PTP @ Meta

Oleg Obleukhov
Production Engineer

Meta

Agenda

Why PTP?

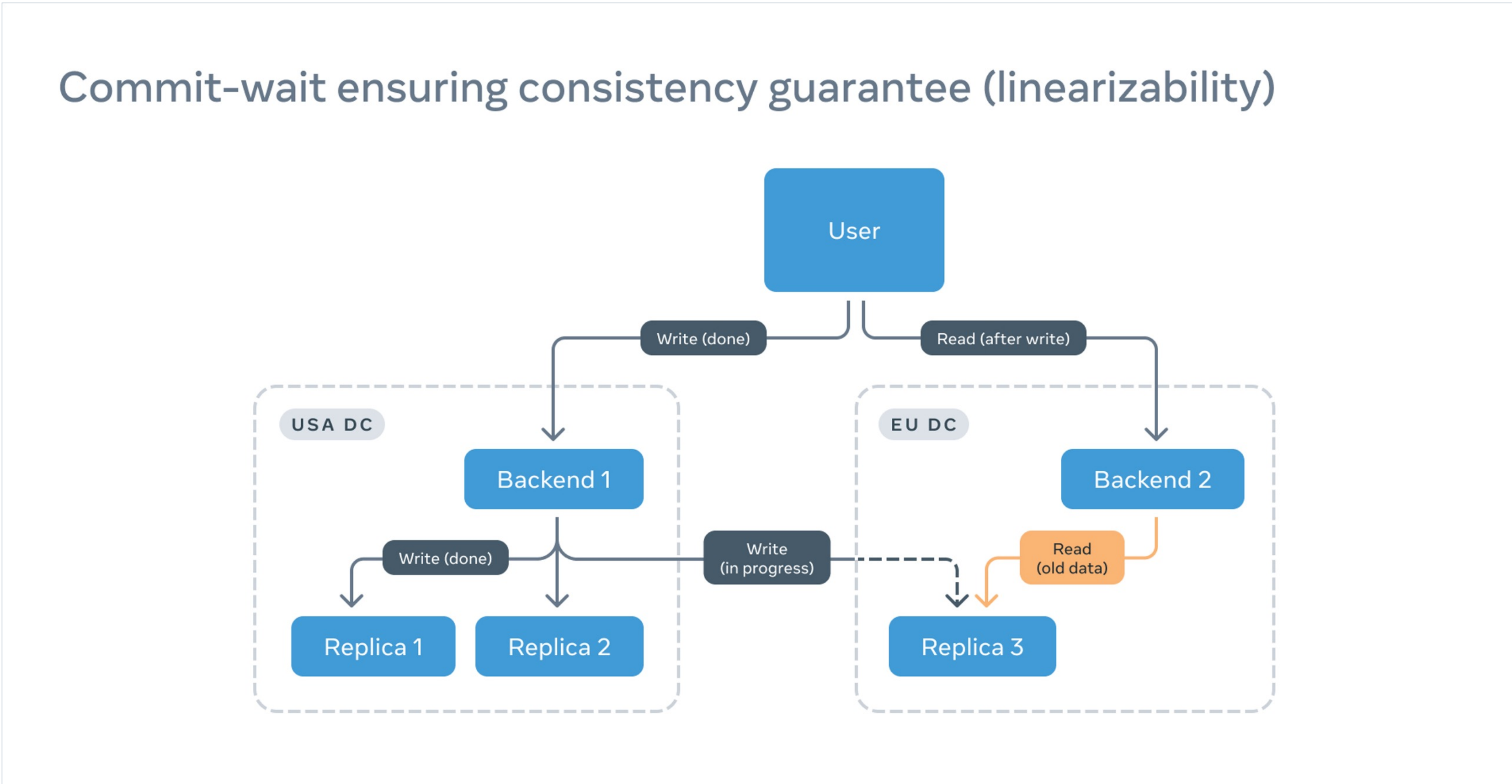
Deploying PTP at scale

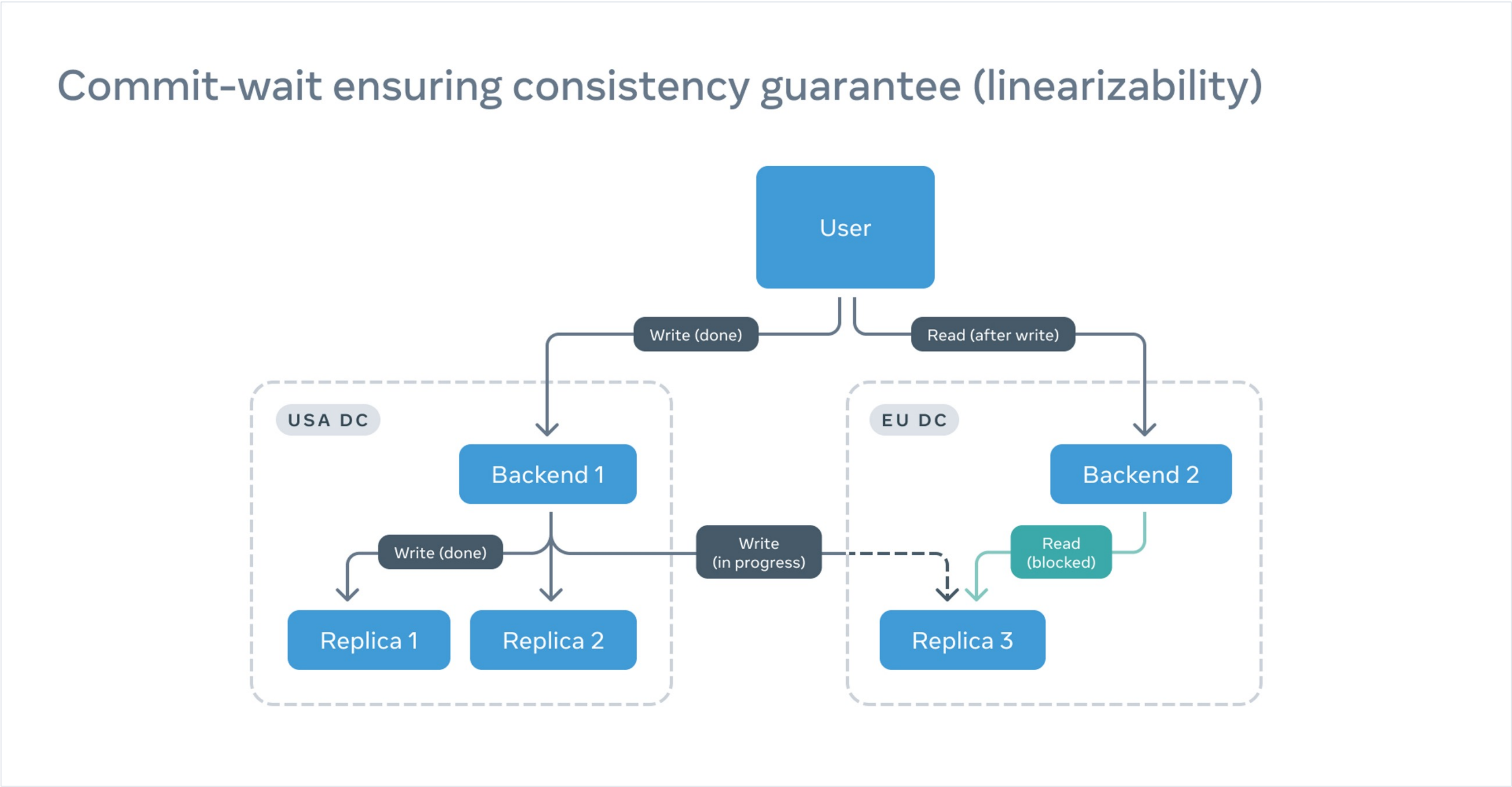
Developing SPTP

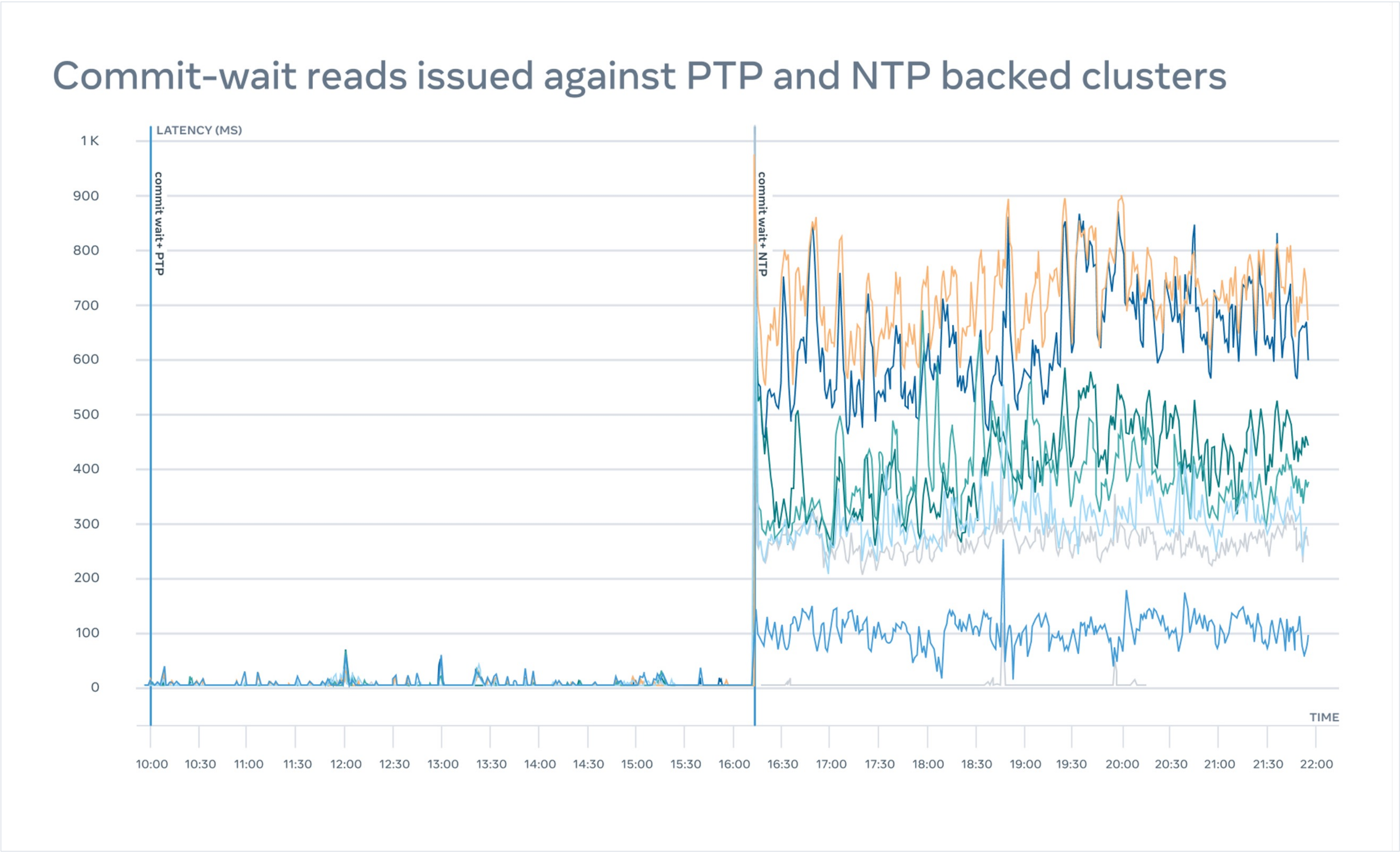
Other usages

01 Why PTP?

01 Why PTP?


Commit-wait ensuring consistency guarantee (linearizability)
User
Write (done)
Read (after write)
USA DC
EU DC
Backend 1
Backend 2
Write (done)
Write
(in progress)
Read
(old data)
Replica 1
Replica 2
Replica 3

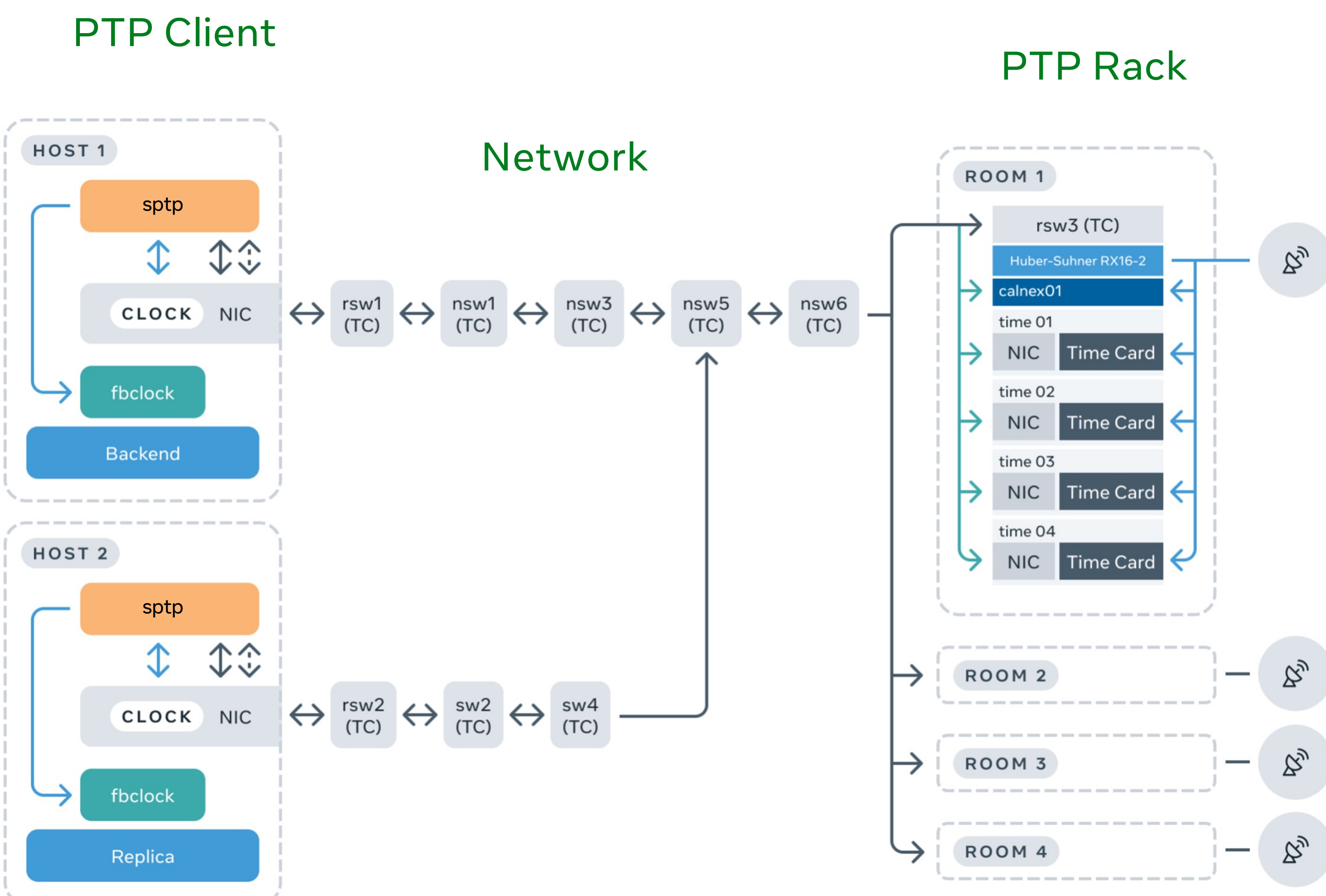

Commit-wait ensuring consistency guarantee (linearizability)
User
Write (done)
Read (after write)
USA DC
EU DC
Backend 1
Backend 2
Write (done)
Write
(in progress)
Read
(blocked)
Replica 1
Replica 2
Replica 3


Commit-wait reads issued against PTP and NTP backed clusters
LATENCY (MS)
1 K
900
800
700
600
500
400
300
200
100
0
commit wait+ PTP
commit wait+ NTP
TIME
10:00
10:30
11:00
11:30
12:00
12:30
13:00
13:30
14:00
14:30
15:00
15:30
16:00
16:30
17:00
17:30
18:00
18:30
19:00
19:30
20:00
20:30
21:00
21:30
22:00

Why PTP

- Read consistency
- Hybrid logical clock (HLC) on scale
- Latency measurement/congestion control
- Event tracing and correlation
- ...

02 Deploying PTP at scale



PTP Client
Network
PTP Rack
HOST 1
sptp
CLOCK
NIC
fbclock
Backend
HOST 2
sptp
CLOCK
NIC
fbclock
Replica
rsw1 (TC)
nsw1 (TC)
nsw3 (TC)
nsw5 (TC)
nsw6 (TC)
rsw2 (TC)
sw2 (TC)
sw4 (TC)
ROOM 1
rsw3 (TC)
Huber-Suhner RX16-2
calnex01
time 01
NIC
Time Card
time 02
NIC
Time Card
time 03
NIC
Time Card
time 04
NIC
Time Card
ROOM 2
ROOM 3
ROOM 4

PTP rack

PTP rack

PTP rack in one of the Meta regions

GNSS antenna in one of the Meta regions

Each rack:

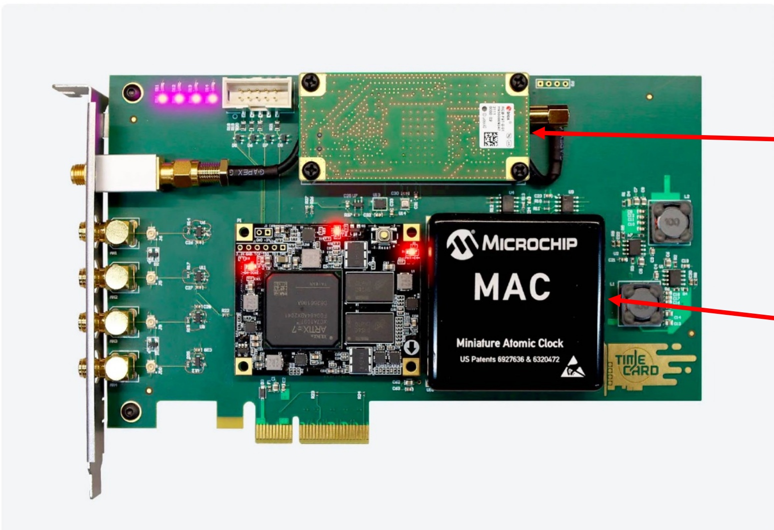
- 4 x Time Appliances
- 1 x Calnex monitoring device
- 1 x Optical antenna
- 1 x TC switch

4 racks per region:

- Independent optical antenna with length compensation
- Independent 2 source power
- Independent monitoring
- Deterministic network distance to the clients (6 hops)

PTP rack

Facebook Time Card



Ublox GNSS receiver supports:

- GPS, Galileo, GLONASS, BeiDou
- 3 independent bandwidths - L1, L2, L5
- Jamming/Spoofing protection
- Operation precision ±12ns

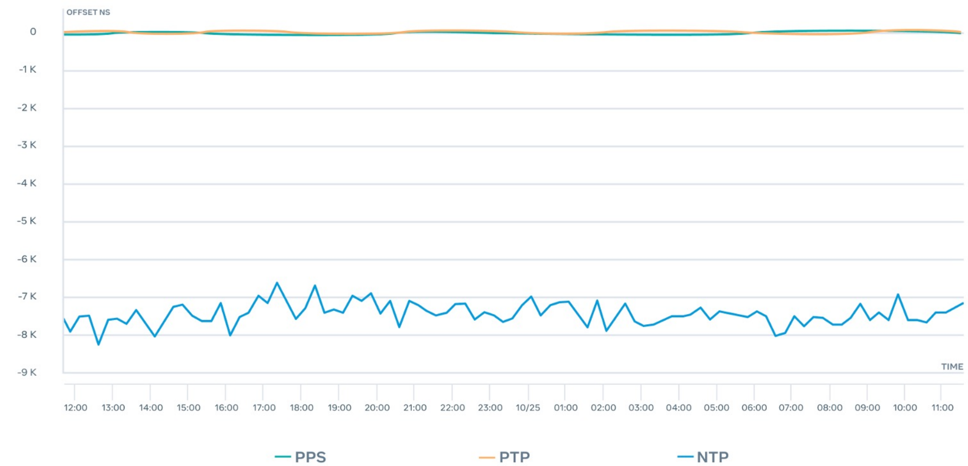
Rubidium Atomic clock ensures <1us / 24 hour drift

- **In practice 1us per 4 days**
- Can run without GNSS for 7 days without breaking an SLA

- 16 appliances per region
- 1 Time Appliance can serve 1.5M QPS
- 1 Time Appliance 100k QPS normal operation

PTP rack

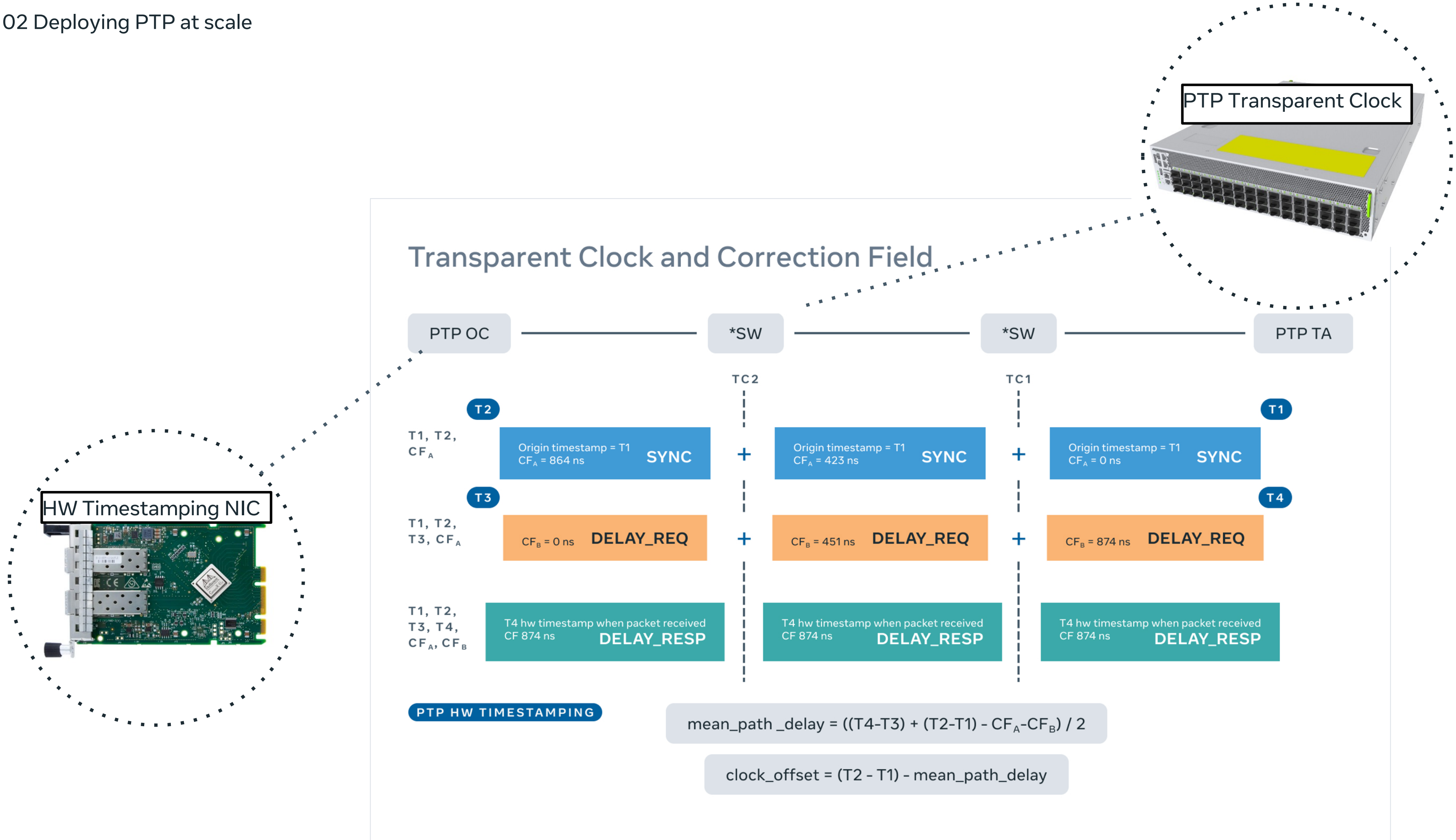
Calnex Sentinel monitoring data



In every rack (4 per region) we have Calnex monitoring solution which is:

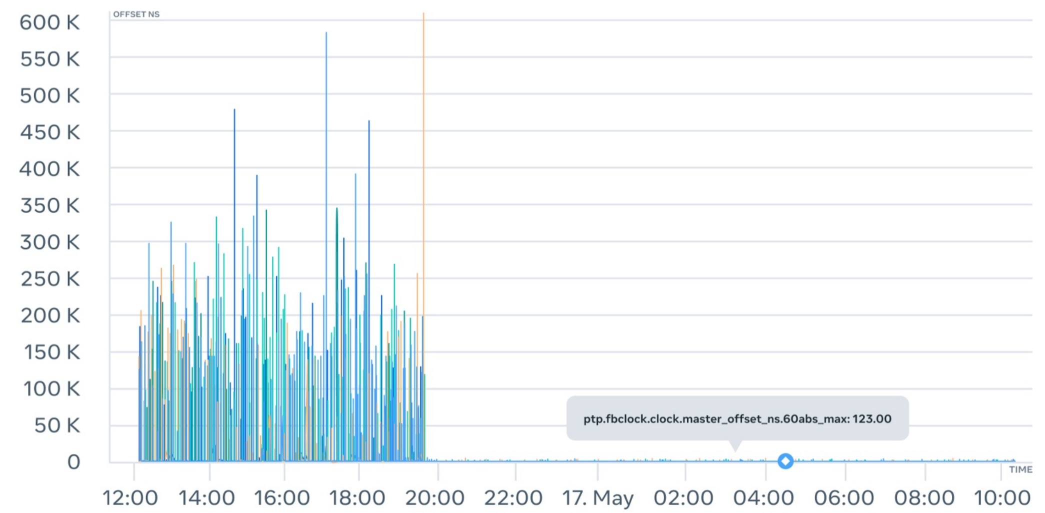
- Testing local Time Appliances with Pulse Per Second (PPS)
- Acting as a PTP and NTP client connected via Network
- Cross checking 3 other racks:
  - Location bias
  - Different network paths
- Exporting data to ODS and Scuba

PTP network


PTP Transparent Clock
Transparent Clock and Correction Field
HW Timestamping NIC
PTP OC
*SW
*SW
PTP TA
TC2
TC1
T2
T1
T1, T2, CF_A
Origin timestamp = T1
CF_A = 864 ns
SYNC
+
Origin timestamp = T1
CF_A = 423 ns
SYNC
+
Origin timestamp = T1
CF_A = 0 ns
SYNC
T3
T4
T1, T2, T3, CF_A
CF_B = 0 ns
DELAY_REQ
+
CF_B = 451 ns
DELAY_REQ
+
CF_B = 874 ns
DELAY_REQ
T1, T2, T3, T4, CF_A, CF_B
T4 hw timestamp when packet received
CF 874 ns
DELAY_RESP
T4 hw timestamp when packet received
CF 874 ns
DELAY_RESP
T4 hw timestamp when packet received
CF 874 ns
DELAY_RESP
PTP HW TIMESTAMPING
mean_path_delay = ((T4-T3) + (T2-T1) - CF_A-CF_B) / 2
clock_offset = (T2 - T1) - mean_path_delay

```
ptp4l[43.662]: offset          -9 s2 freq  -12372 path delay      4114
ptp4l[44.662]: offset          17 s2 freq  -12349 path delay      4114
ptp4l[45.662]: offset          37 s2 freq  -12324 path delay      4078
ptp4l[46.662]: offset         -70 s2 freq  -12420 path delay      4153
ptp4l[47.662]: offset          95 s2 freq  -12276 path delay      4039
ptp4l[48.662]: offset      266776 s2 freq +254434 path delay      4181
ptp4l[49.662]: offset     -430864 s2 freq -363173 path delay    168255
ptp4l[50.662]: offset      -80141 s2 freq -141710 path delay    168255
ptp4l[51.662]: offset      217086 s2 freq +131475 path delay       408
ptp4l[52.662]: offset       16268 s2 freq   -4217 path delay     57459
ptp4l[53.662]: offset        8101 s2 freq   -7504 path delay     57459
ptp4l[54.662]: offset       55912 s2 freq  +42738 path delay      4776
ptp4l[56.305]: offset      -48984 s2 freq  -45385 path delay     19209
ptp4l[56.662]: offset      -37194 s2 freq  -48290 path delay     19209
ptp4l[57.662]: offset       29964 s2 freq   +7710 path delay    -12022
ptp4l[58.662]: offset        9943 s2 freq   -3322 path delay    -12022
ptp4l[59.662]: offset      -19403 s2 freq  -29685 path delay      8279
ptp4l[60.662]: offset        8560 s2 freq   -7543 path delay     -2377
ptp4l[61.662]: offset       -4906 s2 freq  -18441 path delay      6256
ptp4l[62.662]: offset        4197 s2 freq  -10810 path delay      3249
ptp4l[63.662]: offset         979 s2 freq  -12769 path delay      4917
ptp4l[64.662]: offset        1386 s2 freq  -12068 path delay      4917
ptp4l[65.662]: offset        1741 s2 freq  -11297 path delay      4270
ptp4l[66.662]: offset         509 s2 freq  -12007 path delay      4428
ptp4l[67.662]: offset         395 s2 freq  -11968 path delay      4185
ptp4l[68.662]: offset          -7 s2 freq  -12252 path delay      4185
```

Absolute offset values on hosts connected to the switch without Transparent Clock enabled



PTP clients

The PTP client

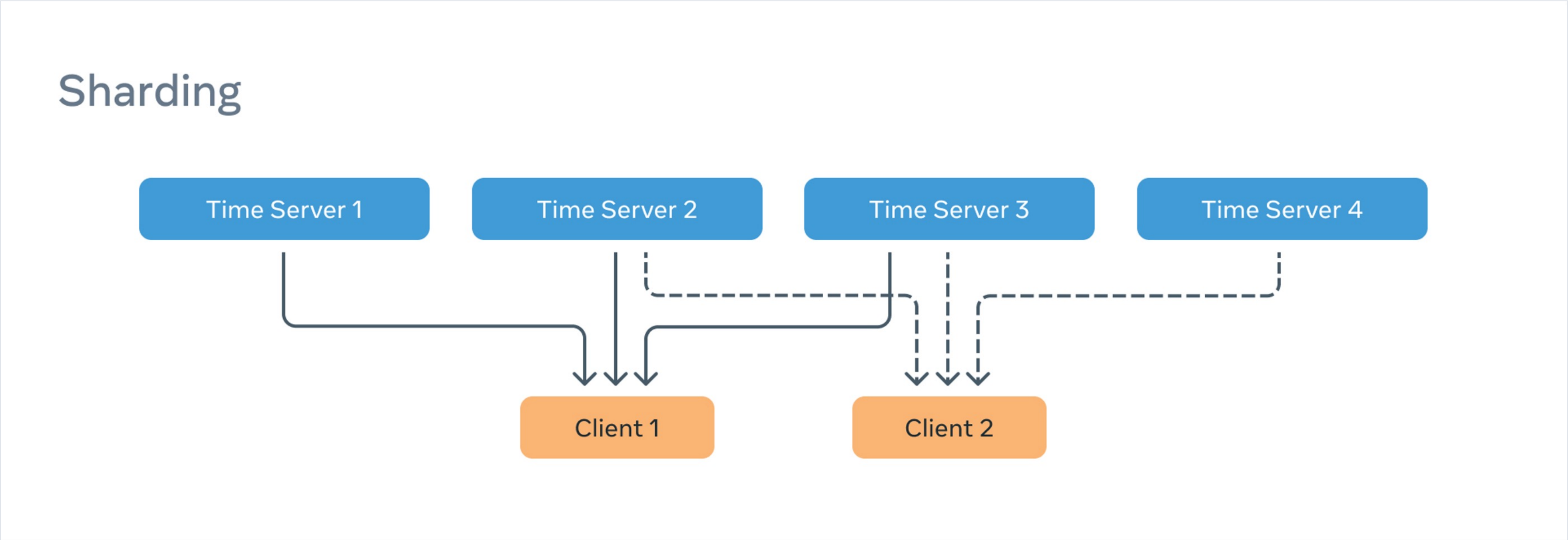
Hardware timestamps

```
$ ethtool -T eth0
Time stamping parameters for eth0:
Capabilities:
        hardware-transmit
        hardware-receive
        hardware-raw-clock
PTP Hardware Clock: 0
Hardware Transmit Timestamp Modes:
        off
        on
Hardware Receive Filter Modes:
        none
        All
```

128 bits	64 bits	64 bits	64 bits
Socket control message header	Software Timestamp	Legacy Timestamp	Hardware Timestamp

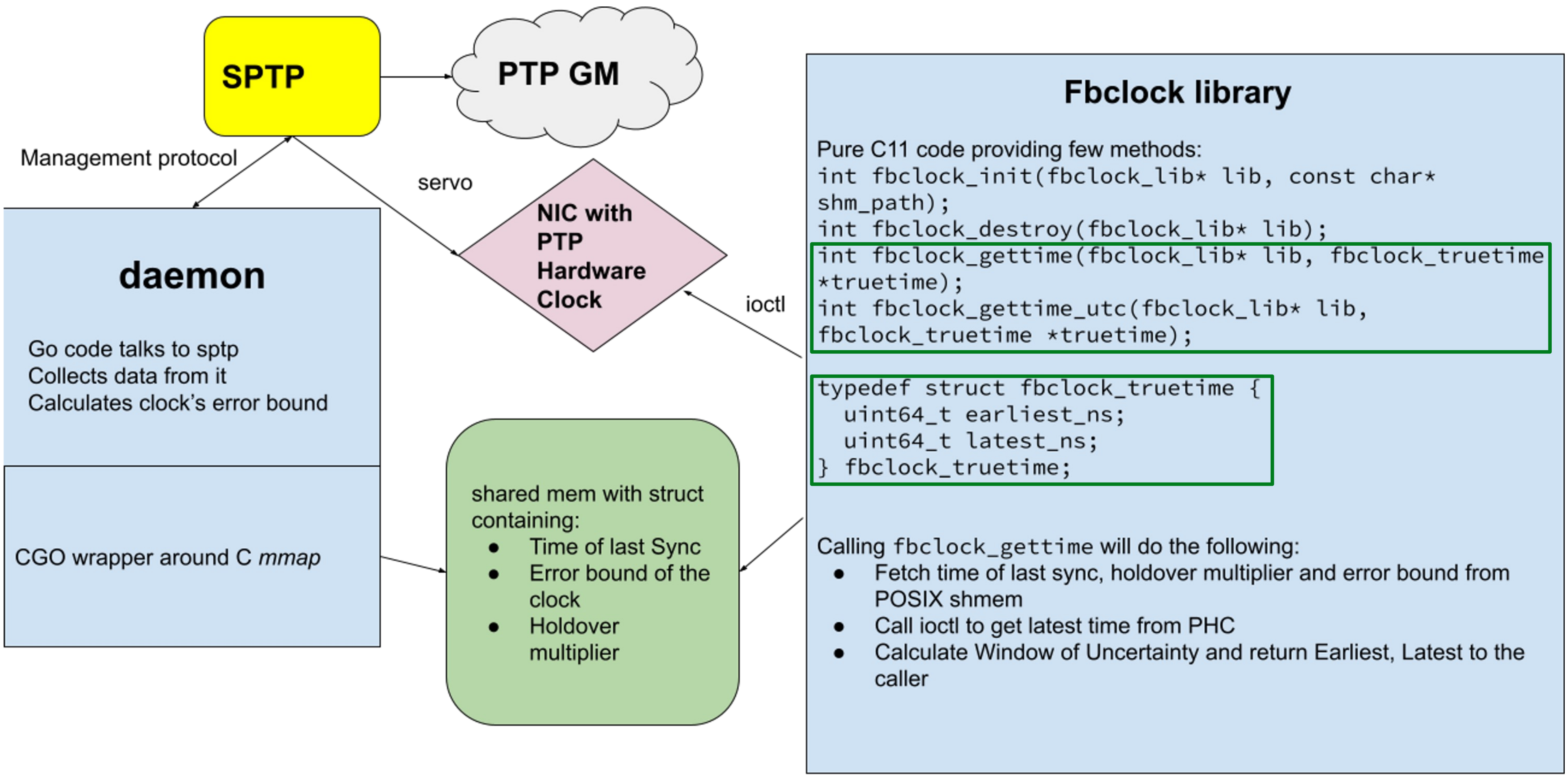
```
ptp4l[40.432]: offset        -16 s2 freq  -13105 path delay      3493
ptp4l[41.432]: offset         -6 s2 freq  -13100 path delay      3493
ptp4l[42.432]: offset          9 s2 freq  -13087 path delay      3493
ptp4l[43.432]: offset         -5 s2 freq  -13098 path delay      3493
ptp4l[44.432]: offset          1 s2 freq  -13093 path delay      3493
ptp4l[45.432]: spike detected => max_offset_locked: 33, setting offset to min_offset_freq_mean: -13065.039314
ptp4l[46.432]: skip 1/15 large offset (>33) 224401
ptp4l[47.432]: offset        -21 s2 freq  -13115 path delay      3493
ptp4l[48.432]: offset          9 s2 freq  -13091 path delay      3493
ptp4l[49.432]: offset         10 s2 freq  -13088 path delay      3493
ptp4l[50.432]: offset         -8 s2 freq  -13103 path delay      3493
```

ptp4l



Schematic representation of sharding

fbclock



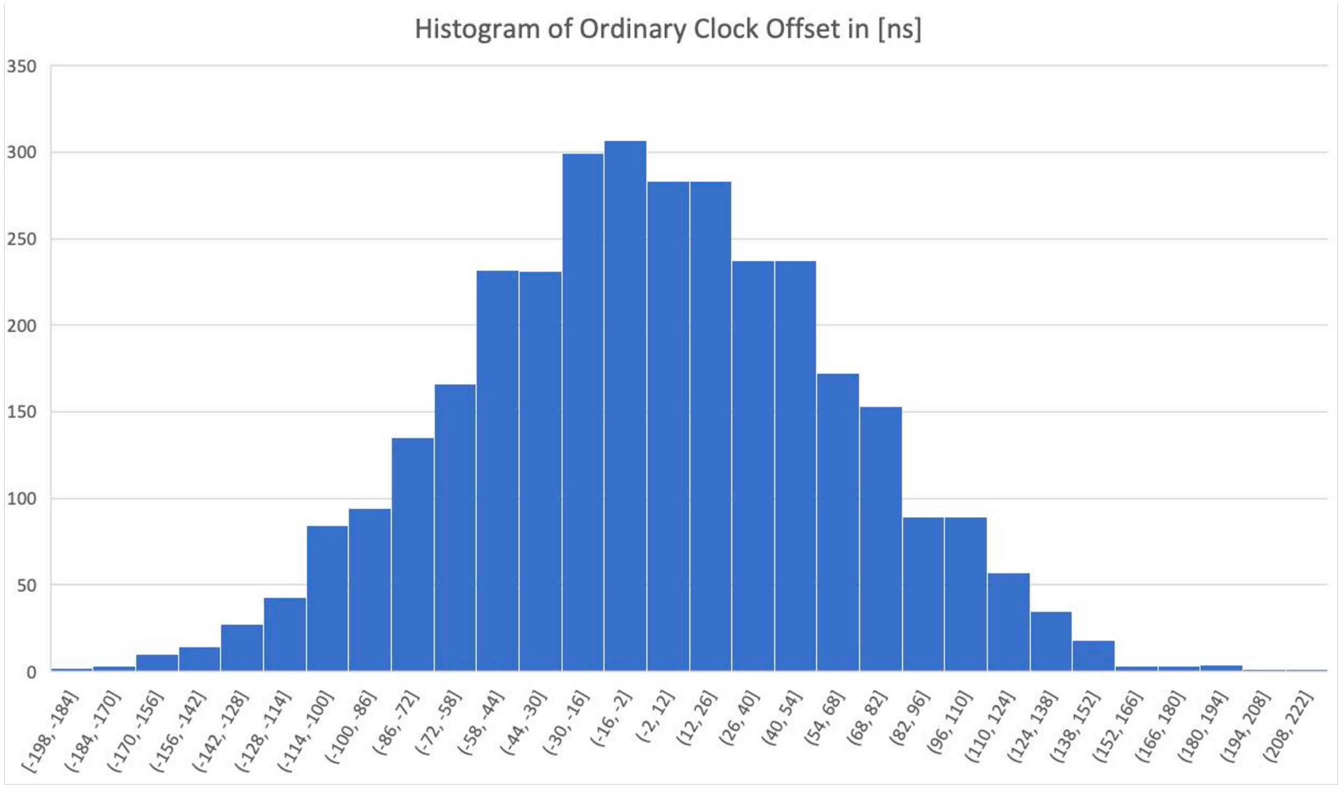
SPTP
PTP GM
Management protocol
servo
NIC with PTP Hardware Clock
ioctl
daemon
Go code talks to sptp
Collects data from it
Calculates clock's error bound
CGO wrapper around C mmap
shared mem with struct containing:
Time of last Sync
Error bound of the clock
Holdover multiplier
Fbclock library
Pure C11 code providing few methods:
int fbclock_init(fbclock_lib* lib, const char* shm_path);
int fbclock_destroy(fbclock_lib* lib);
int fbclock_gettime(fbclock_lib* lib, fbclock_truetime *truetime);
int fbclock_gettime_utc(fbclock_lib* lib, fbclock_truetime *truetime);
typedef struct fbclock_truetime {
uint64_t earliest_ns;
uint64_t latest_ns;
} fbclock_truetime;
Calling fbclock_gettime will do the following:
Fetch time of last sync, holdover multiplier and error bound from POSIX shmem
Call ioctl to get latest time from PHC
Calculate Window of Uncertainty and return Earliest, Latest to the caller

fbclock

`Estimated E2E Variance = [GNSS Variance + MAC Variance + ts2phc Variance] + [PTP4L Offset Variance] = [Time Server Variance] + [Ordinary Clock Variance]`

Estimated E2E Variance = (12ns ^ 2) + (43ns ^ 2) + (52ns ^2) + (61ns ^2) = 8418 which corresponds to 91.7 ns

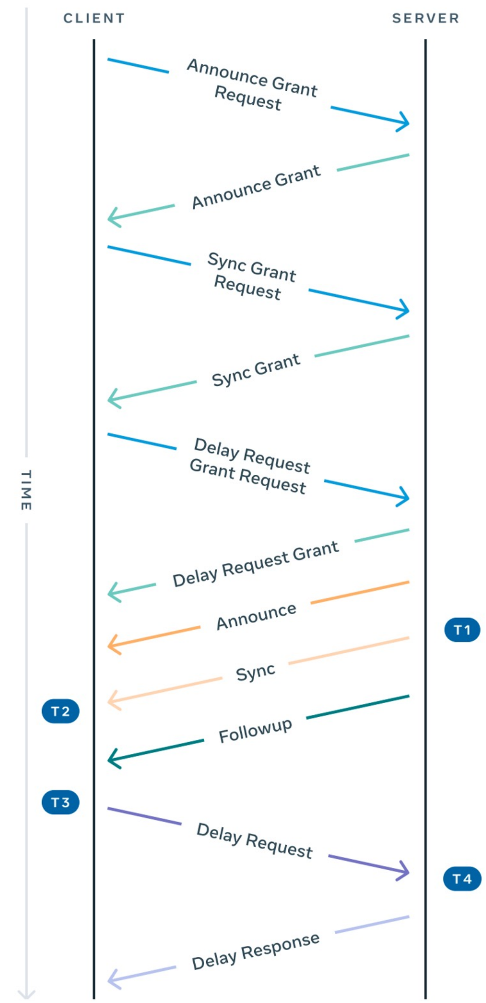
$$Var\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} Var(X_i)$$



03 Developing SPTP

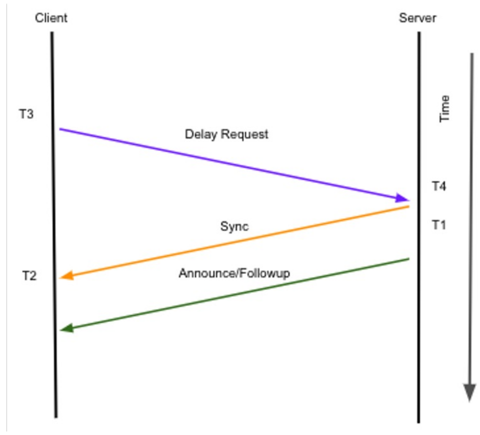
Two-step PTP exchange

- Excessive network communication
- Multicast support requirement for large numbers of clients
- Unicast support has strict capacity limit
- State maintenance on both server and client side
- Individual clients have no control over the communication parameters
- Server driven decision



Simple PTP

- Client sends a Delay Request
- Server responds with a Sync
- Server sends Followup/Announce

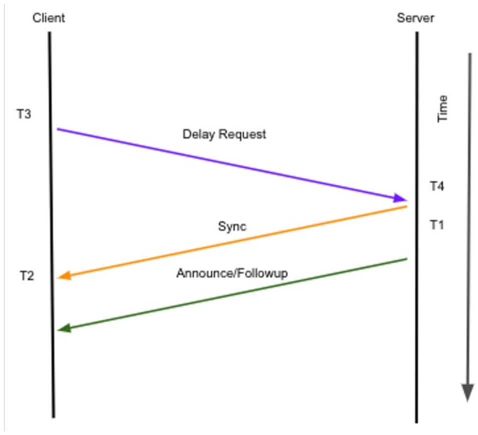


mean_path_delay = ((T4 - T3) + (T2-T1) - CF1 - CF2)/2

clock_offset = T2 - T1 - mean_path_delay

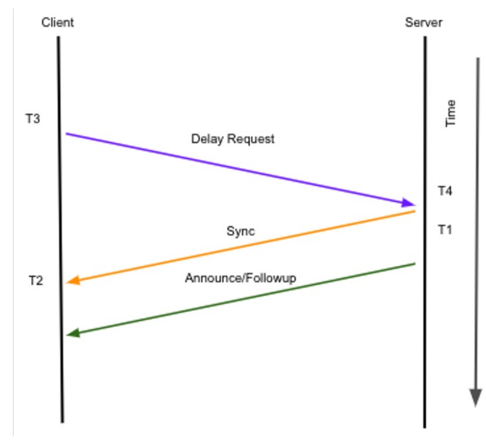
Simple PTP

1. Client sends *DELAY_REQ* effectively initiating an exchange with the Server. The Client records timestamp T3
2. Server records CF_2 from *DELAY_REQ*
3. Server records the RX timestamp T4
4. Server sends *SYNC*. The server adds timestamp T4 in the `originTimestamp` field and records the TX timestamp T1
5. Server sends *ANNOUNCE* with a TX timestamp T1 of the *SYNC* in `originTimestamp` field and CF_2 from *DELAY_REQ* in a `correctionField`.
6. Client records T2 of the received *SYNC* packet, and also CF_1
7. Client records data from *ANNOUNCE* packet, and also CF_2.



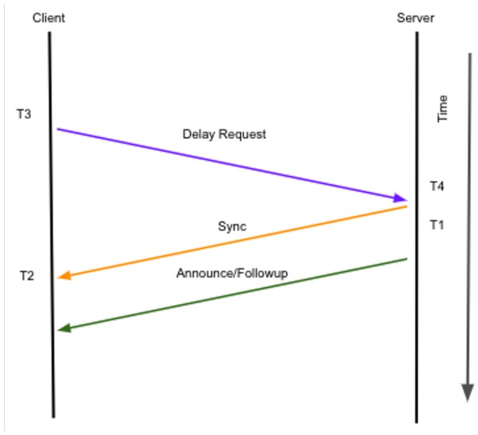
Delay Request

```
Precision Time Protocol (IEEE1588)
    .... 0001 = messageId: Delay_Req Message (0x1)
    .... 0010 = versionPTP: 2
    messageLength: 44
    flags: 0x2400
        0... .... .... .... = PTP_SECURITY: False
        .0.. .... .... .... = PTP profile Specific 2: False
        ..1. .... .... .... = PTP profile Specific 1: True
        .... .1.. .... .... = PTP_UNICAST: True
```



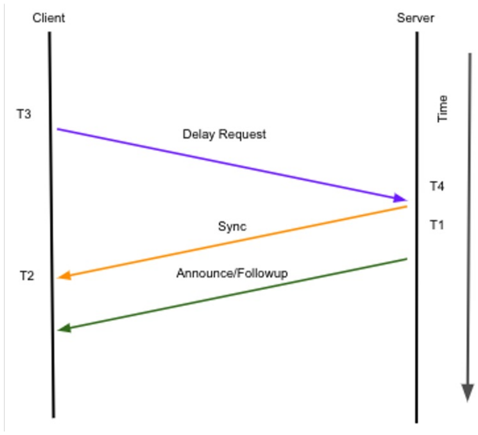
Sync

```
Precision Time Protocol (IEEE1588)
    .... 0000 = messageId: Sync Message (0x0)
    .... 0010 = versionPTP: 2
    messageLength: 44
    correction: 5468.812592 nanoseconds
        correction: Ns: 5468 nanoseconds
        correctionSubNs: 0.812591552734375 nanoseconds
    originTimestamp (seconds): 1695066968
    originTimestamp (nanoseconds): 900335434
```

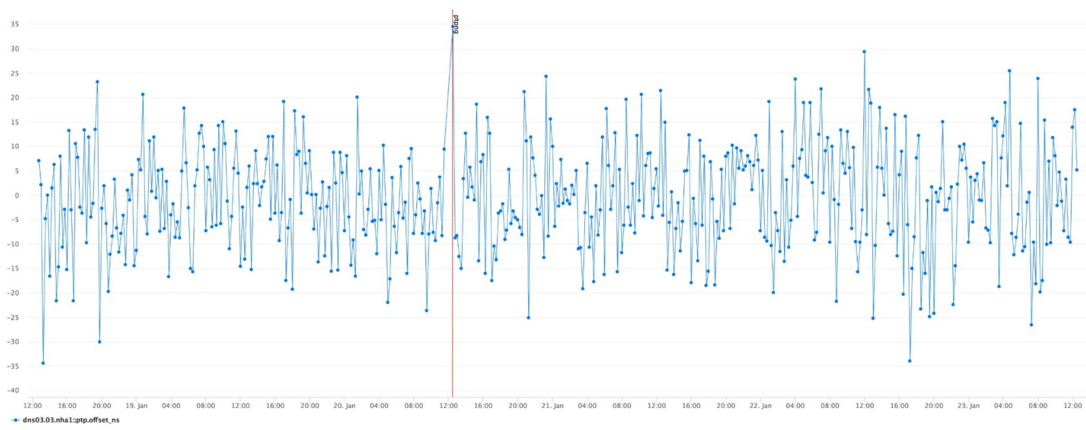


Announce

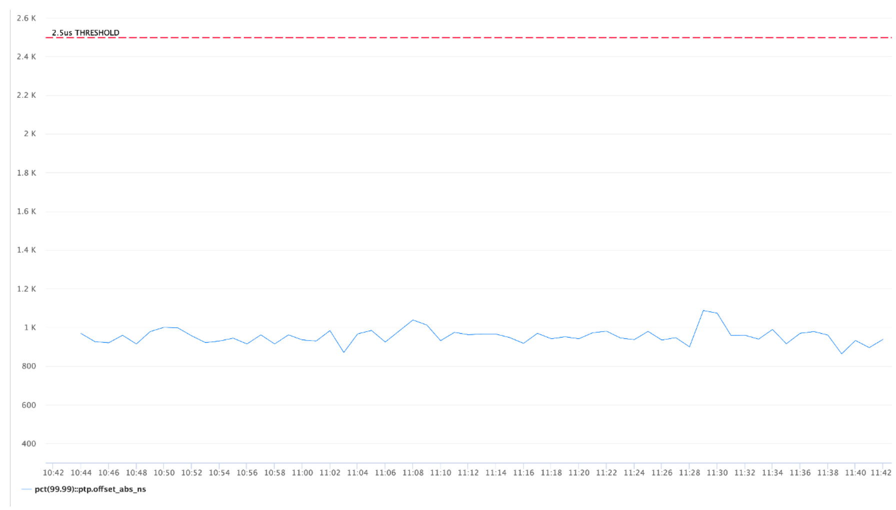
```
Precision Time Protocol (IEEE1588)
    .... 1011 = messageId: Announce Message (0xb)
    .... 0010 = versionPTP: 2
    messageLength: 64
    correction: 5586.562592 nanoseconds
        correction: Ns: 5586 nanoseconds
        correctionSubNs: 0.562591552734375 nanoseconds
    originTimestamp (seconds): 1695066968
    originTimestamp (nanoseconds): 900409436
    originCurrentUTCOffset: 37
    priority1: 128
    grandmasterClockClass: 6
    grandmasterClockAccuracy: The time is accurate to within 100 ns (0x21)
    grandmasterClockVariance: 23008
    TimeSource: GPS (0x20)
```



03 Developing SPTP

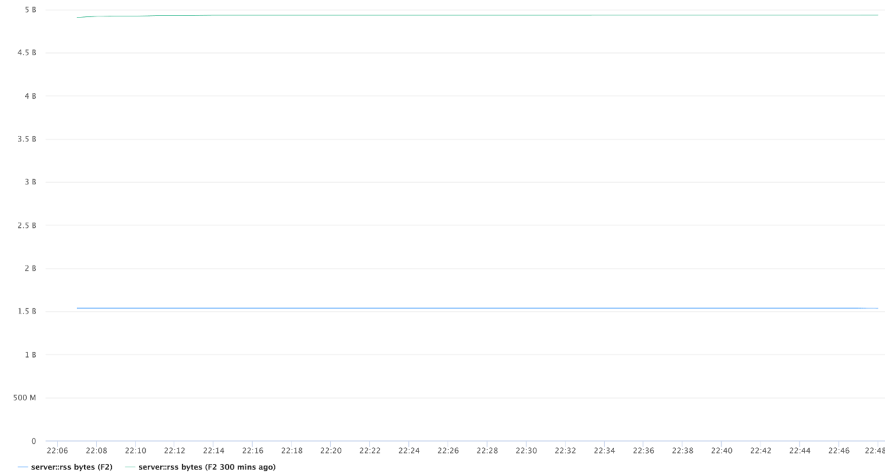


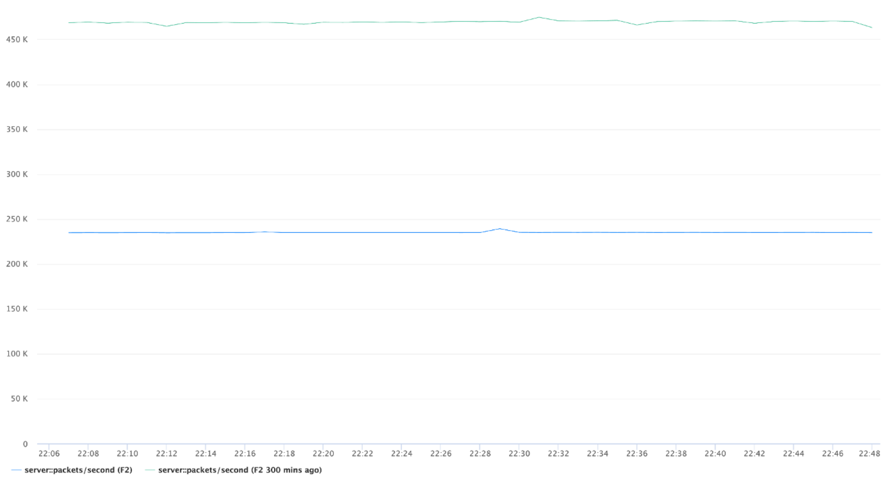
Type	Number	Validation
PTP Servers	16	Calnex Sentinel
Transparent Clocks	5000	Calnex Sentinel Calnex Neo
PTP Client	> 100000	SPTP logs Calnex Sentinel Spirent N4U

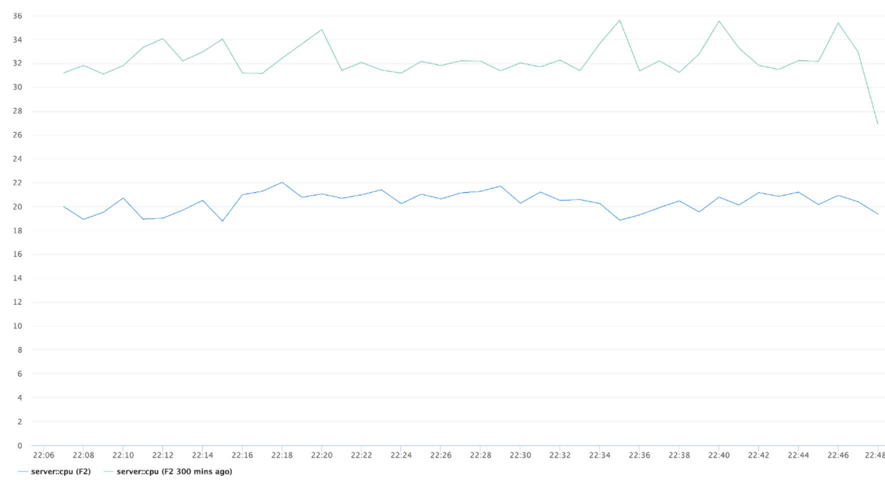


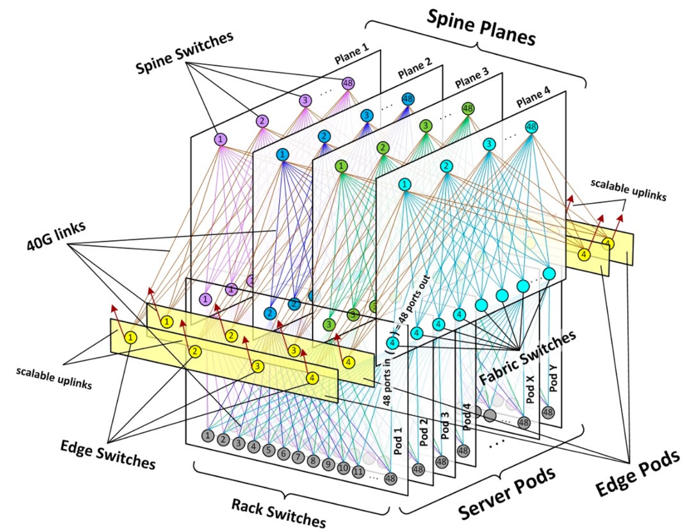
Low resource consumption

- Significant reduction in memory (70%) and CPU (40%) consumption
- Up to 50% reduction in network utilization
- 1.5 million clients per second per server



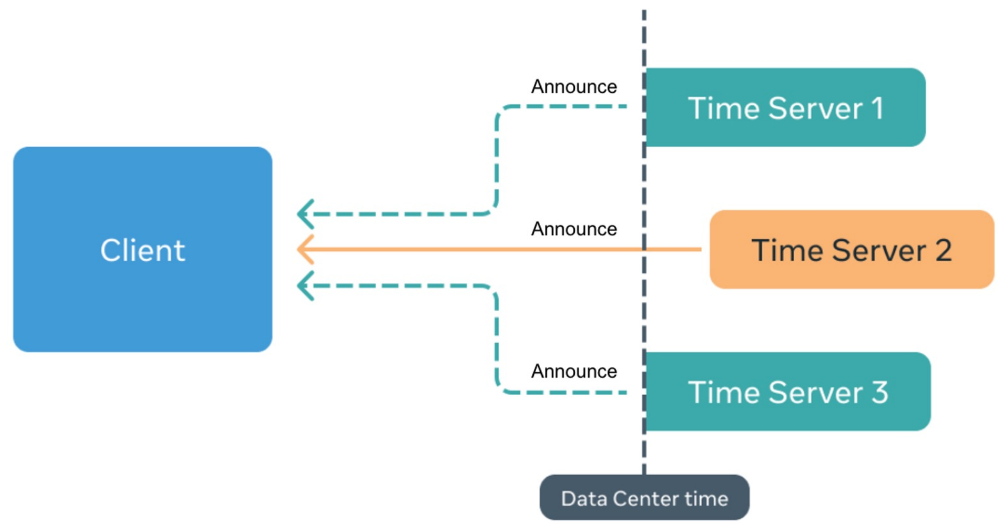






Strong reliability and fault tolerance

- Every exchange is concurrent but independent
- Forward and reverse exchange happen as close as possible
- Path delay is calculated every exchange
- BMCA works



Client driven exchange

- Client controls frequency and duration.
- Synchronous communication
- No “remaining” sync messages

No state

- Simple implementation
- Restart any time

Simple implementation

- No state, transitions etc
- Basic implementation <1000 LOC

No negotiation between client and a server

- No handshake
- Fast start time (1.5 rtt)

Low resource consumption

- Up to 70% reduction in memory and CPU consumption
- Up to 50% network utilization improvement

Using existing underlying hardware support

- Hardware timestamping on NIC works
- Transparent clock works

Less flow control

- Impossible to set different intervals for different types of messages
- Sync and Follow Up/Announce are always bound together

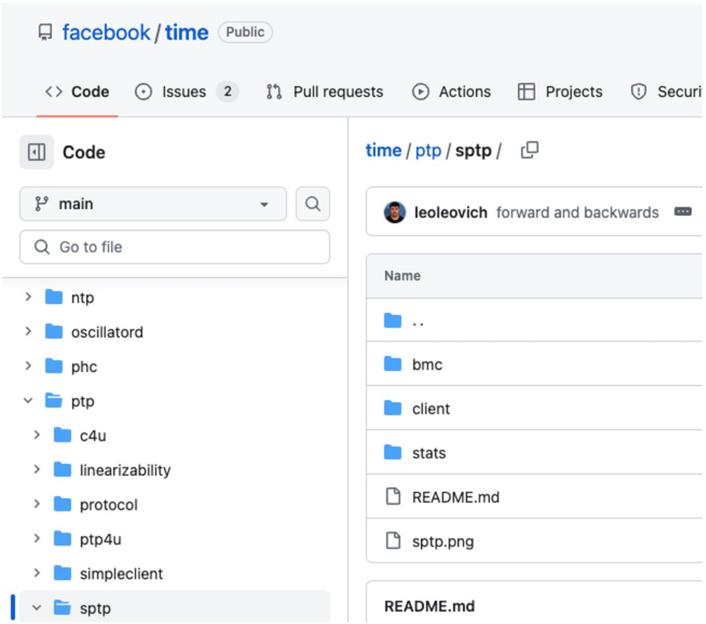
No multicast support

- SPTP is based on unicast and makes no sense with multicast
- Can’t offload work to switches

No negotiation between client and a server

- Less flexibility in negotiation (TLVs are still possible)
- Authentication extensions may reduce performance gains

- Written in modern popular language (Go)
- Client/Server is open sourced
- Good test coverage
- https://github.com/facebook/time/tree/main/ptp

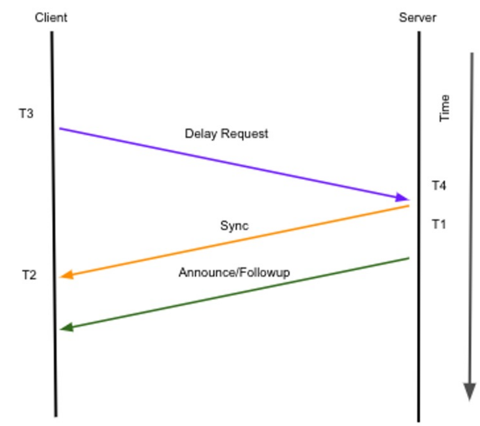


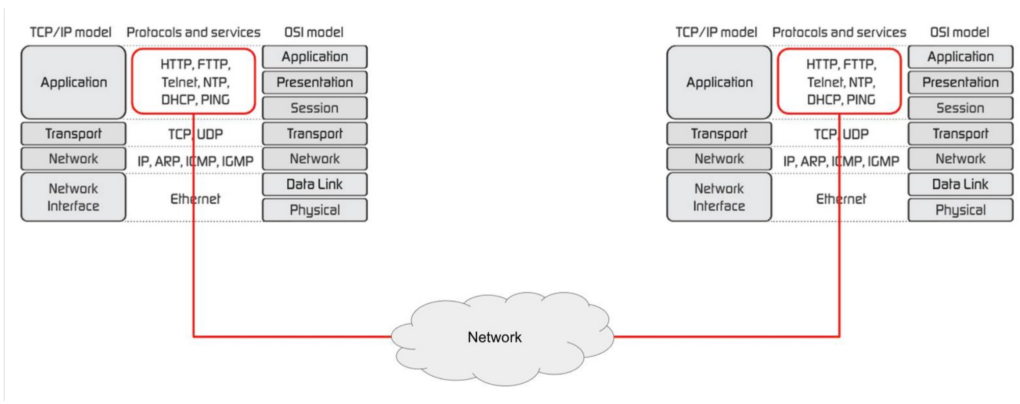
https://sptp.info

04 Other usages

04 Other usages

```
[root@host1 ~]# ping host2
PING host2(host2 (2401:db00::1)) 56 data bytes
64 bytes from host2 (2401:db00::1): icmp_seq=1 ttl=118 time=0.084 ms
64 bytes from host2 (2401:db00::1): icmp_seq=2 ttl=118 time=0.092 ms
64 bytes from host2 (2401:db00::1): icmp_seq=3 ttl=118 time=0.137 ms
64 bytes from host2 (2401:db00::1): icmp_seq=4 ttl=118 time=0.100 ms
64 bytes from host2 (2401:db00::1): icmp_seq=5 ttl=118 time=0.174 ms
```






```
[root@host1 ~]# ptping host2
host2: seq=1 time=38.07µs (->23.767µs + <-14.303µs)
host2: seq=2 time=38.185µs(->23.688µs + <-14.497µs)
host2: seq=3 time=38.033µs(->23.703µs + <-14.33µs )
host2: seq=4 time=38.037µs(->23.698µs + <-14.339µs)
host2: seq=5 time=38.023µs(->23.655µs + <-14.368µs)
```

Announce (quality)

Meta