11-13-1995

Time Synchronization Server

Michael D. Myjak

Nicole L. Densmore

Christopher H. Esser

INSTITUTE FOR SIMULATION AND TRAINING

Contract Number N61339-94-C-0024
CDRL A00D
U.S. Army STRICOM
13 November, 1995

Time Synchronization Server

IST

Institute for Simulation and Training
3280 Progress Drive
Orlando FL 32826

University of Central Florida
Division of Sponsored Research

IST-TR-95-28

INSTITUTE FOR SIMULATION AND TRAINING

Contract Number N61339-94-C-0024
CDRL A00D
U.S. Army STRICOM
13 November, 1995

Time Synchronization Server

IST

Institute for Simulation and Training
3280 Progress Drive
Orlando FL 32826

University of Central Florida
Division of Sponsored Research

IST-TR-95-28

INSTITUTE FOR SIMULATION AND TRAINING

Time Synchronization Server

IST-TR-95-28
November 13, 1995

Prepared For:
U.S. Army STRICOM
N61339-94-C-0024
CDRL A00D

Primary Author:
Michael D. Myjak

[illegible]

Contributing Authors:
Nicole L. Densmore
Christopher H. Esser

Reviewed By:
S. Sureshchandran

[illegible]

Institute for Simulation and Training • 3280 Progress Drive • Orlando, Florida 32826

University of Central Florida • Division of Sponsored Research

1. Introduction

This report is a deliverable item, CDRL A00D, under subtask 3.2.2.6, "Time Synchronization Server," of the U.S. Army Simulation Training and Instrumentation Command (STRICOM) contract #N61339-94-C-0024, entitled "TRIDIS: A TESTBED for Research in Distributed Interactive Simulation."

This report presents the results of the Institute for Simulation and Training's (IST) installation and integration of Network Time Protocol (NTP) and a Global Positioning System (GPS) receiver into the Distributed Interactive Simulation (DIS) TESTBED located in IST's laboratories. The report describes:

- Integration of a Universal Coordinated Time reference into the TESTBED
- Recommendations for implementing NTP in a DIS exercise
- Recommendations on the Integration of NTP using a GPS receiver in DIS exercises

1.1 Abbreviations and Acronyms

The following abbreviations and acronyms are used in the body of this paper:

CD	Carrier Detect
DIS	Distributed Interactive Simulation
DNS	Domain Name Server
GPS	Global Positioning System
IP	Internet Protocol
IST	Institute for Simulation and Training
LAN	Local Area Network
NFS	Network File Server
NTP	Network Time Protocol
PC	Personal Computer
PDU	Protocol Data Unit
PPC	Pulse Per Second
RFC	Request for Comments
STRICOM	U.S. Army Simulation Training and Instrumentation Command
TAIP	Trimble ASCII Interface Protocol
TRIDIS	TESTBED for Research in DIS
UTC	Universal Coordinated Time
WAN	Wide Area Network

1.2 Background

The Network Time Protocol offers a unique approach for achieving reliable time synchronization from a set of possibly unreliable remote time servers. Restated, NTP does not attempt to synchronize the system clocks of multiple hosts to each other. Rather, each server attempts to synchronize to Universal Coordinated Time (abbreviated by the British nomenclature UTC) using the best available source and available transmission paths. This is a fine point which is worth understanding.

A group of NTP-synchronized clocks may be close to each other in time, but this is not a consequence of the clocks in the group having been synchronized to each other. Rather each clock has been synchronized closely to UTC via the best source it has access to. Trying to synchronize a set of clocks to a set of servers whose time is not in mutual agreement may not result in any sort of useful synchronization of the clocks, even if you don't care about UTC [Mills92].

NTP operates on the premise that there is one true standard time, and that if several servers which claim synchronization to standard time disagree about what that time is, then one or more of them must be broken. There is no attempt to resolve differences more gracefully since the premise is that substantial differences cannot exist. In essence, NTP expects that the time being distributed from the root of the synchronization subnet will be derived from some external source of UTC (e.g., a GPS radio clock). This makes it somewhat inconvenient (though not impossible) to synchronize hosts together without a reliable source of UTC to synchronize them to. If your network is isolated and you cannot access other people's servers across the Internet, a GPS receiver may make a good investment.

In the NTP approach, time is distributed through a hierarchy of NTP servers, with each server adopting a "stratum." A stratum indicates how many levels away from an external source of UTC a given server is operating at. Stratum-1 servers, which are the most accurate, have access to some external time source, such as a GPS receiver, which receives its time signal from satellites containing cesium clocks. These servers explicitly provide a standard time service. A stratum-2 server is a time server which is currently obtaining time from a stratum-1 server. A stratum-3 server gets its time from a stratum-2 server, a stratum-4 server from a stratum-3 server, and so on. To avoid long-lived synchronization loops, the number of strata is limited to 15.

Each client in the synchronization subnet (which may also be a server for other stratum clients) chooses exactly one of the available time servers to synchronize to. It is thus possible to construct a synchronization subnet where each server has exactly one source of lower stratum time to synchronize to.

This simple hierarchical tree structure is not an optimal configuration. NTP operates under another premise. The premise that NTP operates under is that each stratum server's time should be viewed with a certain amount of distrust. This is in part due to the drift that is experienced by real clocks. Drift is a variation in skew, which is the frequency difference between two clocks. One of the things that the NTP daemon does when it is first started is to compute the error in the intrinsic frequency, "the drift", of the clock of the computer it is running on. It usually takes a day or so of continual operation in order to compute a good estimate of the drift and a good estimate is needed in order to synchronize the clock closely with the server. Once the initial drift value is computed, it will change slightly during the course of continued operation. The "drift file" is a file designated by the NTP daemon where the current value of the frequency error may be stored. Then if the time server is stopped and restarted, it can reinitialize itself to the previous "drift file" estimate instead of taking an entire day to resynchronize[Mills93].

NTP has access to several sources of lower stratum time (at least three) where it can then apply an agreement algorithm to detect insanity on the part of any one of these. NTP declares a clock to be insane when the data received from that clock is invalid or inconsistent. Normally, when all servers are in agreement, NTP will choose the best server, where "best" is defined in terms of lowest stratum source, the closest server (in terms of network latency) and claimed precision, along with several other considerations.

The implication on server selection is that while one should aim to provide each client with three or more sources of lower stratum time, several of these will only be providing backup service. Some servers may be of lesser quality than others in terms of network delay, precision, and stratum (i.e. a same-stratum peer which receives time from lower stratum sources the local server doesn't access directly can also provide good backup service).

2. Research Goals

The goal of this research project was to implement the Network Time Protocol in the TRIDIS TESTBED. NTP was selected as the method for distributing time in the TESTBED because NTP is an approved Internet Standard protocol (NTP version 2) used to synchronize computer clocks to UTC [RFC 1305]. System clocks which are synchronized to UTC are a fundamental necessity for DIS simulation applications operating in absolute time mode. In version 3, NTP was corrected to always be monotonically increasing (never allows time to run backwards). NTP is continuing to undergo developmental changes to improve its accuracy and to add encryption services (NTP v3.4x). NTP version 3 is currently designated as an experimental protocol with recommended use.

UTC is generally recognized as the authoritative time source. The hypothesis is that a common reference for time, such as UTC, would be beneficial for distributed interactive simulations. The theory is that a set of system clocks synchronized to UTC would eliminate the effects of multiple time zones, network latencies, and altercations with daylight savings time from geographically distributed simulation applications.

As documented in the Standard for Distributed Interactive Simulation [IEEE 1278], a simulation application's system clock must be synchronized to UTC for a DIS exercise to be able to use absolute timestamps. If all nodes in a concurrent DIS exercise use a common, precise reference for time with very high precision (such as a GPS receiver), then simulation messages called protocol data units would be provided with a more accurate timestamp

[Paterson, 95]. A more accurate timestamp would increase the accuracy of the dead reckoning algorithms and therefore improve the overall performance of the simulation.

3. Approach

In order to integrate the latest version of NTP into the TRIDIS TESTBED, the following things needed to occur:

- An accurate time source needed to be identified and procured
- A host system (or systems) needed to be identified to become the authoritative time source or metronome for the TRIDIS TESTBED
- The latest version of NTP needed to be acquired and installed on the identified system
- The NTP server needed to be connected to the identified time source
- A set of experiments needed to be devised to determine the stability of this setup

3.1 Investigate/Acquire Time Signal Sources

The approach taken by the TRIDIS team was to identify the accuracy, implementational cost effectiveness, and ability to achieve time synchronization (of simulation system clocks) for a variety of time sources. A Trimble GPS receiver was purchased by the TRIDIS team to be shared by the Network Time Protocol and another TRIDIS task. At the same time, results of the 10th DIS workshop were published. Dr. Amnon Katz at the University of Alabama had determined that GPS receivers were quite reliable for absolute time sources in distributed simulations. Dr. Katz stated that "the [a] GPS clock maintains an accuracy [of] about three orders of magnitude better, at no added cost..." than any other available time source [Katz, 94a].

3.2 Identify Host Systems

The next step consisted of identifying a host computer to become the authoritative time server for the TRIDIS laboratory. Originally, the Sun Sparcstation-10 on loan from Loral, had been identified as the time server. This workstation was an excellent choice because the original development of the NTP protocol had occurred on a Sun Sparcstation running Sun-OS.

However, a delay was incurred by the TRIDIS team when Encore Systems decided to reclaim their System 91 file server. With the removal of the Encore NFS file system, the previously selected NTP server, the Loral Sparcstation-10, was drafted into service as the new Network File Server (NFS). Additionally, Domain Name Server (DNS) operations were also implemented on this server.

Using the Loral Sparcstation-10 to provide NFS and DNS services meant that the machine was being moved into "production" status. Production machines typically require that maintenance periods and experimental processes be performed during scheduled down-times and on other machines, respectively. This created two problems. First, NTP experiments required that the selected server be periodically rebooted to reset the clock process or interface. This could only be performed during periods of scheduled down time. This could not be scheduled to occur during normal working days and that left no time to perform the experiments. Second, since several DIS projects were now reliant on this machine for multiple services, the CPU process time necessary to devote to NTP would be in jeopardy. The team believed that the possibility existed for NTP's statistical evaluation program to become confused and possibly distort the results of any passive or active experiments. Furthermore, when it came time to experiment with NTP and GPS, current NFS file users might complain if their server began to toggle between "up" and "down" states. The conclusion of the team was that the Loral Sparcstation-10 could no longer be considered for the NTP stratum time server in the TRIDIS laboratory.

The next likely choice for an NTP time server was another Sun machine known as "Jaws." Jaws was a Sun Sparcstation-2 Classic, a lower performance machine which when unloaded, should have been capable of managing the NTP process. However, when the NTP design team was ready to use this DIS Server, they were informed that it was now designated as a standby machine for IST's primary server, "Xeno" and was not available for performing the experiments on NTP. The project was put on hold until a suitable machine could be identified which could run the NTP process and integrate with the Trimble GPS receiver.

Another suggestion for an NTP stratum server machine was a Personal Computer (IBM 80386-PC) running the latest version of LINUX (1.3.9). The TRIDIS team accepted this solution and began installing the LINUX operating system on the new hardware. LINUX required a minimum of 130 megabytes of disk space and 16 megabytes of random access memory.

Several problems were encountered when trying to incorporate NTP and the Trimble GPS receiver with the LINUX operating system. The team took nearly two months to completely resolve the issues. In the end, NTP could run on LINUX and the GPS receiver worked on the PC, but the combination of the two did not work together. After months of electronic mail correspondence with the original authors of LINUX, NTP, and the LINUX NTP development team, it was determined that a new UNIX kernel would need to be built. The NTP process required the use of several operating system level functions in order to communicate with a Trimble GPS receiver and to set and test the PC's system clock. What was missing from LINUX was the "line discipline" necessary to communicate with the Trimble GPS receiver.

Briefly, each GPS receiver provides some sort of serial interface to communicate its data. However, standards and definitions have not yet been universally adopted for GPS communication streams. Each vendor therefore, creates a unique interface definition for transmitting its serial data. In the case of the Trimble GPS receiver, data is transmitted using 10 byte records. The NTP process timestamps the GPS data upon arrival. The NTP process uses this arrival time in its delay calculations prior to updating the system clock.

Before LINUX would be able to communicate with a Trimble GPS receiver, the specific serial line discipline function would have to be developed and integrated into a new LINUX kernel. This would require 6 new internal system calls to be developed for the LINUX operating system. This level of effort was beyond the scope of this TRIDIS task.

In searching for a solution for the LINUX problem it was recommended (several times, in fact) by the LINUX NTP development team that the TRIDIS team migrate stratum time server services over to a Sun workstation running Sun-OS version 4.1.3. Their reasoning was that the Sun was the host machine on which NTP was developed, a known Trimble line discipline existed, and they had no idea when they might get the LINUX NTP software or the Trimble line discipline operational. The LINUX NTP development team said that other GPS receivers had been connected to NTP running LINUX, but the code base within LINUX to support the Trimble GPS receiver line discipline was not going to be available in the near future. The TRIDIS team then decided to abandon the LINUX/PC approach entirely and began looking for another host to operate as a time server.

Two weeks later a suitable machine was located. A Sun Sparcstation-2 running the Sun-OS 4.1.3 was acquired from the Eagle project. The NTP software was installed on this machine and the Trimble GPS receiver was connected. Stratum-1 time synchronization experiments could now begin.

3.3 Integrate NTP and Signal Source into the TESTBED

Prior to the change in allocation of the Loral Sparcstation-10, and prior to receiving the GPS receiver, the team had installed NTP (version 3.4n) on the Sparcstation-10. The first experiment was run using this configuration. The *xntpd* (read as "experimental NTP daemon") was installed and setup to operate as a stratum-2 time server. The NTP configuration file was defined and three NTP stratum servers were entered. These were eagle.tamu.edu at Texas A&M University, black-ice.cc.vt.edu at the University of Vermont, and clock.llnl.gov at Lawrence Livermore National Laboratory. The experimental NTP installation went smoothly and an almost immediate connection to a Stratum-1 time server (at Texas A&M University) was made. Using the TRIDIS data logger, the NTP packets were detected on the network at the rate of one every few seconds.

The *ntpdate* function in debug mode was used to display the synchronization error of the *xntpd* program. Over several months of testing, we were able to determine that the accuracy of the resident drift file was quite good. Time readings from *ntptrace* indicated that the Stratum-2 time server was accurate to 156 microseconds after just 10 days of continual operation. Another iteration of the experiment showed that the server was synchronized to 86 microseconds after 11 days.

After the preliminary installation and testing of the NTP software on the Sparcstation-10, it was necessary to integrate NTP with an authoritative time source. From the NTP documentation and other available literature, it appeared that a GPS receiver would provide the best source for time (UTC). However, locating a GPS receiver which would work with our specific hardware and software environment turned out to be another problem, and perhaps one of the more important lessons learned.

3.4 Integrate NTP and GPS Signal Source into the TESTBED

When the Trimble GPS receiver became available for NTP experiments, we moved the receiver into the laboratory. Immediately we found that the built-in antenna was unable to receive satellite data through the ceiling of the laboratory. We determined that an external roof-mounted antenna was needed to solve the problem. A new antenna was ordered from Trimble and arrived two months later.

When the antenna arrived, another delay was encountered when concern was voiced about an aerial connection to a server located on the IST Local Area Network (LAN). The connection was initially refused because it was felt that the GPS antenna might act as a lightning rod and thus endanger the server and other LAN-based equipment. It took approximately one month to locate a replacement machine to act as the NTP host.

With the GPS receiver installed, a suitable source for UTC was also now available. All that remained was to integrate NTP with the GPS receiver output. The integrated solution would provide a stratum-1 time server for the TESTBED. First, configuration and testing of the GPS receiver was conducted using the Trimble starter kit and a DOS-compatible PC. This permitted the team to configure the receiver for serial output. Specifically, the serial output needed to be set in the Trimble line discipline mode.

NTP requires Trimble GPS receivers to communicate with *xntpd* using a predefined line discipline. To support the Trimble line discipline, specific (but relatively minor) kernel modifications needed to be made to the SUN 4.1.3 operating system. These modifications were documented in the NTP software and are also reprinted in Appendix B.

After the kernel modifications were complete, the Trimble GPS receiver was configured for serial transmission. With the GPS receiver functioning properly, the NTP daemon was started and within approximately 15 minutes of operation, the NTP process had synchronized to the GPS receiver with a stated accuracy of 2 microseconds. The TRIDIS team now had a stratum-1 time server.

4. Experiments

Two experiments were conducted to determine the effects of NTP time stabilization on networked computer systems. The first test involved the use of a Stratum-2 time server connected across a wide area interface to a stratum-1 server (clock.llnl.gov). The second experiment involved a stratum-1 server that was directly connected to the TESTBED.

4.1 Designing the Experiments

The following experiments were devised to determine the utility, effectiveness, reliability and precision of the NTP/GPS application. Four experiments were proposed:

- In the first experiment the system was monitored under normal conditions with no problems introduced to accurately determine the effects of NTP's built-in time stabilization.
- The second experiment was designed to determine what problems, if any, were encountered when the GPS receiver lost power momentarily.
- In the third experiment, the NTP software was reset during normal operation to determine if problems were induced.
- In the fourth experiment power was removed from the network server to determine what effects that had on the time synchronization system.

The built-in commands *ntptrace* , *ntpq*, and *ntpdate* (in debug mode) were used to verify the accuracy of the local host's time. Errors in time were measured as floating point offsets in microseconds from the authoritative time server's clock. For the first experiment, our stratum-2 server's reference clock was connected via the Wide Area Network (WAN). For the second experiment, our stratum-1 server was directly connected to an authoritative clock, GPS. The *ntptrace* command reports on the accuracy of synchronization of the given time server with respect to its parent server or reference clock.

4.2 Conduct The Experiments

All experiments were conducted in the TRIDIS laboratory at IST. Experiments were conducted using stand-alone machines. This eliminated the possibility of another program or project interfering with the NTP process. Timing analysis was performed on an IBM PC using Microsoft Excel.

4.3 The Hypothesis

The NTP GPS time trial experiment was designed to test the resynchronization ability of a stratum-1 time server. This is the time necessary for the NTP agent *xntpd* to resynchronize to the GPS clock. The resynchronization time for the stratum-1 server is similar to the resynchronization time of lower stratum servers without any network latency or propagational delay. This experiment set the reliability for a single LAN implementation of NTP and GPS. The theory behind the experiment was constructed given the following three assumptions.

First, in order to support a geographically distributed simulation exercise using absolute timestamps, each LAN in the exercise would require a connection to the same time source (i.e. UTC). Second, while it is generally recognized that other stratum time servers can and will be used in a simulation exercise, the most accurate time source available (that with the least error) would be a stratum-1 server. Third, in order to synchronize simulation application hosts, an appropriate amount of "up time" would be needed before and during a simulation exercise to stabilize the drift file.

The first assumption, that at least one stratum-1 time server would be required, is a given. A stratum-1 time server is defined as any device designed to serve time which is directly connected to an authoritative time source. Preferable simulation implementation solutions may require additional stratum-1 time servers for each exercise, but we felt that the normal implementational case would be one stratum-1 server for each LAN. Each simulation application would then synchronize its clock to the clock of its peer stratum server. This is also the operational theory behind the use of NTP and DIS.

The second assumption, that other stratum time servers can and will be used in a simulation exercise, is probably quite true because of the relatively low cost of implementing a stratum-2 (or lower) time server. The software is freely available on the Internet and operates on most common platforms in use today. Furthermore, each stratum server is capable of maintaining accurate time during periods when it cannot directly connect to its stratum peer. NTP accomplishes this by continuing to update the host's system clock using the data stored in the NTP drift file.

The third assumption, that a certain amount of "up time" would be needed to stabilize NTP would not be a problem. NTP operates by making small incremental adjustments to the clock increment value and stores this data in the NTP drift file. An indeterminate amount of operation "up time" would be needed in order to stabilize the drift file to some level of accuracy. When the NTP client loses connectivity to its metronome, the data in the drift file is still used to compensate for variances in the resident system clock. When synchronization is re-established, the *xntpd* process continues where it left off. When multiple stratum peers are available, the NTP daemon selects the most accurate time source available.

5. Results

Two experiments were performed. The first experiment was designed to determine the time needed by a stratum-2 (or lower) server to generate a drift file that would maintain the system clock to an accuracy of 1 millisecond. The second experiment was designed to determine how long, on average, a stratum-1 (system peer) would take to resynchronize to UTC after losing connectivity to its GPS receiver.

5.1 Experiment #1

The first experiment involved synchronizing stratum-2 time servers across the Internet. This experiment was important because the simulation community needed an inexpensive and relatively accurate method for creating absolute time stamps. Relative time stamps in DIS exercises have been so badly misinterpreted that they have become (to a greater or lesser degree) a large source of internal error in real time simulations.

Demonstrating that NTP stratum servers could provide an inexpensive source of accurate time to simulation testbeds was the result of this experiment. A Sun Sparcstation-10 was installed with NTP and allowed to settle for two days. At the end of the second day, *ntptrace* was run to indicate the network latency between the local host, and its system peer. The *ntpdate* function was used to indicate the update offset and that showed that the system was synchronized to UTC by just under 1 millisecond. The server was allowed to operate for 10 more days. After the tenth day of continual operation, the stratum-2 time server was synchronized to UTC within 156 microseconds.

This series of NTP experiments were run three consecutive times. At each iteration, the NTP drift file was cleared and the server was rebooted. Each iteration showed that the server was synchronized to 1 millisecond in two days of continual operation. Similar results were achieved after 10 days of operation. During the following months, the server was occasionally reset for system upgrades and software installations. The *xntpd* process never created an entry in the system log indicating a process failure. The program was quite stable and continued to provide consistent results.

5.2 Experiment #2

The second experiment involved testing the resynchronization ability of an NTP stratum server. A stratum-1 time server was selected for this experiment for two reasons: 1) a stratum-1 time server would produce the most accurate results; and 2) all NTP stratum servers operate essentially the same way, implying that resynchronization times would be similar for other lower stratum servers in the network.

Experiment #1 showed that stratum servers can maintain their sanity over long periods of time. When a time server loses time synchronization, its clock is said to be unreliable and is denoted as insane. NTP attempts to circumvent this occurrence by recommending that several (preferably lower level) stratum servers be specified within an NTP configuration file. However, the TRIDIS team believed that many of the simulation administrators would only establish a single clock. They might read this report, for example, and assume that all they need is a GPS receiver and the NTP installation and they are ready to operate using accurate, absolute timestamps. Appropriate backup time servers need to be established to provide fault tolerant NTP distribution services.

If a simulation team purchased a single GPS receiver and built a stratum-1 server, the most likely problem to befall them would be the loss of their time server. In fact, any team which built stratum-2 (or lower) time servers would potentially run into the same or similar problem. So the TRIDIS team wanted to know how long it would take for a stratum server to regain its sanity.

What was not well understood was how long it would take for an insane clock to regain its sanity after resynchronization. This is precisely the fear that a simulation manager might have if he had to suddenly replace his GPS receiver or reboot his stratum time server prior to starting a simulation exercise. The team set out to determine the duration of this critical period of time.

Experimental results were obtained for the resynchronization of an NTP stratum-1 time server and are shown in Table 1. As described above, a Sun Sparcstation-2 running Sun OS 4.1.3 was used to conduct this experiment. The latest version of NTP, version 3.4x, was installed. Next, kernel modifications were required and performed as outlined in the appendices. The Trimble GPS receiver was then connected to the *xntpd* process and allowed to settle for a period of 2 weeks. This insured that the NTP drift file was sufficiently accurate to maintain time on the server when the GPS receiver was lost.

The resynchronization table shows that on average, the NTP stratum-1 server would obtain system peer status in 12 minutes and 54 seconds. This was the average time obtained after thirty-two time trials were attempted. In accordance with standard statistical practice the data in the table does not contain the longest (39 minutes) and the shortest (12 seconds) time trials because they deviated from the norm. They are documented here for completeness.

	Start	Reachable (Insane)	Sys.peer (Sane)	Time to Reachable	Time To Sys Peer	Time to Synchroniz
1	14:20:22	14:24:07	14:32:09	0:03:45	0:08:02	0:11:47
2	9:33:16	9:36:28	9:45:06	0:03:12	0:08:38	0:11:50
3	10:09:32	10:12:42	10:22:20	0:03:10	0:09:38	0:12:48
4	10:22:20	10:25:30	10:35:08	0:03:10	0:09:38	0:12:48
5	10:36:12	10:39:24	10:46:52	0:03:12	0:07:28	0:10:40
6	10:26:48	10:30:38	10:39:36	0:03:50	0:08:58	0:12:48
7	10:23:32	10:26:44	10:34:36	0:03:12	0:07:52	0:11:04
8	11:55:20	11:58:32	12:07:04	0:03:12	0:08:32	0:11:44
9	10:46:52	10:50:04	10:59:40	0:03:12	0:09:36	0:12:48
10	11:47:26	11:53:21	11:56:43	0:05:55	0:03:22	0:09:17
11	12:57:12	13:00:24	13:10:00	0:03:12	0:09:36	0:12:48
12	11:39:08	11:42:20	11:49:48	0:03:12	0:07:28	0:10:40
13	11:49:48	11:53:00	12:02:36	0:03:12	0:09:36	0:12:48
14	12:45:45	12:51:32	12:59:43	0:05:47	0:08:11	0:13:58
15	10:59:40	11:02:52	11:13:32	0:03:12	0:10:40	0:13:52
16	12:15:22	12:18:34	12:28:10	0:03:12	0:09:36	0:12:48
17	12:51:38	12:54:50	13:05:30	0:03:12	0:10:40	0:13:52
18	13:17:14	13:20:28	13:30:02	0:03:14	0:09:34	0:12:48
19	12:44:26	12:49:44	12:57:12	0:05:18	0:07:28	0:12:46
20	13:48:10	13:51:22	14:07:22	0:03:12	0:16:00	0:19:12
21	13:05:30	13:05:42	13:17:14	0:00:12	0:11:32	0:11:44
22	10:47:56	10:57:03	11:07:08	0:09:07	0:10:05	0:19:12
23	11:07:28	11:10:20	11:19:56	0:02:52	0:09:36	0:12:28
24	14:09:48	14:13:48	14:24:28	0:04:00	0:10:40	0:14:40
25	10:40:40	10:43:52	10:52:24	0:03:12	0:08:32	0:11:44
26	14:47:56	14:51:06	14:59:40	0:03:10	0:08:34	0:11:44
27	15:24:12	15:28:28	15:41:16	0:04:16	0:12:48	0:17:04
28	13:32:10	13:35:22	13:44:58	0:03:12	0:09:36	0:12:48
29	13:11:04	13:14:10	13:21:44	0:03:06	0:07:34	0:10:40
30	13:00:12	13:02:24	13:12:08	0:02:12	0:09:44	0:11:56
Averages				**0:03:36**	**0:09:18**	**0:12:54**

Table 1: GPS Synchronization Time Trials

In Table 1, there are three columns labeled Start, Reachable, Sys Peer. The column labeled Start is the time of day that this particular experimental run was started. Start indicates the time of day at which the GPS receiver was reconnected to its power source. Power (110 VAC and battery backup) was removed from the GPS receiver for approximately five minutes. The five minute duration was recommended by Trimble to insure that all memory had cleared and that the GPS receiver would have to recalibrate, locate and lock on to the required number of GPS satellites. When the GPS power was restored, the time was noted as the start time.

The second column marked Reachable, is the time at the GPS receiver began transmitting accurate time. The NTP daemon *xntpd* considers this time to be unreliable because it has received one (or too few) clock updates. The daemon will display this time but will denote it as being insane, meaning unreliable. Reachable time might be significant for DIS field instrument devices or simulation hosts which carry on-board UTC clock generators. If a GPS receiver was used as a source clock in a simulation host, and the receiver lost power or possibly satellite lock-on, then the difference between start time and reachable time would be the time that the GPS receiver was unable to provide time stamp information to the simulation application. On average, the Trimble GPS receiver was able to produce UTC after 3 minutes and 36 seconds.

The third column is marked Sys Peer and represents the time that the *xntpd* daemon needed to "believe" or accept the GPS receiver clock time as valid. After *xntpd* receives several (10-13) consistent clock time readings, as determined by its own system clock and the drift file offsets, *xntpd* denotes the time received from that server as sane. The *xntpd* process denotes this time as sane by using a statistical method of calibration and averaging. The average time taken by *xntpd* to lock on to the receiver and believe it as being a sane, accurate time source was 9 minutes and 18 seconds.

6. Conclusions

As stated in the NTP documentation, the NTP process *xntpd* can synchronize the system's local time using multiple reference sources. Sometimes these time sources will disagree with one another due to errors in their clocks and latency delays which are unaccounted for. NTP will determine which time source, listed in its configuration file, is the closest (in terms of latency, not distance) and most accurate. NTP will then synchronize to that time source. As a client, *xntpd* will manage the local system clock and maintain synchronization to its system peer. If the system peer is synchronized to UTC, then so will the NTP client.

Stratum servers 2-15 will likewise maintain synchronization to their respective system peers prior to becoming stratum servers for a network. Since the same process, *xntpd*, is used by stratum server and NTP client alike, it is reasonable to assume that both will operate in a similar fashion and maintain local time for the respective host. It can be assumed that a network clock will always be available given access to the Internet. If the NTP reachable time is zero, the remaining time to authentication and sanity will be on the order of 10 minutes.

NTP client software can easily be integrated into simulation applications by installing it on the simulation host computer provided the right GPS clock and computer platform are chosen. Many combinations of GPS receiver and UNIX operating system are available. Note that not every GPS receiver can interoperate with every UNIX platform. Special care must be taken to select an appropriate combination. NTP can be transmitted using broadcast or multicast communications, and can be encrypted for secure sites. Local clock management occurs at the rate of once per second using a locally stored drift file. NTP client updates from the server are configurable and by default occur at 20 minute intervals. During an update, the NTP server will send a short burst of packets to the client to calculate the average latency delay of the network. When the last packet is received, it is time stamped and added to the latency delay to compute a new synchronized time. The local clock is then updated accordingly.

NTP stratum servers process time by accepting NTP packets from multiple time sources. Stratum servers use these additional references of "current" time to gauge which server is currently the most accurate. The *xntpd* process continually checks the validity of one time server against another in order to produce a stable, local reference time. The server with the most accurate time and stable delay is used to compute new stratum server time in much the same way that NTP client time is computed.

7. Recommendations

The TRIDIS team had to select hardware for the experiments which was compatible with the Trimble GPS receiver. Indeed, NTP runs on almost every UNIX platform and GPS receivers are available from many different vendors. However, because GPS serial line disciplines and standards are not yet implemented, an NTP solution does not yet exist for every combination of GPS receiver and UNIX platform. The team found that only some GPS receivers will communicate with some NTP implementations on some UNIX platforms. Which GPS receivers work with which NTP/OS combinations should be researched thoroughly prior to procuring either platform or receiver.

According to the DIS standard, in order for a DIS exercise to be able to use absolute timestamps, a simulation application s system clock must be synchronized to UTC. What is missing from this standard is the level of precision or accuracy by which simulation hosts require. The network time protocol has been demonstrated as being adequate for maintaining the system clocks of most simulation hosts participating in a simulation exercise. What needs to be established in the DIS standard is a recommended level of synchronization accuracy.

The *xntpd* daemon for stratum-2 through stratum-15 servers receive time from multiple, preferably lower stratum servers. However, only one time source can be the system s peer an any given point in time. Each stratum server determines its peer by statistically computing

computing both accuracy and latency delay. The system peer selected is the stratum time server deemed best by each NTP client at that moment. DIS simulations which use NTP to distribute time should configure NTP to reference multiple time servers. The *xntpd* process will then choose the most accurate server from the configuration. Further analysis should be conducted using flooded networks to determine if varying network latencies have an impact on time synchronization or time stabilization.

NTP clients (and servers) need some time to settle, possibly for a period of a week or two, in order to establish a unique drift file value for each *xntpd* host. The time required to settle the data contained within the NTP drift file will vary for each NTP client and therefore cannot be empirically measured. This is just as important for stratum-1 time servers as it is for stratum-2 through stratum-15 servers. All stratum servers need time to create an accurate drift file for the local system clock. The time allowed will be different for each platform and will depend on the degree of accuracy required. Properly configured and settled stratum servers can maintain microsecond clock integrity on the order of hours or even days. Simulation exercise managers should be aware of this requirement.

According to the DIS standard, simulation hosts must be synchronized to UTC before they can use absolute timestamps in a simulation exercise. Therefore having a simple method for synchronizing host systems would be quite beneficial to the DIS community. Having a solution which does not require common geographical locations would also be beneficial. The TRIDIS team can recommend that distributed simulations, whether geographically localized or dispersed can use the Network Time Protocol to synchronize system hosts in support of absolute time based simulations. NTP servers make an acceptable metronome and an accurate time source, synchronizing simulation hosts to UTC, thus permitting the use of absolute time stamps within a simulation exercise.

Areas for further research identified by this work are:

Analysis of the effects on DIS traffic of NTP synchronization packets/sequences

Simulation synchronization errors using NTP in Wide Area Network (WAN) configurations

Definition of an accurate UTC timestamp for DIS Protocol Data Units (PDUs)

8. References

[Golner94] Golner, M. and E Pollak, 1994. The Application of Network Time Protocol To Implementing DIS Absolute Timestamp, Proceedings from the *11th DIS Workshop Sept. 26-30 1994*, IST Technical Report IST-CF-94-02, pp. 431-439.

[IEEE1278.1] IEEE 1278.1, 1995. "Standard for Distributed Interactive Simulation -- Application Protocols," Institute of Electrical and Electronics Engineers, Inc., 345 East 47th St., New York, NY 10017.

[Katz94] Katz, A. 1994. The Absolute Clock in the DIS Scheme, Proceedings of *10th DIS Workshop Mar. 14-18, 1994,* IST Technical Report IST-CF-94-01, pp. 1-4, 1994.

[Katz94a] Katz, A. 1994. "Dead Reckoning for Airplanes in Coordinated Flight," Proceedings of the 10th *DIS Workshop Mar.14-18,* 1994, IST Technical Report IST-CF-94-02, 5-13, 1994.

[Katz94b] Katz, A. 1994. Synchronization of Networked Simulators, Proceedings of *11th DIS Workshop Sept. 26-30 1994*, IST Technical Report IST-CF-94-02, pp. 81-87.

[Mills92] Mills, D. 1992. "Network Time Protocol (Version 3) Specification, Implementation and Analysis," Internet Architecture Board Network Working Group report RFC-1305, University of Delaware, 1992.

[Mills93] Mills, D. 1992. "Notes on Xntpd Configuration," NTP Version 3 Distribution Release Notes, University of Delaware, 1993.

[Paterson95] Paterson, D. 1995. Sychrony and Time Representation for Distributed Simulators, Proceedings of the *12th DIS Workshop Mar. 13-17, 1995,* IST Technical Report IST-CF-95-01, pp. 325-328.

[Saunders95] Saunders, R. 1995. It's About Time, It's About Space - Time and Space in DIS, Proceedings of the *12th DIS Workshop Mar. 13-17, 1995,* IST Technical Report IST-CF-95-01, pp. 63-66.

Appendix A Configuring and Testing the GPS Receiver:

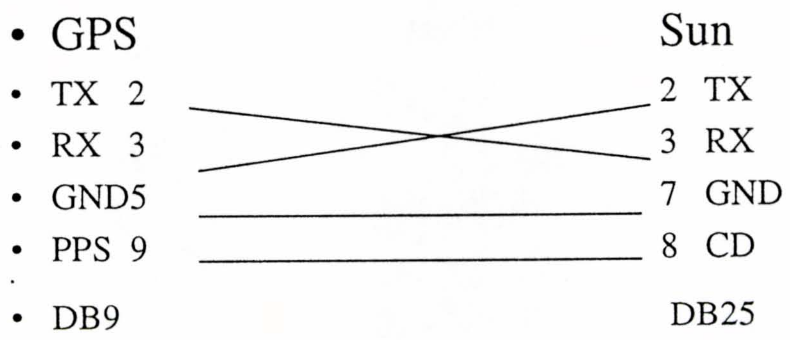
The easiest way to configure the Trimble GPS SV6 receiver is through the software that comes with the starter kit.

1. Connect a PC to the receiver's port 1 using a standard serial cable.

2. Run the cfgps program and put the receiver in TAIP mode.

3. Run the gpssk program and verify the receiver is picking up satellites.

Once you have verified the receiver is functioning properly, you can connect it to the Sun. This requires a special cable. The Trimble GPS receiver sends a PPS (pulse per second) signal on the RING line (pin 9), but *xntpd* expects the PPS to be on CD (carrier detect). This is because CD is a much higher priority interrupt, and we don't want to miss a pulse. See the diagram below for complete cable specifications.



Appendix B Kernel Modifications

B.1 Line discipline

Tty_clk.c contains a generic clock support line discipline. The terminal driver is actually run in raw mode, giving you an eight bit data path. Instead of delivering the data character-by-character, however, the line discipline collects characters until one of two magic characters (your current erase and kill characters) are received. A timestamp is then taken (by calling the function microtime()) and inserted in the input buffer after the magic character and the other information that is available for input by the application. Both select() and SIGIO are supported by the discipline.
To install the clock line discipline the following steps must be followed:

1. Copy tty_clk.c into /sys/sys
2. Edit /sys/sys/tty_conf.c. You will want to include some facsimile of the following lines:

```
#include "clk.h"

#if NCLK > 0

int     clkopen(), clkclose(), clkwrite(), clkinput(), clkioctl();

#endif


#if NCLK > 0

        { clkopen, clkclose, ttread, clkwrite, clkioctl,

          clkinput, nodev, nulldev, ttstart, nullmodem,

        /* 10-CLKLDISC */

          ttselect },

#else

        { nodev, nodev, nodev, nodev, nodev,

          nodev, nodev, nodev, nodev, nodev,

          nodev                                 },

#endif
```

3. Edit /sys/h/ioctl.h and include a line (somewhere near where other line disciplines are defined) like:

```
#define         CLKLDISC    10              /* clock line discipline */
```

The `10' should match what you used in /sys/sys/tty_conf.c.

4. Edit /sys/conf/files and add a line which looks like:

   sys/tty_clk.c   optional clk

5. Edit the configuration file for the machine you want to use the clock line discipline on to include the following: **pseudo-device   clk 4**

6. Run config, then make clean, then make depend, then make vmunix. Then reboot the new kernel.

B.2 Pulse Per Second Support

In order to use the 1pps pulse to correct both the UNIX clock offset and frequency, the time between when the external clock signals 1pps and when this module records the UNIX clock should be as small as possible and should be repeatable. AT&T STREAMS code makes this very difficult to achieve

The allocb/wput overhead adds about 700us of latency on a SS-1+ and the stream 'scheduler' combined with doing everything at a relatively low interrupt priority adds about +-400us of jitter. So, to get a high quality time stamp, this module uses STREAMS (only) to get the ioctl request.

The module gets the 1pps signal by avoiding STREAMS code entirely and inserting itself directly into the hardware interrupt service path for the 1pps pulse. It does this by stealing the modem control interrupt from the Sun's zs serial driver. This is not a generic streams module. It is designed to work with Sun's zs driver under SunOS 4. Expect that this will not work with other versions such as Solaris-2. Expect it not to run on anything but a Sun.

Note that the distribution was designed for a clock where '1 second' occurs on the leading edge of the pulse, where on the Trimble SV6 the second starts on the trailing edge. Therefore you should change the line:

```
if ((s0 & ZSRR0_CD) != 0) {
```

in ppsclock_intr to:

```
if ((s0 & ZSRR0_CD) == 0) {
```

to grab the time stamp on the trailing edge of the pulse.

B.3 Installation

These are some notes on the installation of the SunOS 4 PPS clock streams module.

This directory contains:

README	- this file
RELEASE	- version of this release
CHANGES	- description of differences between releases
magnavox.ps	- PostScript schematic (Magnavox rs422/rs232 converter)
b-and-b.ps	- PostScript schematic (B&B rs422/rs232 converter)
Makefile	- compilation rules
ppstest	- ppsclock test program (works with Magnavox MX4200 GPS you can probably use this as sample code illustrating the use of this streams module if you have some other kind of clock).
sys	- SunOS 4 kernel modules
sys/genassym	- genassym program for object-only (non-source) sites

B.4 Kernel Configuration

1. Copy sys/sundev/ppsclock.c to /sys/sundev.
2. Copy sys/sys/ppsclock.h to /sys/sys (and /usr/include/sys if it is separate directory).
3. If you want to use the fast microtime module, copy sys/sun4c/microtime.s to /sys/sun4c.
4. Use sys/sun/str_conf.c.patch to patch /sys/sun/str_conf.c.
   Alternately, manually install the following lines in the appropriate places:

```
#include "zs.h"
          [...]
          #if   NZS > 0
                  #include "sys/ppsclock.h"
                  extern struct streamtab ppsclockinfo;
          #endif
          [...]
          #if   NZS > 0
                  { PPSCLOCKSTR,  &ppsclockinfo },
          #endif
```

5. Use sun4c/conf/files.patch to patch /sys/sun4c/conf/files.

   Alternately, manually install the following line in the appropriate place:

```
sundev/ppsclock.c     optional zs device-driver
```

In addition, if you want to use the fast microtime module, use:

sun4c/conf/files.microtime.patch to patch /sys/sun4c/conf/files

Alternately, manually install the following line in the appropriate place:

```
sun4c/microtime.s     standard
```

If you are using SunOS 4.1.3 and wish to support the sun4m architecture, apply the corresponding patches from the sun4m tree.

6. If you want to use the fast microtime module, and have full SunOS 4 source, use /sys/os/kern_clock.c.patch to patch /sys/os/kern_clock.c. Alternately, edit /sys/os/kern_clock.c and bracket the code for uniqtime() with the following two lines:

```
#if !defined(sun4c) && !defined(sun4m)
#endif
```

If you DO NOT have full SunOS 4 source but still want to use the fast microtime, change the symbol table entry in for _uniqtime to _Uniqtime as follows:

```
hell 1 % cd /sys/sun4c/OBJ                    # or /sys/sun4m
hell 2 % mv kern_clock.o kern_clock.o.virgin
hell 2 % cp kern_clock.o.virgin kern_clock.o
hell 3 % chmod +w kern_clock.o
hell 5 % strings -o -a kern_clock.o | grep -w _uniqtime
7892 _uniqtime
hell 6 % adb -w kern_clock.o
?m 0 0xffffffff 0
0t7892?s
_dk_ndrive+0x12cc:            _uniqtime
.?x
_dk_ndrive+0x12cc:            5f75
.?w5f55
_dk_ndrive+0x12cc:            0x5f75        =       0x5f55
.?s
_dk_ndrive+0x12cc:            _Uniqtime
^D
```

7. If you do not have a full implementation of SunOS 4 source code, you are missing the source code to genassym. In order to correct this problem copy the "mini" genassym source from genassym/genassym.c to /sys/sun4c (and into /sys/sun4m if you have SunOS 4.1.3 and wish to support the Sun4m architecture).

   Next, use sun4c/conf/Makefile.src.patch to patch /sys/sun4c/conf/Makefile.src (and possibly /sys/sun4m/conf/Makefile.src).

Alternately, manually install the following rules in the prototype Makefile(s).

```
assym.s: ${MACHINE}/genassym.c
        ${CC} -E ${CPPOPTS} ${MACHINE}/genassym.c > ./a.out.c
        cc ${COPTS} ./a.out.c
        ./a.out >assym.s
        rm -f ./a.out ./a.out.c
```

8. Configure, build, and boot the new kernel.

Appendix C Compile and Install Xntpd3.4x

With the GPS clock connected and the kernel modifications done all that's left to do is compile and install the *xntpd*.

1. Make sure that you have all necessary tools for building executables. These tools include cc or gcc, make, awk, sed, tr, sh, grep, egrep and a few others. Not all of these tools exist in the standard distribution of today's UNIX versions (compilers are likely to be an extra product). For a successful installation all of these tools should be accessible via the current path.

   By default, if there is no Config.local, the system will generate one to support a local reference clock (which is run off the system clock).

   To set up for a radio clock, type "**make refconf**" and answer the questions about PLL, PPS and radio clock type. If this is the first use of the ref clock, don't forget to make suitable files in /dev/. For a Trimble SV6 make a symbolic link callerd refclock-0 in /dev/ to the port to which the clock is attached (i.e. ln -s /dev/ttya /dev/refclock-0).

   For custom tailored configuration, copying Config.local.dist to Config.local and editing Config.local to suit the local needs is necessary. Or use one of the makes listed above and then make the necessary modifications for the correct radio clock type. Config.local can also be used to override common settings like the AUTHDEFS= which is used to select very specific configurations. Please use this feature with care and don't be disappointed if it doesn't work the way you expect.

2. Type "make" to begin the compilation. Expect only a few or no warnings using cc and only a moderate level of warnings using gcc. Note on some UNIX platforms the use of gcc can result in quite a few complaints about system header files and type problems within *xntpd* code. This is usually the case when the OS header files are not up to ANSI standards or GCCISMs. There may, however, still be some inconsistencies in the code.

   Each time you change the configuration a script that interacts with your hardware and software will be run to build the actual configuration files. If the script fails, it will give you a list of machines it knows about. You can override the automatic choice by changing to the directory ../machines and typing "make makeconfig OS=<machine>", where <machine> is one of the file names in the ../machine directory.

   The shell script will attempt to find the gcc compiler and, if found, will use it instead of the cc compiler. You can override this automatic choice by changing to the directory ../machines and typing "make makeconfig COMP=<compiler>", where <compiler> is one of the file names in the ../compilers directory. This can be combined with the OS argument above.

The configuration step can be separately invoked by typing "make makeconfig".

Note that any reconfiguration will result in cleaning the old program and object files.

3. Assuming you have write permission on the destination directory, type "make install" to install the binaries in that directory. Presently, at the IST TESTBED, this includes the programs *xntpd* (the daemon), *xntpdc* (an xntpd-dependent query program), *ntpq* (a standard query program), *ntpdate* (an rdate replacement for boot time date setting and sloppy time keeping) and *xntpres* (a program which provides name resolver support for some *xntpd* configurations).

4. Create /etc/ntp.conf with support for a local clock and any server peers that looks something like this.

```
# /etc/ntp.conf
server 127.127.8.0  mode 137        # Support for local clock (127.127.type.unit)
                                    # type 8 = GENERIC
                                    # unit 0 = /dev/refclock-0
                                    # mode 137 = Mode 9 (Trimble SV6/TAIP) + Mode
128 (PPS)
# peer servers for redundancy
server 128.115.14.97                # clock.llnl.gov
server 128.173.14.71                # black-ice.cc.vt.edu
server 128.2.250.95                 # clock-1.cs.cmu.edu
server 165.91.72.27                 # eagle.tamu.edu

driftfile /etc/ntp.drif             # File to keep system clock drift value
```

5. Start *xntpd.*

```
/usr/local/bin/xntpd -c /etc/ntp.conf
```

To start *xntpd* at boot time insert the following lines in /etc/rc.local just after the network interfaces get started.

```
#
# Start xntpd
#
if [ -f /etc/ntp.conf ]; then
        echo "Starting NTP daemon...."
        /usr/local/bin/ntpdate -v (server1) (server2) (server....)
        sleep 5
        /usr/local/bin/xntpd -c /etc/ntp.conf
fi
```

0000070