Running PTP at scale

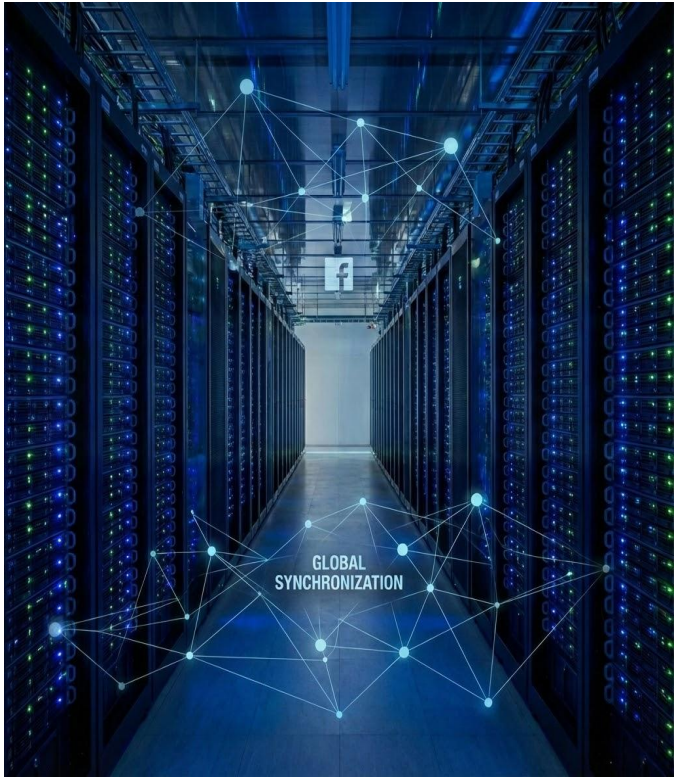
LINUX PLUMBERS 2025
Tokyo, Japan

Vadim Fedorenko
Software Engineer

Meta

Why do we need PTP?

- Distributed DBs (HLC)
- Event tracing and correlation
- Congestion control
- …



What is the scale?

- Millions of machines…
- ~10us - precision of time provided

What is the scale?

- Millions of machines…
- ~10us - precision of time provided

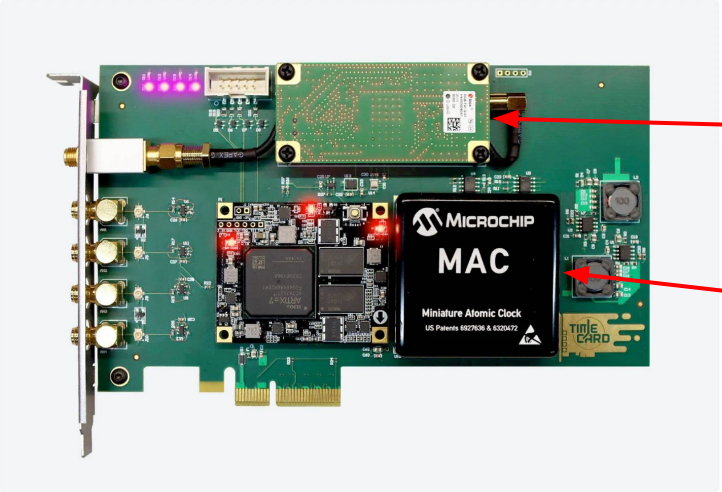
What do we run?

- OCP Time Appliances with Time Cards
- PTP4U - Server Side (Grand Master)
- SPTP - Client Side
- FBClock - library to get

  Window-of-Uncertainty

https://github.com/facebook/time

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

- Time Appliance can serve 1.5M QPS

PTP rack

PTP rack in one of the Meta regions

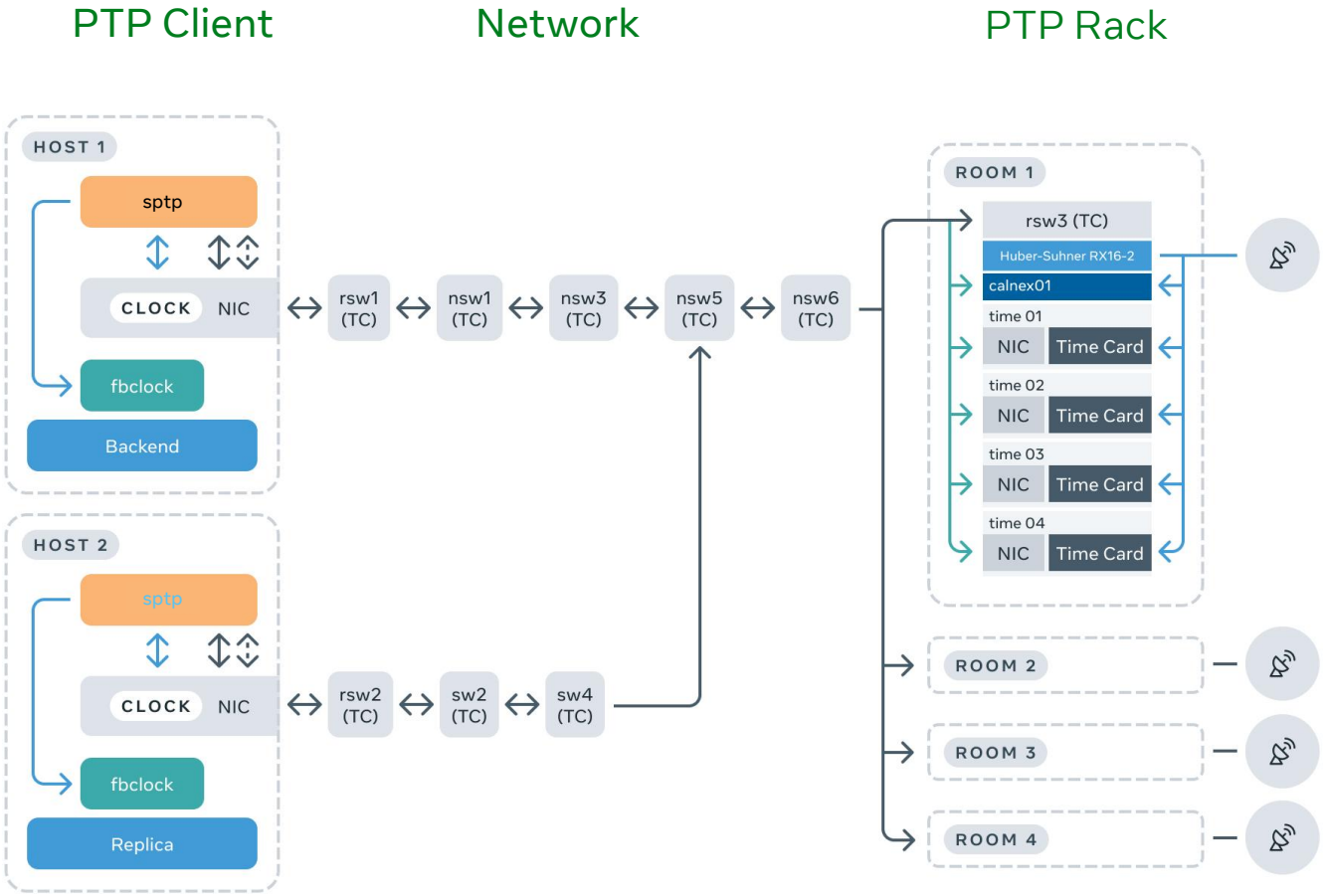
GNSS antenna in one of the Meta regions

Each rack:

- Time Appliances
- Calnex monitoring device
- Optical antenna

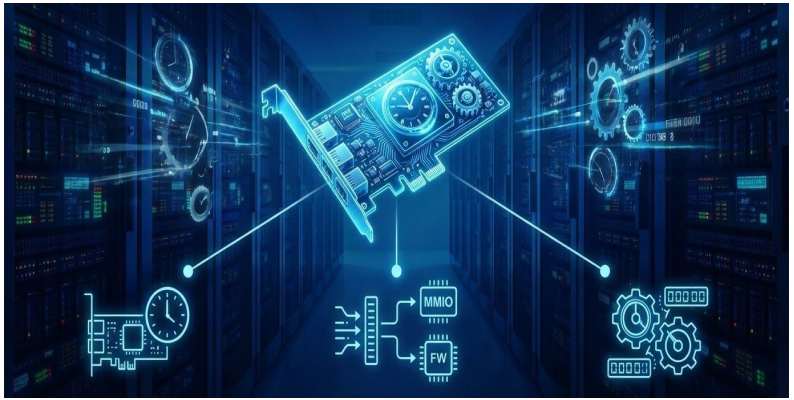
Several racks per region:

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


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

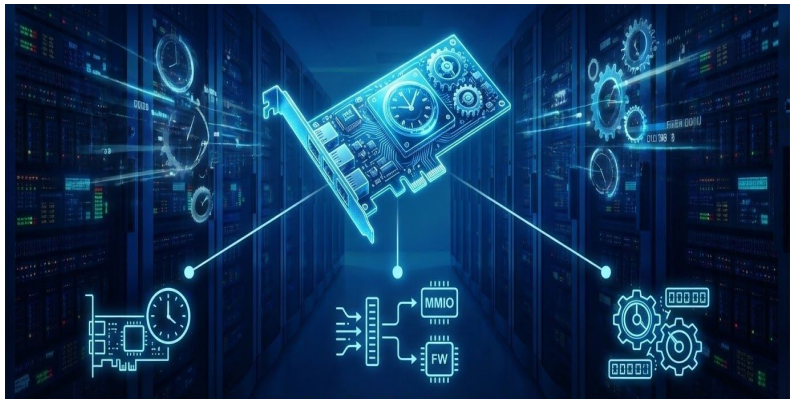
What is PHC (PTP Hardware Clock)?

- NIC with Timestamp Counter
- Exported via Memory-mapped registers
  - or via special FW command
- Cyclecounter and timecounter structures



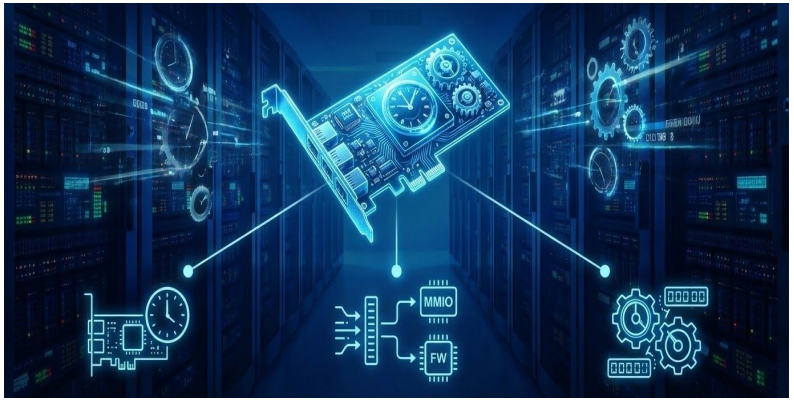
What is PHC (PTP Hardware Clock)?

- NIC with Timestamp Counter
- Exported via Memory-mapped registers
  - or via special FW command
- Cyclecounter and timecounter structures
- Set of callbacks:
  - gettime64()/gettimex64()
  - adjtime()/adjfreq()/adjphase()
  - settime64()
  - getcrosststamp() (optional)



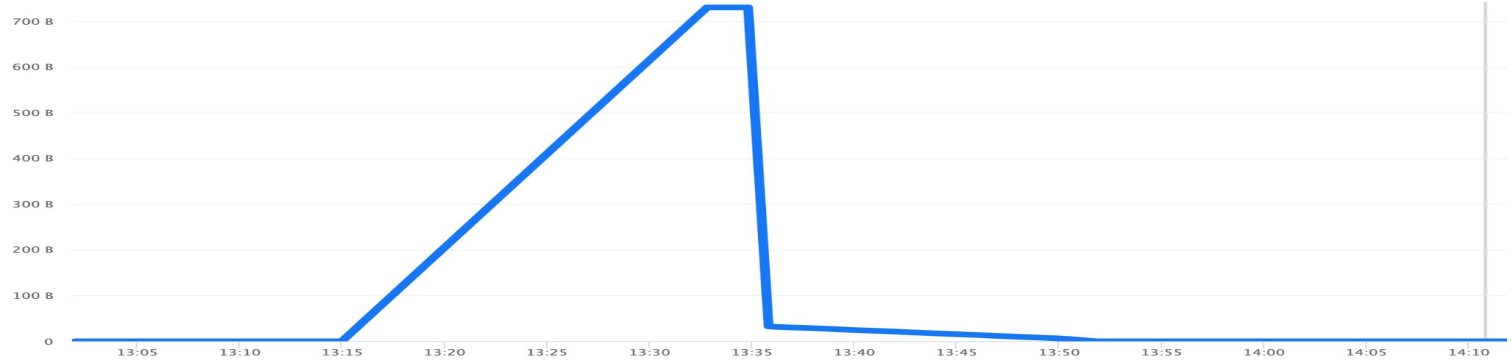
What is PHC (PTP Hardware Clock)?

- NIC with Timestamp Counter
- Exported via Memory-mapped registers
  - or via special FW command
- Cyclecounter and timecounter structures
- Set of callbacks:
  - gettime64()/gettimex64()
  - adjtime()/adjfreq()/adjphase()
  - settime64()
  - getcrosststamp() (optional)
- Watchdog to check overflows and to keep timecounter accurate



Time counter keeper

Time counter keeper




```
I110012:18:02.298608 sptp.go:468] offset        -34 servo LOCKED freq  -29548 path delay 3181
I110012:18:03.406484 sptp.go:468] offset         20 servo LOCKED freq  -29576 path delay 3181
I110012:18:04.397362 sptp.go:468] offset          9 servo LOCKED freq  -29580 path delay 3181
I110012:18:05.396749 sptp.go:468] offset        -18 servo LOCKED freq  -29556 path delay 3181
W110012:18:14.507007 sptp.go:373] tick took 9110ms, which is outside of expected +-10% from
the interval 1000ms
I110012:18:14.507126 sptp.go:468] offset -8589934415 servo FILTER freq  -29576 path delay 3181
I110012:18:15.507043 sptp.go:468] offset -8589934427 servo FILTER freq  -29576 path delay 3181
I110012:18:16.507340 sptp.go:468] offset -8589934391 servo FILTER freq  -29576 path delay 3181
```

Time counter keeper

```
INIT_DELAYED_WORK(&timer->overflow_work, mlx5_timestamp_overflow);
if (timer->overflow_period)
    schedule_delayed_work(&timer->overflow_work, 0);
```

The watchdog runs on system workqueue…

Time counter keeper

```
struct ptp_clock_info {
        struct module *owner;
        char name[PTP_CLOCK_NAME_LEN];
        .....
        long (*do_aux_work)(struct ptp_clock_info *ptp);
};
```

The issue was fixed:

e61e6c415ba9 net/mlx5: use do_aux_work for PHC overflow checks

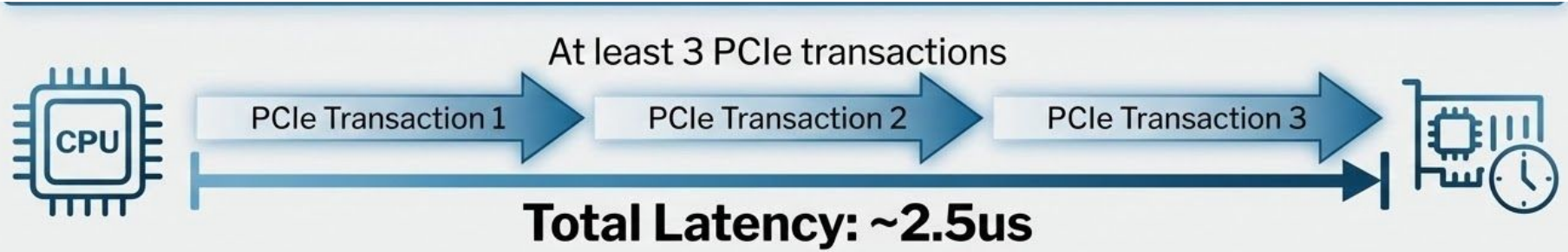
Get time from PHC

Get time from PHC

```
/* read high dword twice to detect overrun */
do {
        high = ioread32(dev->addr + SYSTEM_TIME_HIGH);
        ptp_read_system_prets(sts);
        low = ioread32(dev->addr + SYSTEM_TIME_LOW);
        ptp_read_system_postts(sts);
} while (high != ioread32(dev->addr + SYSTEM_TIME_HIGH));
system_time = (((u64)high) << 32) | ((u64)low);
```

Get time from PHC

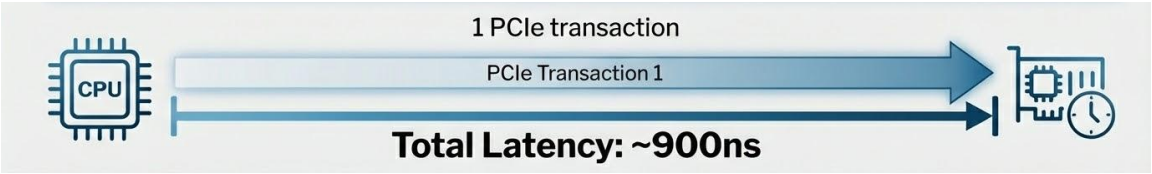
```
/* read high dword twice to detect overrun */
do {
        high = ioread32(dev->addr + SYSTEM_TIME_HIGH);
        ptp_read_system_prets(sts);
        low = ioread32(dev->addr + SYSTEM_TIME_LOW);
        ptp_read_system_postts(sts);
} while (high != ioread32(dev->addr + SYSTEM_TIME_HIGH));
system_time = (((u64)high) << 32) | ((u64)low);
```



Get time from PHC

```
ptp_read_system_prets(sts);
ts = ioread32(dev->addr + SYS_TIME_LOW);
ptp_read_system_postts(sts);
time = (u64)READ_ONCE(dev->old_time) << SYS_TIME_SHIFT;
cycles = (time & SYS_TIME_HIGH_MASK) | ts;
if (ts < (time & SYS_TIME_LOW_MASK))
        cycles += SYS_TIME_LOW_MASK + 1;
```

Optimize by keeping high bits cached in the driver private data and using **timecounter_cyc2time()** with reading low part only - **<1us**



Get time from PHC

An example of how that can be done in the driver:

```
bb2ef9b92bdf bnxt_en: cache only 24 bits of hw counter
c7a21af711e8 bnxt_en: optimize gettimex64
```

Even faster with the latest patch from Eric Dumazet (queued for post 6.19-rc1):

```
<hashid> time/timecounter: inline timecounter_cyc2time()
```

Serializing racy timecounter

Serializing racy timecounter

5.97%	**[kernel]**	**[k] native_queued_spin_lock_slowpath**
1.80%	[kernel]	[k] cpuidle_enter_state
1.63%	[kernel]	[k] __netif_receive_skb_core.constprop.0
1.51%	[kernel]	[k] __raw_spin_lock_irqsave
1.46%	[kernel]	[k] __mod_memcg_state

Serializing racy timecounter

5.97%	**[kernel]**	**[k] native_queued_spin_lock_slowpath**
1.80%	[kernel]	[k] cpuidle_enter_state
1.63%	[kernel]	[k] __netif_receive_skb_core.constprop.0
1.51%	[kernel]	[k] __raw_spin_lock_irqsave

In the worst case scenario it is really bad:

55.78%	**[kernel]**	**[k] native_queued_spin_lock_slowpath**
3.10%	[kernel]	[k] intel_idle_xstate
1.94%	[kernel]	[k] bnxt_get_rx_ts_p5
1.70%	[kernel]	[k] tcp_gro_receive
1.36%	[kernel]	[k] bnxt_rx_pkt
1.30%	[kernel]	[k] _raw_spin_lock_bh

Serializing racy timecounter

```
spin_lock_irqsave(&ptp->ptp_lock, flags);
ns = timecounter_cyc2time(&ptp->tc, ts);
spin_unlock_irqrestore(&ptp->ptp_lock, flags);
```

Serializing racy timecounter

```
spin_lock_irqsave(&ptp->ptp_lock, flags);
ns = timecounter_cyc2time(&ptp->tc, ts);
spin_unlock_irqrestore(&ptp->ptp_lock, flags);
```

```
do {
        seq = read_seqbegin(&ptp->ptp_lock);
        ns = timecounter_cyc2time(&ptp->tc, ts);
} while (read_seqretry(&ptp->ptp_lock, seq));
```

Serializing racy timecounter

```
do {
        seq = read_seqbegin(&ptp->ptp_lock);
        ns = timecounter_cyc2time(&ptp->tc, ts);
} while (read_seqretry(&ptp->ptp_lock, seq));
```

Full fix can be found in:

```
6c0828d00f07 bnxt_en: replace PTP spinlock with seqlock
```

Re-implementation of FIFOs

Re-implementation of FIFOs

The list of patches with fixes:

3a50cf1e8e51 mlx5: fix possible ptp queue fifo use-after-free

e435941b1da1 mlx5: fix skb leak while fifo resync and push

3178308ad4ca net/mlx5e: Make tx_port_ts logic resilient to out-of-order CQEs

Re-implementation of FIFOs

The list of patches with fixes:

3a50cf1e8e51 mlx5: fix possible ptp queue fifo use-after-free

e435941b1da1 mlx5: fix skb leak while fifo resync and push

3178308ad4ca net/mlx5e: Make tx_port_ts logic resilient to out-of-order CQEs

.....

c790275b5edf bnxt_en: fix atomic counter for ptp packets

8f7ae5a85137 bnxt_en: improve TX timestamping FIFO configuration

1e7962114c10 bnxt_en: Restore PTP tx_avail count in case of skb_pad() error

Next steps

Next steps

- HW Timestamp Configuration tests

Next steps

- HW Timestamp Configuration tests
- E2E tests with HW timestamps
- Regression tests?


Meta