

# Real World Time Source Failover with TimeKeeper

**FSMLabs, Inc. Copyright Finite State Machine Labs Inc.**  
2011 All rights reserved.

TimeKeeper is well known for providing highly accurate (sub-microsecond) time by synchronizing to a variety of time sources and driving that directly into the application. What this article will demonstrate is how this is done in practice with specific examples.

## 1 Background and basics

Let's start with the basics. TimeKeeper synchronizes local time to a variety of sources, and also acts as a server of that synchronized time. Some examples of sources include:

- NTP (versions 1-4)
- PTP/IEEE 1588 (versions 1 and 2)
- PCI/PCI-E GPS/PPS/IRIG cards

Prior to version 5.0, TimeKeeper could track one of the above sources - it could be an NTP or PTP client, or synchronize to a local bus card with a GPS/PPS/IRIG feed. It could also serve NTP or PTP from that provided time.

Starting with version 5.0, TimeKeeper can now track many time sources simultaneously, and sources can be prioritized. Need to be able to fall back on NTP if your PTP services go down? Track a PTP source as a 'primary' source and an NTP server (or a whole list of them) as a fallback. This document will demonstrate exactly how to do this.

Alternatively, for timing verification, TimeKeeper can be configured to point to all of your NTP and PTP servers, with a local PCI card as the primary source. TimeKeeper will crosscheck all of the sources against each other (including the primary) to ensure that none of them are diverging and providing bad time to clients. This not only provides timing security and audit proof, it also shows the quality of the various timing services available on the network.

## 2 TimeKeeper Setup

Now let's get to specifics - how this works in the real world. First, a simple example where TimeKeeper should track a public NTP server as a client. Here's all that is needed in TimeKeeper's `/etc/timekeeper.conf`:

Listing 1: NTP Client Configuration

```
1 SOURCE0() {
2     NTPSERVER=pool.ntp.org
3 }
```

Once saved, start/restart TimeKeeper:

Listing 2: Startup

```
1 bash /etc/init.d/timekeeper restart
```

That's it! Similarly, this is how to configure a PTP client:

Listing 3: PTP Client Configuration

```
1 SOURCE0() {
2     PTPDOMAIN=80
3     PTPCLIENTVERSION=2
4 }
```

Restart TimeKeeper as in the NTP example and you're done. Pointing to a local bus card follows the same form, and that will be demonstrated later.

It really is that easy - a couple lines of configuration identifying your clock source, and TimeKeeper will track it to the best of the ability of the hardware/-driver/distribution, automatically, to sub-microsecond levels. Since TimeKeeper also 'owns' all of the timing calls on the system, time is pushed directly to applications even faster than via the stock Linux methods allow, without any source code changes.

### 3 Tracking Multiple Sources

Pointing to a timing source is fine, as long as that source never goes down. For systems that rely on highly accurate time, all the time, TimeKeeper's multi-source tracking feature provides a way to identify a very secure, resilient, and auditable path to timing data.

For example, a trading system may need to track a Symmetricom S350 acting as PTP server on domain 40 but fall back to an NTP server at 10.10.30.45 if that PTP server is not available. Here's how to do it:

Listing 4: 2 Source Configuration

```
1 SOURCE0() {
2     PTPDOMAIN=40
3     PTPCLIENTVERSION=2
4 }
5 SOURCE1() {
6     NTPSERVER=10.10.30.45
7 }
```

TimeKeeper will follow SOURCE0 (PTP domain 40) automatically, making use of the best timestamping available (including hardware timestamps) while also maintaining a model of SOURCE1 (NTP). If the PTP server stops providing timing information, TimeKeeper will fall back to the NTP server automatically. Once the PTP source comes back, TimeKeeper will revert to the higher priority source without any interruption in service. This happens entirely transparently aside from log entries and SNMP traps. Applications do not need to be aware of the issue.

As it runs, TimeKeeper tracks information about declared sources, so it's easy to compare the accuracy of both the PTP and NTP server against any point in time.

Here's a similar example using a local bus card as the primary (highest priority) source with several PTP and NTP servers. This can also provide failover capabilities but is more useful as an aggregator - by reviewing the results, it's easy to demonstrate how accurate (or inaccurate) the available timing services are.

Listing 5: Aggregator Configuration

```
1 SOURCE0() {
2     PPSDEV=symmetricom
3 }
4 SOURCE1() {
```

```

5      PTPDOMAIN=40
6      PTPCLIENTVERSION=2
7      IFACE=eth1
8  }
9  SOURCE2() {
10     PTPDOMAIN=80
11     PTPCLIENTVERSION=2
12     IFACE=eth3
13 }
14 SOURCE3() {
15     NTPSERVER=10.10.30.45
16 }
17 SOURCE4() {
18     NTPSERVER=10.10.40.23
19 }

```

The local bus card (a Symmetricom device) will provide the time that TimeKeeper syncs to. At the same time, it also tracks a PTP server on domain 40 via `eth1`. We'll discuss specific interfaces more shortly. Additionally, a PTP server on domain 80 is tracked on `eth3`.

Many PTP users also have existing NTP infrastructure - so in this case TimeKeeper will also track and quantify how accurate these 2 NTP servers are.

With these features, from one location, you have a view of multiple timing sources on your network, in addition to the ability to seamlessly transition from one to another as needed.

## 4 Reviewing collected data

We've covered how to identify and track multiple timing sources - but how exactly do you make use of this information? TimeKeeper provides the data in a simple format, and provides tools to visualize it.

If the main goal is to apply redundancy to your network via multiple paths, protocols, and servers, TimeKeeper will handle the work for you, noting (via SNMP traps and log files) when a primary time source was lost and regained, and what it did in the meantime. It also provides a continual track of the various identified time sources over time.

If your main goal is to have TimeKeeper act as an aggregator/auditor, the clock source data will be what you want to track. Here's what TimeKeeper provides:

- `/var/log/timekeeper` - A general log that notes configuration data, options enabled, and significant events, like when different sources are selected.
- `/var/log/timekeeper_0.data` - Timing information about SOURCE0, including time samples over time and how it compares to the local synchronized time source. TimeKeeper documentation provides more detail on the fields available.
- `/var/log/timekeeper_0.discard` - For SOURCE0, this provides a list of time samples discarded over time due to network noise and other reasons.
- `/var/log/timekeeper_1.data` - Timing information for SOURCE1.
- `/var/log/timekeeper_1.discard` - Time samples discarded for SOURCE1.
- ... And so on for every source identified in TimeKeeper's configuration.

This provides a simple audit trail for analysis. Plotting tools are provided to easily visualize the behavior of each identified source - these tools are detailed in TimeKeeper's documentation. Since the log files are just text, they can also be collected together elsewhere, like an Excel spreadsheet.

Let's visualize how well TimeKeeper makes use of hardware timestamps on a Intel 82580 NIC. To ensure the accuracy of the hardware timestamp, multiple sources are identified. One source is the PTP server that will be tracked as the primary, and the other is a local serial device with a GPS/PPS input. Both sources are plotted together in order to crosscheck. First, the configuration:

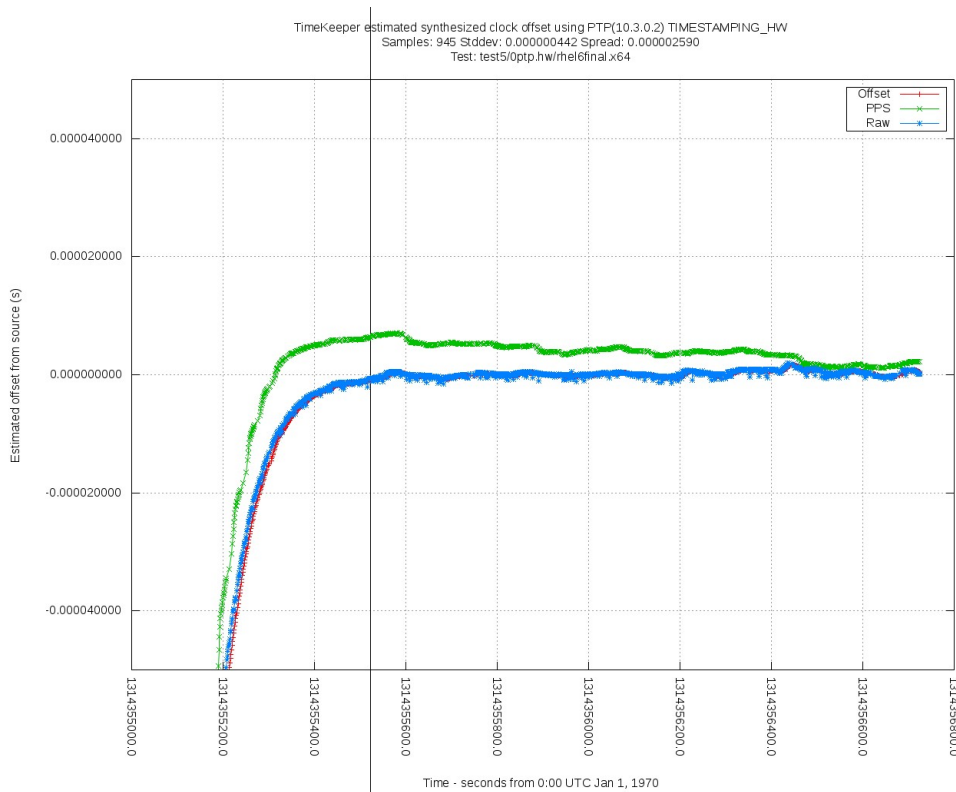
Listing 6: Hardware Timestamp with Reference PPS Configuration

```

1 SOURCE0() {
2     PTPDOMAIN=0
3     PTPCLIENTVERSION=2
4     IFACE=eth2
5 }
6 SOURCE1() {
7     PPSDEV=/dev/ttyS1
8 }

```

TimeKeeper visualizes the clock source results as follows:



Using multiple sources allows us to see exactly how well our PTP time source compares to the reference GPS data, and also how the clock on the NIC converges over time. TimeKeeper, while tracking multiple sources, takes in PTP data and serves local time to approximately 1 microsecond of the reference clock, which, coincidentally, is the limit of accuracy available with an off the shelf serial device.

## 4.1 Major and Minor Time

Here's an example of how flexible multiple sources can be. Consider this scenario - a PPS source is available to all systems that need highly accurate time, but the PPS is just that: a PPS. It does not have any time of day information. However, the network does have an NTP server available. This NTP server has time of day information but can't be trusted to be accurate even down to single digit milliseconds, let alone microseconds.

What's really needed here is a meld of the two timesources - the PPS can provide accurate sub-second information, and the NTP server can provide the time of day/month/year. TimeKeeper allows this by making use of multiple sources:

Listing 7: Merging Major and Minor Time

```
1 SOURCE0() {
2     PPSDEV=spectracom
3     MAJORTIME=SOURCE1
4 }
5 SOURCE1() {
6     NTPSERVER=10.3.8.10
7 }
```

A Spectracom local bus card provides the PPS in SOURCE0. SOURCE1 is the NTP server. To combine them together, SOURCE0 notes that the major time, like the time of day information, should come from SOURCE1. TimeKeeper then merges the major time from SOURCE1 into the minor time from SOURCE0 as a single accurate time source.

## 5 Serving data with TimeKeeper

Serving NTP or PTP with TimeKeeper is handled similarly to the above examples. In the same configuration file (`/etc/timekeeper.conf`), NTP and PTP can be enabled by adding:

Listing 8: Serve PTP and NTP

```
1 SERVENTP=1
2 SERVEPTP0() {
3     PTPSERVERVERSION=2
4     PTPSERVERDOMAIN=80
5     PTPSERVERSYNCRATE=0.9
6 }
```

Why PTP is identified as '0' will be covered shortly. It only takes a couple lines to enable the server options. NTP is a simple switch - turn it on and you're done. With PTP, the version of IEEE1588 to support, the domain to broadcast on, and the rate at which to send out sync packets must be identified. Here .9Hz is selected, or one sync every 1.1ms.

As you may have guessed, TimeKeeper serves the local time it has synced to, and that local time depends on what it is configured to track. Let's look at a fully functional configuration with failover built in:

Listing 9: Serve PTP and NTP with Multiple Sources

---

```

1 SOURCE0 () { PTPDOMAIN=80; PTPCLIENTVERSION=2; IFACE=eth0; }
2 SOURCE1 () { PTPDOMAIN=20; PTPCLIENTVERSION=2; IFACE=eth1; }
3 SOURCE2 () { NTPSERVER=10.5.3.45; }
4 SERVENTP=1
5 SERVEPTP0() {
6     PTPSERVERVERSION=2
7     PTPSERVERDOMAIN=10
8     PTPSERVERSYNCRATE=0.9
9 }

```

The syntax follows standard shell rules - so multiple directives can be declared on the same line. Here TimeKeeper will track a PTP server on domain 80, visible via `eth0`. It will also track a PTP server on domain 20, visible via `eth1` in case the first PTP server goes offline. If both of those fail, it will follow an NTP server.

Regardless of which source is active, TimeKeeper will deliver time via NTP, and also act as a PTP server on domain 10, emitting packets on the device that owns the default route at a rate of one every 1.1ms. Clients of this PTP server can trust they will get accurate time regardless of any upstream server failures.

As with other configurations, TimeKeeper keeps the clock information for each source over time for analysis and auditing.

## 6 Bridging Networks with PTP Boundary Clocks

The above configuration turned on PTP with the syntax `SERVEPTP0`. The reason for this is that TimeKeeper can be configured to act as a boundary clock. If there is only one primary network, there is no need to specify where the PTP messages are to be sent - TimeKeeper will make use of Linux's standard routing to determine where to send the data.

In the case where PTP traffic should be transferred from one network to another, or taken in on one network and repeated on multiple separate networks, TimeKeeper allows multiple PTP servers to be named. Here's an example:

Listing 10: Serve PTP and NTP with Multiple Sources

```

1 SOURCE0 () { PTPDOMAIN=0; PTPCLIENTVERSION=2; IFACE=eth0; }
2 SOURCE1 () { NTPSERVER=10.5.3.45; }
3 SERVEPTP0() {
4     PTPSERVERVERSION=2
5     PTPSERVERDOMAIN=0

```

```
6     PTPSERVERSYNCRATE=0.9
7     IFACE=eth1
8 }
9 SERVEPTP1() {
10     PTPSERVERVERSION=2
11     PTPSERVERDOMAIN=0
12     PTPSERVERSYNCRATE=0.9
13     IFACE=eth2
14 }
```

In this configuration, PTP domain 0 data visible on eth0 will be tracked as a primary source, with an NTP fallback. TimeKeeper, as it tracks the primary, will act as a PTP server for domain 0 on eth1 and eth2 separately. PTP clients downstream will communicate with this server for delay requests, sync messages, followups, etc., rather than overloading the primary upstream domain 0 server.

## 7 Wrapup

TimeKeeper is built from the ground up to provide highly accurate time synchronization through the 'last mile', directly into the application. With version 5.0, FSMLabs has built in multiple source tracking into the core of the product. This allows accurate reporting and analysis at any time, while providing customers with the ability to seamlessly add redundancy to their time sensitive applications.

Still have questions? Contact us at [support@fsmllabs.com](mailto:support@fsmllabs.com).