NETWORK TIME FOUNDATION

Nerdear.la

24 September 2024

Network Time Protocols
and Network Time Foundation

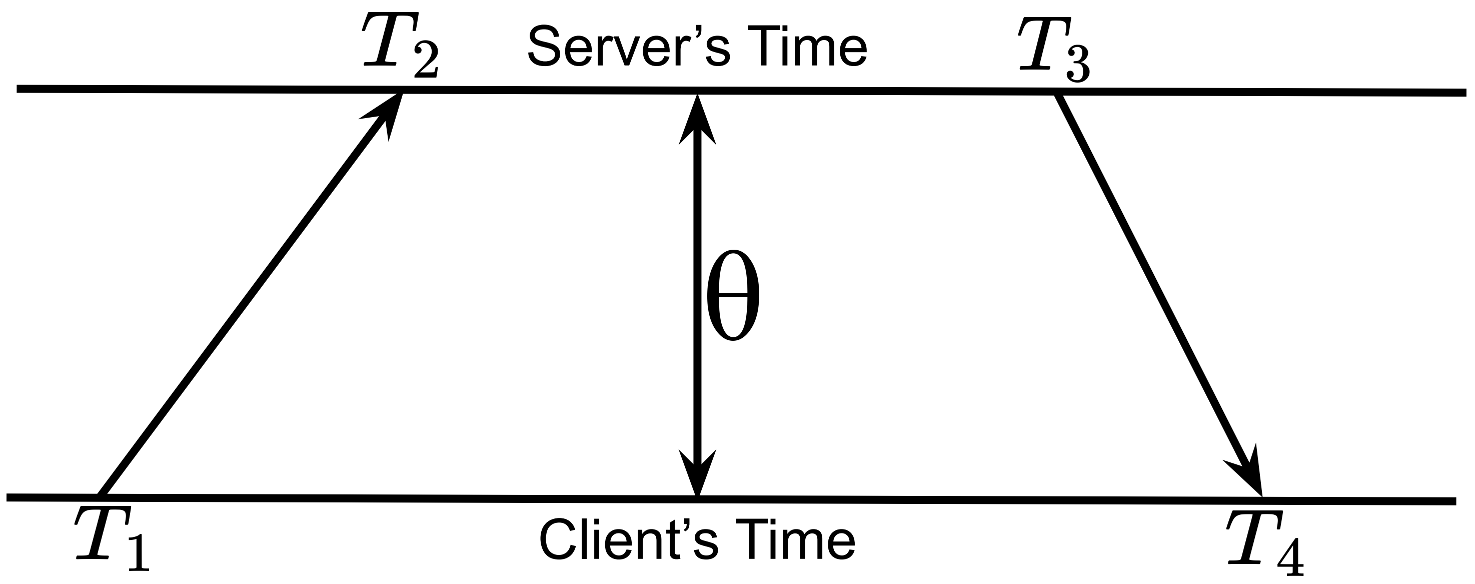
How it First Began

In the late 1970s/early 1980s David L. Mills, PhD stopped working on early network routing protocols and was looking for an interesting new project. He noticed how the lack of time synchronization between computers was a Big Problem:

> It is somewhere between difficult and impossible to correlate events that happen on different systems if their clocks are not synchronized.

He chose to work on network time synchronization.

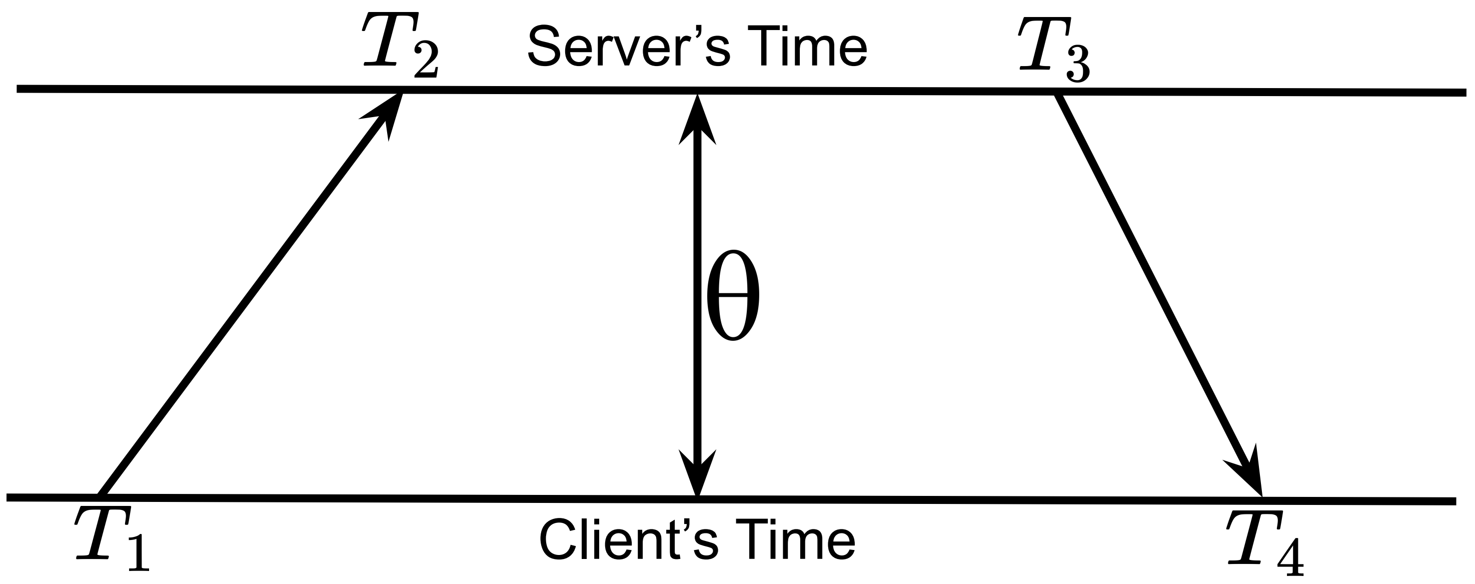
The Mechanics of NTP



Time Offset: $\theta = \frac{1}{2}[(T_2 - T_1) + (T_3 - T_4)]$

Network Delay: $\delta = (T_4 - T_1) - (T_3 - T_2)$

The Mechanics of NTP

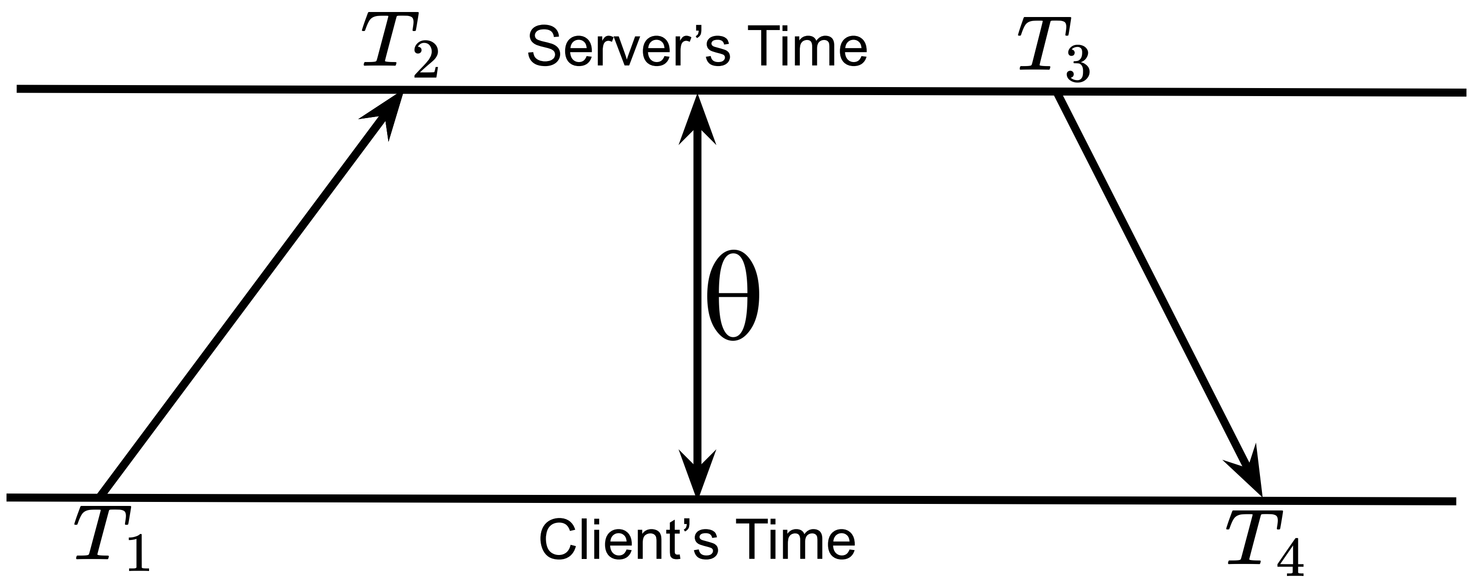


Time Offset: $\theta = \frac{1}{2}[(T_2 - T_1) + (T_3 - T_4)]$
Network Delay: $\delta = (T_4 - T_1) - (T_3 - T_2)$

Client and Server time match. If $T_1=1$, $T_2=2$, $T_3=3$, $T_4=4$ then:

Offset/$\theta = \frac{1}{2}[(2-1)+(3-4)] = \frac{1}{2}[1+ -1] = \frac{1}{2}(0) = 0$ seconds
Delay/$\delta = (4 - 1) - (3 - 2) = 3 - 1 = 2$ seconds

The Mechanics of NTP

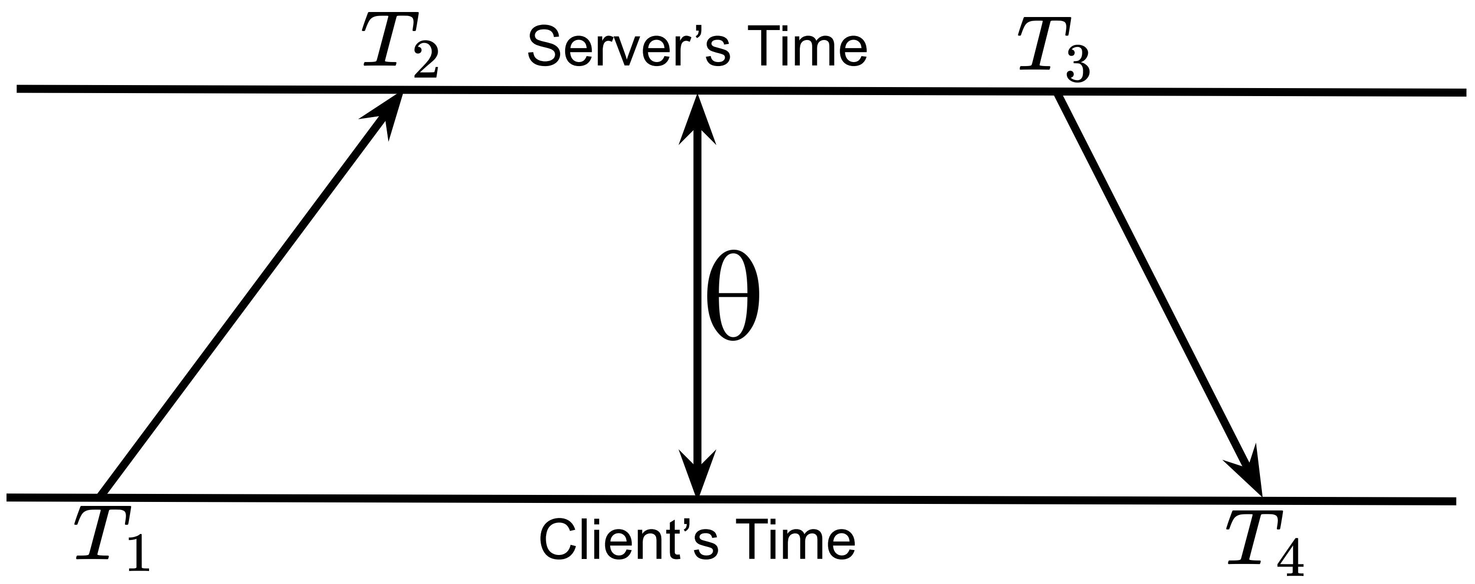


Time Offset: $\theta = \frac{1}{2}[(T_2 - T_1) + (T_3 - T_4)]$
Network Delay: $\delta = (T_4 - T_1) - (T_3 - T_2)$

Client behind Server by 10 seconds. $T_1=1$, $T_2=12$, $T_3=13$, $T_4=4$:

Offset/$\theta = \frac{1}{2}[(12-1)+(13-4)] = \frac{1}{2}[11+ 9] = \frac{1}{2}(20) = 10$ seconds
Delay/$\delta = (4 - 1) - (13 - 12) = 3 - 1 = 2$ seconds

The Mechanics of NTP



Time Offset: $\theta = \frac{1}{2}[(T_2 - T_1) + (T_3 - T_4)]$
Network Delay: $\delta = (T_4 - T_1) - (T_3 - T_2)$

Client's clock runs twice as fast as the Server's clock.
$T_1=0$, $T_2=1$, $T_3=2$, $T_4=6$:

Offset/$\theta = \frac{1}{2}[(1-0)+(2-6)] = \frac{1}{2}[1 + -4] = \frac{1}{2}(-3) = -1.5$ seconds
Delay/$\delta = (6 - 0) - (3 - 1) = 6 - 2 = 4$ seconds

NETWORK TIME FOUNDATION

Adjusting Clocks

We’ve seen how, if the client and server clocks are running at the same rate, we can calculate the offset that will make the client’s clock match the server’s clock.

That’s a good first step. The next step is to adjust the client’s clock rate to better track correct time.

How can we do this?

Adjusting Clocks 2

To determine the difference between the client’s clock rate and the server’s clock rate, we wait a while and again query the server’s time.

Since we know how long it has been since our last query and we know the new offset, simple division tells us the rate at which the client’s clock is different from the server’s clock. This value tells us how many (fractional) seconds per second to adjust the client’s time rate.

NETWORK TIME FOUNDATION

Adjusting Clocks 3

For example, let’s say we fully applied any offset correction from our previous NTP time query.

100 seconds later we issue another NTP time query, and we learn that we have to apply an offset correction of 1 millisecond.

1 millisecond/100 seconds is a correction rate of 10 microseconds per second.

Adjusting Clocks 4

For example, let’s say we fully applied any offset correction from our previous NTP time query.

100 seconds later we issue another NTP time query, and we learn that we have to apply an offset correction of 1 millisecond.

1 millisecond/100 seconds is a correction rate of 10 microseconds per second.

So we apply the 1msec offset correction and tell the system clock to adjust its rate by 10µsec/sec.

In another 100 seconds we “repeat the dance.”

The poll interval

The number of seconds between polls, the “poll interval”, is internally tracked via ‘n’, which is 2^n seconds long. In NTP, ‘n’ can range from 3 (8 seconds) to 17 (about 1.5 days’ time). The default poll interval is 6, or 64 seconds.

If the timekeeping quality of a host is “good”, we can make even finer time adjustments by waiting longer between our polls. Likewise, if the time quality degrades we reduce the polling interval.

About Network Delay

Please recall:

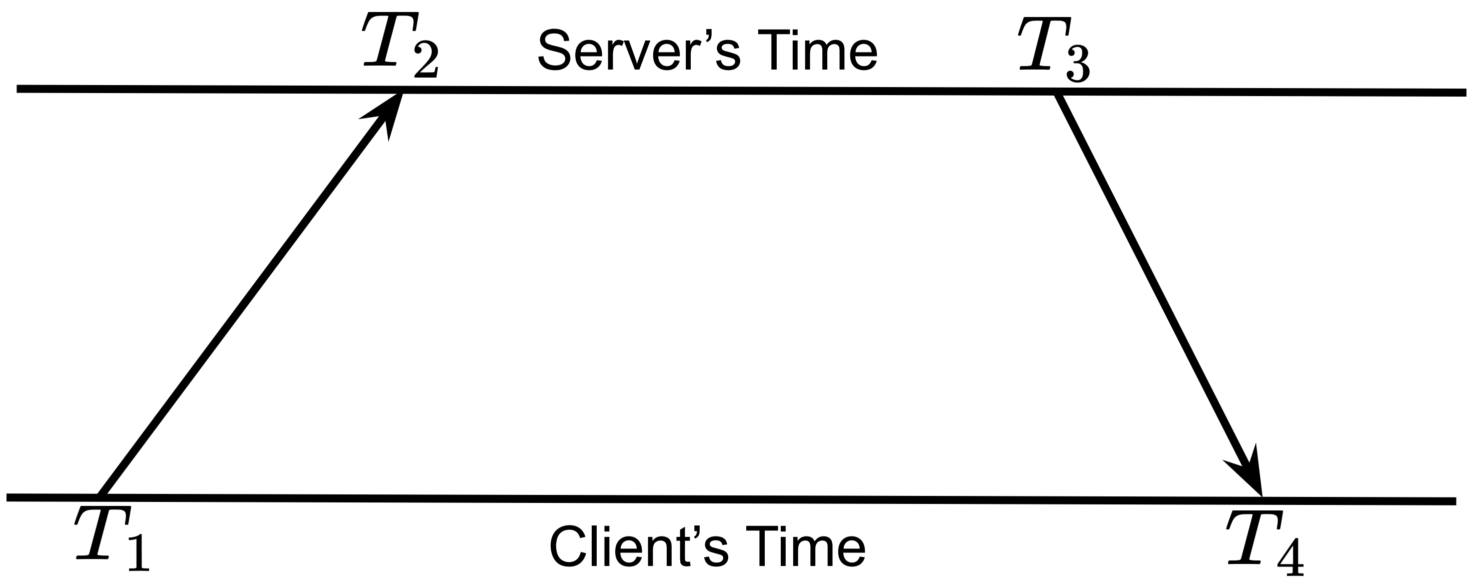
Network Delay: $\delta = (T_4 - T_1) - (T_3 - T_2)$

The model assumes symmetric delay times between the client and server.

What would be needed to accurately measure each one-way path delay?

What if that value changes, dynamically?

Packet Transmit Time



T_1 and T_3 represent "Packet Transmit Times".

Packet Transmit Time

If we build a packet and transmit it:

```
…
pkt.xmit_time = clock_gettime();
send(dest, pkt);
```

it’s obvious that some additional time passes between the call to `clock_gettime()` and the time `send()` actually transmits the packet.

PTT and Authentication

```
…
pkt.xmit_time = clock_gettime();
authenticate(pkt);
send(dest, pkt);
```

And if we want to add an authentication hash to the packet, even more time is needed.

This additional time needs to be added to T_1 and T_3 to “correct” the calculations.

But how can we do that?

PTT and Authentication

```
…
pkt.pre_xmit_time = clock_gettime();
authenticate(pkt);
send(dest, pkt);
pkt.post_xmit_time = clock_gettime();
```

The `post_xmit_time` works for T_1 because we can use it instead of the exchanged T_1 value.

But `post_xmit_time` is actually a bit too long.

Some systems can better "tag" the actual transmit file in the `send()` call.

It doesn't help us correct T_3. But there are ways…

PTT and Authentication

```
…
pkt.pre_xmit_time = clock_gettime();
authenticate(pkt);
send(dest, pkt);
pkt.post_xmit_time = clock_gettime();
```

Communicating the correct T_3 needs other mechanisms. Options include:

- NTP’s Interleave mode
- Transmit `post_xmit_time` in an extension field in the next packet

NETWORK TIME FOUNDATION

Modem Technology

The Bell 212A 300/1200 baud modem was standardized in 1979. In 1984, 2400 baud modems became available. These modems were generally built with discrete components. These modems exhibit constant processing delay time.

The Public Switched Telephone Network (PSTN) was real copper wires. A modem connection between two systems went over what amounts to an uncongested "pipe" on a static network path.

Modems & NIST/ACTS

US NIST used modems and the PSTN to deliver very accurate time with their Automated Computer Time Service, ACTS, starting in 1988.

ACTS sends a * character as an on-time mark, and notes how long it takes from sending the * to its “echo” coming back. ACTS calculates the delay, and when that calculation is stable, it switches to sending a # timed to arrive at the client at the correct time.

NETWORK TIME FOUNDATION

Phone Technology

As time passed, “real” modems became more complex, and “software modems” (softmodems) also arrived. These advances introduced variable processing delay times.

The combination of “technology advances” with modems and the PSTN changing from copper wires to digital has drastically reduced the usefulness of NTP over a modem connection.

NETWORK TIME FOUNDATION

Network Technology

Early local networks were 10Mbps, and then 100Mbps. The network cards and kernel drivers for these speeds offer stable performance.

Gigabit (and faster) network interfaces process their data using interrupt coalescing and component processing techniques that have more random/variable processing performance.

NETWORK TIME FOUNDATION

RS232 Technology

Reference clocks often used an RS-232 serial connection to communicate time. There might also be a Pulse-Per-Second (PPS) signal, often sent on the Data Terminal Ready (DTR) line.

This meant that a kernel that supported TTY STREAMS drivers could get excellent NTP time synchronization over a real serial connection.

Real serial ports became USB serial ports.

NETWORK TIME FOUNDATION

Technology Advances!

We’ve just discussed three places where technology advances have happened, modems and telephone lines, network interfaces and network “media”, and USB serial ports.

There are consequences to changes. Tradeoffs.

As complexity increases … complexity increases!

Fun with time…

Imagine small devices that have well-synchronized time via WiFi connectivity. ½ of the network delay (δ) tells us how far apart each device is.

3.3ns is about 1m of distance.

If the network delay to the “other” device is 330ns, that means the “other” device is somewhere on the surface of a sphere with a radius of 100m centered on this device.

… Fun with time

Imagine small devices that have well-synchronized time via WiFi connectivity. ½ of the network delay (δ) tells us how far apart each device is. 3.3ns is about 1m of distance. If the network delay to the other device is 330ns, that means the "other" device is somewhere on a sphere with a radius of 100m centered on this device.

If we exchange the distance (½δ) to each of our neighbors with each of our neighbors, we can calculate the overlapping spheres. With 3 devices we can calculate the more specific location possibilities based on the intersection of the spheres. As the number of devices increases, we get a better idea of where each neighbor "lives".

NETWORK TIME FOUNDATION

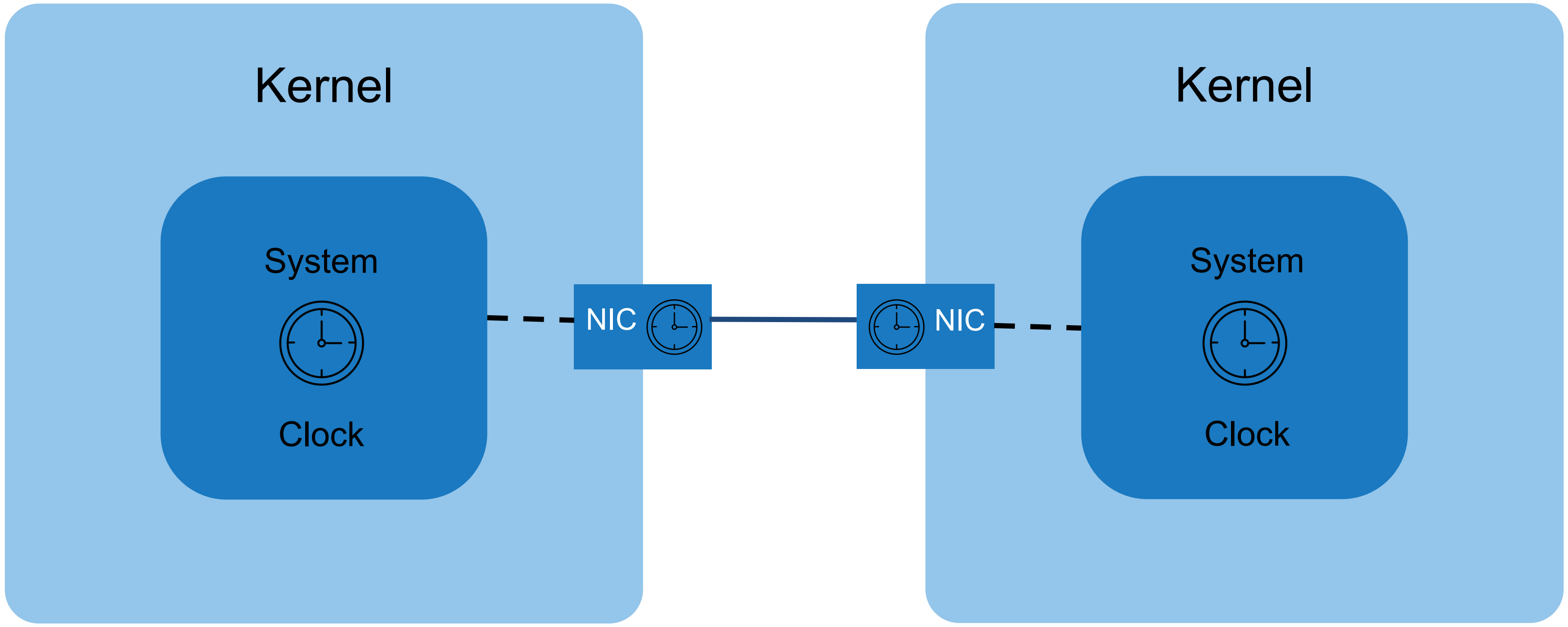
Foundation of NTP

NTP keeps time in the heart of the kernel. To do this, it syncs the time using network packets and/or direct IO (like serial and PPS connections) to a Reference Clock (refclock). That communication goes thru a variety of "layers" to exchange data with the time partner. These layers add noise/jitter to the time sync process, which reduces accuracy. The network path might have some asymmetry. Any asymmetric network connection degrades the quality of time synchronization.

Foundation of PTP...

PTP works by exchanging the time on a clock chip built in to a network interface. If the PTP NIC on one machine talks over a directly-connected network cable to a PTP NIC on another machine, that connection is both direct and symmetric, and it is easy to closely synchronize the PTP clock chips on these two devices.

But that's the best, simplest, and perhaps least useful/interesting configuration.


Kernel
System
Clock
NIC
NIC
Kernel
System
Clock

…Foundation of PTP

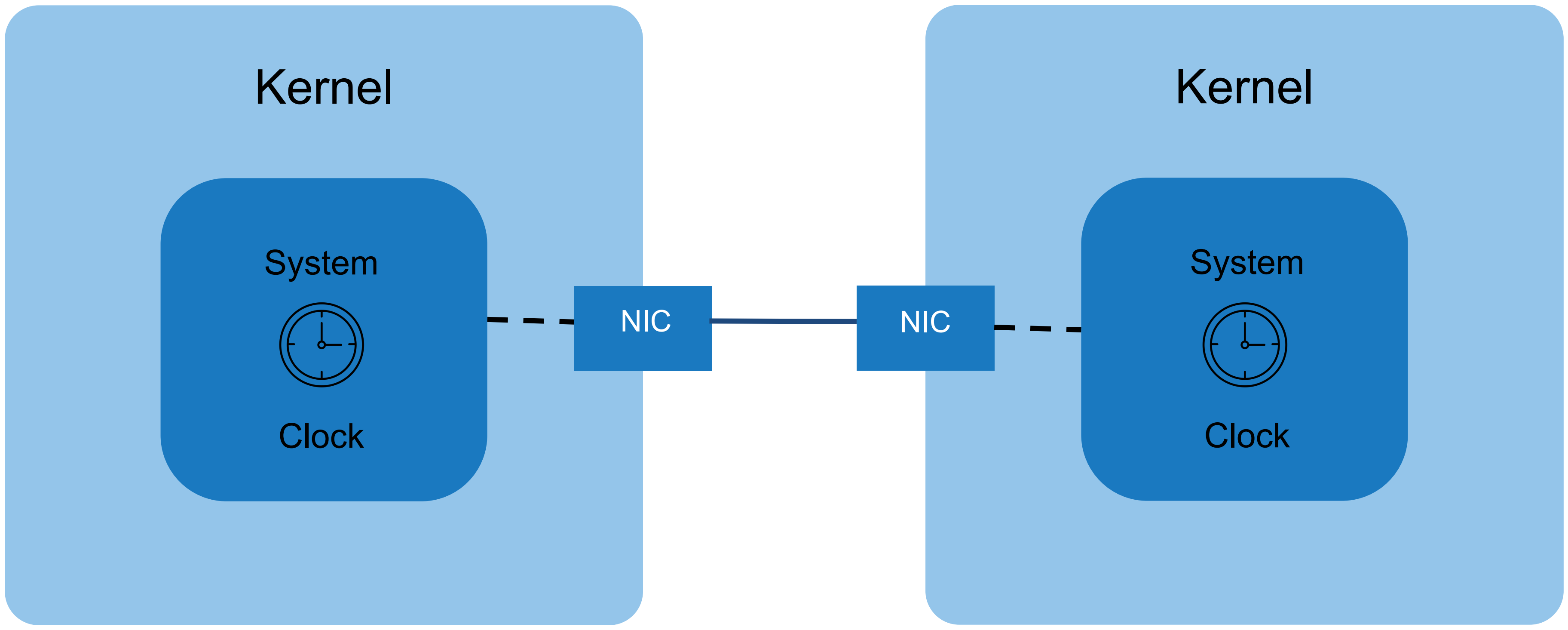
Also, one needs synchronized time on a more generally accessible clock than the one on the NIC chip. One needs easily accessible synchronized time in many places on a system. To get the time from the PTP clock on a NIC chip to the system clock in the heart of the kernel, time needs to be communicated over a variety of "layers" to get it to the system clock. These layers add noise/jitter to the time sync process, which somewhat degrades accuracy.

Foundations …

With both NTP and PTP, there is noise and jitter added whenever time is 'moved around' as part of the internal time synchronization process.

Design of NTP

NTP was designed to transfer time at the network level. Each hop in the network introduces more noise and jitter. If two NTP systems are directly connected to each other with a cable, like the simplest PTP case, the theoretical difference between NTP and PTP can be reduced to the noise and jitter between the NIC and the system clock. In a LAN setting, NTP can easily synchronize clocks to the sub-millisecond level. It can do even better with things like hardware timestamps in the NIC.


Kernel
System
Clock
NIC
NIC
Kernel
System
Clock

Design of PTP

PTP was designed to transfer time over a bus. In the simplest configuration, NIC to NIC, time can be synchronized to a handful of nanoseconds. Time transfer degrades somewhat if there is a single PTP-capable switch involved. Time degradation increases with the addition of every additional PTP-capable switch. If any non-PTP-capable switch is used, high-accuracy time transfer is lost.

NTP and PTP

With both NTP and PTP, there is noise and jitter added whenever time is 'moved around' as part of the internal time synchronization process.

NETWORK TIME FOUNDATION

NTP ...

NTP was designed to synchronize time at the network level.

Local-area, wide-area, and even in space.

The better the quality of the network, the better the quality of time synchronization.

Each hop in the network introduces more noise and jitter.

… NTP

If two NTP systems are directly connected to each other with a cable, like the simplest PTP case, the theoretical difference between NTP and PTP can be reduced to the noise and jitter between the NIC and the system clock.

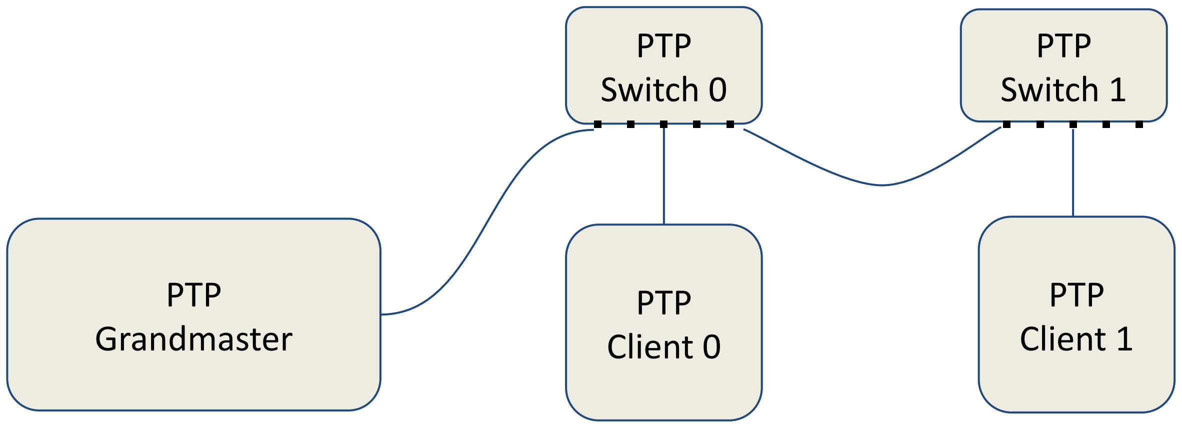
Over a LAN, NTP can easily synchronize clocks to the sub-millisecond level. It can do better with things like hardware timestamps in the NIC.

NETWORK TIME FOUNDATION

PTP

PTP was designed to synchronize time over a bus.

In the simplest configuration, NIC to NIC, time can be synchronized to a handful of nanoseconds. Time transfer degrades somewhat if there is a single PTP-capable switch involved. Time degradation increases with the addition of every additional PTP-capable switch. If any non-PTP-capable switch is used, high-accuracy time transfer is lost.


PTP
Switch 0
PTP
Switch 1
PTP
Grandmaster
PTP
Client 0
PTP
Client 1

NETWORK
TIME FOUNDATION

Convergence…

NTP might take a whole week to converge to within a few milliseconds to a remote server over the Internet. But its servo can do that, slowly averaging out all of the packet delay variation. This works even over WiFi links.

PTP can't do that. At least, not LinuxPTP. The servos assume hardware time stamping and a well behaved LAN, without much packet delay variation.

NETWORK
TIME FOUNDATION

… Convergence

With a well behaved LAN, PTP can converge to a few dozen nanoseconds within a few seconds, by sending many packets. NTP can't do that.

So NTP and PTP operate on the same principles, but they are tuned to different use cases.

With enough effort, NTP could perform as well as PTP in a LAN setting, and PTP could work over the public Internet with WiFi links, just like NTP.

Fun Story #1

Once upon a time (but not all that long ago), in a land far, far away, something like this might have happened:

There was a master time lab that had long used an older computer to serve NTP to the public. By the time it was done booting, communication with the master clock was stable and the box was ready to answer time queries. One day, this server box was upgraded. When they turned it on the new box booted up much faster. It started answering NTP requests before communication with the master clock was ready. For a short while it told folks the time was 1 Jan 1970 at 00:00:xx.

Fun Story #2

An underground optical cable connects a (very good) clock at a NIST building in Boulder Colorado, USA to a laboratory at the nearby University of Colorado. On a hot day the length of this cable grows by 11 mm (less than half an inch). Researchers must monitor the temperature along the cable run and account for the (very measurable) difference this makes. Light travels about 200,000 km/sec (124,274 miles/sec) in an optical cable, so in a nanosecond it travels about 200 mm (8 inches). This clock is so good that on a hot day they can tell that the light in the optical cable takes .05 nanoseconds longer to travel the entire length of the cable. That’s .000 000 000 05 seconds.

NTF Projects

NTF now supports the following:

- The NTP Project
- Ntimed (a new, clean NTP implementation)
- Khronos (provable realtime bounds on valid time)
- LinuxPTP (a PTP implementation for Linux)
- libptpmgmt (a TLV management library for LinuxPTP)
- Several SyncE projects (frequency synchronization)
- General Timestamp API (much better timestamps)

NETWORK
TIME FOUNDATION

Quick Thank-Yous

I’d like to thank:

Laura Jones Crossey, Distinguished Professor of Earth and Planetary Sciences at the University of New Mexico, for helping me with the graphics slides

Pedro Aznar and Tania Maria, for their music. I doubt either of them will ever know I feel this way :)
I could list many more here…

Mills-Spring Fund

The Mills-Spring Fund at Network Time Foundation was established in 2024, with the blessings of his family, to honor and perpetuate the legacy and efforts of David L. Mills, Internet Pioneer and architect of the Network Time Protocol.

Donations to the Mills-Spring Fund assist Network Time Foundation in sponsoring the development and maintenance of the open source timekeeping protocols used daily by billions of users.

https://www.nwtime.org/mills-spring/