text{MBV} \cdot \text{BOC})$$

MEOD – Mask External Output Data. When this bit is 0, the DATA bit of the ECDR will masked out and will not contribute to the external output value.

MEI – Mask External Input. When this bit is 0, the logical value (TRUE/FALSE) of the external input will be masked out and will not contribute to the external output value.

MBV – Mask Bin Overflow. When this bit is 0, the bin overflow condition (BOC) will be masked out and will not contribute to the external output value. The BOC is set when a histogram bin overflows; the BOC is cleared when a 1 is written to bit BV of the ISCR register.

MODE – These bits select where the trigger packet occurs in a trace:

- 00 – trigger is at the beginning of the trace
- 01 – trigger is at the middle of the trace
- 10 – trigger is at the end of the trace
- 11 – reserved

QUAL – These bits determine the qualifier (after the trace is enabled by the HTCR register) needed for a packet to be the trigger packet:

- 00 – Any packet will trigger a trace.
- 01 – External input. If the external input is TRUE when a packet arrives, that packet will be the trigger.
- 10 – Histogram bin overflow. Any packet which causes a histogram bin overflow will trigger a trace.
- 11 – The logical-OR of the previous two conditions.

When trace is enabled (HTCR.TEN = 1) and a qualifying packet arrives, HTCR.TRG is set to 1. This in turn will cause ISCR.TP to be set.

ECDR — External Connector Data Register — 0x0010

Bits	Field	Description
31..1		Unused. Reads as 0 bits. Writes are ignored.
0	DATA	Data

DATA – Logical Output Data. (1 = logical TRUE; 0 = logical FALSE). See the MEOD field of the TTCR register for details.

PCLR — Packet Category / LED Register — 0x0014

Bits	Field	Description
31..5		Unused. Reads as 0 bits. Writes are ignored.
4	LED	LED Indicator.
3..0	CAT	Category.

LED – LED Indicator. Writing a 1 or a 0 turns the user LED on or off, respectively.

CAT – Category. Each incoming packet is assigned a 4-bit category which can be used to differentiate among packet types. If the SurfBoard is configured for local category (the Category mux of the application shift register is 0), incoming packets will be assigned the category of this 4-bit field. When remote category is selected, the top four bits of the first word of the incoming packet is used as the category. Because there are three stages in the packet processing pipeline, up to three packets may be processed with the previous value of the category register.

ISCR — Interrupt Status / Control Register — 0x0018

Bits	Field	Description	
31	PEND	Interrupt Pending.	(read only)
30..14		Unused. Reads as 0 bits. Writes are ignored.	
13	RTV	Receive Time Overflow	(read only)
12	LU	Latency Underflow	(write 1 to clear)
11	LV	Latency Overflow	(write 1 to clear)
10	PV	Pipeline Overflow	(write 1 to clear)
9	WCV	Word Counter Overflow	(write 1 to clear)
8	BV	Bin Overflow	(write 1 to clear)
7	FV	FIFO Overflow	(write 1 to clear)
6	FF	FIFO Full	(read only)
5	FAF	FIFO Almost Full	(read only)
4	FPF	FIFO Partially Full	(read only)
3	FNE	FIFO Not Empty	(read only)
2	EXT	External Input	(write 1 to clear)
1	TP	Trigger Packet	(write 1 to clear)
0	TC	Trace Complete	(write 1 to clear)

There are 14 different conditions which may cause an interrupt. The PEND bit will be a one when the board is actively asserting one of the IRQ lines on the EISA bus. The PEND bit is the sign bit of a typical 32-bit signed integer so that testing if this board has caused an interrupt can be made by checking the sign of the ISCR register. Note that the interrupt conditions (bits 13..0) reflect actual status and are not masked by the IEMR.

PEND – Interrupt Pending. This status bit is a 1 when the SurfBoard is actively asserting an interrupt on the EISA bus.

RTV – Receive Time Overflow. The 44-bit receive time counter has overflowed. To clear this condition, the receive time counter must be reset using a global reset signal from the SNI board.

LU – Latency Underflow. This indicates that a packet was received before it was sent (as indicated by the timestamps). In this case the latency for the packet will be reported as zero. To clear this condition, write a 1 in this bit position to this register.

LV – Latency Overflow. This indicates that a packet was received with a latency greater than can be represented by the 24-bit window. In this case the latency for the packet will be reported as all ones. To clear this condition, write a 1 in this bit position to this register.

PV – Pipeline Overflow. This indicates that packets have arrived too fast for the SurfBoard to process and that at least one packet has been dropped. By design, this should "never happen". It is included to facilitate testing and debugging of the hardware. To clear this condition, write a 1 in this bit position to this register.

WCV – Word Counter Overflow. This indicates that the 32 bit word counter has wrapped around zero. To clear this condition, write a 1 in this bit position to this register.

BV – Bin Overflow. A histogram bin count has wrapped around to 0. The index of the histogram bin which overflowed will be written to the FIFO. To clear this condition, write a 1 in this bit position to this register. This (sticky) bit is known as the Bin Overflow Condition (BOC) and is also used to determine the external connector output (see the TTCR register for details).

FV – FIFO Overflow. An attempt was made to write a histogram bin address (caused by a histogram bin overflow) to the FIFO when it was full. To clear this condition, write a 1 in this bit position to this register. Note that writing to the FIFO from the EISA bus via the HFDR register will not cause a FIFO overflow.

FF – FIFO Full. The FIFO is full. To clear this condition, either read at least one item from the FIFO or reset the FIFO using the HTCR register.

FAF – FIFO Almost Full. The FIFO has fewer available slots than specified by the full offset in the HFCR register. To clear this condition, either read items from the FIFO until the full offset threshold is crossed or reset the FIFO using the HTCR register.

FPF – FIFO Partially Full. The number of occupied slots in the FIFO is at least that specified by the empty offset in the HFCR register. To clear this condition, either read items from the FIFO until the empty offset threshold is crossed or reset the FIFO using the HTCR register.

FNE – FIFO Not Empty. The FIFO has at least one occupied slot. To clear this condition, either read items from the FIFO until it is empty or reset the FIFO using the HTCR register.

EXT – External Input. There has been a FALSE-to-TRUE transition on the external input. The TTCR.EIP bit controls the definition of TRUE and FALSE. Note that if the external input is at a constant level, toggling the TTCR.EIP will effectively cause a transition. To clear this condition, write a 1 in this bit position to this register.

TP – Trigger Packet. A trigger packet has arrived. (Specifically, HTCR.TEN was a 1 and a 0 to 1 transition of HTCR.TRG occurred.) The TPIR register contains the index in the trace memory of the packet. To clear this condition, write a 1 in this bit position to this register.

TC – Trace Complete. When the trace mode is set to either trigger at the beginning or to trigger at the middle, the trace is complete when the trace memory is full. When the trace mode is set to trigger at the end, the trace is complete when the trigger packet arrives. To clear this condition, first clear the underlying condition by either setting TECR to 0 for MODE 00 and 01 or clearing the HTCR.TRG for MODE 10, then write a 1 in this bit position to this register.

IEMR Interrupt Event Mask Register 0x001C

Bits	Field	Description
31..14		Unused. Reads as 0 bits. Writes are ignored.
13	RTV	Receive Time Overflow.
12	LU	Latency Underflow.
11	LV	Latency Overflow.
10	PV	Pipeline Overflow.
9	WCV	Word Counter Overflow.
8	BV	Bin Overflow.
7	FV	FIFO Overflow.
6	FF	FIFO Full.
5	FAF	FIFO Almost Full.
4	FPF	FIFO Partially Full.
3	FNE	FIFO Not Empty.
2	EXT	External Input.
1	TP	Trigger Packet.
0	TC	Trace Complete.

These bits correspond precisely to the bits in the ISCR register. When a bit is a 1 (unmasked), it allows the corresponding event to cause an interrupt. When a bit is a 0 (masked), the corresponding event will not cause an interrupt. Note that a masking an interrupt will not change the status value read from the ISCR register.

When the board is disabled by either a power-on reset or setting the ENABLE bit of the EBCR register to zero, the IEMR register is reset to all zeros. All interrupts will be masked and the previous value of the IEMR register will be lost.

HTCR Histogram / Trace Command Register 0x0020

Bits	Field	Description
31..6		Unused. Reads as 0 bits. Writes are ignored.
5	PRS	Pipeline Reset.
4	FRS	FIFO Reset.
3	TRG	Trigger Occurred.
2	WEN	Word Counter Enable.
1	HEN	Histogram Enable.
0	TEN	Trace Enable.

PRS – Pipeline Reset. When this bit is a 1, the packet pipeline state machines are held in a reset state. When this bit is set back to a 0, the next word of data from the SNI board will be interpreted as the beginning of a new packet. This reset is needed only if the last packet from the SNI board was not properly terminated with an end-of-data signal and there was no EISA bus reset (reboot) to reset the packet pipeline state machines.

FRS – FIFO Reset. When this bit is a 1, the Histogram Index FIFO is held in a reset state. When this bit is set back to a 0, the FIFO will be empty, the FIFO control registers will be set to their default values, and the control register pointer will be reset to point to the LSB of the Empty Offset. At power on, this bit is set to 1. Therefore, in order to use the FIFO, this bit must be set to a 0 in software.

TRG – Trigger Occurred. This bit is set when a trigger packet arrives. It can be set to aid in debugging and to restore the state of the SurfBoard.

WEN – Word Counter Enable. (1 = enable; 0 = disable). This bit enables the word counter. At power on, this bit is reset to 0.

HEN – Histogram Enable. (1 = enable; 0 = disable). This bit enables the histogram function. At power on, this bit is reset to 0.

TEN – Trace Enable. (1 = enable; 0 = disable). This bit enables the trace function. At power on, this bit is reset to 0.

WCDR Word Count Data Register 0x0024

Bits	Field	Description
31..0	DATA	Data

8-bit or 16-bit writes to this register are ignored.

DATA – This register contains the number of 32-bit words received.

When this counter overflows the WCV bit in the ISCR register is set. At the highest word rate (limited by the 8.3 Mwords/sec EISA bandwidth), the counter will roll over every 9 minutes.

HFCR Histogram FIFO Control Register 0x0028

Bits	Field	Description
31..8		Unused. Reads as 0 bits. Writes are ignored.
0	DATA	Data

DATA – FIFO Data.

This register accesses the FIFO offset registers. They define how full or empty the FIFO must be before it asserts the FAF (FIFO almost full) or the FPF (FIFO partially full) flags.

HFDR — Histogram FIFO Data Register — 0x002C

Bits	Field	Description
31..9		Unused. Reads as 0 bits. Writes are ignored.
8	LSB	Least Significant Byte Indicator.
7..0	DATA	Data

DATA – Each time this register is written, the value of DATA is clocked (using `CCLK`) into the `DIN` pin of the FPGA. Reading this register returns the value of the last stage in the shift register and does not move any bits.

LSB – Least Significant Byte Indicator. An 8-bit FIFO is used to store 24-bit histogram indices using 3 slots per index. This bit indicates that the current 8-bit value is the LSB of the 24-bit index. The LSB is read first.

DATA – FIFO Data. If the FIFO is empty, any data read will be undefined. If the FIFO is full, any data written will be ignored.

Note that reading or writing to the FIFO will effect the FIFO status bits in the ISCR and could cause a corresponding interrupt. Note that an 8-bit read will ignore the LSB field and that an 8-bit write will store an unknown value into the LSB field.

TICR — Trace Index Count Register — 0x0030

Bits	Field	Description
31..22		Unused. Reads as 0 bits. Writes are ignored.
21..0	DATA	Data

8-bit or 16-bit writes to this register are ignored.

DATA – This 22-bit index contains the index where the next 64-bit trace value will be written.

The actual size of the DATA field in this register can be 4, 18, 20, or 22 bits when the the setting of the 2-bit TSIZE field of the ACDR shift register is 00, 01, 10, or 11, respectively. When TSIZE specifies a register size smaller than 22 bits, the high order bits will read as zero and writes to these upper bits are ignored.

TECR — Trace Event Count Register — 0x0034

Bits	Field	Description
31..23		Unused. Reads as 0 bits. Writes are ignored.
22..0	DATA	Data

8-bit or 16-bit writes to this register are ignored.

DATA – This 23-bit counter indicates how many events have been recorded in the trace memory since starting the trace.

TPIR	Trigger Packet Index Register	0x0038

Bits	Field	Description
31..22		Unused. Reads as 0 bits. Writes are ignored.
21..0	DATA	Data

DATA – This 22-bit index indicates which 128-bit trace value corresponds to the trigger packet.

A.7 Trace Memory

The trace memory space begins at offset 0 from MBASE in the EBCR register and is 64 MBytes in size. Each packet which is traced requires 16 bytes of memory. The SurfBoard supports 32-bit wide (72 pin) SIMMs of 4 MBytes, 16 MBytes, or 64 MBytes. This allows 2^{18}, 2^{20}, or 2^{22} packets to be sampled, respectively. Each sample is 128 bits and is stored as four 32-bit words as follows:

Trace Memory Word (Offset 0)

31..26	25	24	23..0
Sender ID	V	U	Latency

V: Latency Overflow, U: Latency Underflow

Trace Memory Word (Offset 1)

31..28	27..18	17..12	11..0
Category	Size	Color	Receive Time (High)

Trace Memory Word (Offset 2)

31..0
Receive Time (Low)

Trace Memory Word (Offset 3)

31	30..29	28..4	3..0
I	P	Packet Address (bits 26..2)	B

I: Interrupt, P: Presence, B: Byte Enables

The presence bits can be used to indicate where valid trace data exists in the trace memory. These bits can be initialized by software to be non-zero; the trace hardware will always store a zero in these bits when a packet arrives. The Shrimp hardware only uses bits 26..2 of the address. This limits the locations where a packet can be stored to the low 128 MBytes of physical address space.

A.8 Histogram Memory

The histogram memory space begins at offset 0x04000000 from MBASE in the EBCR register and is 64 MBytes in size. The SurfBoard supports 32-bit wide (72 pin) SIMMs of 4 MBytes, 16 MBytes, or

64 MBytes. This allows up to 2^{20}, 2^{22}, or 2^{24} histogram bins each containing a 32-bit count. Data from incoming packets is extracted and distilled to create the bin address which is incremented. When a histogram bin increments from 0xFFFFFFFF to 0 (wraps around), the bin address is stored in the histogram address FIFO. This bin overflow can optionally trigger a trace and/or interrupt the CPU. The histogram memory must be initialized before use. The memory can be randomly accessed and any value can be preloaded into each location.

A.9 Programming Sequences

A.9.1 Initialization

After either a power-on reset or an EISA reset, the board will be disabled and will only respond to accesses on the slot specific I/O mapped registers EPIR and EBCR. The ENABLE.EBCR bit will be cleared. As a result, the IEMR register will be cleared masking all interrupts.

The EISA configuration sequence automatically assigns the interrupt level, assigns the memory address space for the registers and memory, and enables the board using the EBCR register.

The device driver must initialize the board. The device driver also determines the memory size of the trace and histogram memory.

A.9.2 Collecting Data

To begin an experiment the board is accessed as follows:

- Write a 0 to the IEMR register to mask all interrupts.
- Disable data collection using the HTCR register.
- Load the application shift register using the ACDR register.
- Set the external I/O, trace mode, and qualifier using the TTCR and ECDR registers.
- Initialize the histogram memory.
- Initialize TICR and TECR registers.
- Clear interrupt sources of interest by:
  - writing to the ISCR register
  - ensuring that the FIFO is empty by using the HIFR register.
  - ensuring that the external input line is not asserted.
  - ensuring that the word counter is not full by writing to the HTCR register.
- Set the user LED and the local category with PCLR.
- Enable interrupts as appropriate using the IEMR register.
- Enable data collection with the HTCR register.

B Software Specification

B.1 Driver Interface

Interaction with the SurfBoard is provided by a character mode device driver. The following is a list of functions available in the Surf library which control the SurfBoard through the driver. The Calling sequences and return values can be found in the header file `surflib.h`.

`surf_Open`	Open the SurfBoard device.
`surf_Open`	Open the SurfBoard device.
`surf_Close`	Close the SurfBoard device.
`surf_GetReg`	Read a specific SurfBoard register.
`surf_PutReg`	Write a specific SurfBoard register.
`surf_GetTSize`	Get the size of the trace memory in bytes.
`surf_GetTMem`	Copy a block of trace memory to local memory.
`surf_PutTMem`	Copy a block of local memory to trace memory.
`surf_GetHSize`	Get the size of the histogram memory in bytes.
`surf_GetHMem`	Copy a block of histogram memory to local memory.
`surf_PutHMem`	Copy a block of local memory to histogram memory.
`surf_ConfigHAS`	Load a bitstream into the Histogram Address Selector FPGA.
`surf_WaitException`	Waits until an exception (interrupt) occurs or until a timeout has occurred.
`surf_ClearException`	Clears the exception state in the driver.

B.2 Utility Programs

B.2.1 User Interface to the SurfBoard

The `surf` utility provides a user interface to the SurfBoard. In addition to modifying register values and loading the histogram address configuration FPGA, it can transfer data between the histogram and trace memory and files.

Here is an example of a short session using the `help`, `show`, and `quit` commands:

```
node1:~$ surf
> help
Available Commands:
   QUIT                           -  Quit Program
   EXIT                           -  Quit Program
   HELP                           -  Show this message
   SHOW                           -  Display Registers
   DO    <filename>               -  Execute Commands in file
   CLEAR <memtype>                -  Clear Memory
   FPGA  <filename>               -  Load FPGA configuration
   LOAD  <memtype>   <filename>   -  Load Memory from file
   SAVE  <memtype>   <filename>   -  Save Memory to file
   <register> = <value>           -  Modify Register

Where:
   <memtype>   - TRACE, HISTOGRAM, or FIFO
   <register>  - A register or register.field
   <value>     - An unsigned integer.  Hex by default.
                 End with decimal point to specify decimal.
> show
FCCR  00000EE8   TSPD E   HSPD E   DONE 1   ERR 0   CFG 0   PROG 0
```

```
ACDR  02006080   SENDID 1  CATMUX 0  LAT_S 00  LAT_N C  TSIZE 2
TTCR  00000000   EIP  0   EOP  0   MEOD 0   MEI 0   MBV 0   MODE 0   QUAL 0
ECDR  00000000   DATA 0
PCLR  00000000   LED  0   CAT  0
ISCR  00000002                                                  TP
IEMR  00000000
HTCR  00000000   PRS  0  FRS  0  TRG 0  WEN 0  HEN 0  TEN 0
HFCR  --------   EMPTY 007  FULL 007
TPIR  00000000
TICR  00000207
TECR  00000207   (      519.)
WCDR  0000020D   (      525.)
> quit
node1:~$
```

B.2.2 Trace and Histogram Data Processing Utilities

This section gives brief descriptions of utility programs which can process trace or histogram format files produced by the `surf` program (previous section). Because the SurfBoard can quickly generate large amounts of data, we needed to develop several utility programs which could manipulate the data as a preprocessing step to plotting.

A histogram file is composed of lines containing two fields separated by a space: a 6-digit hexadecimal bin address and an 8-digit hexadecimal count. A trace file is composed of lines containing five fields separated by spaces: a 6-digit hexadecimal index followed by four 8-digit hexadecimal values which are offsets 0, 1, 2, and 3 of the 128-bit trace value (see Section A.7). In addition, in either format file, a pound sign (`#`) begins a comment which extends to the end of a line. Blank lines are allowed and are ignored.

A common format used by spreadsheets and plotting programs, is a comma separated value (CSV) file. Several of these utilities process CSV files.

`hist3dcsv`	Produce a multidimensional histogram in CSV format from a histogram file.
`histfold`	Fold a (multidimensional) histogram file by combining the bin counts of bins whose addresses are identical when masked with a given value.
`traceCheck`	Analyze a trace file and gives a brief report. The report includes a record count, a count of instances when the receive time decreases, count of latency flags, histogram by sender ID.
`traceFilter`	Filter a trace file. Trace entries can be filtered on ranges of sender ID, latency, size, and address page.
`makeHist`	Convert a trace file to a histogram file. (Uses a histogram mapping of the 10-bit packet size concatenated with the low 12 bits of latency.)
`sizeHist`	Produce a packet size histogram in CSV format from a trace file.
`latHist`	Produce a latency histogram in CSV format from a trace file.
`pageHist`	Produce a page address histogram in CSV format from a trace file.
`reduceXY`	Reduce a comma separated file (with two fields) as a preprocessing step for graphing. The command line options include the X and Y ranges, the physical size of the final graph, and the resolution of the output device. Redundant points which would plot to the same point are removed. There is a histogram mode which combines the counts from bins which would overlap in the final graph.
`rxtVlat`	Read a trace file from standard input and produces a comma separated receive time vs. latency data file. Input lines are truncated to 80 characters.
`scale`	Scale fields in CSV value file.