

# Measurement of TimeKeeper and PTPd performance

Cort Dougan

cort@fsmllabs.com

*Chief Technical Officer, Finite State Research, LLC*

Self-reported estimates of the accuracy of time synchronization software can be misleadingly incorrect so here we detail how we independently test the accuracy of time synchronization and show the results. Any time synchronization installation and testing should not exclusively rely on self-reported software accuracy estimates.

## 1 Our Motivation - The Problem

Very few sites that rely on time synchronization systems have tested them. When asked how they know that the software/hardware is doing what it's expected to be doing the response is usually "the documentation says so" or "the software itself reports all is ok". There are very few tools available to easily test time synchronization or easily reproduced methods for testing and this is a likely cause for this problem. In this paper we present some simple methods of doing testing that do not require special equipment or techniques. Our tests show that one of the more popular time synchronization systems (PTPd) exhibits some undesirable behavior including poor tracking of a single time source, significant offset from a desired time for long periods and erratic movement of time. Running untested time synchronization software is a very dangerous thing since it exposes applications to unpredictable behavior of the system clock that are not apparent in the self-reported estimates of accuracy. This can be especially dangerous for sites that have external or customer requirements for time synchronization.

## 2 TimeKeeper 4.19

Fig.1 shows TimeKeeper during a 9 hour test under extremely heavy load and alternating unidirectional network traffic - a worst-case scenario for time synchronization.

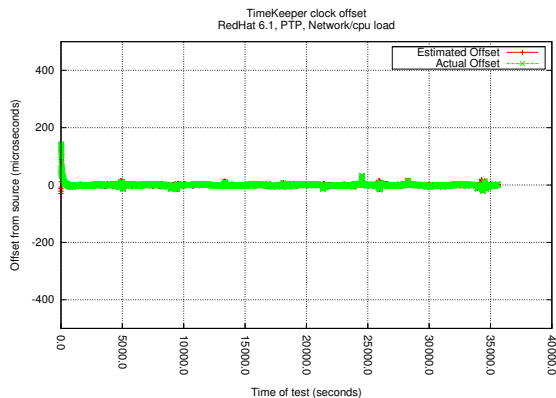


Figure 1: TimeKeeper NTP - 9 hours

Fig.2 is a 2 hour TimeKeeper test under heavy load using the PTP protocol. Performance is not as good as when using the NTP protocol since the PTP protocol is far less tolerant of varying workload on the network and is more susceptible to alternating load.

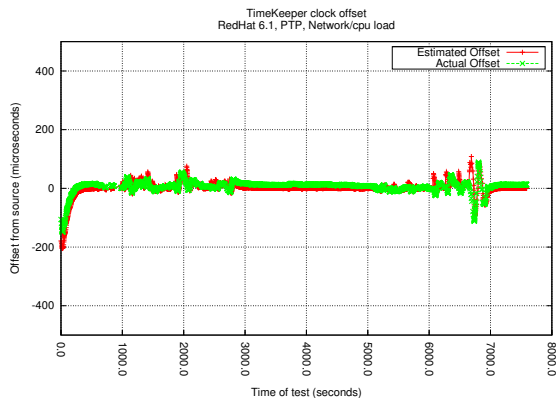


Figure 2: TimeKeeper PTP - 2 hours

### 3 Open Source PTPd 2.1.0

Fig.3 shows a 2 hour test of PTPd 2.1.0 with the same load on the system and network as with TimeKeeper.

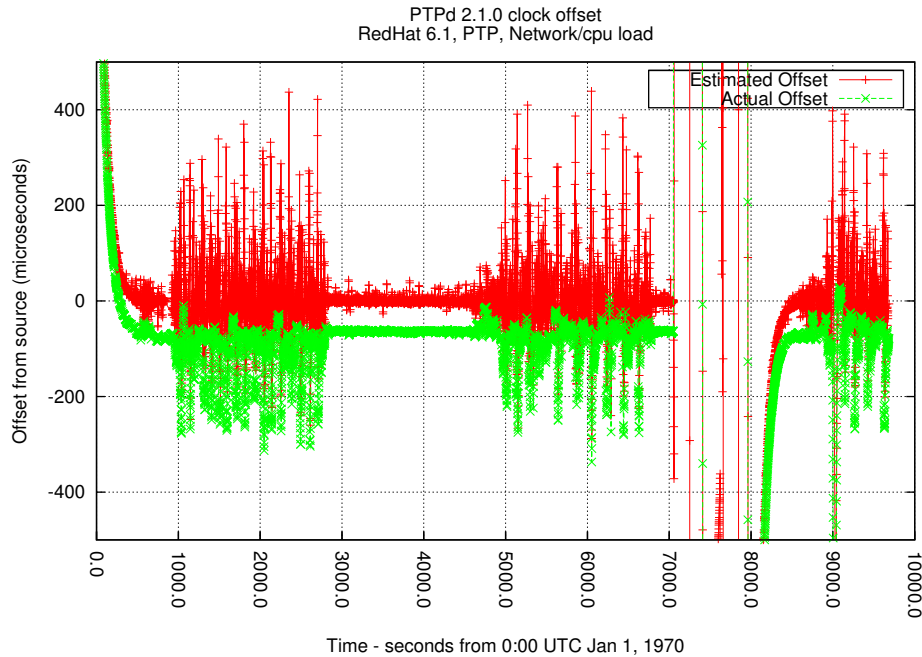


Figure 3: PTPd - 2 hour

### 4 Cost of reading time

Calls to read the system time require some amount of time to execute and that in turn causes inaccuracies in the time being returned and can limit the rate at which the time can be read. A marginal change in the cost to read the time can impact applications heavily that make frequent time reads. Here we call `clock_gettime()` 100,000 times and measure the time required to complete the call and display it as a histogram. Calls under standard Linux and PTPd(Fig.4) require about 30% longer to complete than when running TimeKeeper(Fig.5). TimeKeeper works by providing time to the entire system without any modification applications necessary but if

one is willing to change an application to take advantage of an even higher performance call to `clock_gettime()` then even faster reads are possible - shown in Fig.6.

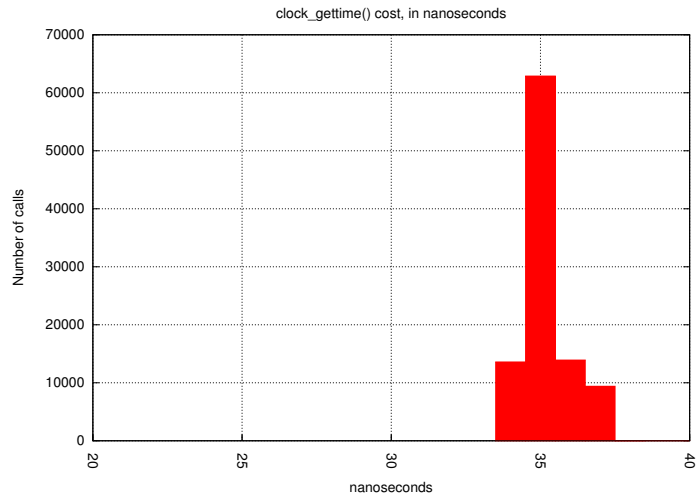


Figure 4: PTPd - cost of reading time

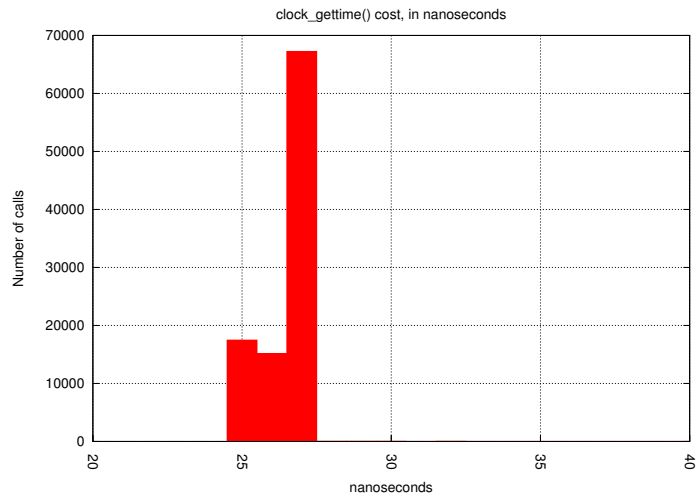


Figure 5: TimeKeeper - cost of reading time

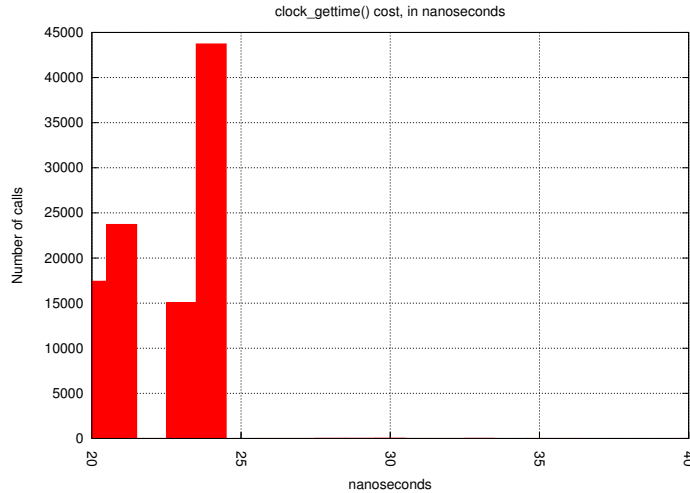


Figure 6: TimeKeeper - direct call

## 5 Test Setup and Methodology

There are many ways to verifiably test the accuracy of a time synchronization system but the method described here is designed to be easily reproduced very quickly without special test equipment. This test used a single time source (GPS) to provide accurate time to the system acting as a time synchronization server and to the computer acting as a client. The server synchronized its clock with the GPS provided time and then provided its time over the network to the client. The client synchronized its clock to the server over the network. The accuracy of the synchronization was measured by comparing the time difference between the GPS time provided to the client and system clock on the client that was managed by the software under test. That showed the accuracy of the time provided from an authoritative source, through the server, across the network and to the client.

The software under test was used on the server to provide time over the network and to consume that time on the client. Since PTPd when acting as a server is not able to directly synchronize the local clock to an external time source (it instead requires the local clock already be synchronized) another piece of software was used to do that.

One test was run for 9 hours to simulate a normal business day but others were run long enough to show typical behavior. Statistics collected

were over that entire period with no datapoints discarded. Both machines were Supermicro, two 4-core Intel CPUs (8 cores total) running at 2.13 GHz installed with RedHat Enterprise 6.1 and E1000 82580 ethernet cards (with the latest driver installed). The network connecting the two systems was a shared 1Gb ethernet connection through a switch (Cisco SR2024) on an isolated network.

## 5.1 Software settings

Software was installed and used without changes to default settings. The IEEE 1588 (Precision Time Protocol) mode tests used “Annex J” values only since testing with other settings showed the open source PTPd package failed to synchronize properly.

## 5.2 Accuracy of the testing tools

The GPS used in this test transmitted a RS232 NMEA 0148 string every second to provide “time of day” and a pulse-per-second signal to indicate the top of each second. The manufacturer claims a 1 microsecond accuracy for the pulse.

The pulse-per-second line was routed to the RS232 port on the server and client under test. A loadable kernel Linux module was used to read the pulse and perform the comparison of system time and the time that the pulse arrives at the client. It was able to do this accurately by isolating a CPU on the system, disabling interrupts and polling continuously for the pulse and then immediately reading the system time when the pulse was detected. The RS232 UART required 1.2 microseconds for every read and this in addition to the accuracy of the GPS limits the accuracy of the measurement to roughly 2.2 microseconds total. While not ideal for production use this is perfectly adequate for these tests. Measuring the time from the GPS in this way ensures that interrupt latency or heavy system activity does not interfere with the accuracy of the read of the pulse-per-second signal.

## 5.3 System Load

Each of these tests was alternating between an “idle” system and an intentional load to mimic periods of heavy activity and idle time that a system may see during a typical day. The script used for testing generates a very

large amount of CPU, network and disk activity. The network between the two systems was very heavily loaded with multiple streams of traffic between the server and client that alternated direction to create asymmetrical loading.