FSMLabs
Enterprise Real-Time

Avoiding pitfalls and misinformation in time synchronization by using TimeKeeper®

Sidestep the roadblocks to reliable timing

In our deployments of time sync solutions we have seen many people struggle with misinformation and roadblocks that shouldn't be allowed to hold people back. Lack of easy testing procedures make things harder and more mysterious than they should be. This document captures the common pitfalls (and solutions), explains some common sources of misinformation, and gives a good understanding of how to look at time sync tools and concepts.

Before TimeKeeper, timing solution complexities kept reliable timing out of the hands of people that need it. Having timing expertise was a prerequisite to deploying systems. In some environments, with enough time/money, some of the hurdles may be overcome. In our experience though, timing is just a piece of the puzzle and spending months becoming a timing expert isn't an option. There's usually a clear need to deploy a solution and move on. TimeKeeper is built to let users do precisely that and not spend months working on science projects.

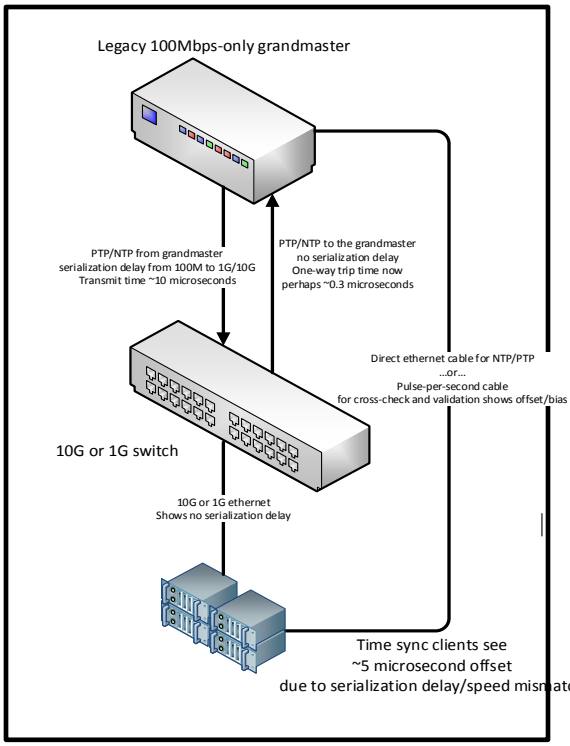
Some or all of the following probably sounds familiar: Applications need to get more accurate time than they're getting right now. There are some existing legacy NTP servers, but they don't deliver reliable time more accurate than 10s of milliseconds in the best scenario. There's a push to move to PTP, but it's not clear how a complex multicast protocol is going to work across sites and through firewalls to different networks and colos. A lot of questions aren't answered, like:

- 10G networking is the norm, but these legacy time servers are 100mbit or at best 1G. How do clients detect and correct for network asymmetries caused by this mismatch?
- How do we make sure clients get the best timestamping possible? What needs to happen so that clients use hardware timestamps? How can we confirm that the hardware timestamping cards are working properly?
- Accurate time on the network is key, but how do I make sure that my application's time is accurate? Custom timing API calls are not an option.
- The PTP spec declares some failover capabilities, but it's difficult to understand what happens in every scenario. If a failure occurs, diagnosing and describing what clocks were selected and why after the fact isn't realistic. Can the failover process be clearly defined so there's no need to guess at what could happen?
- There are time servers deployed already, and they can't just be thrown out. How can they be leveraged for protocol/performance upgrades?

- NTP is deployed today, but there's a need to move to PTP. Is there a way to slowly and reliably deploy PTP while keeping a safe fallback?
- How is all of this managed? Is it automatic? Do I need more tools?
- Along those lines, how do I verify and test that no matter where in the network the client is, time is accurate?
- I'm being told NTP is inaccurate – but don't see why it should be?

Let's take this apart question by question.

Link speed mismatches – why do they matter?



If your time server is 100Mbps or 1G but your switch is 10G then speed mismatches ***will*** cause silent time errors in production. Anywhere there's a change in network speed, like 100mbit to 1G, or 1G to 10G, in any direction, significant timing accuracy is being lost. This introduces an error of 5us or more, depending on the mismatch.

The reason this occurs is that most cut-through switches are cut-through going from faster to slower speed, but store and forward in the other direction. This means that the time packets going from the client to the server will take a different amount of time than when they go from the server to the client. This creates a difficult to detect network asymmetry. So there is a silent error on the client – although it is one that TimeKeeper can detect and correct for by pulling in additional time sources to cross-check time (perhaps a pulse-per-second signal). Another solution is to use timing servers with the correct link speeds to match modern networks. TimeKeeper GrandMaster appliances come with both 1G and 10G links to prevent this from ever being a problem for FSMLabs customers.

How do my clients use the most accurate timestamping available? And why does this matter?

Thankfully, this is a minefield that TimeKeeper customers can avoid, since it can consume weeks or months of engineering time. To calculate accurate time via PTP or NTP, accurate timestamping of packets is very important. This can be done in software, and optimally, in hardware if supported by the hardware and software. Most "PTP aware" cards available are

not only PTP aware but are able to timestamp NTP packets as well – giving you flexibility on protocols without sacrificing accuracy.

TimeKeeper automatically makes use of hardware timestamping if the hardware capability is present and the timestamp data is correct. Hardware timestamping just works out of the box with cards from Intel, Mellanox, Solarflare, Broadcom, and others under TimeKeeper. The hardware timestamps are validated at all times, and if at any point they are found to be incorrect, TimeKeeper automatically transitions over to software timestamps seamlessly.

With this out of the box hardware timestamping, applications get access to more accurate data, allowing for a more accurate clock. There's no need to match special daemons to special hardware revisions, or become an expert in Linux kernel driver management. TimeKeeper just works, and provides the best time available based on the underlying hardware.

My APPLICATION needs accurate time, not some timing daemon or the network card.

This is exactly right – and it's a fact commonly ignored by many time providers. We call this the 'last micrometer' of time distribution and it's what TimeKeeper excels at. Delivering accurate time to the NIC is fine. Having a perfectly accurate clock on a PCI-E card is great but your application isn't using that. Having a perfectly accurate clock in a protocol daemon on the host is great too, but most solutions, even if they calculate perfect time (which they do not), the daemon just tries to steer the OS clock. None of this really addresses the actual accuracy and speed needed by the application.

TimeKeeper takes a different approach here, so that it can take that accurate time on the wire and deliver it directly to the application. Automatically, the best timestamping technology is used to build accurate time to deliver to applications. Instead of handing that off and hoping the OS moves in the right direction, TimeKeeper takes control of time management from the OS. This resolves many longstanding Linux bugs while allowing applications to get more accurate time, all without ever having to worry about custom APIs.

This means that that accurate time goes directly from TimeKeeper to your application, instead of being pushed around the system. As a bonus, TimeKeeper's implementation of the standard Linux timing calls deliver time faster, providing more accurate time to the application and at the same time freeing up processor cycles.

Resiliency and failover – how do I make sure I'm covered?

This is a complex undertaking without TimeKeeper. With NTP clock A, PTP clock B, PTP clock C, and remote NTP cross check clock D, how do you specify what sources to track, in what order? If you're just deploying PTP, you may want to track NTP from clock A, and fail over to NTP from clock D, but still track PTP while you test your infrastructure.

Or, after PTP is deployed, clock B's PTP might be preferred, so that clock C is used only if B goes offline. If both go offline, track clock A then clock D.

Here's where things get complex – without TimeKeeper, failover between protocols is difficult, fragile at best, and requires in-house tooling to detect problems and start/stop daemons since no other solutions combine NTP, PTP, bus devices, and all other protocols into a single system as TimeKeeper does. (If you're wondering what happens to your application's time when the clock control gets handed from one daemon to another, and how long it may take to settle, you're seeing the very start of a long chain of problems.)

Even failover within protocols is complex without TimeKeeper. PTP's best-master-clock algorithm provides some definition of which PTP clock will be preferred, and this allows you to configure the network so clock B is selected as the primary PTP source. However, the BMC algorithm may decide based on changing network parameters that C is a better choice, against your wishes or intent. With all of the moving parts in modern networking, relying on just the BMC algorithm requires vigilance to ensure that your clients are tracking the clocks you want them to track. For some real-world examples of how the non-TimeKeeper implementation of PTP can cause problems due to network problems that causes violation of FINRA regulations see:

http://tagus.inesc-id.pt/~pestrela/timip/Challenges_deploying_PTPv2_in_a_Global_Financial_company.pdf

The group writing that paper required a team of smart and dedicated people, working for 2 years without TimeKeeper to deploy PTP. After that amount of work they still have points of failure in their network that they cannot overcome. TimeKeeper is designed to handle these issues so that production deployments can happen in hours or days, and it resolves a number of the remaining complaints in the article as a bonus.

TimeKeeper makes the issue of failover a non-problem, on both the GrandMaster and on the client. Just lay out your list of time sources in your preferred order, and TimeKeeper will automatically track them, switching clocks only when needed – like when a clock is lost or when it's found to be invalid, due to a failing component or a GPS attack. This leverages the PTP best-master-clock algorithm but builds on it to provide clear failover scenarios and allows you to override ambiguous BMC behavior you want to avoid. An example configuration with well defined behavior for deploying PTP in your network while also maintaining NTP as a primary time could be:

1. Track NTP from clock A
2. Track multicast PTP from domain 0 (PTP clock B)

3. Track unicast PTP from PTP clock C in a remote datacenter
4. Track NTP from clock D

Let's be specific here - TimeKeeper's actual configuration here would look like this on a client system:

```
SOURCE0() { NTPSERVER=10.0.3.4; }
SOURCE1() { PTPCLIENTVERSION=2; PTPDOMAIN=0; }
SOURCE2() { PTPCLIENTVERSION=2; PTPDOMAIN=0; PTPSERVER=10.50.20.4; }
SOURCE3() { NTPSERVER=10.0.4.5; }
```

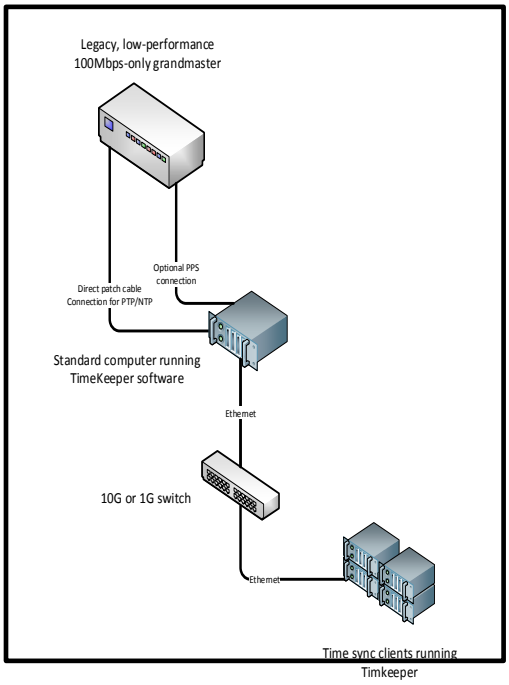
This simple configuration provides an unambiguous failover scenario between multiple clocks in multiple datacenters with multiple protocols.

What about our existing investment?

The easiest solution is to add a TimeKeeper GrandMaster to solve these problems but that's not always possible. Deployed timing components, like legacy NTP and PTP servers, can't always be written off given the investment in them. The trouble is, the time they deliver may simply be too noisy, silently biased due to the device having less-than-modern connectivity (see the asymmetry discussion above), or may not provide the management capabilities needed in a modern network. Often times the poor quality NTP protocol performance that these devices provide is just a limitation of the free software downloaded from the internet that they use to implement NTP. That's something that TimeKeeper can fix without much investment at all.

Time clients still need to get accurate time, and TimeKeeper has some simple mechanisms to make this happen. In some situations TimeKeeper can detect noisy clocks and treat them as 'low quality', extracting a more stable signal from the noise. This can be explicitly configured as well.

TimeKeeper also works as a standalone software server, like a boundary clock or a stratum server. Putting a TimeKeeper instance (in software) close to a noisy source stabilizes a noisy clock so a clean signal can be delivered to downstream clients, leaving the existing server in place and unmodified during a deployment. TimeKeeper software servers can even bridge different link speeds, isolating and correcting for asymmetries. Bringing legacy servers 'up to speed' in this way lets you continue to leverage that investment.



What's the best way to transition to PTP with minimal risk and not "jump all at once"?

Let's look back at the scenario we laid out earlier. NTP servers exist on the network and PTP GrandMasters are being deployed, but it's risky to just cut over to PTP all at once. What if there are clock configuration errors? What if the new servers can't handle the load? Are all of the multicast routing changes ironed out?

TimeKeeper is designed to make this transition one that happens at your pace, and in a precise and controlled way. You can continue to track NTP as your primary time source, and just add in a number of other sources on your clients, like we did above:

- Primary source – NTP from clock A
- Secondary source – PTP from clock B
- Tertiary source – PTP from clock C
- Track NTP from clock D

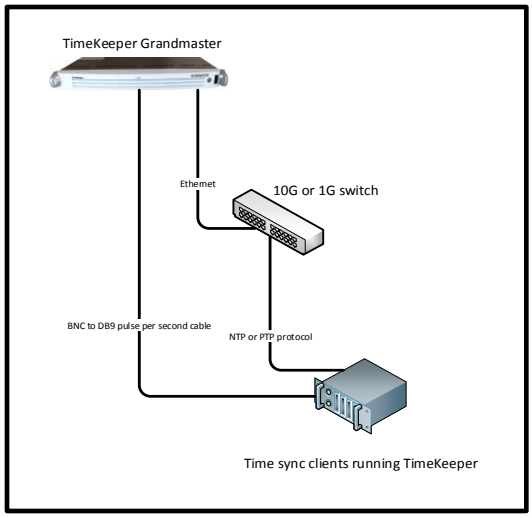
There are many benefits here – the client continues to use NTP, but tracks PTP at the same time. Not only does this stress your PTP server, it also validates that any network changes needed to get PTP to the client are in place. More importantly, TimeKeeper provides detailed clock behavior data on each clock from the perspective of each client over time. With this data, the NTP and PTP feeds can be compared over the same time period on the same machine, so you can compare clock accuracy. If the PTP and NTP clocks don't agree, problems can be diagnosed and resolved without ever forcing the client to switch to PTP. Once they agree, the setup can be left in place for extended testing, and when the time comes, switching to PTP is a matter of changing the time source order. Move PTP higher in the list, move NTP down, and you've deployed PTP, while still maintaining an easy and clear failover path. This change can happen on a per-network, per-site, or per-client basis, all based on your preference.

How is all of this managed?

Managing thousands of clients is already hard enough. Building an accurate timing network should not create more work. Very few people have the time, ability, or desire to write their own log check scripts targeting a number of different timing tools, and their own alarm logic that will detect and act on every single possible failure scenario.

TimeKeeper solves this problem by integrating seamlessly with enterprise tools. SNMP support is provided out of the box, including a full SNMP MIB that can be used to connect management consoles to both TimeKeeper clients and GrandMasters. Syslog alerts, email, traps – it all just works out of the box, so there's no need to write your own log monitoring tools.

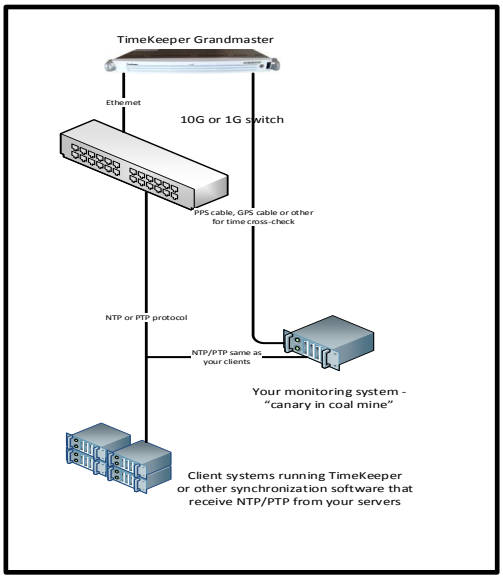
Additionally, TimeKeeper automatically collects management data from all of the client clocks on the network, and makes them all viewable from a single pane of glass. With this tool, you can see how all of your clients are behaving. Alarms can be automatically sent if any of the clients exceed a defined accuracy threshold. Again, this means users can identify what constitutes an error, configure it, and move on, without having to understand and parse log files, or create any specialized tools.



How is the data verified?

This is a question that's not asked enough. Too many times users install a tool that reports great accuracy and that's the end of the testing. Dangerous is a polite way to describe this. What if there's a silent asymmetry? What if there's a discrepancy between PTP and NTP? What if the calculated time is jittery, or off by hundreds of microseconds, but the tool reports accuracy to 300 nanoseconds? This happens more commonly than you'd think.

It doesn't matter what client is used – free clients, TimeKeeper, or even a local dedicated hardware solution. Verification is key to having an accurate timing network, and that's why TimeKeeper is built to allow for easy validation.



FSMLabs can also provide other test scenarios, but with a direct PPS line to the client, TimeKeeper can track the PPS and any other clock sources on the network, including time from the device providing the PPS. Comparing the resulting data allows you to easily detect network asymmetries and routing issues, so that they can be resolved. TimeKeeper of course does let you bias particular time sources, but it's relatively rare to need this option unless there is a legacy time server upstream. Instead, it's much more common to use TimeKeeper to diagnose and resolve fundamental issues so that you don't have to spend your time finding the correct bias for this week's network topology.

Verification can also be continual – by placing TimeKeeper at remote endpoints of your network, you can be alerted immediately if a topology or routing change begins to affect the quality of your client's time sync. As with everything else, this is done with common off the shelf tools, requiring little to no maintenance effort.

Why is NTP so inaccurate?

Now we've come to the not-so-state-of-the-art: the prevalent NTP misdirection. It's common to be told that 'NTP can't provide accuracy better than X milliseconds', usually by someone who happens to be selling PTP and wants you to buy new switches that are PTP aware, or new network clocks and client hardware. This is simply not true. NTP is a time protocol and carries similar information to PTP. Both have their benefits and downsides. However, both are capable of delivering time accuracy in the sub-microsecond range, and TimeKeeper's NTP implementation has been doing so for years.

Many people, including several PTP vendors, confuse the NTP protocol and implementation - perhaps intentionally. The protocol allows for accuracy, but the implementations tend to be very inaccurate. **NTP can be and is the same accuracy as IEEE 1588 with the right tools.** NTP and IEEE 1588 (PTP) are simply network protocols that exchange time information with nanosecond resolution. As protocols, they have the same potential for nanosecond accuracy. Many vendors claim NTP is less accurate because when referring to "NTP" they mean the program NTPd from the University of Delaware that has been the de facto standard for NTP implementations for years. That program was never designed to be as accurate as TimeKeeper or as aggressive at maintaining the correct time. Many hardware devices out there that serve NTP, devices that boast very high quality clocks and GPS backed time, still internally rely on the NTPd program that is not able to able to deliver accurate time. The problem is magnified when people use NTPd on the client also. With TimeKeeper (software or hardware) you can see equally accurate time (100s of nanoseconds in some cases) from NTP and PTP. To verify that you can even run both protocols at the same time and compare as long as your time server is able to deliver accurate NTP and PTP. Again, TimeKeeper GrandMasters deliver NTP with sub-microsecond accuracy.

If you're avoiding NTP due to accuracy concerns, it's best to evaluate that decision very carefully since the complexity and cost of PTP is significant. TimeKeeper can deliver and track NTP very accurately, and in many cases, this is the best way to greatly increase accuracy in a timing network with a minimum amount of change.

Contact FSMLabs

TimeKeeper, TimeKeeper GrandMasters, TimeKeeper Server Software, and TimeKeeper Client Software are all available from FSMLabs and FSMLabs' resellers. For purchase information or for a live demonstration of TimeKeeper please contact FSMLabs at sales@fsmlabs.com.