Timed Installation and Operation Guide

Riccardo Gusella, Stefano Zatti, James M. Bloom

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

Kirk Smith

Engineering Computer Network
Department of Electrical Engineering
Purdue University
West Lafayette, IN 47906

Introduction

The clock synchronization service for the UNIX 4.3BSD operating system is composed of a collection of time daemons (*timed*) running on the machines in a local area network. The algorithms implemented by the service is based on a master-slave scheme. The time daemons communicate with each other using the *Time Synchronization Protocol* (TSP) which is built on the DARPA UDP protocol and described in detail in [4].

A time daemon has a twofold function. First, it supports the synchronization of the clocks of the various hosts in a local area network. Second, it starts (or takes part in) 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]. The next paragraphs are a brief overview of how the time daemon works. This document is mainly concerned with the administrative and technical issues of running *timed* at a particular site.

A *master time daemon* measures the time differences between the clock of the machine on which it is running and those of all other 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 the accuracy of the 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 are able to maintain a single network time in spite of the drift of clocks away from each other. The present implementation keeps processor clocks synchronized within 20 milliseconds.

To ensure that the service provided is continuous and reliable, it is necessary to implement an election algorithm to elect a new master should the machine running the current master crash, the master

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 CSELT Corporation of Italy. 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 other machines [1,2].

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.

The machines that are gateways between distinct local area networks require particular care. A time daemon on such machines may act as a *submaster*. This artifact depends on the current inability of transmission protocols to broadcast a message on a network other than the one to which the broadcasting machine is connected. The submaster appears as a slave on one network, and as a master on one or more of the other networks to which it is connected.

A submaster classifies each network as one of three types. A *slave network* is a network on which the submaster acts as a slave. There can only be one slave network. A *master network* is a network on which the submaster acts as a master. An *ignored network* is any other network which already has a valid master. The submaster tries periodically to become master on an ignored network, but gives up immediately if a master already exists.

Guidelines

While the synchronization algorithm is quite general, the election one, requiring a broadcast mechanism, puts constraints on the kind of network on which time daemons can run. The time daemon will only work on networks with broadcast capability augmented with point-to-point links. Machines that are only connected to point-to-point, non-broadcast networks may not use the time daemon.

If we exclude submasters, there will normally be, at most, one master time daemon in a local area internetwork. During an election, only one of the slave time daemons will become the new master. However, because of the characteristics of its machine, a slave can be prevented from becoming the master. Therefore, a subset of machines must be designated as potential master time daemons. A master time daemon will require CPU resources proportional to the number of slaves, in general, more than a slave time daemon, so it may be advisable to limit master time daemons to machines with more powerful processors or lighter loads. Also, machines with inaccurate clocks should not be used as masters. This is a purely administrative decision: an organization may well allow all of its machines to run master time daemons.

At the administrative level, a time daemon on a machine with multiple network interfaces, may be told to ignore all but one network or to ignore one network. This is done with the *−n network* and *−i network* options respectively at start-up time. Typically, the time daemon would be instructed to ignore all but the networks belonging to the local administrative control.

There are some limitations to the current implementation of the time daemon. It is expected that these limitations will be removed in future releases. The constant NHOSTS in /usr/src/etc/timed/globals.h limits the maximum number of machines that may be directly controlled by one master time daemon. The current maximum is 29 (NHOSTS − 1). The constant must be changed and the program recompiled if a site wishes to run *timed* on a larger (inter)network.

In addition, there is a *pathological situation* to be avoided at all costs, that might occur when time daemons run on multiply-connected local area networks. In this case, as we have seen, time daemons running on gateway machines will be submasters and they will act on some of those networks as master time daemons. Consider machines A and B that are both gateways between networks X and Y. If time daemons were started on both A and B without constraints, it would be possible for submaster time daemon A to be a slave on network X and the master on network Y, while submaster time daemon B is a slave on network Y and the master on network X. This *loop* of master time daemons will not function properly or guarantee a unique time on both networks, and will cause the submasters to use large amounts of system resources in the form of network bandwidth and CPU time. In fact, this kind of *loop* can also be generated with more than two master time daemons, when several local area networks are interconnected.

Installation

In order to start the time daemon on a given machine, the following lines should be added to the *local daemons* section in the file */etc/rc.local*:

```
if [ -f /etc/timed ]; then
        /etc/timed flags & echo -n ' timed' >/dev/console
fi
```

In any case, they must appear after the network is configured via ifconfig(8).

Also, the file */etc/services* should contain the following line:

```
timed       525/udp         timeserver
```

The *flags* are:

-n network — to consider the named network.

-i network — to ignore the named network.

-t — to place tracing information in */usr/adm/timed.log*.

-M — to allow this time daemon to become a master. A time daemon run without this option will be forced in the state of slave during an election.

Daily Operation

Timedc(8) is used to control the operation of the time daemon. It may be used to:

- measure the differences between machines' clocks,
- find the location where the master *timed* is running,
- cause election timers on several machines to expire at the same time,
- enable or disable tracing of messages received by *timed*.

See the manual page on *timed* (8) and *timedc* (8) for more detailed information.

The *date(1)* command can be used to set the network date. In order to set the time on a single machine, the *-n* flag can be given to date(1).

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. R. Gusella and S. Zatti, *The Berkeley UNIX 4.3BSD Time Synchronization Protocol*, UNIX Programmer’s Manual, 4.3 Berkeley Software Distribution, Volume 2c.