The NTP protocol operations in basic client/server, symmetric and broadcast modes have been described in previous chapters. As pointed out in the previous chapter, the highest accuracy obtainable requires the use of driver or hardware timestamps; however, this requires intervention in the kernel driver and/or network interface. This chapter describes two new modes, interleaved symmetric and interleaved broadcast, which can improve accuracy without modifying these components. As in previous NTP versions, NTP Version 4 with interleaved capability is compatible with NTPv4 without this capability and with all previous NTP versions.

In the reference implementation the receive timestamp is captured shortly after the input device interrupt to avoid input latencies due to buffering and input queuing operations. According to the nomenclature adopted in the previous chapter, this would be described as a drivestamp. The transmit timestamp is ordinarily captured just before computing the message digest and then adding the buffer to the output queue. In this method the output latencies due to queueing and buffering operations can become significant. In this chapter the transmit timestamp is captured shortly after the output device interrupt, so would properly be described as a drivestamp. However, in this case the transmit drivestamp is not available when the packet header fields are determined, so is sent in the transmit timestamp field of the following packet. This is basically how the interleaved modes work and is the topic of this chapter.

To distinguish between timestamps captured in various ways, timestamps determined when an input packet removed from the input queue or when an output packet is added to the output queue are called softstamps, while timestamps determined at interrupt time are called drivestamps. While the choice of receive softstamp or drivestamp affects accuracy, it does not affect the protocol operations. On the other hand, while the choice of transmit softstamp or drivestamp affects accuracy, the choice does affect the protocol operations. In order to emphasize the difference in the diagrams in this chapter, softstamps appear with an asterisk (*) while drivestamps do not. Since all timestamps used in this chapter are drivestamps, it will lessen clutter if we avoid the distinction and simply call them timestamps.

Interleaved symmetric mode is an extension of basic symmetric mode, while interleaved broadcast mode is an extension of the basic broadcast mode described in Chapter 2. The interleaved modes require servers and peers to retain state, so there is no interleaved client/server mode. Mode selection for configured symmetric peers and broadcast servers is determined by a configuration option; mode selection for passive peers and broadcast clients is determined automatically by the protocol. Interleaved broadcast mode is compatible with basic broadcast.
NTP Interleaved Modes

253

Basic/Interleaved Protocol State Machines and Flow Charts

mode; a server configured for interleaved broadcast mode can support both interleaved and basic broadcast clients at the same time and on the same wire.

The plan for this chapter is first to describe the protocol state machines, including the packet header variables, state variables and flow charts. As the protocol state machine must support both the basic mode operations as well, all the modes will be described in an integrated way. This description is followed by a detailed description of each mode along with examples of operation. Finally, an example is presented that shows how the protocol automatically reverts to basic mode when confronted with an NTP implementation that does not support interleaved modes.

16.1 Basic/Interleaved Protocol State Machines and Flow Charts

The integrated modes require two protocol state machines: one for the transmit process, which runs for each transmit packet, and the other for the receive process, which runs for each receive packet. The processes can run in one of four modes: basic symmetric and basic broadcast modes, and interleaved symmetric and interleaved broadcast modes. The state machines described in this section support all of these modes depending on a configuration option. Ordinarily, the transmit and receive state machines run in the same mode; however, as described later, a transmit process configured for interleaved symmetric mode will automatically switch to basic symmetric mode if confronted by a peer incapable of interleaved operation.

Figure 16.1 shows three packet header variables, $t_{org}$, $t_{rec}$ and $t_{xmt}$; eight state variables rec, dst, aorg, borg, xmt, x, b and h; and three peer variables $\theta$, $\delta$ and $\beta$, used in the flowcharts to follow. The state variables persist from packet to packet; the packet variables, which have subscripts for clarity, persist for the life of the packet only. All the state variables, except $x$, $b$ and $h$, and all the packet variables, carry timestamp values; $b$ takes values 0 and 1, $x$ takes values $0$, $+1$ and $-1$, and $h$ is a counter used during error recovery in some modes. Some state variables are not used in some modes. By convention in the following, the pseudo-state variable clock designates the system clock at the time of reading. The peer variables represent the product of the protocol, $\theta$ is the clock offset, $\delta$ the roundtrip delay and $\beta$ the unicast/broadcast offset, as described below.
The flow charts in Figure 16.2 show the transmit operations in all modes. Note that x serves both as a switch to indicate basic modes or interleaved modes, as well as in interleaved modes a means to alternate the transmit timestamp between aorg and borg. It is set at configuration time to 0 for basic mode or +1 for interleaved mode. Note also that in interleaved broadcast mode torg ordinarily zero in basic broadcast mode, it is hijacked to hold the previous transmit timestamp. This is how a broadcast client recognizes whether interleaved mode is available. Clients conforming to the NTPv4 and previous specifications ignore torg and trec.

The flow charts in Figure 16.3 show the receive operations in the basic modes, while the flow charts in Figure 16.4 show the receive operations in interleaved modes. In this and later flow...
NTP Interleaved Modes

16.2 Basic Symmetric Mode

The basic modes conform to the current NTPv4 specification and reference implementation. Following is an example illustrating typical operation in this mode, starting from an unsynchronized condition. Figure 16.5 illustrates the most general case where peers A and B independently measure the offset and delay relative to each other. Note this is basically the same as shown in Figure 3.2, but including much additional information on the interaction with the interleaved modes and emphasizing the use of softstamps. Each packet transmitted is shown as an arrow with the receive timestamp at the head and the transmit timestamp at the tail. The grey boxes hold timestamps captured from the system clock; the other boxes hold values copied from them, other state variables or packet variables.

The labels shown are after arrival or departure of a packet. Keep in mind that, upon arrival of a packet, a state variable might be needed before update from a packet variable.

- When a packet is received, $t_{xmt}$ is copied to $rec$ and the receive timestamp to $dst$.
- When a packet is transmitted, $rec$ is copied to $t_{org}$, $dst$ to $t_{rec}$ and the transmit timestamp to $aorg$ and $t_{xmt}$.

---

```c
if (x > 0)
  T1 = aorg
else
  T1 = borg
T2 = rec
T3 = t_{xmt}
T4 = dst
if (t_{org} == 0 || T1 == 0 || T2 == 0 || T3 == 0)
  SYNC
else if (t_{org} != 0 && t_{org} != T4) {
  h = 2
  BOGUS
} else if (h == 0) {
  \theta = [(T2 - T1) + (T3 - T4)] / 2
  \delta = (T4 - T1) - (T3 - T2)
}
```
After the state variables have been updated, the timestamps are $T_1 = t_{org}$, $T_2 = t_{rec}$, $T_3 = t_{xmt}$ and $T_4 = dst$. As in Chapter 2, these timestamps can be used to calculate the clock offset and roundtrip delay.

It is important to note that the protocol operations are symmetric, in that A can determine the offset and delay of B relative to A and at the same time B can determine the offset and delay of A relative to B. In Figure 16.5, A first sends $T_1 \rightarrow T_2 (0, 0, T_1)$ to B. Later, B sends $T_3 \rightarrow T_4 (T_1, T_2, T_3)$ to A. At $T_4$ the timestamps $T_1, T_2, T_3$ and dst variable $T_4$ are available to compute the offset and delay of B relative to A. Later, A sends $T_5 \rightarrow T_6 (T_3, T_4, T_5)$ to B. In a similar fashion, at $T_6$ the timestamps $T_3, T_4, T_5$ and dst variable $T_6$ are available to compute the offset and delay of A relative to B. The protocol is said to be synchronized when both A and B have computed the offset and delay; that is, at $T_6$.

In symmetric modes each peer independently polls the other peer, but not necessarily at identical intervals. Thus, one or the other peer might receive none, one or more than one packet between polls. This can result in lost or duplicate packets and even cases where packets cross each other in flight, resulting in out-of-sequence or bogus packets. In addition, provisions must be made to reset and restart the protocol, if necessary. There are three sanity checks designed to detect duplicate, unsynchronized or bogus packets.

1. The packet is a duplicate if $t_{xmt}$ matches $xmt$ saved from the last packet received. This might occur due to a retransmission in the network or a malicious replay without modification. The proper response is to ignore the packet without modifying the state variables. Thus, if the network or an intruder replays the $T_3 \rightarrow T_4$ packet, this test would discard it.
2. The packet is unsynchronized if either $t_{org}$ is zero or $t_{rec}$ is zero or $t_{xnt}$ is zero. This could result if the transmitter has not yet synchronized, or if the implementation is defective. The proper response is to ignore the packet, but update the state variables.

3. The packet is bogus if $t_{org}$ does not match $org$. This might occur due to packet loss, reorder or a malicious replay with modification. The proper response is to ignore the packet, but update the state variables.

The bogus test is actually a strong nonce that binds a previously transmitted packet to the reply. For this reason every $t_{org}$ should be different. As the resolution of this variable is less than one nanosecond and, in principle, packets cannot be sent that fast, this is feasible. However, in many cases the resolution of the system clock is insufficient. In these cases the low-order insignificant bits of the timestamp should be filled with a random bit string. Not only does this insure that every timestamp will be different, it also helps to avoid averaging bias.

Although not shown in Figure 16.5, $org$ is set to zero after the bogus test in order to deflect a replay of the first packet in client/server mode. Thus, if the network or an intruder were to replay the $T_1 \rightarrow T_2$ packet causing $B$ to generate a spurious $T_3 \rightarrow T_4$ packet, the bogus test would discard it. Thus, in client/server mode the protocol is protected against replays of either the $T_1 \rightarrow T_2$ or $T_3 \rightarrow T_4$ packets. In symmetric modes replay of the $T_1 \rightarrow T_2$ packet would be caught as a duplicate by $B$. The protocol is inherently resistant to lost packets and overlapped packets. For instance, if the $T_3 \rightarrow T_4$ packet is lost, the next set of timestamps $T_5, T_6, T_7$ and $T_8$ are available to compute offset and delay. If packets $T_3 \rightarrow T_4$ and $T_5 \rightarrow T_6$ overlap, $A$ will discard the latter as bogus and use the next set of timestamps as before.

**16.3 Interleaved Symmetric Mode**

In interleaved modes the transmit timestamp is captured after the packet has been sent, so it is sent in the next following packet. This can be done using the two-step or interleaved protocol described in this section. The trick, however, is to implement the interleaved protocol without changing the NTP packet header format, without compromising backwards compatibility and without compromising the error recovery properties. Following is a typical example of operation, starting from an unsynchronized condition.

Figure 16.6 illustrates the most general case where interleaved symmetric peers $A$ and $B$ independently measure the offset and delay relative to each other. Note that the receive (even-numbered) timestamp is available immediately after the packet has been received, but the transmit (odd-numbered) timestamp is available only after the packet has been transmitted. In contrast to the basic protocol, which requires one complete round to calculate offset and delay, the interleaved round requires two basic rounds. The interleaved round that begins at $t_1$ is not complete until $t_8$, while the interleaved round that begins at $t_5$ is not complete until $t_{12}$. However, the rate of offset/delay calculations is the same as the basic protocol. The NTP packet header fields are the same in the interleaved protocol as in the basic protocol, but carry different values.
Each peer requires the state variables defined in Figure 16.1. The transmit state machine operates as in Figures 16.2, while the receive state machine operates as in Figure 16.4.

- When a packet is received, $t_{\text{rec}}$ is copied to $\text{rec}$ and the receive timestamp to $\text{dst}$.
- When a packet is transmitted, $\text{rec}$ is copied to $\text{torg}$ and $\text{dst}$ to $t_{\text{rec}}$. If $x = +1$, the transmit timestamp for the previous transmitted packet is copied to $\text{aorg}$ and $\text{borg}$ is copied to $t_{\text{xmt}}$. If $x = -1$, the transmit timestamp for the previous transmitted packet is copied to $\text{borg}$ and $\text{aorg}$ is copied to $t_{\text{xmt}}$.

Upon receipt and before the state variables have been updated, the timestamps are $T_2 = \text{rec}$, $T_3 = t_{\text{xmt}}$ and $T_4 = \text{dst}$. If $x = +1$, $T_1 = \text{aorg}$; if $x = -1$, $T_1 = \text{borg}$. After this, $t_{\text{rec}}$ is copied to $\text{rec}$ and the receive timestamp to $\text{dst}$.

As in the basic interleaved mode, there are three tests to detect duplicate, unsynchronized or bogus packets.

1. The packet is a duplicate if $t_{\text{xmt}}$ matches $\text{xmt}$ saved from the last packet received. This is the same test as in basic interleaved mode. The proper response is to discard the packet without modifying the state variables.
2. The packet is unsynchronized if either $t_{\text{org}}$ is zero or $T_1$ is zero or $T_2$ is zero or $T_3$ is zero. This test is slightly different, due to the requirement that old state variables be expunged when a peer restarts the protocol. The proper response is to ignore the packet, but update the state variables.

3. The packet is bogus if $t_{\text{org}}$ does not match $T_4$. The proper response is to ignore the packet, but update the state variables. This test is slightly different, due to the different placement of the state variables. The proper response is to ignore the packet and the next one received, but update the state variables.

A bogus packet interleaved symmetric mode is much more expensive than in basic symmetric mode. The protocol state machine extends over two basic rounds, which means that recovery will last that long. The strategy chosen is to update the state variables, but otherwise ignore the timestamps for the bogus packet and the next one as well. By that time the old state variables have been flushed and replaced with fresh ones. This is the purpose of the $h$ state variable. It is set to 2 when a bogus packet is found and is decremented by 1 for each received packet, but not below zero. The state variables are updated, but timestamps not used if $h > 0$.

### 16.4 Interleaved Broadcast Mode

In NTP basic broadcast mode the client responds to the first broadcast received by executing a number of client/server mode protocol rounds in order to calibrate the offset difference between the broadcast spanning tree topology and the unicast spanning tree topology. In the reference implementation the rounds are continued several times in order to refine the measurements and complete the security protocol, if required. In client/server mode the protocol is inherently resistant to lost, duplicate or misordered packets, but in interleaved broadcast mode special considerations apply to avoid disruptions due to these causes.

Recall from Section 15.4 that, in IEEE 1588 Precision Time Protocol (PTP), the master broadcasts a Sync message to the slaves, which capture the receive timestamp $T_2$. Immediately thereafter, the master broadcasts a Follow_up message including the transmit timestamp of the Sync message $T_1$. Some time later, each client separately sends a Delay_req message to the master and captures the transmit timestamp $T_3$. The master returns a Delay_resp message containing the receive timestamp $T_4$ for the Delay_req message. The client collects these timestamps to calculate the offset and delay, as in the NTP protocol with the offset sign inverted. Subsequently, the master broadcasts a Sync/Follow_up pair of messages from time to time and the slaves apply a correction equal to one-half the measured delay to determine the true offset.

The problem with the IEEE PTP applied to large networks is that the packet routing for broadcast packets and unicast packets can be far different. Unicast packets are usually routed via a shortest-path spanning tree, while broadcast packets can be routed either by distance vector multicast (DVM) protocol or by protocol-independent multicast (PIM) protocol; each have different subgraph aggregation rules. In PTP $T_1$ and $T_2$ are determined via the broadcast subgraph, while $T_3$ and $T_4$ are determined via the unicast subgraph. While the delays on the outbound and inbound paths of the unicast subgraph are often substantially the same, the delays between the same two points on the broadcast subgraph can be substantially different.
The NTP interleaved broadcast protocol faces this problem in a fundamentally different way. The actual transmit timestamp $T_1$ for one broadcast packet is sent in the following broadcast packet. The client saves the receive timestamp $T_2$ for the previous packet and uses the associated transmit timestamp $T_1$ in the current packet. The client applies a correction measured during the client/server calibration phase and determines the current clock offset.

Figure 16.7 shows a typical scenario where the clock is updated as each broadcast packet arrives. In basic broadcast mode the transmit softstamp $txmt$ is used to determine the clock offset. On the other hand, the transmit timestamps $aorg$ and $borg$, which alternate with each packet, are captured at output device interrupt time, so do not have output queueing and buffering latencies. The interleaved broadcast mode uses the same packet header format as the basic mode, but includes in $torg$ the transmit timestamp for the previous broadcast packet. As each packet is received, $torg$ contains the transmit timestamp for the previous packet, $borg$ the corresponding receive timestamp and $aorg$ the corresponding transmit softstamp. While transmit softstamps are not used directly in interleaved broadcast mode, they are included to support both basic and interleaved modes with the same packet stream and to detect lost packets in interleaved mode. In the current NTP specification and NTP versions implemented prior to the interleaved protocol support, the $torg$ and $trec$ packet variables are unused in broadcast mode and are ordinarily set to zero.

In Figure 16.7, B is the broadcast server and A is the broadcast client. Packets sent at $T_1$, $T_3$, $T_{11}$ and $T_{13}$ use broadcast mode (5), while the packet sent at $T_5$ uses client mode (3) and the packet sent at $T_7$ uses server mode (4). As in basic broadcast mode, the stateless calibration round $T_5 - T_8$ uses no server state variables and the timestamps $T_6$ and $T_7$ are essentially coincident. While only one stateless round is shown, there can be a number of them to refine the measurements using the
NTP Interleaved Modes

Error Detection and Recovery

NTP clock filter and data grooming algorithms, as well as the optional Autokey security protocol. During the calibration phase broadcast packets are ignored.

For simplicity in the presentation, assume that the first broadcast is not received by the client. Clients that support interleaved broadcast mode will note that \( t_{\text{org}} \) is nonzero in later broadcasts and in that case set state variable \( b \) to 1. The broadcast client now executes one or more ordinary stateless client/server mode rounds, which in this example results in a unicast clock offset \( \theta_U \) at \( T_8 \). In the reference implementation, stateless rounds continue until the synchronization distance drops below the selection threshold, normally in about four rounds.

If upon the arrival of the next broadcast at \( T_{12} \) \( b \) is nonzero, the broadcast clock offset is

\[
\theta_B = t_{\text{org}} - dst.
\]

The difference between the unicast offset and the broadcast offset represents the bias of the unicast subgraph delay relative to the broadcast subgraph delay. The peer bias variable

\[
\beta = \theta_U - \theta_B
\]

and \( b \) is set to zero. In order to simplify the calculations in the various modes of operation, the delay computed in the unicast rounds is replaced by \( \delta = 2 \beta \). In interleaved broadcast mode this packet calculates the delay and initializes the state variables, but is not otherwise used to calculate other peer variables. For packets arriving at \( T_{14} \) and later, the peer offset is

\[
\theta = \theta_B + \frac{\delta}{2}.
\]

By construction, the difference between the transmit timestamp \( t_{\text{org}} \) and the corresponding softstamp \( a_{\text{org}} \) represents the message digest, queuing and buffering latencies in the broadcast server, \( \varepsilon = t_{\text{org}} - a_{\text{org}} \). If \( \varepsilon \) is less than zero or greater than the largest expected latency, a packet has been lost or replayed. In such cases, the state variables are updated, but not the peer variables.

16.5 Error Detection and Recovery

When a packet is lost or appears unsynchronized, bogus or crossed in basic symmetric mode, the strategic response is to simply drop it. In interleaved symmetric mode duplicate packets are ignored; unsynchronized and bogus packets update the state variables only. While not shown in the flow charts, a hold-off counter \( h \) is used to avoid updating the peer variables for this and the next packet, even if it appears valid. It is set to 2 for bogus packets and decrements by 1 for subsequent packets. This causes the bogus packet and the next packet received to update only the state variables, but otherwise be ignored.

As previously mentioned, the interleaved modes have been implemented and tested in the reference implementation. However, a code surveyor might find it difficult to follow the code flow, as it is intertwined with the authentication and rate management subsystems. The implementation is based on a simple test program used to verify correct behavior under normal and abnormal conditions with errors of various types. As an aside, this programs was used to
generate the packet traces of Figure 16.8. Here, peer A starts in basic symmetric mode, while peer B starts in interleaved symmetric mode. To make the example more interesting, both peers start at the same time, but B hears A at $T_2$ before A hears B at $T_4$. By rule and the state machine in Figure 16.4, packets received at $T_2$, $T_4$, and $T_6$ are unsynchronized and only the state variables are updated. However, the packets received at $T_8$ and $T_{10}$ are declared bogus by A, since $t_{org}$ does not match $a_{org}$ as required by the basic symmetric state machine. Likewise, the packet received at $T_{10}$ is declared bogus by B, since $t_{org}$ does not match $rec$. However, $t_{org}$ matches $a_{org}$, so B knows that A is in basic symmetric mode. B switches to that mode and both A and B continue in that mode.

This is not the only error/recovery scenario, of course, but it does verify that the state machines can settle their arguments without producing an undetected error. In fact, correct protocol behavior has been demonstrated in all cases involving no mode switch, even under an artificial dropped packet error rate of ten percent.

**16.6 Measured Performance with the Interleaved Modes**

As mentioned previously, the interleaved modes have been implemented and tested in the reference implementation. This section contains a performance analysis using seven test and production machines on test and production LANs. The synchronization paths operate in basic, symmetric, interleaved symmetric and interleaved broadcast modes and with various combinations of symmetric key and Autokey cryptographic means. The goal of the testing program is to quantify the accuracy and evaluate the latencies due to various causes.
There are two networks available for testing: the Backroom LAN, a 100-Mb/s switched Ethernet with very little traffic other than the test traffic, and the Campus LAN, a 100-Mb/s switched Ethernet with two very busy servers and a NTP traffic volume of well over 1000 packets per second. The test and production machines used on the LANs are summarized in Table 16.1. The

<table>
<thead>
<tr>
<th>Name</th>
<th>CPU</th>
<th>Ethernet NIC</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baldwin</td>
<td>Sun UltraSPARC III</td>
<td>10/100/1000 Mb/s</td>
<td>Solaris 5.10</td>
</tr>
<tr>
<td>Bridgeport</td>
<td>Sun UltraSPARC III</td>
<td>10/100/1000 Mb/s</td>
<td>Solaris 5.10</td>
</tr>
<tr>
<td>Howland</td>
<td>Intel P II, 1.0 GHz</td>
<td>RealTek 8139 10/100 Mb/s</td>
<td>FreeBSD 6.1</td>
</tr>
<tr>
<td>Macabre</td>
<td>Intel P 4x2, 2.8 GHz</td>
<td>Broadcom BCM5751 10/100/1000 Mb/s</td>
<td>FreeBSD 6.1</td>
</tr>
<tr>
<td>Mort</td>
<td>Intel P 4x2, 2.8 GHz</td>
<td>Broadcom BCM5751 10/100/1000 Mb/s</td>
<td>FreeBSD 6.1</td>
</tr>
<tr>
<td>Pogo</td>
<td>Sun Ultra 5.1</td>
<td>10/100 Mb/s</td>
<td>Solaris 5.10</td>
</tr>
<tr>
<td>Rackety</td>
<td>Intel P II, 500 MHz</td>
<td>3COM 3C905B 10/100 Mb/s</td>
<td>FreeBSD 6.1</td>
</tr>
</tbody>
</table>

Backroom LAN shown in Figure 16.9 includes three switches (one not shown) interconnected by 100-Mb/s fiber repeaters. As the lengths of the fiber segments are less than 30 m, the propagation delays are well below the microsecond, so are neglected. The GPS server is a Meinberg LANtime M600, which includes an integrated NTP server and Ethernet port. This network is used for program development and testing only, so carries very little production traffic other than NTP.

The Backroom LAN includes three hosts: Macabre, Mort and Howland. Macabre and Mort hosts are identical Pentium machines operating in interleaved symmetric mode with each other, as well as client/server mode with the GPS server. Each includes an interleaved broadcast server in the configuration. Howland operates as an interleaved broadcast client. Each of the three hosts use two associations as shown in the figure. Each association includes as peer variables the measured offset $\theta$, delay $\delta$, jitter (RMS offset differences) $\varphi$ and output queuing delay $\varepsilon$, all in milliseconds.
In addition, interleaved broadcast client Howland includes the bias $\beta$ calculated in the preceding section. The interleaved associations are all operated using Autokey cryptography. The delays reported in this case are calculated in the calibration rounds.

Inspection of the results suggests that Macabre and Mort can maintain the system clock within a few microseconds of each other and within low tens of microseconds with the GPS server. Slower Howland can maintain the system clock relative to the servers within low tens of microseconds. Further inspection suggest the jitter measured over the network (two or three LAN segments) is in the low tens of microseconds.

One interesting result of the experiment is the output queuing delay, about 15 $\mu$s for Macabre and Mort and about twice that for Howland. The experiment included Autokey, so the measured delays include the message digest computation. For the interleaved broadcast client, the value of $\beta$ is roughly one-half the roundtrip delay $\delta$, as expected. From these results the input queuing delays can be calculated from the measured roundtrip delay and the calculated LAN delays. Note that the output queuing delays do not contribute to the calculations. For Macabre and Mort the roundtrip path between them includes four LAN hops, where each LAN hop is about 8 $\mu$s plus 2 $\mu$s for the switch, for a total of 40 $\mu$s roundtrip transmission delay. For a roundtrip delay of roughly 100 $\mu$s, this leaves 60 $\mu$s for the input queuing delay split between two NICs. These measurements are typical of lightly loaded backwater LANs in which the performance improvement using the interleaved modes is only a few tens of microseconds.

The scenario can be quite different with slower machines, heavily loaded LANs and large Ethernet switches contributing modest to large delay variations. The Campus LAN shown in Figure 16.10 includes one large Ethernet switch serving about two dozen subnets with several hundred production servers and clients and a few test hosts. The particular LAN referred to here is one of these subnets. It includes two busy NTP servers: Pogo and Rackety, and two test hosts: Bridgeport and Baldwin, dedicated to the experiment. Collectively, the NTP servers support over a thousand clients in the global Internet with a traffic level of several hundred packets per second. In the figure Pogo and Rackety operate with each other in interleaved symmetric mode, while Bridgeport and Baldwin operate with each other in interleaved symmetric mode. Both Bridgeport and Baldwin operate in client/server mode with Rackety. As in the Backroom LAN experiment, Autokey is used for all interleaved associations.
In addition to the interleaved associations shown, both Pogo and Rackety are configured for individual Spectracom 8183 GPS receivers which produce both a serial timecode and a PPS signal. The serial timecode is processed by a reference clock driver, while the PPS signal is processed by the kernel PPS discipline described in Chapter 8. This produces an exceptionally clean reference offset which hovers near zero as shown, but occasionally surges up to 3 \( \mu s \). The roundtrip delay for reference clock drivers is by design zero and the jitter is at the minimum, limited by the machine precision.

While Baldwin and Bridgeport have identical architecture and operating systems, they share a busy switch with a core throughput of several GB/s, so the jitter and delay variations are somewhat higher than the Backroom LAN. However, note that the roundtrip delay is three times higher than the Backroom LAN and the output queuing delay is twice as high, suggesting that the Ethernet NICs or the NIC driver may be aggregating interrupts in order to reduce the interrupt load. The roundtrip delays measured by both Pogo and Rackety are roughly 600 \( \mu s \), while the jitter and output queuing delays are about the same as Baldwin and Bridgeport.

The interesting thing about these data is that the offsets of Pogo and Rackety are about the same but of opposite sign, which reveals an asymmetric path between the machines, most certainly due to the Ethernet NIC and/or driver structure. The results show Rackety offset roughly \(-180 \mu s\) relative to Pogo and Pogo offset roughly \(+180 \mu s\) relative to Rackety, which can only be accounted for if the difference in delay from Rackety to Pogo is 360 \( \mu s \) larger than the delay from Pogo to Rackety. By calculation, the input queuing plus transmission delays from Pogo to Rackety are 120 \( \mu s \) while the delays on the reciprocal direction are 480 \( \mu s \). This highlights the extreme case where differences in NIC hardware and driver design can have significant affect on overall network accuracies.

### 16.7 Parting Shots

It is tempting to speculate on an interleaved symmetric design for the Proximity-1 space link protocol described in the next chapter. In the current hardware design the timestamps are captured in a way very similar to IEEE 1588; that is, they are captured upon passage of a designated octet at the hardware frame level. As in IEEE 1588, this requires intricate hardware and software provisions which may not be justified if the expected accuracy is at the millisecond level.

A Proximity-1 frame can have up to 2048 octets and is transmitted at rates from a 1 kb/s to 128 kb/s. Frames are first processed by a Reed-Solomon encoder then by a convolutional encoder which operates at one-half rate. We can assume that, once the transmission rate and frame length are known, the transmission delay can be calculated. However, the transmission rates and frame lengths are often far different in the reciprocal directions. Assuming a Unix-like design for the spacecraft computer system, and the timestamp capture techniques described in the previous chapter, and the interleaved symmetric protocol described in this chapter, accuracies less than a millisecond should be achievable, even with whopping output queuing delays of several minutes.