The Berkeley UNIX® Time Synchronization Protocol

Riccardo Gusella, Stefano Zatti, and James M. Bloom

Computer Systems Research Group
Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA 94720

Introduction

The Time Synchronization Protocol (TSP) has been designed for specific use by the program *timed*, a local area network clock synchronizer for the UNIX 4.3BSD operating system. Timed is built on the DARPA UDP protocol [4] and is based on a master slave scheme.

TSP serves a dual purpose. First, it supports messages for the synchronization of the clocks of the various hosts in a local area network. Second, it supports messages for the election that occurs among slave time daemons when, for any reason, the master disappears. The synchronization mechanism and the election procedure employed by the program timed are described in other documents [1,2,3].

Briefly, the synchronization software, which works in a local area network, consists of a collection of *time daemons* (one per machine) and is based on a master-slave structure. The present implementation keeps processor clocks synchronized within 20 milliseconds. A *master time daemon* measures the time difference between the clock of the machine on which it is running and those of all other machines. The current implementation uses ICMP *Time Stamp Requests* [5] to measure the clock difference between machines. The master computes the *network time* as the average of the times provided by nonfaulty clocks.[1] It then sends to each *slave time daemon* the correction that should be performed on the clock of its machine. This process is repeated periodically. Since the correction is expressed as a time difference rather than an absolute time, transmission delays do not interfere with synchronization. When a machine comes up and joins the network, it starts a slave time daemon, which will ask the master for the correct time and will reset the machine's clock before any user activity can begin. The time daemons therefore maintain a single network time in spite of the drift of clocks away from each other.

Additionally, a time daemon on gateway machines may run as a *submaster*. A submaster time daemon functions as a slave on one network that already has a master and as master on other networks. In addition, a submaster is responsible for propagating broadcast packets from one network to the other.

To ensure that service provided is continuous and reliable, it is necessary to implement an election algorithm that will elect a new master should the machine running the current master crash, the master terminate (for example, because of a run-time error), or the network be partitioned. Under our algorithm, slaves are able to realize when the master has stopped functioning and to elect a new master from among themselves. It is important to note that since the failure of the master results only in a gradual divergence of clock values, the election need not occur immediately.

All the communication occurring among time daemons uses the TSP protocol. While some messages need not be sent in a reliable way, most communication in TSP requires reliability not provided by the

This work was sponsored by the Defense Advanced Research Projects Agency (DoD), monitored by the Naval Electronics Systems Command under contract No. N00039-84-C-0089, and by the Italian CSELT Corporation. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the Defense Research Projects Agency, of the US Government, or of CSELT.

[1] A clock is considered to be faulty when its value is more than a small specified interval apart from the majority of the clocks of the machines on the same network. See [1,2] for more details.

underlying protocol. Reliability is achieved by the use of acknowledgements, sequence numbers, and retransmission when message losses occur. When a message that requires acknowledgment is not acknowledged after multiple attempts, the time daemon that has sent the message will assume that the addressee is down. This document will not describe the details of how reliability is implemented, but will only point out when a message type requires a reliable transport mechanism.

The message format in TSP is the same for all message types; however, in some instances, one or more fields are not used. The next section describes the message format. The following sections describe in detail the different message types, their use and the contents of each field. NOTE: The message format is likely to change in future versions of timed.

Message Format

All fields are based upon 8-bit bytes. Fields should be sent in network byte order if they are more than one byte long. The structure of a TSP message is the following:

1) A one byte message type.

2) A one byte version number, specifying the protocol version which the message uses.

3) A two byte sequence number to be used for recognizing duplicate messages that occur when messages are retransmitted.

4) Eight bytes of packet specific data. This field contains two 4 byte time values, a one byte hop count, or may be unused depending on the type of the packet.

5) A zero-terminated string of up to 256 ASCII characters with the name of the machine sending the message.

The following charts describe the message types, show their fields, and explain their usages. For the purpose of the following discussion, a time daemon can be considered to be in one of three states: slave, master, or candidate for election to master. Also, the term *broadcast* refers to the sending of a message to all active time daemons.

Adjtime Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	**Version No.**	**Sequence No.**	
Seconds of Adjustment			
Microseconds of Adjustment			
Machine Name			
. . .			

Type: TSP_ADJTIME (1)

The master sends this message to a slave to communicate the difference between the clock of the slave and the network time the master has just computed. The slave will accordingly adjust the time of its machine. This message requires an acknowledgment.

Acknowledgment Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	**Version No.**	**Sequence No.**	
(unused)			
(unused)			
Machine Name			
. . .			

Type: TSP_ACK (2)

Both the master and the slaves use this message for acknowledgment only. It is used in several different contexts, for example in reply to an Adjtime message.

Master Request Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	**Version No.**	**Sequence No.**	
(unused)			
(unused)			
Machine Name			
. . .			

Type: TSP_MASTERREQ (3)

A newly-started time daemon broadcasts this message to locate a master. No other action is implied by this packet. It requires a Master Acknowledgment.

Master Acknowledgement

Byte 1	Byte 2	Byte 3	Byte 4
Type	**Version No.**	**Sequence No.**	
(unused)			
(unused)			
Machine Name			
. . .			

Type: TSP_MASTERACK (4)

The master sends this message to acknowledge the Master Request message and the Conflict Resolution Message.

Set Network Time Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	**Version No.**	**Sequence No.**	
Seconds of Time to Set			
Microseconds of Time to Set			
Machine Name			
. . .			

Type: TSP_SETTIME (5)

The master sends this message to slave time daemons to set their time. This packet is sent to newly started time daemons and when the network date is changed. It contains the master's time as an approximation of the network time. It requires an acknowledgment. The next synchronization round will eliminate the small time difference caused by the random delay in the communication channel.

Master Active Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	**Version No.**	**Sequence No.**	
(unused)			
(unused)			
Machine Name			
. . .			

Type: TSP_MASTERUP (6)

The master broadcasts this message to solicit the names of the active slaves. Slaves will reply with a Slave Active message.

Slave Active Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	**Version No.**	**Sequence No.**	
(unused)			
(unused)			
Machine Name			
. . .			

Type: TSP_SLAVEUP (7)

A slave sends this message to the master in answer to a Master Active message. This message is also sent when a new slave starts up to inform the master that it wants to be synchronized.

Master Candidature Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	**Version No.**	**Sequence No.**	
(unused)			
(unused)			
Machine Name			
. . .			

Type: TSP_ELECTION (8)

A slave eligible to become a master broadcasts this message when its election timer expires. The message declares that the slave wishes to become the new master.

Candidature Acceptance Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	**Version No.**	**Sequence No.**	
(unused)			
(unused)			
Machine Name			
. . .			

Type: TSP_ACCEPT (9)

A slave sends this message to accept the candidature of the time daemon that has broadcast an Election message. The candidate will add the slave's name to the list of machines that it will control should it become the master.

Candidature Rejection Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	**Version No.**	**Sequence No.**	
(unused)			
(unused)			
Machine Name			
. . .			

Type: TSP_REFUSE (10)

After a slave accepts the candidature of a time daemon, it will reply to any election messages from other slaves with this message. This rejects any candidature other than the first received.

Multiple Master Notification Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	**Version No.**	**Sequence No.**	
(unused)			
(unused)			
Machine Name			
. . .			

Type: TSP_CONFLICT (11)

When two or more masters reply to a Master Request message, the slave uses this message to inform one of them that more than one master exists.

Conflict Resolution Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	**Version No.**	**Sequence No.**	
(unused)			
(unused)			
Machine Name			
. . .			

Type: TSP_RESOLVE (12)

A master which has been informed of the existence of other masters broadcasts this message to determine who the other masters are.

Quit Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	**Version No.**	**Sequence No.**	
(unused)			
(unused)			
Machine Name			
. . .			

Type: TSP_QUIT (13)

This message is sent by the master in three different contexts: 1) to a candidate that broadcasts an Master Candidature message, 2) to another master when notified of its existence, 3) to another master if a loop is detected. In all cases, the recipient time daemon will become a slave. This message requires an acknowledgement.

Set Date Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	**Version No.**	**Sequence No.**	
Seconds of Time to Set			
Microseconds of Time to Set			
Machine Name			
. . .			

Type: TSP_SETDATE (22)

The program *date* (1) sends this message to the local time daemon when a super-user wants to set the network date. If the local time daemon is the master, it will set the date; if it is a slave, it will communicate the desired date to the master.

Set Date Request Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	**Version No.**	**Sequence No.**	
Seconds of Time to Set			
Microseconds of Time to Set			
Machine Name			
. . .			

Type: TSP_SETDATEREQ (23)

A slave that has received a Set Date message will communicate the desired date to the master using this message.

Set Date Acknowledgment Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	**Version No.**	**Sequence No.**	
(unused)			
(unused)			
Machine Name			
. . .			

Type: TSP_DATEACK (16)

The master sends this message to a slave in acknowledgment of a Set Date Request Message. The same message is sent by the local time daemon to the program *date(1)* to confirm that the network date has been set by the master.

Start Tracing Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
. . .			

Type: TSP_TRACEON (17)

The controlling program *timedc* sends this message to the local time daemon to start the recording in a system file of all messages received.

Stop Tracing Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
. . .			

Type: TSP_TRACEOFF (18)

Timedc sends this message to the local time daemon to stop the recording of messages received.

Master Site Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
. . .			

Type: TSP_MSITE (19)

Timedc sends this message to the local time daemon to find out where the master is running.

Remote Master Site Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	**Version No.**	**Sequence No.**	
(unused)			
(unused)			
Machine Name			
. . .			

Type: TSP_MSITEREQ (20)

A local time daemon broadcasts this message to find the location of the master. It then uses the Acknowledgement message to communicate this location to *timedc*.

Test Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	**Version No.**	**Sequence No.**	
(unused)			
(unused)			
Machine Name			
. . .			

Type: TSP_TEST (21)

For testing purposes, *timedc* sends this message to a slave to cause its election timer to expire. NOTE: *timed* is not normally compiled to support this.

Loop Detection Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	**Version No.**	**Sequence No.**	
Hop Count	**(unused)**		
(unused)			
Machine Name			
. . .			

Type: TSP_LOOP (24)

This packet is initiated by all masters occasionally to attempt to detect loops. All submasters forward this packet onto the networks over which they are master. If a master receives a packet it sent out initially, it knows that a loop exists and tries to correct the problem.

References

1. R. Gusella and S. Zatti, *TEMPO: A Network Time Controller for Distributed Berkeley UNIX System*, USENIX Summer Conference Proceedings, Salt Lake City, June 1984.

2. R. Gusella and S. Zatti, *Clock Synchronization in a Local Area Network*, University of California, Berkeley, Technical Report, *to appear*.

3. R. Gusella and S. Zatti, *An Election Algorithm for a Distributed Clock Synchronization Program*, University of California, Berkeley, CS Technical Report #275, Dec. 1985.

4. Postel, J., *User Datagram Protocol*, RFC 768. Network Information Center, SRI International, Menlo Park, California, August 1980.

5. Postel, J., *Internet Control Message Protocol*, RFC 792. Network Information Center, SRI International, Menlo Park, California, September 1981.