2 Traffic Measurements

The Ethernet local-area network is a broadcast medium: every station must be able to monitor all communication in order to select frames destined to its own address. This makes it easy to obtain traffic measurements of single-wire Ethernets without disruption of the measured system: a single (powerful) machine can be instructed to listen to all messages without modifying or altering their arrival times. This technique was first used by Shoch and Hupp [80] and later reused numerous times [7, 15, 88]. The same passive listening technique can be easily adapted to ring networks [24], and, with a little more work, to star networks such as a single-node Datakit [54].

This is also the method we used in a previous study of the traffic of an Ethernet in the Computer Science Division of the University of California, Berkeley, in which we focused on the properties of the aggregate traffic produced by diskless workstations that communicate with their file servers [33]. Beside the specificity of the topic investigated, that study was notable for the high accuracy of its timing information and for the very small packet loss. A detailed account of the Berkeley study is reproduced in Appendix A.

A more recent work at Bellcore pushes the limits of clock accuracy and packet loss farther [93]. Its authors needed a measurement system that could detect broadcast storms: the sudden increase in network traffic caused by many hosts that answer a broadcast message.

For a number of reasons, which we will discuss in Section III, but primarily to look beyond diskless workstations and the small machines of the Berkeley environment, we decided to conduct a new study in an industrial environment with more powerful machines at Sun Microsystems, where we took traces of packet protocol information. The structure of the new measurement system is similar to the one we devised for the Berkeley study: a dedicated UNIX machine with a customized Ethernet interface driver collected packet headers and packet arrival information that were stored on disk and later moved to tapes for offline analysis. For the Sun study, we used the fastest machine available at Sun and a hardware clock with a resolution of 1 microsecond. Mainly because of bus contention, the accuracy of the timing information in the traces is lower than the resolution of the clock: the maximum errors are estimated to be smaller than 100 microseconds. This accuracy is more than sufficient for the estimation of statistical quantities and the modeling effort we intend to carry out in this study.

This chapter begins with a brief description of the main results of our Berkeley measurements, which we frequently refer to in our analyses of the data that is the topic of this study. We then describe the new measurement software, concentrating on the hardware high-resolution clock and our modification to the Ethernet driver. We then explain the formats of the packet traces; since we had to compress the traces to be able to store on-line most of the recorded information for ease of processing, we also describe the compressed record formats. We next analyze in depth the timing errors caused by the variations in the access time to the bus shared by the CPU and the Ethernet I/O interface. Finally, we describe the aggregate traffic on the Sun Ethernet in terms of network load, packet length distribution, and packet interarrival times.

II. Berkeley Measurements

We had previously taken measurements of packet traffic on an Ethernet in the Computer Science Division of the University of California, Berkeley, in which we studied the aggregate traffic produced by more than 100 machines. Although the network environment, composed of Sun, Xerox, and VAX computers, was heterogeneous, most network traffic was generated by diskless Sun machines. For this reason, we focused on those workstations and their software. We analyzed and discussed network load, packet length distribution, source-destination traffic patterns, and interarrival time distributions, and focused on three protocol families: the Transmission Control Protocol (TCP), the Sun Network Disk protocol (ND), and the Network File System protocol (NFS) [33].

The most striking conclusion of the Berkeley study was that diskless workstations may indeed generate enough traffic to load substantially a local-area network of the current generation. The mean network utilization was low (averaging 6.5 percent over 24 hours), but bursts generated short-term peak utilization that exceeded 33 percent of the Ethernet raw bandwidth.

Our measurements showed that a workstation's traffic can be subdivided in three broad categories: character traffic from the workstation to other machines; paging traffic generated by the workstation's virtual memory to a remote paging device; and file access traffic to remote file servers. A workstation's behavior will depend on the characteristics of each of these three types of traffic. These components were easily identifiable because a different protocol was employed for each component. Character traffic generates many small packets but no substantial network utilization. File access to a remote file server generates bursts of traffic lasting several seconds, which may demand bandwidths of roughly 120 Kbytes per second, or about 10 percent of the Ethernet raw bandwidth. Paging traffic, which accounted for maximum network utilization levels of 20 to 25 percent over 1-second intervals between a single client workstation and a file server, has been greatly increased by small physical memories (by today's standards a 4-Mbyte memory, common in the Berkeley network, is small indeed) and, possibly, by sub-optimal performance of the virtual-memory algorithms. Increasing memory sizes may decrease the level of paging traffic; however, since there is a tendency for applications to grow as more and more memory becomes available, it is likely that paging traffic will remain a noticeable component in future diskless workstation traffic. Paging traffic could be reduced, however, by improved buffering schemes; one such scheme, which calls for a global caching area for both virtual memory and the file system, has been implemented in Mach [94] and in release 4.0 of Sun UNIX.

We also observed that the interarrival time distribution of the aggregate traffic differs substantially from that of a Poisson process. We suggested that this was probably attributable to correlations between arrivals, but did not analyze the possible correlations.

III. Sun Measurements

The Berkeley study provided us with certain insights into the nature of network traffic and experience in traffic measurements. For studying the problem of the variability of network traffic, we felt that new measurements on a different network—we chose a network at the Sun Microsystems headquarters in Mountain View, California—needed to be done. The Sun network presented some important hardware and software advances. First, the machines in the Berkeley network (mostly Sun 3/50s with 4 Mbytes of memory) were being superseded by more powerful models with larger physical memories, leading to a smaller proportion of paging traffic and a lower level of network traffic altogether. Second, the recently released SunOS 4.0 had changed the virtual memory algorithms, managing to provide private root file system partitions to client workstations without employing the ND protocol, a low-level protocol tailored to work with the Ethernet local-area network, which in the Berkeley measurements accounted for 52 percent of the traffic. Third, following the work of V. Jacobson [40], Sun had improved the performance of NFS, increasing its throughput. We anticipated that, in a SunOS 4.0 environment, the NFS protocol would be more pervasive, accounting for a larger proportion of the total traffic than the 16 percent we measured in the Berkeley network. In addition, we felt that it would be worth analyzing the effects of some features of the Sun network that were not present in the Berkeley network. For instance, at Sun Microsystems many machines have local disks, which makes those workstations independent of the network for running UNIX and its virtual memory. Moreover, the Sun computing environment is more integrated than the Berkeley environment: machines (with disks and diskless) mount file systems from a variety of file servers and from other disk-based workstations. Finally, although the clocks used in the Berkeley and Sun measurements have similar resolution (1 and 0.5 microseconds), the accuracy of the Sun clock is, as discussed below, better by at least one order of magnitude. Thus, a second study in this new environment would provide more general results and conclusions more readily adaptable to other types of network environments.

The new measurement system was similar in structure to the old one, consisting of a powerful stand-alone machine, a Sun-4/280, whose kernel had been modified to trace all packets and that had been equipped with a hardware counter that was physically assembled on a small board plugged into the machine's socket normally reserved for an optional encryption chip. The device used was an Am9513 counter chip, driven by a 2 MHz crystal, and was programmed to increment one of its five 16-bit registers every 0.5 microseconds. (We did not use the remaining four registers.) Register values thus ranged from 0 to 65535 cyclically and were used in combination with the kernel time to produce a high-resolution clock. The data path to the counter chip was 8-bit wide, making the reading of a register value to occur in two steps, and was on a bus shared by the CPU, the memory, and the Ethernet interface.

We added a routine, `microtime`, to the instrumented kernel to produce a timestamp when called according to the following C programming language type definition:

```
typedef struct {
     struct timeval ts_tv;
     unsigned short ts_ticks;
} Timestamp;
```

The data structure `timeval`, used to represent timing information in Berkeley UNIX 4.2BSD and Sun UNIX, provides separate variables for the seconds and for the microseconds:

```
struct timeval {
     long tv_sec;
     long tv_usec;
};
```

The UNIX kernel variable `time`, which stores a value that approximates the wall-clock time, is of type `timeval` and is incremented by SunOS on a Sun-4 every 10 milliseconds. The routine `microtime`, called with a pointer of type `Timestamp` as a single argument, simply recorded the UNIX time along with the current value of the counter register from the Am9513 chip in the object addressed by the pointer argument.

The routine that read the counter chip actually had to cross the bus several times in order to obtain a reading. First, the counter value was saved in a chip's auxiliary register so that slower reading cycles would not affect its value. This required writing three 8-bit words in sequence, each in a separate bus cycle, to the chip's control and data registers. Second, the chip had to be told about which auxiliary register to make available to its data I/O port. This again required writing three separate words to the control and data registers. Finally, as mentioned above, two reading cycles were necessary to obtain a 16-bit counter register value through the 8-bit data path, bringing the total number of bus accesses to eight for each timestamp.

Derivation of packet interarrival times from the timestamps first required some processing. This was done by calculating separately the differences between two consecutive packets' UNIX times and between their counter-register values. (The differences in the counter values are taken modulo 2^{16}, by which, if x_1 and x_2 are two such values, we mean that $D = x_1 - x_2$ if $x_2 \geq x_1$ or $D = 2^{16} + x_1 - x_2$ if $x_2 < x_1$.) Then, we can use these differences to evaluate R, the number of times the counter has wrapped around. Table 2.1 shows how this is performed. The interarrival time in microseconds is then $D/2 + 2^{15}R$.

This relatively complicated timing system was chosen, over a simpler one in which the counter chip would have maintained by itself a full microsecond-accurate clock by chaining three of its 16-bit counters together, in order to reduce the time needed to read the clock and thus increase the accuracy of the measurements. Using three 16-bit counters would have required reading the counter device six times through the slow 8-bit data path to the socket into which the counter circuit was plugged; in contrast, under our system, the overhead was confined to offline processing.

We measured the time it took to obtain a timestamp using a short program (which we ran in kernel space during the testing phase of our modified kernel) containing the following code segment:

```
. . .
ptr = &hrc[0];
do {
     microtime(ptr);
} while (++ptr < &hrc[4096])
. . .
```

In this code, `hrc` and `ptr` are defined respectively as an array of size 4096 and a pointer, both of type `Timestamp`. In the 'do' loop, between calls to `microtime`, the Sun-4 C compiler generated object code with seven instructions, which the machine executed in 13 clock cycles or 0.78 microseconds. After running the code segment, we computed the differences between subsequent values contained in the elements of the array. In 97.3 percent of the cases the difference was 13 microseconds; in 2.5 percent, 14 microseconds. There were eight values of differences greater than 14 microseconds, the largest being 80 and 108 microseconds. Except for the first interarrival time of the series, which was of 21 microseconds because the code was not yet in the instruction cache, these larger intervals were caused by bus contention during accesses to the counter register. Recall that the Ethernet device with its busy DMA controller was on the same bus. Using more than one counter in the Am9513 chip would have increased the probability of bus contention delay, since more register accesses would have been necessary. Based on the measurements above, we estimate that a minimum of 12 to 13 microseconds are needed to obtain a timestamp with `microtime`.

TABLE 2.1. DIFFERENCES BETWEEN UNIX TIMES (IN MS) AND COUNTER VALUE READINGS

DIFFERENCE BETWEEN UNIX TIMES	RANGE OF DIFFERENCES BETWEEN COUNTER VALUES		NUMBER OF TIMES COUNTER HAS WRAPPED AROUND
	FROM	TO	
0	0	19999	0
10	1	39999	0
20	20001	59999	0
30	40001	14463	0, or 1 if between 0 and 14463
40	60001	34463	0, or 1 if between 0 and 34463
50	14465	54463	1
60	34465	8927	1, or 2 if between 0 and 8927
70	54465	28927	1, or 2 if between 0 and 28927
80	8929	48927	2
90	28929	3391	2, or 3 if between 0 and 3391

IV. The Ethernet Driver

We modified the Ethernet driver to extract protocol information from each Ethernet packet and to store it with a timestamp in a system's file, from where it could be dumped onto tapes. Although we cannot publish the code of our modified driver, since it is owned by Sun Microsystems, we will describe here some details of its implementation.

The Sun-4 used for the measurements had two network interfaces, one on the CPU board and one available as a card on the VME bus. We disabled the VME interface and used the on-board Ethernet controller as it accesses a bus with larger bandwidth. The on-board Ethernet controller, based on the Intel 82586 chip [37], was programmed to receive, without CPU supervision, all Ethernet frames that were transmitted. The 82586 coprocessor moves each frame it receives to one of a large number of buffers in kernel memory using DMA. It uses for temporary storage a small 16-byte first-in-first-out queue to prevent a temporarily busy bus from disrupting data reception. This organization is superior to one in which the controller buffers a frame entirely in its internal memory before moving it to the computer memory since it reduces latency in data reception. Upon being informed by an interrupt of the successful transfer, the CPU extracted the Ethernet packet header and the portion of the data containing protocol information, and time-stamped the record with the time information described above.

The driver employed a double buffering technique: it kept two 256-Kbyte buffers in kernel memory in which it stored the packet information it was collecting. When the first buffer was full, it switched to the second one, concurrently awakening a sleeping kernel process, which ran only when there was no network interrupt pending or being served, for the asynchronous transfer of the data in the first buffer to a trace file on disk. This procedure alternated between the two buffers.

When a trace file grew to 100 Mbytes—a size we had chosen because it fit in one standard tape at 6250 bpi, and was the size of the three disk partitions we had dedicated to the measurements—the kernel driver automatically switched to another file. Dumping the data onto tape presented no problem since it took only a few minutes, while filling up one file took from about 50 to 80 minutes, depending on the time of day.

An important goal of our measurements was to reduce packet loss to a minimum. There are two major causes for packet losses: overruns of the small FIFO in the controller caused by a busy bus, and lack of buffers in main memory to receive the data. We programmed the controller to initiate data transfer to the bus when the data in the FIFO queue had reached 12 bytes and allocated 250 buffers in kernel memory for the frames. The Intel Ethernet controller monitored the bus transfers and kept counts of the frames discarded due to bus overruns and a shortage of buffer space. The frames lost were typically a few units over a day, when several million frames were received. Apparently, the speed of the CPU, the bandwidth of the on-board bus to which the Ethernet controller was connected, and the memory allocated to receive frames were sufficient.

V. Packet Traces

The packet traces are composed of variable-size records, each of which is preceded by a standard header. The format of the header is shown below:

```
struct emt_record {
     unsigned long dst;
     unsigned long src;
     unsigned short type;
     unsigned short length;
     Timestamp mtime;
     unsigned short bcount;
     long sequence;
};
```

The first two fields of this structure are used to store the least significant four bytes of the six-byte Ethernet source and destination addresses. In general, these four bytes may not identify uniquely each machine. However, since the most significant three bytes of an Ethernet address identify the interface maker, they are likely to be redundant in a network of machines of the same type. We have verified, by comparing directly each machine's address against each other, that the lower four bytes of the addresses do identify our workstations uniquely.

The `type` field is also taken from the Ethernet frame and identifies the higher-level protocol in the frame's data portion, the length of which is stored in the `length` field of the record above. The fifth field, `mtime`, is used to store the timestamp; the `bcount` field, which indicates how many bytes of the original frame follow the trace header, is a pointer to the next packet's header; and the sequence number, which we included to discover how many records would be lost in case of recording errors in the secondary-level magnetic storage media, is incremented by one with every received frame.

The decision of how many bytes to save for each packet was made by scanning the type field of the Ethernet header, making assumptions about higher level protocols, and trading off space for time. For instance, if an IP packet was of type UDP, we assumed it was transporting NFS data and saved enough bytes to record the largest possible NFS header. Careful, complete protocol demultiplexing would have resulted in longer execution time during trace collection, not justifying the space saved.

VI. Data Compression

Typically, the amount of information in the traces can be as large as 100 Mbytes per hour of network traffic. The analyses and results shown in this study required reading and processing the packet traces countless times. Without keeping the information on secondary random-access storage our work would have not been possible. Given the severe space limitations of our computing environment, it was necessary to compress the traces in order to store several hours of network activity on line. The formats that we chose for the compressed data reflect the requirements of most, though not all, of our analyses: in most cases we needed only information about the source and destination addresses, the packet length, and the arrival times.

The format of the records of the compressed data is variable: there are four different formats, which are identified by the first two bits of each record. The source and destination addresses required eight bits each. Even though a single Ethernet can have more than 256 interfaces (each one with its own unique address), commonly Ethernets are class-B IP subnets, for which the last byte of the IP address is reserved for the hosts (or, more precisely, for the host interfaces). On the Sun network we monitored, all machines have (distinct) IP addresses. Packet length information (a number between 46 and 1500), required 11 bits. Finally, we encoded the timing information in the form of an interarrival time: the interval of time between a packet and the packet preceding it.

While converting the packet arrival time information, we also changed the original times, which we recorded in the traces as the times when packets were received, to the times when the transmission on the Ethernet first started. This was done by subtracting from each recorded packet arrival time the packet transmission time, which is proportional to the packet length.

The first type of format, identified by two zeros in the most significant bits (the *format type bits*), encodes the interarrival time in a total of 35 bits—32 bits in the second four bytes of the record, and three bits, indicated by A, B, and C, just after the two format type bits. This allows us to represent time differences of about nine and a half hours, more than enough since workstations are never silent for this long a time. (In this and the following pictorial representation of formats, we show 32 bits on each line.)

```
-----------------------------------------------------------------
|0|0|A|B|C|  length (11 bits)   |    source     |  destination  |
-----------------------------------------------------------------
|                        time difference                        |
-----------------------------------------------------------------
```

If a packet followed the exact same path of the packet that immediately preceded it, i.e., moved from the same source machine to the same destination, there is no need to store the source and destination fields in the record. Instead, we use a second type of format, identified by ``01'' in the format type bits, which instruct trace-analysis programs to produce a packet with the same source and destination addresses as the previous packet's:

```
-----------------------------------------------------------------
|0|1|  length (11 bits)   |      time difference (19 bits)      |
-----------------------------------------------------------------
```

If a packet moved between the same pair of machines as the preceding one, but in the opposite direction, we employ a third, ``10'', format:

```
-----------------------------------------------------------------
|1|0|  length (11 bits)   |      time difference (19 bits)      |
-----------------------------------------------------------------
```

In these two formats, we reserved for the time difference 19 bits, capable of representing interarrival times of up to about half a second. This should be sufficient for most cases, though, especially in traces of traffic of single workstations, interarrival times may be larger. When this occurs, we revert to the first format for the record, even if the address fields of the two packets are the same.

Using interarrival times helps to save considerable space, but has the disadvantage of requiring sequential reads of all the previous records in order to obtain the arrival time of a specific packet. To obviate this problem, we chose a fourth format based on the first format but expanded to include the arrival time, expressed in the standard Berkeley UNIX 4.2BSD convention in which the time is represented in seconds and microseconds since midnight, January 1, 1970. (The x's indicate unused bits.)

```
-----------------------------------------------------------------
|1|1|A|B|C|  length (11 bits)   |    source     |  destination  |
-----------------------------------------------------------------
|                        time difference                        |
-----------------------------------------------------------------
|                            seconds                            |
-----------------------------------------------------------------
|x|x|x|x|x|x|x|x|x|x|x|x|          microseconds (20 bits)       |
-----------------------------------------------------------------
```

A record with this format was inserted at the beginning of each compressed trace file (we kept on line compressed versions of the original traces divided into files representing approximately one hour each). To determine the arrival time of a given packet, we only need to refer back to the first record of the file in which the packet's record is located, and not the the beginning of the trace. In addition, we used this format to begin each trace containing a subset of the global traffic that we extracted from the aggregate traces.

It is important to notice that we use this type of data encoding for both the full traces that we have collected on the Sun network and the traffic of individual workstations (or any subset of the aggregate traffic) that we extract from the general traces for additional, more specialized processing (such as estimating the interarrival time distribution of a particular machine). Thus, we only needed to design a single set of programs to read and manipulate all the various possible traces.

With these data-compression techniques, we used only about one third of the storage space that would have been needed if we had represented each packet with time, source, destination, and length fields as in the original traces. With respect to the original traces, we need only one sixteenth of the storage space, though, of course, we lose information since in the compressed traces we do not represent protocol data.

VII. Timing Errors

Timestamping may not always accurately record the arrival time of a packet. Although the counter-based clock has high resolution, latencies in interrupt response time and lags in accessing the bus can lead to delays in the recording of timestamps. For instance, if a packet (the receipt of which triggers the posting of an interrupt to the CPU) is followed by the smallest Ethernet packet (64 bytes), and the two packets are separated by the minimum interpacket time permitted (9.6 microseconds), the CPU has only 67.2 microseconds to process the first packet before the next interrupt occurs (51.2 microseconds for the 46 bytes of data and the 18 bytes of header and checksum, 9.6 microseconds for the interframe time, and 6.4 microseconds for the Ethernet preamble, which always precedes the first bit of a frame and is used to synchronize the clock of the receiver with the clock of the transmitter). Since 13 microseconds are necessary to record a timestamp, should the CPU take more than 54.2 microseconds to process the first packet, the second packet would be timestamped later than it should be. (This is so because UNIX serves only one network interrupt at a time.) Notice also that in this situation, since the timestamp of the second arrival time is delayed, the interarrival time between the second packet and the third packet may be shorter than the real interarrival time. Because the driver only needs to perform few operations per received frame, this hypothetical situation should occur infrequently.

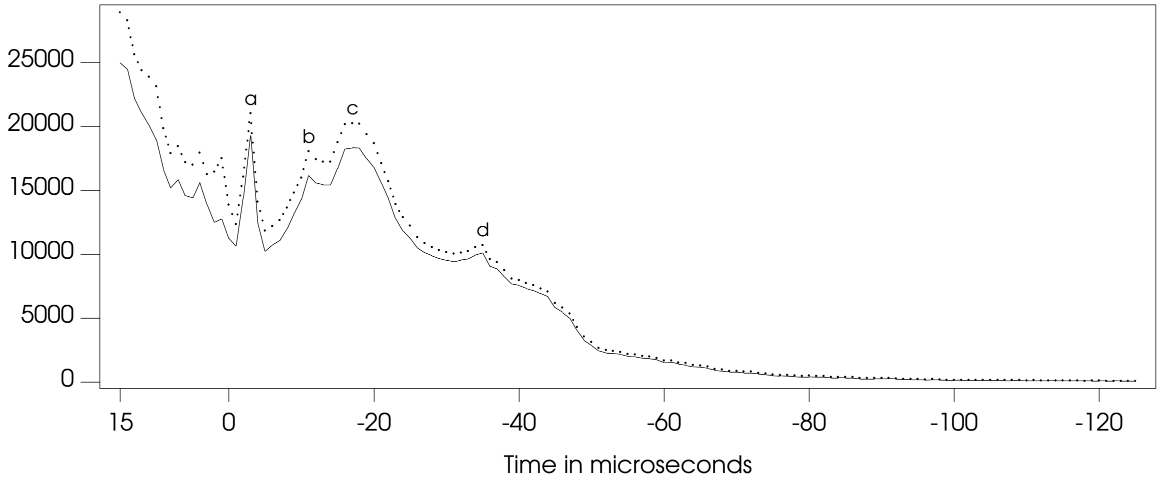
During testing, we found the timestamps of some packets to be such that the transmission of each of these packets should have begun before the arrival of the previous packet—an impossible situation. If we define the *interpacket time* to be the time between the first bit of a packet and the last bit of the previous packet, this situation is characterized by negative interpacket times. Clearly, negative interpacket times must have been artificially created by measurement errors. In this section we will address two issues: bounding the size of the errors and determining the mechanisms that delay the timestamps.

To determine the size of the measurement errors, one needs to compare real values with measured values. Although no actual values are available, we can reasonably hypothesize what the actual interpacket times were for a class of packets pairs that generated errors. Specifically, these are packet pairs with unrelated source and destination addresses. Assuming that communications between unrelated pairs of machines are statistically independent—an assumption that we will discuss in detail in the next chapter—it is intuitive that, given any marginal probability distribution of the interarrival times of packets between two stations, the probability that a ready-to-transmit station will find the network busy is directly (though perhaps not linearly) related to the length of the packet in transmission. The majority of negative interpacket time errors were caused by pairs of packets in which the first packet had the maximum size of 1500 data bytes. Thus, it is reasonable to assume that each second packet was queued up, waiting for the large-sized packet in front of it to reach its destination. Since an Ethernet interface ready to transmit samples the medium continuously and attempts transmission only after the wire has been idle for at least 9.6 microseconds, a behavior that has been termed 1-persistent [48], a packet that is queued at one interface should generate, in the absence of collisions, an interpacket time of 16 microseconds (9.6 microseconds of the minimum interframe spacing plus 6.4 microseconds necessary to transmit the frame's preamble). Thus, interpacket times of less than 16 microseconds for two packets in our traces with unrelated source and destination addresses should in fact be exactly 16 microseconds. (Some authors define two packets with minimum interpacket time as *back-to-back* packets.)

In Figure 2.1 we show, represented by a solid line, the number of interpacket times less than 16 microseconds including negative times, for these pairs of unrelated packets belonging to a 10-hour trace, which we analyze in the remainder of this chapter and use in the rest of the thesis. About 9.8 percent of all packets in our traces—a sizable number—were affected by this type of errors. We also show, as a dotted curve, the total number of errors for all packets, including those generated by pairs of packets between the same two of machines. This curve follows the shape of the unrelated-packet curve, suggesting that packets between the same two machines may also be caused by measurement errors involving back-to-back packets. In fact, in SunOS 4.0, back-to-back packets can occur between two machines: as reported in [40], improvements in the Ethernet throughput were achieved mainly by reducing the interpacket gap during packet fragmentation.

We observe that the large majority of errors in Figure 2.1 is between 15 and −85 microseconds (some 99.2 percent of the timing errors are within this interval). In the discussion that follows, we will also show that there some back-to-back packet pairs for which the measured interpacket times are larger than 16 microseconds. These too are caused by measurement errors; however, they are many fewer than the errors in Figure 2.1 and do not extend towards positive times as much as those extend towards negative times. We can therefore conclude that in our traces with probability 0.99 the absolute value of the measurement errors is smaller than 100 microseconds. In addition, in building the compressed traces, which we have used for all timing analyses, we have adjusted the measured arrival times of all the packet involved in Figure 2.1 so that their interpacket times become 16 microseconds. In doing so, in all probability we have reduced the

FIGURE 2.1. ERRORS IN THE TIMING MEASUREMENTS



value of the average error. Based on these estimates, we can state that the accuracy of these measurements is fully adequate for all analyses we perform in the following chapters.

The conclusion above is predicated upon the assumption that first packets are long; as Figure 2.2 shows, 72 percent in fact are longer than 500 bytes. In Figure 2.2 we plot the packet length distribution of the first packets in the pairs that generated the interpacket time errors; in Figure 2.3, the distribution of the length of the second packets of those pairs. (These data are from the 10-hour trace.) Notice that the median of the distribution of the first packets is 1500 bytes. This, as observed above, represents a case of length biasing: since it is more likely that the minimum interpacket time occurs after a long packet, a timing error of this type is more probable after a long packet.

FIGURE 2.2. PERCENTAGE OF PACKETS VS. PACKET LENGTH
FIRST PACKETS IN PAIRS

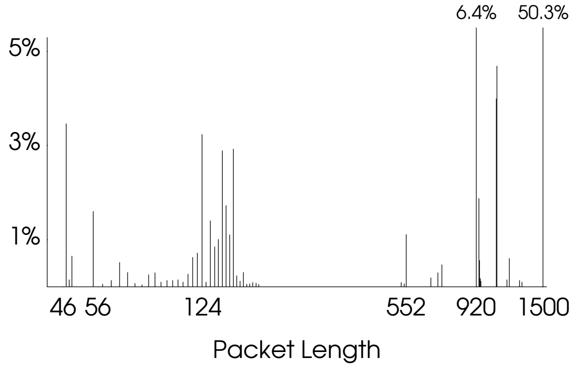
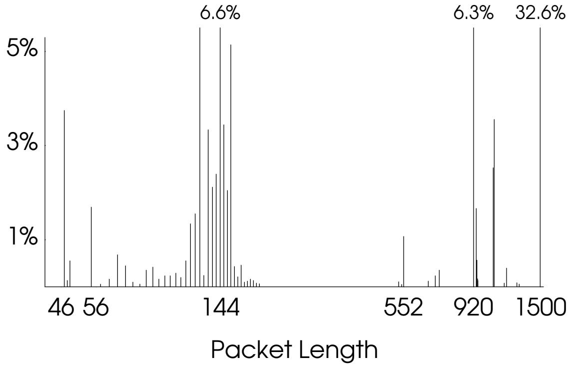


FIGURE 2.3. PERCENTAGE OF PACKETS VS. PACKET LENGTH
SECOND PACKETS IN PAIRS



These two figures should be compared to Figure 2.15, which shows the packet length distribution of all the packets in the 10-hour trace. Unlike Figure 2.2, Figure 2.3 is quite similar to Figure 2.15, suggesting that the lengths of the second packets may be independent of the lengths of the preceding interpacket time. In order to explore the hypothesis of independence, we computed the distribution of the lengths of consecutive interpacket-time errors. Errors that occur in sequence one after another generate dependencies in the packet lengths. To see this, consider, for instance, two errors in a row. Three consecutive packets in the aggregate traces are involved, the second and the third of which will both be considered among those in Figure 2.3. Since the lengths of packets exchanged between two machines are autocorrelated, as we will see in Chapter 5, and since there is a high probability that subsequent packets in the aggregate trace are between the same stations, there may be dependencies between two subsequent timing errors and the packet lengths of the associated packets.

We found that 84.9 percent of the errors are single, that is, the interpacket times that precede them and those that follow them are not in error. In 13.2 percent of the cases there are two errors in a row, in 1.6 percent three errors, and in 0.2 percent four errors. Since only a fraction of the 15.1 percent of two or more consecutive errors are caused by packets exchanged between the same pairs of machines (the probability we mentioned in the previous paragraph is less than 1), the global effect of consecutive errors on Figure 2.3 is negligible. Thus, we conclude that the packets of Figure 2.3 represent a random sampling of all the packets in the traces.

Next, we would like to identify and understand the mechanisms by which these short interpacket times were generated. The presence of peaks in Figure 2.1, some of which we have labeled, indicates that errors do not occur randomly, but are produced by some recurring conditions. We noticed that packet pairs in which the second packet was 46 data bytes generated errors of value –3 microseconds (i.e., peak *a*) very frequently, suggesting that perhaps the length of the second packet was a primary factor responsible for the peaks.

To determine whether or not errors were related to the length of the second packet in a packet pair, we plotted the number of errors occurring for various second packet lengths. Figures 2.4 and 2.5 show the results for two of the most common, shortest second packet lengths, 46 and 56 data bytes. (These two data lengths were chosen because they occur frequently, and hence allow the error estimation to be statistically meaningful.) These graphs prove that in Figure 2.1 the peak at –3 microseconds, labeled *a*, is caused by the errors associated with 46-byte second packets and that the glitch at –11 microseconds, labeled *b*, is instead caused by 56-byte packets.

We will explain the shapes of Figures 2.4 and 2.5 with the help of the timing diagram of Figure 2.6. In this diagram, which illustrates the transmission of a 46-byte frame (and a 56-byte one) separated from the previous frame by the minimum interframe time, we show a possible scenario for the situation that leads to the errors in Figures 2.4 and 2.5. The unit of the time axis is one microsecond, and the tick marks are spaced two microseconds apart. The time origin is arbitrarily set at the time when the last bit of the first frame (of the two frames that will generate the interpacket time error) arrives on the network. Each event is represented by a labeled arrow and a time value. Events above the time axis are Ethernet events; those below the axis are events—either measurements, or quantities derived from measurements, or machine events—that occur in the measurement system.

FIGURE 2.4. ERRORS IN THE TIMING MEASUREMENTS
SECOND PACKETS OF 46 Data Bytes

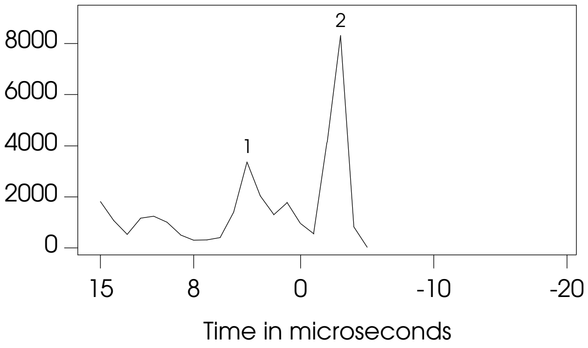


FIGURE 2.5. ERRORS IN THE TIMING MEASUREMENTS
SECOND PACKETS OF 56 DATA BYTES

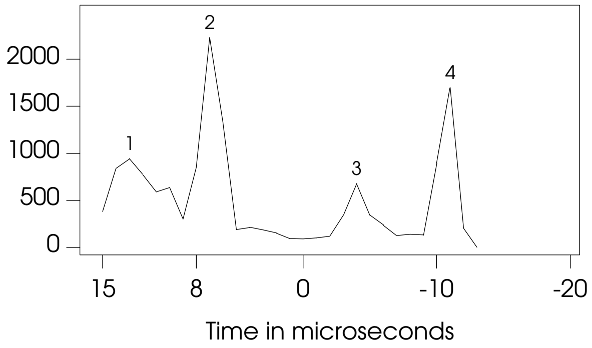
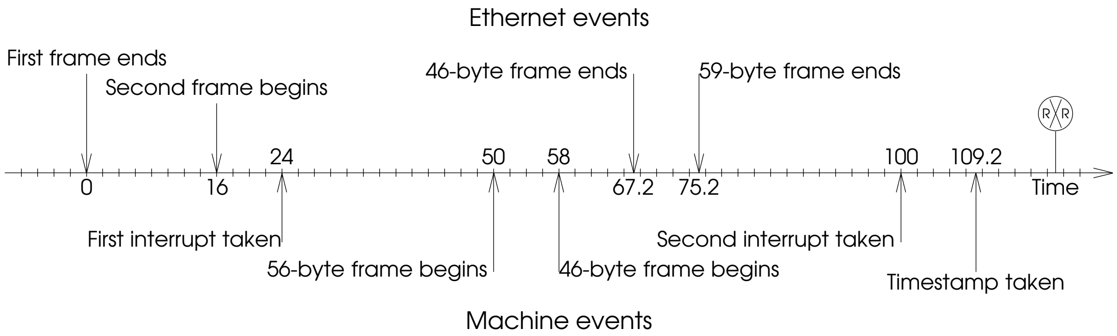


FIGURE 2.6. TIMING DIAGRAM OF INTERPACKET TIME ERRORS
SECOND PACKETS OF 46 AND 56 BYTES



The first frame is fully received at 0 microseconds, and, assuming that the CPU is idle—a reasonable assumption since most of the first packets are 1500 bytes implying that the previous interrupt occurred at a time smaller than −1230 microseconds in our reference system—the corresponding interrupt is signaled shortly thereafter. We indicated that the interrupt occurs at 24 microseconds because of the interrupt latency time—the several dozens of instructions that UNIX executes in order to switch context to the interrupt-handling routine and then to the device-driver entry point—and because the Ethernet controller chip has to update various data structures and pointers in the computer's memory, which is done through DMA, before posting the interrupt. (Most likely this is an optimistic estimate; however, whether or not the interrupt actually occurs a little later is not important for our purpose of deriving qualitative conclusions from these ballpark estimates.) The first bit of the second Ethernet frame is transmitted at 16 microseconds, after the minimum interframe spacing of 9.6 microseconds and the transmission of the 6.4-microsecond Ethernet preamble. Therefore, the timestamp routine, called soon after the interrupt handler gives control to the Ethernet driver, is executing while the second frame is being received by the Ethernet interface device.

To timestamp the first packet requires, as observed above, crossing the bus, which is now busy moving the data of the second frame from the interface device to the buffers in main memory. Although the counter value is copied to an auxiliary register early in the timestamping process, three bus accesses are required before this can happen, increasing the value of the timestamp. Moreover, all memory accesses for the Ethernet driver code also require bus cycles, causing additional delay before the timestamp can be taken. Although we do not have measurements of bus utilization during our Ethernet packet tracing and cannot provide direct data to assess the value of these timestamp delays, we can cite measurements on a Sun-3/50 [92] that showed that the Ethernet-interface DMA engine loaded the bus up to 70 percent during the transfer of Ethernet frames into memory. The result of this study supports our interpretation of the delays.

If the frame's data length is 46 bytes the last bit of the frame is received at 67.2 microseconds; if the length is 56 bytes, the last bit is received at 75.2 microseconds. We will assume that the Ethernet will become idle after receiving the second frame; again, this is a reasonable assumption, as we have seen that the large majority of errors are single. Under this assumption, the timestamps of the second packets are not delayed.

Now assume that servicing the interrupt of the first packet is completed by the time the 46-byte frame is received. Then, all things being equal, the interrupts associated with a 46-byte frame and with a 56-byte one will be separated by an interval of time equal to the interval between the two frames: 8 microseconds. Similarly, between the two associated timestamps there will be 8 microseconds. Finally, it is plain that the starting times of the two frames, reconstructed from the timestamps, will coincide. But if this were true, since the 0's on each x-axis in Figures 2.4 and 2.5 coincide with the beginning of each frame in Figure 2.6, Figures 2.4 and 2.5 would have to show peaks at the same times, which is not the case. Thus, we conclude that the servicing of the first frame's interrupt extends beyond the time when the 56-byte frame is fully received and, hence, that the interrupts generated by both 46-byte and 56-byte frames must occur at the same time.

We have illustrated this situation graphically in Figure 2.6, which also shows the reconstructed times for the beginning of the transmission of the two frames at 50 and 58 microseconds. Consider peak 1, at 4 microseconds before the beginning of the second frame of 46-bytes in Figure 2.4. In Figure 2.6, this point would be at 54 microseconds, that is 4 microseconds after the beginning of the 56-byte frame. Thus, we recognize that peak 1 in Figure 2.4 is associated with peak 3 in Figure 2.5. The association is caused by the same amount of delay in the timestamp of the first frame. Through a similar line of reasoning, we can show that peak 2 in Figure 2.4 and peak 4 in Figure 2.5 are also associated. This gives us evidence that the timestamping of the first frame is delayed by discrete, fixed intervals depending on how many times the three bus accesses are delayed during bus arbitration.

Peaks 1 and 2 in Figure 2.5 occur when the timestamps of the first frames are delayed by smaller amounts. In these circumstances, the interpacket times of 46-byte frames are larger than the 16-microsecond minimum and, though not shown in Figure 2.4, still must be considered errors. We have verified that these errors are present but, at least in this particular case, they are fewer. However, in general, the error intervals of figures such as Figure 2.4 and 2.5 need to be extended to include valid-interpacket-time errors.

Figures 2.7 and 2.8, analogous to Figure 2.4 and 2.5, refer to errors generated by packet pairs in which the second packets are 124 and 144 bytes, respectively. We can see that the peak labeled 2 in the 124-byte figure produces a small bump labeled *d* at –35 microseconds in Figure 2.1.

FIGURE 2.7. ERRORS IN THE TIMING MEASUREMENTS SECOND PACKETS OF 124 Data Bytes

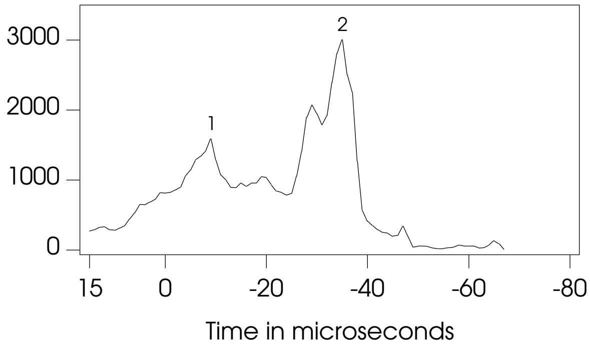
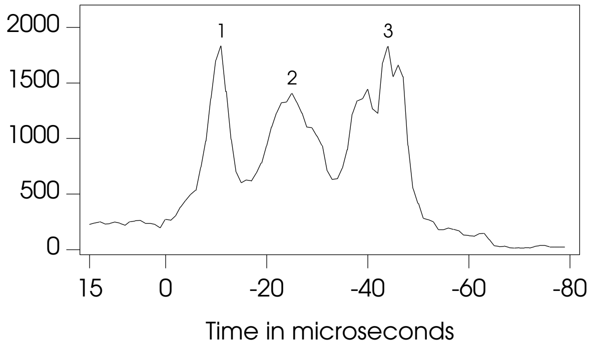


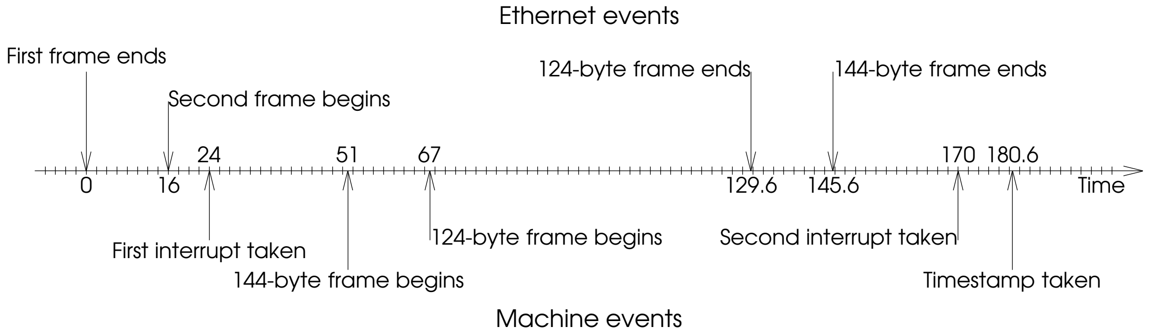
FIGURE 2.8. ERRORS IN THE TIMING MEASUREMENTS SECOND PACKETS OF 144 DATA BYTES



Here, it is more difficult to interpret the peaks. In Figure 2.9, in which we show the timing diagram of the transmission of a 124-byte and a 144-byte packet, we have again indicated that the timestamps for the two packets occur at the same time. These are NFS packets of which we saved 124 bytes in the traces (a number of bytes that is guaranteed to contain the longest NFS header). It is important to observe that since the SPARC is a RISC architecture and does not have memory-to-memory instructions, the data from the network buffer must first be moved to a CPU register and then from the register to the trace buffer, crossing the bus twice in short sequence. Thus, for these packets the interrupt servicing time is much longer than for the short 46 or 56-byte TCP packets. Because of these facts, the beginnings of transmission of the two packets are 16 microseconds apart, which can be recognized as the difference between peak 1 in Figure 2.7 and peak 2 in Figure 2.8. However, this does not appear to be the case if we compare peak 2 in Figure 2.7 at –35 microseconds after the beginning of the 124-byte packet, and peak 3 in Figure 2.8 at –44 microseconds after the beginning of the 144-byte packet. These two peaks are only 9 microseconds apart. The double-pointed peaks in both figures suggest that in some instances the interrupts for the 124-byte packets and for the 144-byte packets occur after equally long delays, but in other cases after delays of different lengths.

Figures 2.10 and 2.11 show the errors associated with pairs of packets in which the second packets are respectively 920 and 1500 bytes. The two graphs are virtually identical, indicating that the time when interrupts are received depends on packet lengths. They are also very similar to graphs (not shown) for packet lengths of 1064 and 1072 bytes. The mechanisms generating the errors are the same as those we have discussed previously. Here, again, the timestamp associated with the first arrival is delayed, while the second timestamp, taken when the Ethernet and consequently the CPU bus are idle, is not. Whereas in the case of short packets one could argue that the delays are bounded by the packet transmission times, since after the completion of a packet's

FIGURE 2.9. TIMING DIAGRAM OF INTERPACKET TIME ERRORS
SECOND PACKETS OF 124 AND 144 BYTES



transmission the bus becomes idle, allowing the timestamp process to proceed without additional delay, in the case of very long packets arbitrarily long delays could occur. But this is not the case, since the great majority of the delays in both figures are smaller than 40 microseconds, indicating that the maximum amount of delay is bounded. Our model of delays attributable to bus contention capture this phenomenon: bus contention is resolved in a finite time since bus arbitration does not allow starvation. Thus, the timestamp of the first packet when the second packet is long occurs within a short time. The long tails may be due to other factors such as the effects of collisions, which increase interpacket times by some 50 microseconds during which the channel is jammed. The initial decreasing slope of the curves indicates that the second timestamp may also be delayed. The explanation is simple: since the second packet is long, there is a high probability that a third packet becomes queued during its transmission.

FIGURE 2.10. ERRORS IN THE TIMING MEASUREMENTS
SECOND PACKETS OF 920 Data Bytes

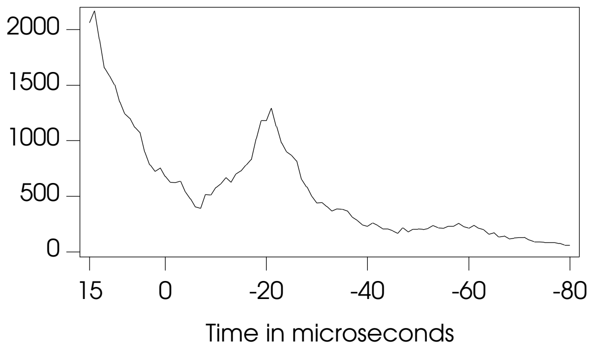
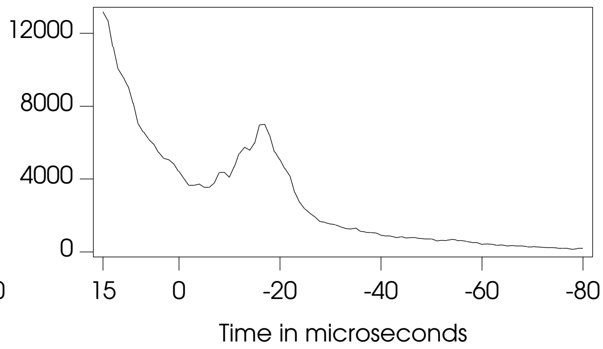


FIGURE 2.11. ERRORS IN THE TIMING MEASUREMENTS
SECOND PACKETS OF 1500 DATA BYTES



Many questions are still not fully resolved. For instance, without additional measurements, it is difficult to assess the precision of the numbers we have used in our argumentation about the timing diagrams of Figures 2.6 and 2.9. Nevertheless, we think we have presented evidence of the types of mechanisms that are behind the interpacket time errors. In summary, we found that the errors are caused by readings of counter values delayed on the single bus used in the CPU board to access both the counter chip and the Ethernet interface device. The magnitude of the errors depends on how many of the counter accesses necessary to obtain a timestamp are delayed. We estimated that, with probability 0.99, the errors of our clock–counter combination are, in absolute value, smaller than 100 microseconds.

VIII. The Sun Engineering Network

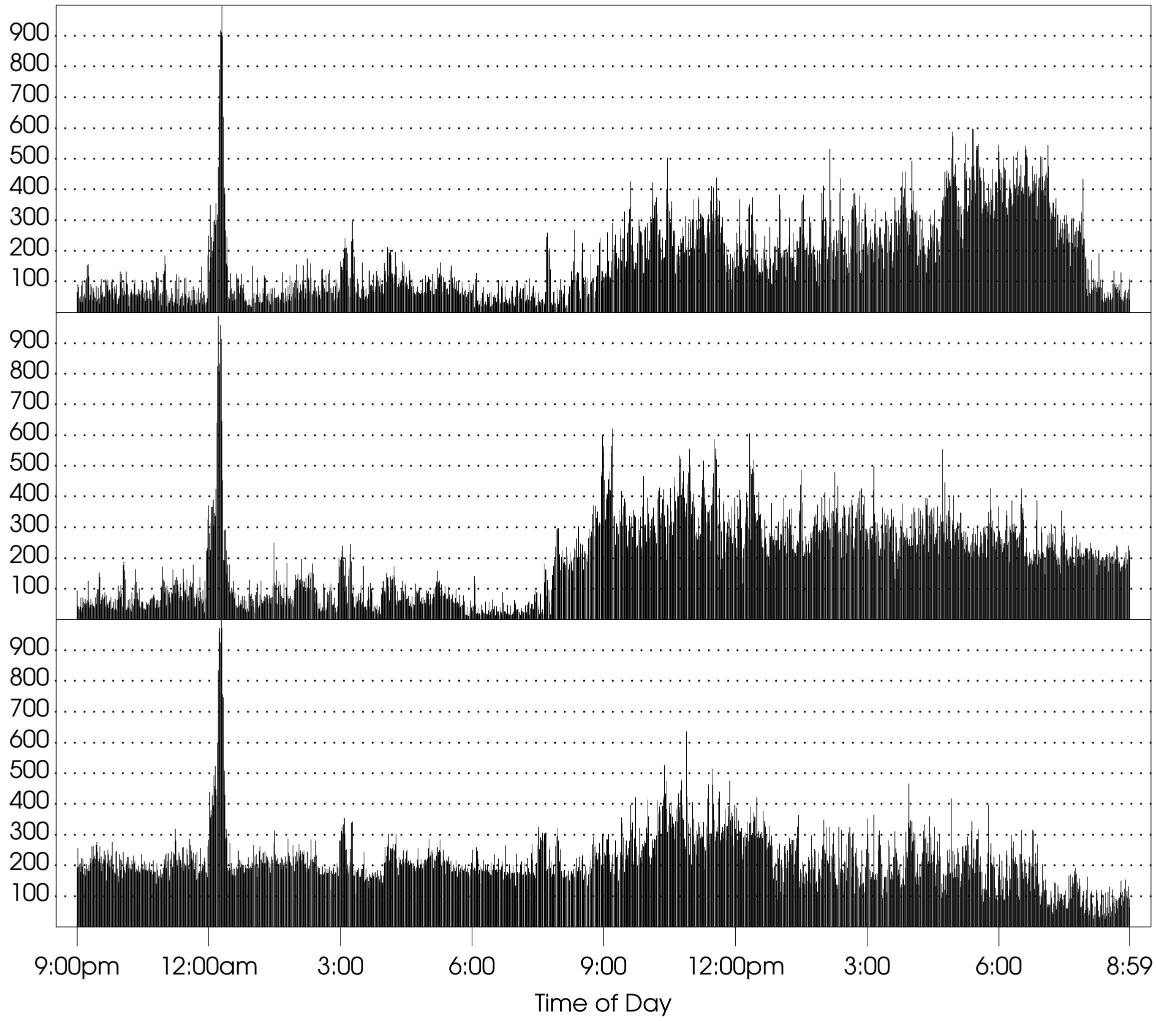
In order to get a picture of the workload of the Sun Ethernet, we monitored the load over 24-hours for several days during the testing phase of the measurement system. Figure 2.12 shows the packet rate over 72 consecutive hours. Notice that the activity, unlike the activity in a university environment shown in Figure A.1, is concentrated during office hours. During the night hours the average packet rate falls to roughly one packet per second per workstation. This background activity consists primarily of diskless workstations periodically flushing to disk their dirty virtual-memory pages and modified pages in the file cache, and of programs, left running by users who are no longer in their offices, that monitor the load on other workstations or file servers by probing them periodically. In addition, UNIX has a mechanism to run ``housekeeping'' programs, such as the commands to clean up directories that contain temporary files, at midnight every day. Since the clocks of all machines are approximately synchronized, this generates the peak load seen shortly after midnight. The higher load during the night in the last 24 hours in Figure 2.12 was caused by a set of benchmark programs that were left running until morning.

Because the objective of our research was to analyze how the user workload determines the traffic behavior and to build models that capture the variability of the arrival processes, we decided to trace only working hours and not the night periods. On the basis of a preliminary study of which the data in Figure 2.12 is part, we selected a typical day for our analysis. In the remainder of this chapter and in the rest of the thesis, the data presented and discussed will be based on traces that we collected on April 10, 1989 during a period starting at 8:15 am and lasting 10 hours and 20 minutes. The full trace amounted to about 850 Mbytes of information. We kept on line and loaded segments of the original data when necessary for more specialized analysis. No packet loss was experienced in this particular trace, which is the same data set whose timing errors were discussed in the previous section; our conclusion there was that the errors in the recorded arrival times were smaller than 100 microseconds. In the remainder of this chapter we will discuss some general statistics of the aggregate traffic in the Sun network; these results should be compared to the results in Appendix A, which describes the aggregate traffic of the Berkeley network.

The Sun Engineering network in which the data were collected consisted of about 130 machines and six major file servers. (We will differentiate between client workstations, which are single-user machines, and file servers, which have no users of their own, but provide file services to client workstations.) Table 2.2 provides some basic information on some of the workstations of the Sun network. (To ensure security of the Sun network and to protect the privacy of individual users, we have not used the actual machine names; the names used throughout the thesis are fabricated ones.)

Figure 2.13 displays the network load during the 10 hours of our data set. Each vertical line shows the percent mean network utilization over a 30-second interval, that is, the fraction of the 30-second period during which machines successfully transmitted Ethernet

FIGURE 2.12. SUN ENGINEERING NETWORK PACKET RATE (PACKETS/SEC)
THREE CONSECUTIVE WEEKDAYS (FROM TOP TO BOTTOM)



frames. (In computing the utilization, we have included the time spent in transmitting the checksums and Ethernet-frame preambles, as well as packet headers and data bytes.) The highest peaks reach 40 percent, indicating that for several seconds the Ethernet load was nearly half of the network's total capacity.

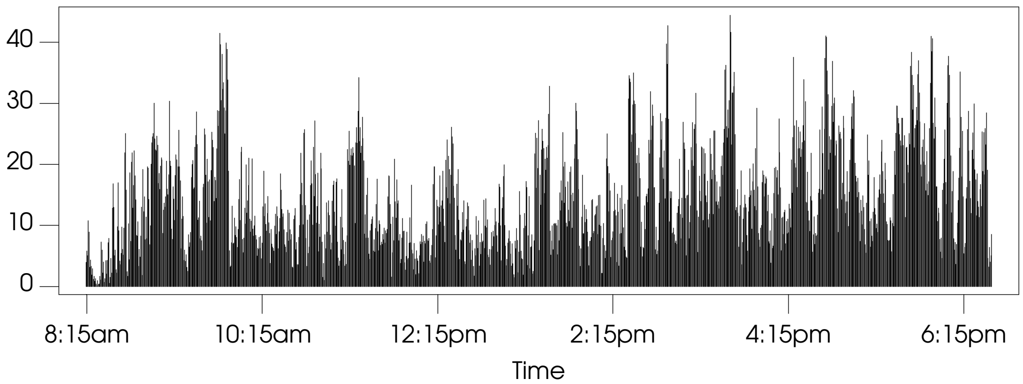
At these levels one would expect a high number of collisions. We have not measured collisions and are not concerned here with the question of estimating the collision rate except for the way in which it affects workstations' network access and the correlations it induces in the traffic among simultaneously communicating workstations. We will deal with these issues in Chapter 3, when studying the sources of correlation in the traffic.

The average load over the 10 hours in Figure 2.13 is 14.6 percent. Much of it involved the six major file servers. As previously noted, at Sun Microsystems it is customary for workstation users to mount several file systems from many different file servers onto their diskless-machine file space. Furthermore, many clients have local disks that in some

TABLE 2.2. WORKSTATIONS WHOSE TRAFFIC IS USED IN THE EXAMPLES

WORKSTATION	TYPE	MEMORY SIZE	OS VERSION	LOCAL DISKS
station 1	Sun-3/160 server	20 Mbytes	Sun-UNIX 3.5	2 disks
station 2	Sun-3/50 client	4 Mbytes	SunOS 4.1	diskless
station 3	Sun-3/50 client	4 Mbytes	SunOS 4.1	diskless
station 4	Sun-4/110 client	8 Mbytes	SunOS 4.0	2 disks
station 5	Sun-3/280 server	8 Mbytes	SunOS 4.0	4 disks
station 6	Sun-4/260 client	16 Mbytes	SunOS 4.0	1 disk
station 7	Sun-4/280 server	32 Mbytes	SunOS 4.0	1 disk
station 8	Sun-4/280 client	32 Mbytes	SunOS 4.0	3 disks
station 9	Sun-4/280 server	32 Mbytes	SunOS 4.0	2 disks
station 10	Sun-3/75 client	8 Mbytes	SunOS 4.1	diskless
station 11	Sun-4/260 client	32 Mbytes	SunOS 4.0	1 disk
station 12	Sun-3/50 client	4 Mbytes	SunOS 4.1	diskless
station 13	Sun-3/50 client	4 Mbytes	SunOS 4.1	diskless
station 14	Sun-3/50 client	8 Mbytes	SunOS 4.0	diskless
station 15	Sun-4/110 client	8 Mbytes	SunOS 4.0	1 disk
station 16	Sun-4/260 client	16 Mbytes	SunOS 4.0	1 disk
station 17	Sun-3/50 client	4 Mbytes	SunOS 4.0	diskless
station 18	Sun-3/110 client	16 Mbytes	SunOS 4.0	1 disk
station 19	Sun-3/280 client	16 Mbytes	SunOS 4.0	2 disks
station 20	Sun-4/110 client	8 Mbytes	SunOS 4.0	1 disk
station 21	Sun-3/280 server	16 Mbytes	SunOS 4.1	2 disks
station 22	Sun-3/50 client	4 Mbytes	SunOS 4.0	diskless
station 23	Sun-4/280 server	32 Mbytes	SunOS 4.0	2 disks
station 24	Sun-4/110 client	8 Mbytes	SunOS 4.0	diskless
station 25	Sun-3/50 client	4 Mbytes	SunOS 4.0	diskless
station 26	Sun-4/260 client	16 Mbytes	SunOS 4.0	1 disk
station 27	Sun-3/260 client	16 Mbytes	SunOS 4.0	1 disk
station 28	Sun-3/280 server	16 Mbytes	SunOS 4.0	2 disks

FIGURE 2.13. NETWORK UTILIZATION (PERCENTAGES)



cases are used to export files to other machines, and thus, at Sun, even personal workstations may function as file servers for other workstations. As a result, the patterns of file access of individual users are quite complicated.

On the average, no single client workstation sent or received much more than 2 percent of the total number of packets over the entire 10-hour trace. File servers generated a higher proportion of the traffic because they interact with many clients simultaneously. This is shown in Table 2.3, which lists the most active workstations. The second column and third column show what percent of the total number of packets and what percent of the total number of bytes respectively had a source address belonging to the machine listed in the left column. Similarly, the last two columns represent the percentages of packets and data bytes received by the workstation. Thus, each of these four columns would add up to 100 percent when complete. Note that the workstations listed (28 out of a total of 130) account for about 80 percent of all packets.

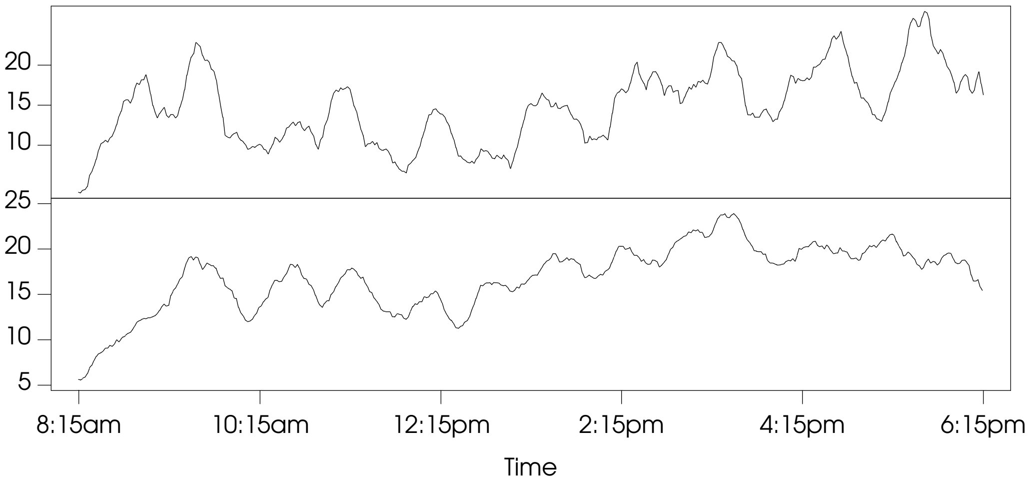
TABLE 2.3. LIST OF SOURCE–DESTINATION TRAFFIC (PERCENTAGES)

WORKSTATION	SOURCE		DESTINATION	
	PACKETS	BYTES	PACKETS	BYTES
station 21 (s)	18.6	19.3	15.4	12.6
station 7 (s)	8.3	9.2	6.2	4.1
station 23 (s)	7.6	10.8	5.6	6.6
station 28 (s)	6.8	6.7	5.9	3.7
station 5 (s)	4.8	6.7	3.1	2.9
station 9 (s)	3.1	3.1	2.4	0.9
station 13	2.4	2.5	2.8	3.6
station 27	2.0	3.0	1.1	0.7
station 11	2.0	0.4	2.3	1.2
station 25	1.9	2.4	2.8	4.5
station 14	1.7	2.1	2.6	4.2
station 3	1.6	1.8	2.4	3.5
station 15	1.6	2.0	1.6	2.0
station 8	1.6	1.8	1.4	1.5
station 20	1.4	1.1	1.2	0.6
station 22	1.3	1.8	1.3	1.9
station 1 (s)	1.3	0.1	2.1	2.7
station 26	1.3	1.5	1.8	2.7
station 2	1.3	1.6	1.5	2.2
station 19	1.2	1.3	1.1	1.0
station 4	1.2	0.6	0.9	0.6
station 24	1.1	0.7	1.2	0.9
station 10	1.1	0.7	1.1	0.7
station 16	1.0	0.9	2.8	5.0
station 18	1.0	0.7	1.0	0.6
station 6	1.0	0.5	1.1	0.8
station 17	0.8	1.0	1.2	1.8
station 12	0.7	0.6	0.9	0.9
Total	79.7	84.9	74.8	74.4

As we can see, the traffic in this network was rather balanced: no single workstation dominated over the others. In contrast, in the Berkeley study we found that a single diskless workstation on occasion generated 20 to 25 percent of the network load over short intervals of time and accounted for more than 16 percent of the total data bytes over a day. The balanced load in the Sun network suggests that there is a better match between hardware—machine power, local disks, large memory configuration—and user workload than in the Berkeley environment. In a university environment, students often are assigned limited resources, which they tend to overwork even if they have to pay the price of poor performance.

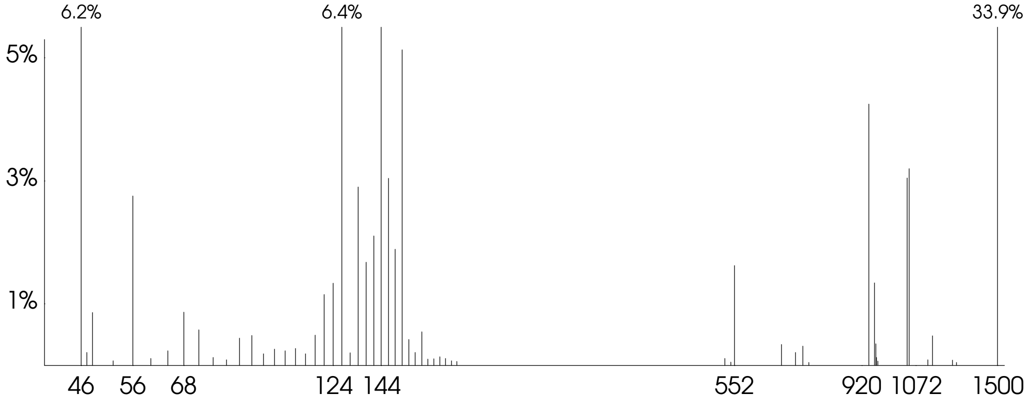
There is clearly a 24-hour periodicity in Figure 2.12 and a shorter periodicity in Figure 2.13. In the next chapter, we will examine in depth the question of whether or not the traffic produced by individual stations is stationary. Here, we are interested in a related question: whether the trends in the aggregate traffic are connected to the number of active workstations. If so, the nonstationary trends in Figure 2.13 can be predicted by the number of active workstations for some definition of ``active''. At any given time, one can define a workstation as active if it has transmitted at least a specified number of bytes or packets over a given interval. The ``best'' results were obtained when we considered active those workstations that sent more than 4 Kbytes in 1-second intervals. Even in this case, however, the association between the number of active workstation and network load was weak, as qualitatively illustrated in Figure 2.14, which shows a smoothed version of the load in Figure 2.13 and the smoothed number of workstations that transmitted more than 4 Kbytes in 1-second intervals. (The smoothing was performed in both cases by averaging together 40 consecutive values.) This is not surprising when we recall that different workstations produce different loads, and that even a single workstation generates varying loads during the day.

FIGURE 2.14. COMPARISON OF PERCENT NETWORK LOAD (TOP) AND NUMBER OF ACTIVE WORKSTATIONS (BOTTOM)



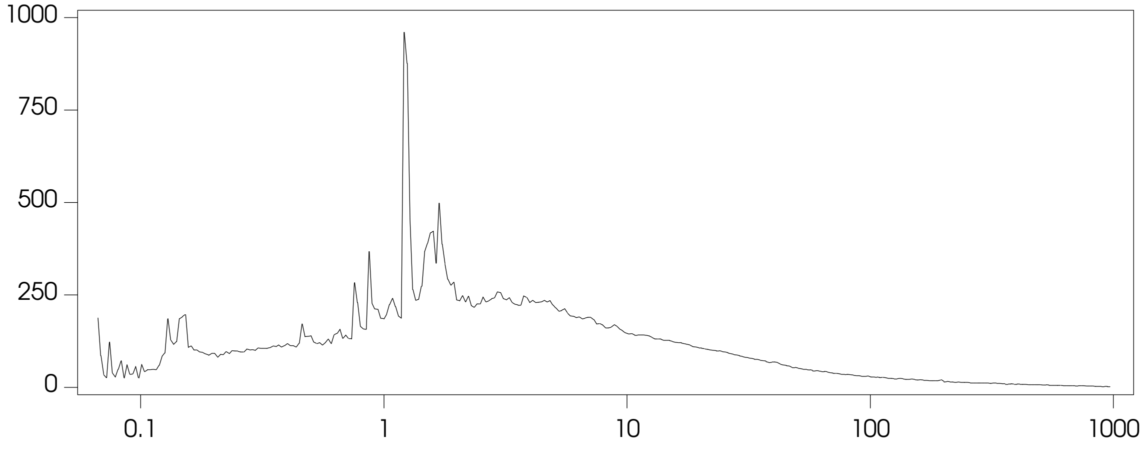
It is interesting to compare the network's packet length distribution with that of the Berkeley study. Figure 2.15 shows the percentage of packets transmitted in the Sun network as a function of the packet length. Its counterpart for the Berkeley network is Figure A.3. Figure 2.15 is very interesting. As expected, there are many fewer small packets than in the Berkeley network, in which many users accessed machines on other networks through remote login programs, producing small TCP packets for keyboard characters and for the associated protocol acknowledgements. (TCP produces in this figure the lines at 46 and 1064 bytes.) Also predictable is the much higher peak at 1500 data bytes, caused by the substitution of ND with NFS as the transport protocol for paging traffic. (Notice that ND is still being used by the server *station 1*; this protocol produces the lines at 48 bytes and 1072 bytes.) We observed that, except for the peaks mentioned above produced by TCP and ND, NFS generates all the other peaks, an indication of the homogeneity of the Sun environment. The higher weight of the mid-sized packets, those between 100 and 200 data bytes, which are NFS remote procedure call protocol packets, is more interesting. Together, the packets in this interval constitute 34.7 percent of the total. We had predicted in [33] that this segment of the packet-length distribution would have increased as more and more applications use RPC protocols.

FIGURE 2.15. PERCENTAGE OF PACKETS VS. PACKET LENGTH



Figures 2.16 and 2.17 show histograms of the interarrival times and of the interpacket times. In both graphs, the abscissa is logarithmic and on the ordinate we plot the square root of the number of events (interarrival and interpacket times respectively) that fall in a particular interval.

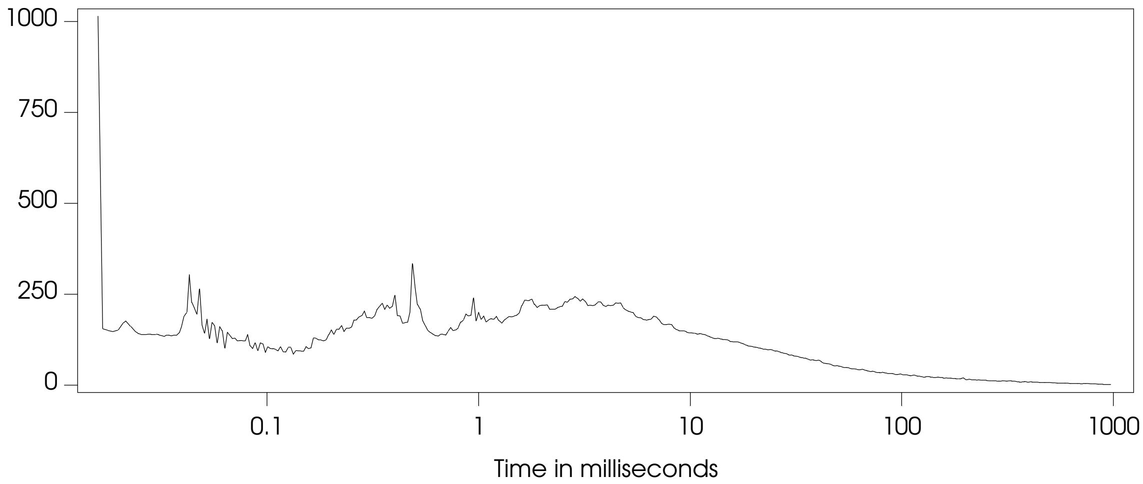
FIGURE 2.16. HISTOGRAM OF INTERARRIVAL TIMES – AGGREGATE TRAFFIC



The highest peak in Figure 2.16 is generated by consecutive 1500 byte packets. As we have seen in Section VII, most of these packet pairs have unrelated source and destination addresses, and are separated by the minimum interpacket times. Thus, most of them appear in the peak near the origin in Figure 2.17.

The histograms in Figures 2.16 and 2.17 are proportional to estimates of the marginal density probability functions of the interarrival and interpacket times. These estimated functions are difficult to model directly as they appear to be the superposition of discrete and continuous components. Although we understand the significance and the origin of

FIGURE 2.17. HISTOGRAM OF INTERPACKET TIMES – AGGREGATE TRAFFIC



the peaks, we can think of no simple analytical representation for them. The value of these histograms is the negative result that they make explicit: times between arrivals are not exponentially distributed,

There are significant correlations between subsequent interarrival times, but we will postpone the study of their second order properties to Chapter 3, in which we look at statistical descriptions of the arrival processes generated by individual workstations.

IX. Summary

In this chapter we have described and illustrated our measurement methodology and techniques. Measurements were taken through passive listening of all packets on an Ethernet local-area network. We used a dedicated machine with a modified network driver that collected packet protocol information and dumped it to large files, which were later transferred to magnetic tapes. The machine had a 2-MHz hardware counter that was used as a high-resolution clock. However, as our detailed timing error-analysis section has shown, the clock values in the traces are not as accurate as the original clock's 0.5 microsecond resolution. These timing errors, the great majority of which are less than 100 microseconds, are still small for the type of analyses we carry out in this study.

We have discussed the basic properties of the aggregate traffic of the Sun network mainly to place in a broader perspective the per-workstation statistical description of the traffic, which will be the topic of Chapter 3. The reader may also find it useful to compare directly these results to our previous analyses of traffic measurements in a university environment, which we reproduce in Appendix A.