Precision Time Protocol on Linux
~ Introduction to linuxptp

Ken ICHIKAWA
FUJITSU LIMITED.
LinuxCon Japan 2014
Agenda

- Background
- Overview of Precision Time Protocol (PTP)
- About PTP on Linux
- Tips
- For easy trial or development
Background

- Event ordering is very important
  - for incident analysis, performance analysis and so on
- Event ordering is based on timestamps
- Timestamps are collected from multiple servers
  ⇒ Clock synchronization is important
- If precision and accuracy of clock synchronization are bad, event ordering can reverse against actual time
NTP is not enough

- NTP provides millisecond level synchronization
  - Maybe enough for remote machines, but not enough for locally cooperating machines
  - Many events occur in a millisecond in multiple servers
    $\Rightarrow$ Event ordering will frequently reverse

- Need another protocol
  - Higher precision and accuracy
  - Not need to synchronize large area, but local servers and devices
Example of wrong event ordering

Event Ordering based on Actual Time

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Event 1</td>
</tr>
<tr>
<td>5</td>
<td>Event 2</td>
</tr>
<tr>
<td>7</td>
<td>Event 3</td>
</tr>
</tbody>
</table>

Event Ordering based on Timestamp

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Event 1</td>
</tr>
<tr>
<td>6</td>
<td>Event 3</td>
</tr>
<tr>
<td>8</td>
<td>Event 2</td>
</tr>
</tbody>
</table>

reverse!
Example of correct event ordering based on better clock synchronization

---

**Actual Time**

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

**Server A**

<table>
<thead>
<tr>
<th>diff: -0.001</th>
<th>Event 1</th>
<th>Event 2</th>
<th>Event 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.999</td>
<td>2.999</td>
<td>4.999</td>
<td>6.999</td>
</tr>
<tr>
<td>3.999</td>
<td>5.999</td>
<td>7.999</td>
<td></td>
</tr>
</tbody>
</table>

**Server B**

<table>
<thead>
<tr>
<th>diff: +0.003</th>
<th>Event 1</th>
<th>Event 2</th>
<th>Event 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.003</td>
<td>3.003</td>
<td>5.003</td>
<td>7.003</td>
</tr>
<tr>
<td>4.003</td>
<td>6.003</td>
<td>8.003</td>
<td></td>
</tr>
</tbody>
</table>

**Event Ordering based on Actual Time**

- Time 3: Event 1
- Time 5: Event 2
- Time 7: Event 3

**Event Ordering based on Timestamp**

- Time 2.999: Event 1
- Time 5.003: Event 2
- Time 6.999: Event 3

*correct!*
Agenda

- Background
- **Overview of Precision Time Protocol (PTP)**
  - What’s PTP
  - Term explanation
  - About packet timestamp
- About PTP on Linux
- Tips
- For easy trial or development
Precision Time Protocol (PTP)

- Standardized protocol, IEEE1588
- Synchronize the clocks in local computing systems and devices
- Microsecond to sub-microsecond accuracy and precision
- Administration free
  - Capability to autonomously decide time server (master)
    - called Best Master Clock Algorithm (BMCA)
Term explanation

- An example of PTP network
Grandmaster Clock (Ordinary Clock)

- Original time source for the PTP network
- Typically synchronize its clock to external time (GPS, NTP and so on)
- End point of PTP network is called Ordinary Clock
Term explanation

**Boundary Clock**

- Typically it’s a switch
- Synchronize its clock to a master
- Serve as a time source to other (slave) clocks
- May become Grandmaster clock if current Grandmaster is lost

- Master: serve as a time source
- Slave: synchronize to another clock
Slave Clock (Ordinary Clock)

- Synchronize its clock to a master (to the boundary clock in this example)
- May become Grandmaster clock if current Grandmaster is lost
Packet timestamp

- Time offset between master and slave clocks is calculated based on timestamps at packet sending and receiving

  \[ offset = t_2 - t_1 - \frac{1}{2}(t_{ms} + t_{sm}) \]

  \[ = t_2 - t_1 - \frac{1}{2} \{(t_4 - t_1) - (t_3 - t_2)\} \]

- Packet timestamp accuracy is important for PTP
Ideally, we want timestamps of the time just sending (or receiving) packet.

But in reality, there is deference between timestamp timing and packet sending (or receiving) timing.
Type of timestamping

- Software timestamping
  - Timestamp at Application or OS layer
  - Get time from system clock
  - Error is relatively huge

Software Timestamping

```
<table>
<thead>
<tr>
<th>Application</th>
<th>OS</th>
<th>MAC</th>
<th>PHY</th>
</tr>
</thead>
</table>
```

timestamp
Sys Clock

transmit
error
To Achieve High Precision

- **Hardware Timestamping**
  - Hardware assisted timestamp at PHY or MAC layer
  - Get time from PTP Hardware Clock (PHC) on NIC
  - Minimize error

---

**Software Timestamping**

- Application
  - OS
    - MAC
    - PHY

- timestamp

- transmit

---

**Hardware Timestamping**

- Application
  - OS
    - MAC
    - PHY

- timestamp

- PHC

- transmit

- small error
Agenda

- Background
- Overview of Precision Time Protocol (PTP)
- About PTP on Linux
  - Kernel features
  - User-land application: Linuxptp
- Tips
- For easy trial or development
Linux kernel assists PTP

- The protocol itself is implemented on user-land

- Kernel features for PTP
  - Socket option SO_TIMESTAMPING for packet timestamping
  - PHC subsystem
    - Allow to access PHC via clock_gettime/settime/adjtime system calls
  - Some drivers support Hardware and/or Software timestamping (e.g. e1000e, igb, ixgbe, and so on)
The Linux PTP Project

- Project developing user-land applications for PTP
- Maintainer: Richard Cochran
  - He has implemented many Linux kernel features for PTP
  - Linuxptp is reliable and correctly using the kernel features for PTP
- Red Hat, Intel, SUSE, Fujitsu, etc. have been participating the development
Linuxptp Applications

- ptp4l
  - Implementation of PTP (Ordinary Clock, Boundary Clock)

- phc2sys
  - Synchronize two clocks (typically PHC and system clock)

- pmc (PTP Management Client)
  - Send PTP management messages to PTP nodes
Implementation of PTP

- Ordinary / Boundary clock
- Hardware / Software timestamping
- Delay request-response / Peer delay mechanism
- IEEE 802.3 (Ethernet) / UDP IPv4 / UDP IPv6 network transport
Synchronize two clocks (typically PHC and system clock)

When you are using Hardware timestamping:
- ptp4l adjusts PHC
- phc2sys adjusts system clock
How about software timestamping

When you are using Software timestamping:

- ptp4l directly adjusts system clock
- phc2sys is not needed
Typical usage of ptp4l

- Start as a slave node
- Use eth0 to send/receive messages
- Use /etc/ptp4l.conf as configuration file

# ptp4l -i eth0 -f /etc/ptp4l.conf -s

Specify network interface to use.
Specify slave only mode. Otherwise, this node can be master.
Specify configuration file to use. Otherwise, default configuration is used.
Observe synchronization of ptp4l

- Log is handed over to syslog
- Or, you can print it into stdout by using –m option

<table>
<thead>
<tr>
<th>Port</th>
<th>Status</th>
<th>Offset</th>
<th>Frequency</th>
<th>Path Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INITIALIZING to LISTENING</td>
<td>-3318 s0 freq +0</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>UNCALIBRATED to SLAVE</td>
<td>389 s2 freq</td>
<td>533</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>INITIALIZING to LISTENING</td>
<td>-3343 s1 freq -8378</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>UNCALIBRATED to SLAVE</td>
<td>2344 s2 freq -10722</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>INITIALIZING to LISTENING</td>
<td>-2344 s2 freq -10722</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>UNCALIBRATED to SLAVE</td>
<td>-18 s2 freq -9099</td>
<td>545</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>INITIALIZING to LISTENING</td>
<td>641 s2 freq -8446</td>
<td>513</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>UNCALIBRATED to SLAVE</td>
<td>570 s2 freq -8324</td>
<td>533</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>INITIALIZING to LISTENING</td>
<td>-3343 s1 freq -8378</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>UNCALIBRATED to SLAVE</td>
<td>-2344 s2 freq -10722</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>INITIALIZING to LISTENING</td>
<td>-3318 s0 freq +0</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>UNCALIBRATED to SLAVE</td>
<td>389 s2 freq</td>
<td>533</td>
<td></td>
</tr>
</tbody>
</table>

Offset between Master and Slave (PHC)
Typical usage of phc2sys

- Adjust system clock based on eth0’s PHC
- Wait until ptp4l starts synchronization to the master

By specifying network interface to –s option, related PHC is automatically selected. Or, you can directly specify PHC like –s /dev/ptp0

```
# phc2sys -s eth0 -c CLOCK_REALTIME -w
```

Specify the clock you want to adjust. CLOCK_REALTIME is system clock.

Wait until ptp4l’s synchronization.
Send PTP management messages to PTP nodes
- GET action: Get current values of data
- SET action: Update current values of variables
- CMD action: Initiate some events

PTP management messages are specified in IEEE1588

Many PTP devices have not supported management messages yet
- Also linuxptp has not supported many SET and CMD messages yet
Typical usage of `pmc`

- Send a message to localhost’s node
- Get values of `CURRENT_DATA_SET`

```
# pmc -u -b 0 'GET CURRENT_DATA_SET'
```

- Indicates to use Unix Domain Socket. UDS is used to receive PTP management messages from localhost.

- `-b` specifies allowance number of boundary hops. In this case, management messages is sent only localhost.
An example of synchronization between Linux servers

- Directly connect two Linux servers (Grandmaster and Slave)
- Use hardware timestamping

Setup

Linux server (Grandmaster)  ---  Linux server (Slave)
An example of synchronization between Linux servers

- The offsets between the PHCs, observed by ptp4l

```
max 116 ns
min -89 ns
RMS 31.19 ns
```

very stable!

```
offset (ns)
```

```
elapsed time (s)
```
Agenda

- Background
- Overview of Precision Time Protocol (PTP)
- About PTP on Linux
- Tips
  - Workaround against bad GM behavior
  - Improve system clock stability
- For easy trial or development
Bad GM behavior

- I encountered a Grandmaster product
- The GM sometimes occurs a few hundred microsecond level errors

Setup

GM product (Grandmaster)  ----------------  Linux server (Slave)
Bad GM behavior

- The offsets between the GM and the Linux server’s PHC, observed by ptp4l

max 352129 ns (= 352 us)
min -231644 ns (= -232 us)
RMS 13664.71 ns (= 14 us)
What’s happening?

- GM sends a timestamp including huge error
  - It appears as a huge plus offset
- ptp4l changes PHC’s frequency too much depending on the offset

![Graph showing offset (ns) vs. elapsed time (s)]
What’s happening?

- GM’s next timestamp does not include so much error
- But, PHC’s frequency was changed too much
  ⇒ A huge minus offset appears against normal GM’s timestamp

![Graph showing offset (ns) vs elapsed time (s) between GM and PHC]
Observe the influence to PHC

- The offsets between PHC and system clock observed by phc2sys
  - There are similar offsets
  \[ \Rightarrow \text{Introduce worse system clock stability} \]

How to avoid this issue?
ptp4l has servo mechanism

- ptp4l has PI (proportional-integral) controller servo
  - A kind of feedback loop
  - Determine frequency set to PHC

- Kp and Ki are tuning parameters (proportional gain and integral gain)

\[ P = K_p \cdot e(t) \]
\[ I = K_i \int e(\tau) \, d\tau \]

Figure: Basic block of Proportional + Integral controller. (excerpt from wikipedia)
Tuning sensitivity

- Anyway, by tuning the servo parameters, we can adjust how clock sensitivity react to the offset.
- To change the configuration, edit ptp4l’s configuration file:
  - `pi_proportional_const` for Kp
  - `pi_integral_const` for Ki

Default configuration file is `/etc/ptp4l.conf` in Fedora
There are two default configurations

- For hardware timestamping
  - $K_p$ 0.7
  - $K_i$ 0.3
  $\Rightarrow$ Sensitive

- For software timestamping
  - $K_p$ 0.1
  - $K_i$ 0.001
  $\Rightarrow$ Insensitive

Previous result used this configuration $\Rightarrow$ Sensitive

Try this one to prevent over-reacting
Use software timestamping config

- We tried software timestamping configuration though ptp4l used hardware timestamping

- Minus direction offsets disappeared
  - PHC’s frequency is not changed too much

<table>
<thead>
<tr>
<th>offset (ns)</th>
<th>elapsed time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-120000</td>
<td>-70000</td>
</tr>
<tr>
<td>-20000</td>
<td>-80000</td>
</tr>
<tr>
<td>30000</td>
<td>130000</td>
</tr>
<tr>
<td>1</td>
<td>127</td>
</tr>
<tr>
<td>127</td>
<td>253</td>
</tr>
<tr>
<td>253</td>
<td>379</td>
</tr>
<tr>
<td>379</td>
<td>505</td>
</tr>
<tr>
<td>505</td>
<td>631</td>
</tr>
<tr>
<td>631</td>
<td>757</td>
</tr>
<tr>
<td>757</td>
<td>883</td>
</tr>
<tr>
<td>883</td>
<td>1009</td>
</tr>
<tr>
<td>1009</td>
<td>1135</td>
</tr>
<tr>
<td>1135</td>
<td>1261</td>
</tr>
<tr>
<td>1261</td>
<td>1387</td>
</tr>
<tr>
<td>1387</td>
<td>1513</td>
</tr>
<tr>
<td>1513</td>
<td>1639</td>
</tr>
<tr>
<td>1639</td>
<td>1765</td>
</tr>
<tr>
<td>1765</td>
<td>1891</td>
</tr>
<tr>
<td>1891</td>
<td>2017</td>
</tr>
<tr>
<td>2017</td>
<td>2143</td>
</tr>
<tr>
<td>2143</td>
<td>2269</td>
</tr>
<tr>
<td>2269</td>
<td>2395</td>
</tr>
<tr>
<td>2395</td>
<td>2521</td>
</tr>
<tr>
<td>2521</td>
<td>2647</td>
</tr>
<tr>
<td>2647</td>
<td>2773</td>
</tr>
<tr>
<td>2773</td>
<td>2899</td>
</tr>
</tbody>
</table>

Minus direction offsets disappeared between GM and PHC
Observe from phc2sys

PHC looks stable
⇒ system clock will be also stable

between PHC and sys clock
Agenda

- Background
- Overview of Precision Time Protocol (PTP)
- About PTP on Linux
- Tips
  - Countermeasure against bad GM behavior
  - Improve system clock stability
- For easy trial or development
Improve system clock stability

- Dynamic ticks make system clock stability worse
  - Dynamic ticks disable periodic timer tick interrupt
  - It is a useful feature to power saving but...
  - Error correction mechanism in kernel doesn’t aware dynamic ticks

- You can disable dynamic ticks
  - Specify nohz=off in kernel boot option
  - (nohz=on is default)
The offsets between PHC and system clock, observed by phc2sys

- **nohz=on**
  - max 9550 ns
  - min -8134 ns
  - RMS 1589.3 ns

- **nohz=off**
  - max 32 ns
  - min -26 ns
  - RMS 9.3 ns

1000 times better!
Zoom graph of nohz=off

nohz=off
max 32 ns
min -26 ns
RMS 9.3 ns

between PHC and sys clock

offset (ns)
elapsed time (s)
Miroslav Lichvar and John Stultz are working to fix the issue

<table>
<thead>
<tr>
<th>title</th>
<th>author</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>2nd patch</td>
<td>“[PATCH] [RFC] timekeeping: Rework frequency adjustments to work better w/ nohz”</td>
<td>John</td>
</tr>
<tr>
<td>3rd patch</td>
<td>“[PATCH 0/3] timekeeping: Improved NOHZ frequency steering”</td>
<td>John</td>
</tr>
</tbody>
</table>
Agenda

- Background
- Overview of Precision Time Protocol (PTP)
- About PTP on Linux
- Tips
  - Countermeasure against bad GM behavior
  - Improve system clock stability
- For easy trial or development
  - Try linuxptp on qemu-kvm
FYI: Running linuxptp on qemu-kvm

- You can try linuxptp on qemu-kvm without NICs supporting PTP

- Note:
  - Run two Virtual Machines (for GM and Slave)
  - Recommend to use OS supporting linuxptp
    - e.g. recent Fedora, RHEL6.5, 7.0
  - Use virtual NIC emulating e1000
    - e1000 driver supports software timestamping
  - Don’t forget to define appropriate firewall rules (or disable firewalls) to allow multi-cast
  - Don’t expect high precision and accuracy
Fujitsu

shaping tomorrow with you