IEEE 1588 & PTP

USING EMBEDDED LINUX SYSTEMS

Created by Insop Song

- This is NOT an advance topic of 1588 and PTP
- Presenter is NOT one of the PTP application developers
- This IS a collection of information that the presenter learned on 1588 and PTP
- This IS an introductory material for using IEEE 1588 and PTP on Linux

CONTENTS

TIME SYNCHRONIZATION

- Alignment of time within distributed nodes

- critical for real-time applications, control and measurement systems, and voice and video network

- Frequency, phase, time sync between distributed nodes

TIME SYNC IS IMPORTANT:

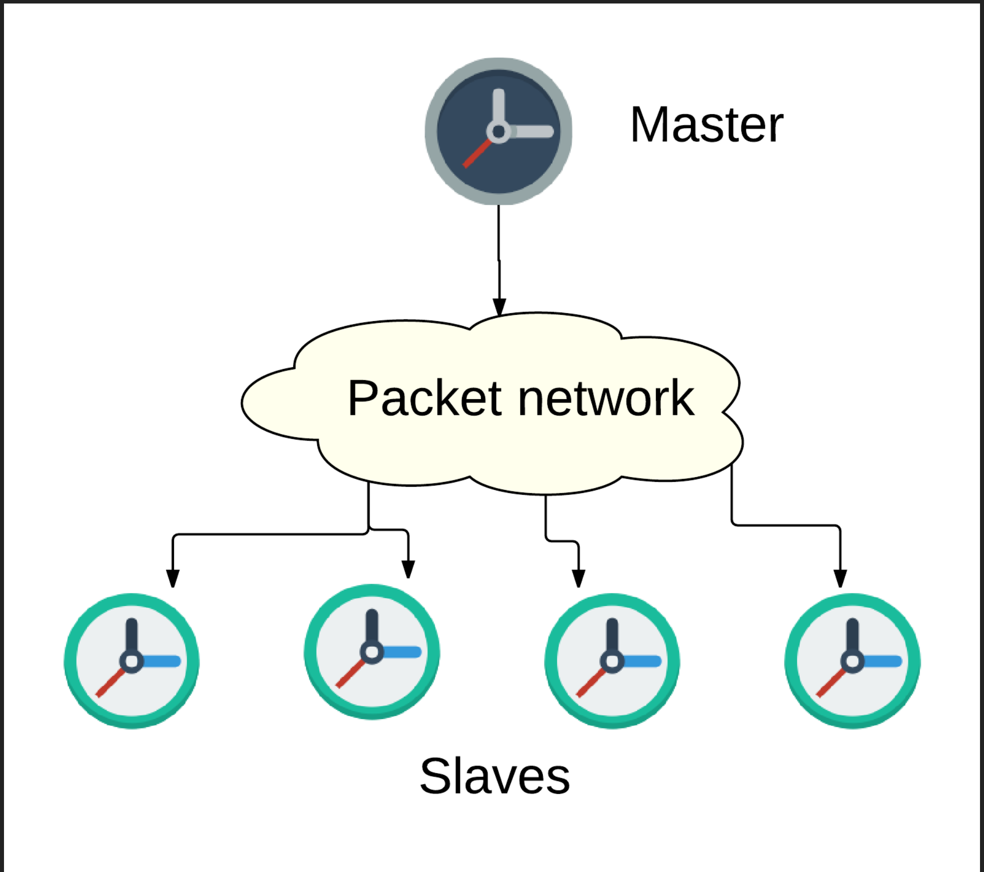
- to coordinate actions
- to trigger measurement
- to reference events

OVERVIEW: IEEE 1588

IEEE 1588

- Distribution of precise time information over packet-based network
- Sync frequency and phase between network connected nodes
- Offers high accuracy (sub micro sec) over network
- "Server (master)" clock sends packets to slave to sync time

IEEE 1588



NTP (NETWORK TIME PROTOCOL)

- Widely used in server and client environment for many years
- Provides milisecond level accuracy
- Some applications require higher accuracies

COMPARISON BETWEEN NTP AND 1588

	NTP	IEEE 1588
Communication	Internet	LAN
Accuracy	msec	< usec
H/W support	no	usually required, doable without

APPLICATION AREAS

- Automation control systems
- Measurement and test systems
- Telecommunication

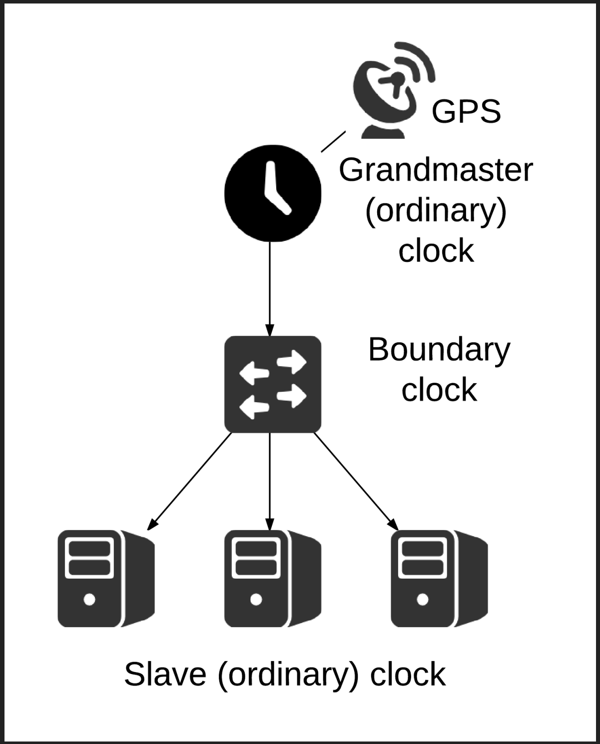
OVERVIEW:PTP

PRECISION TIME PROTOCOL

Terminology

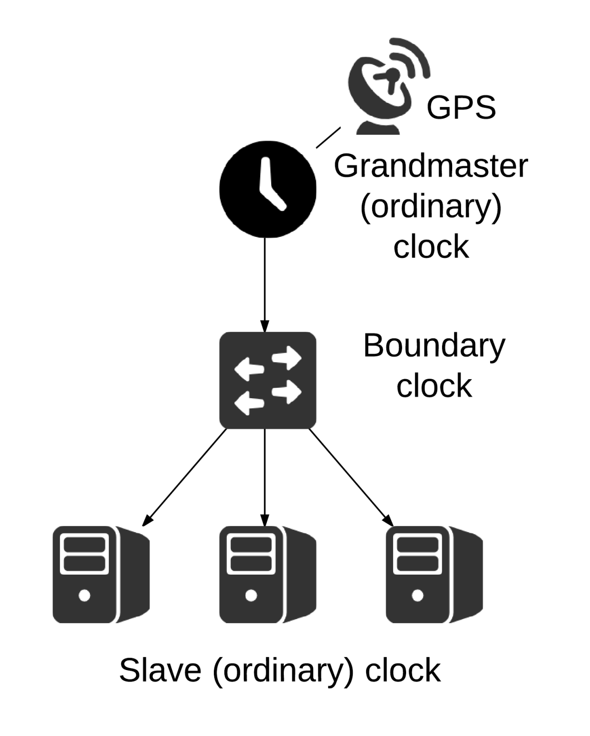
- Grandmaster clock (Ordinary clock)
- Boundary clock
- Slave clock (Ordinary clock)

GRANDMASTER CLOCK



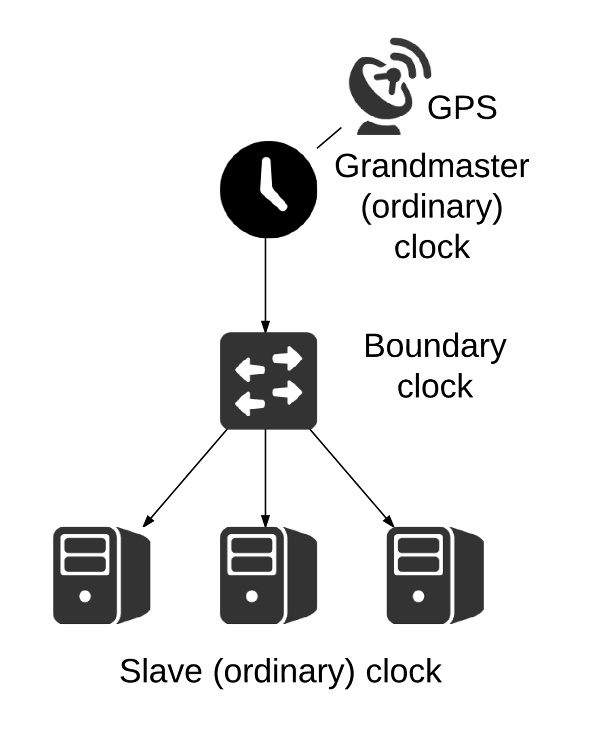
- Time source
- has accurate time source, such as GPS

BOUNDARY CLOCK



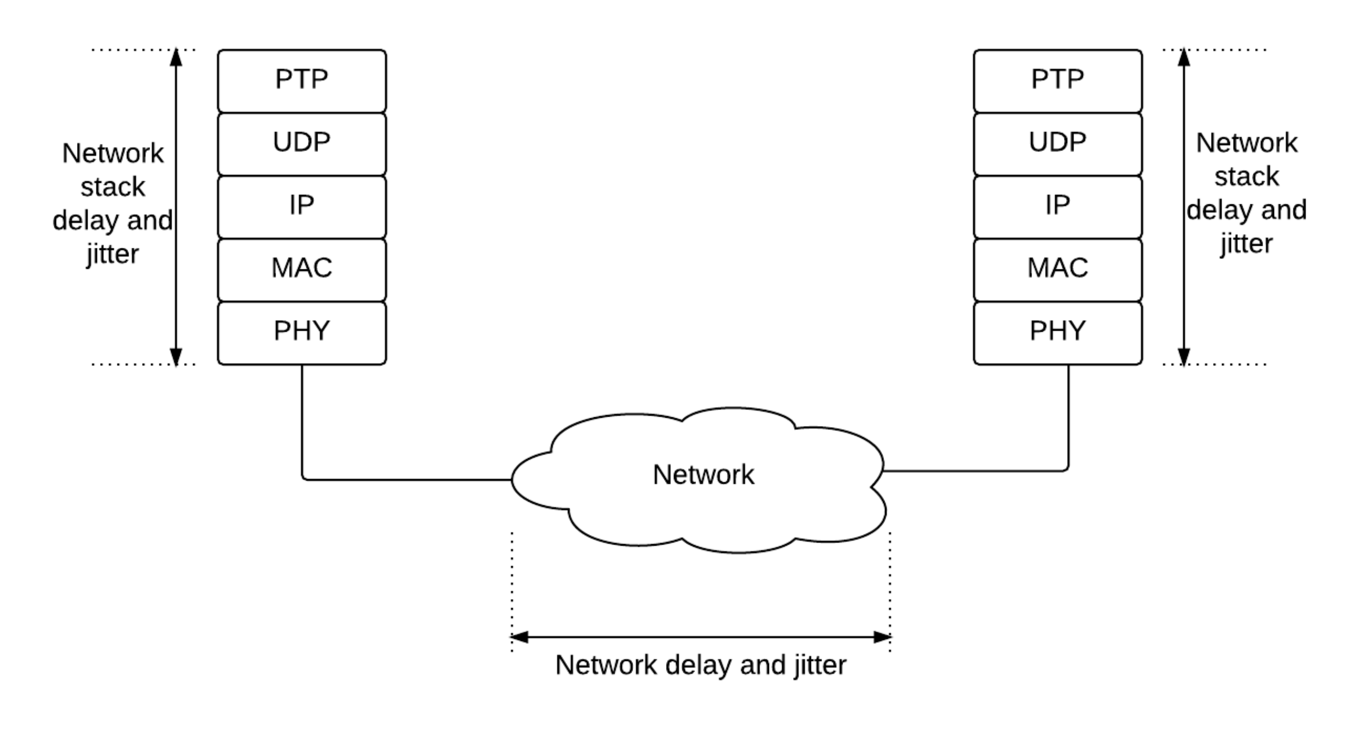
- Sync clock to master
- Serve as a master to slaves
- Switch/Router devices

SLAVE CLOCK



- Sync clock to master

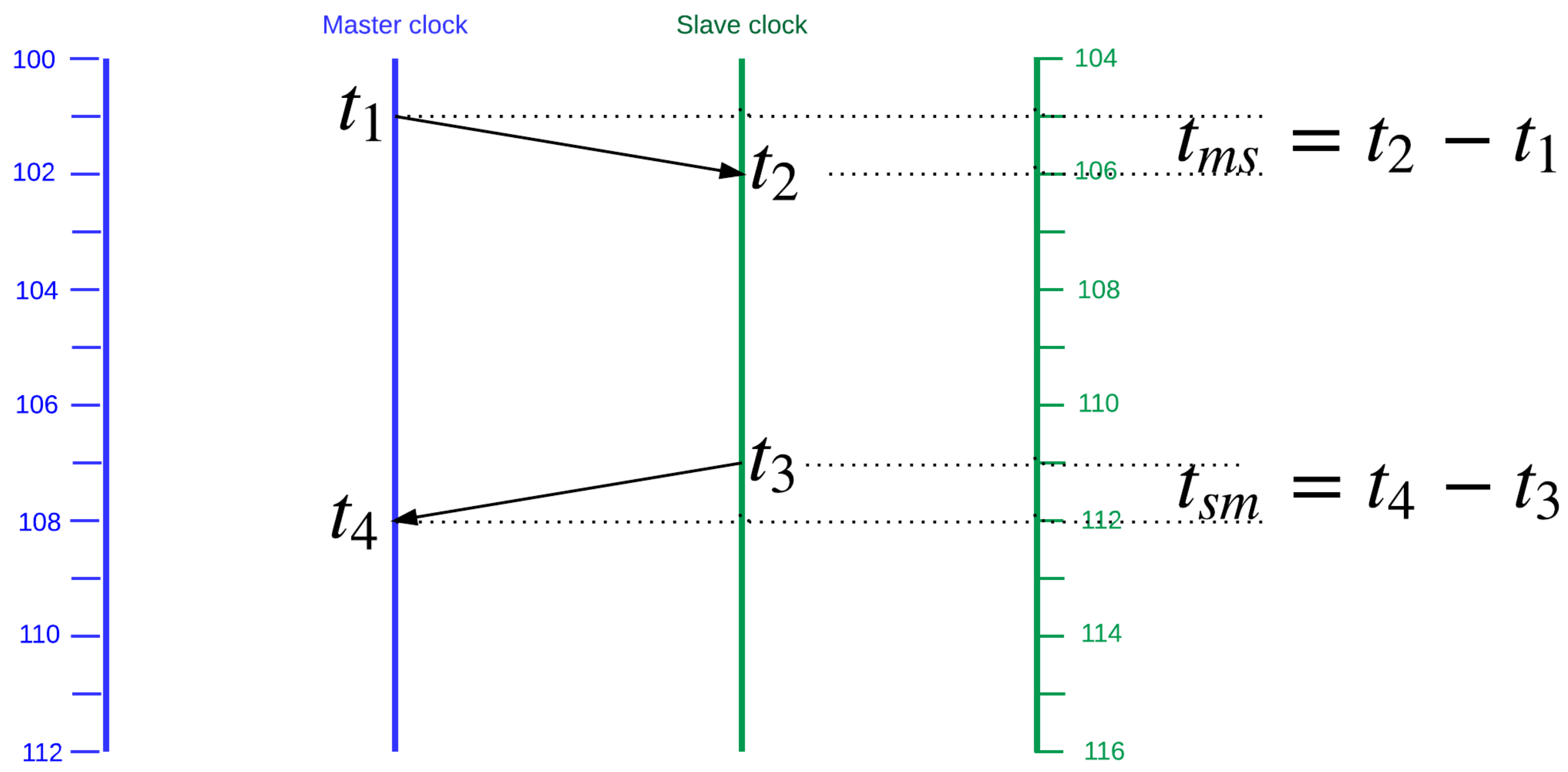
NETWORK/PROTOCOL STACK DELAY AND JITTER



CASES OF DISTRIBUTED CLOCKS

- Two clocks changes at the same rate, but they are 10 min apart
- Two clocks started at the same time, but changes at different rates
- Two clocks started at the same time, changes at the same rates

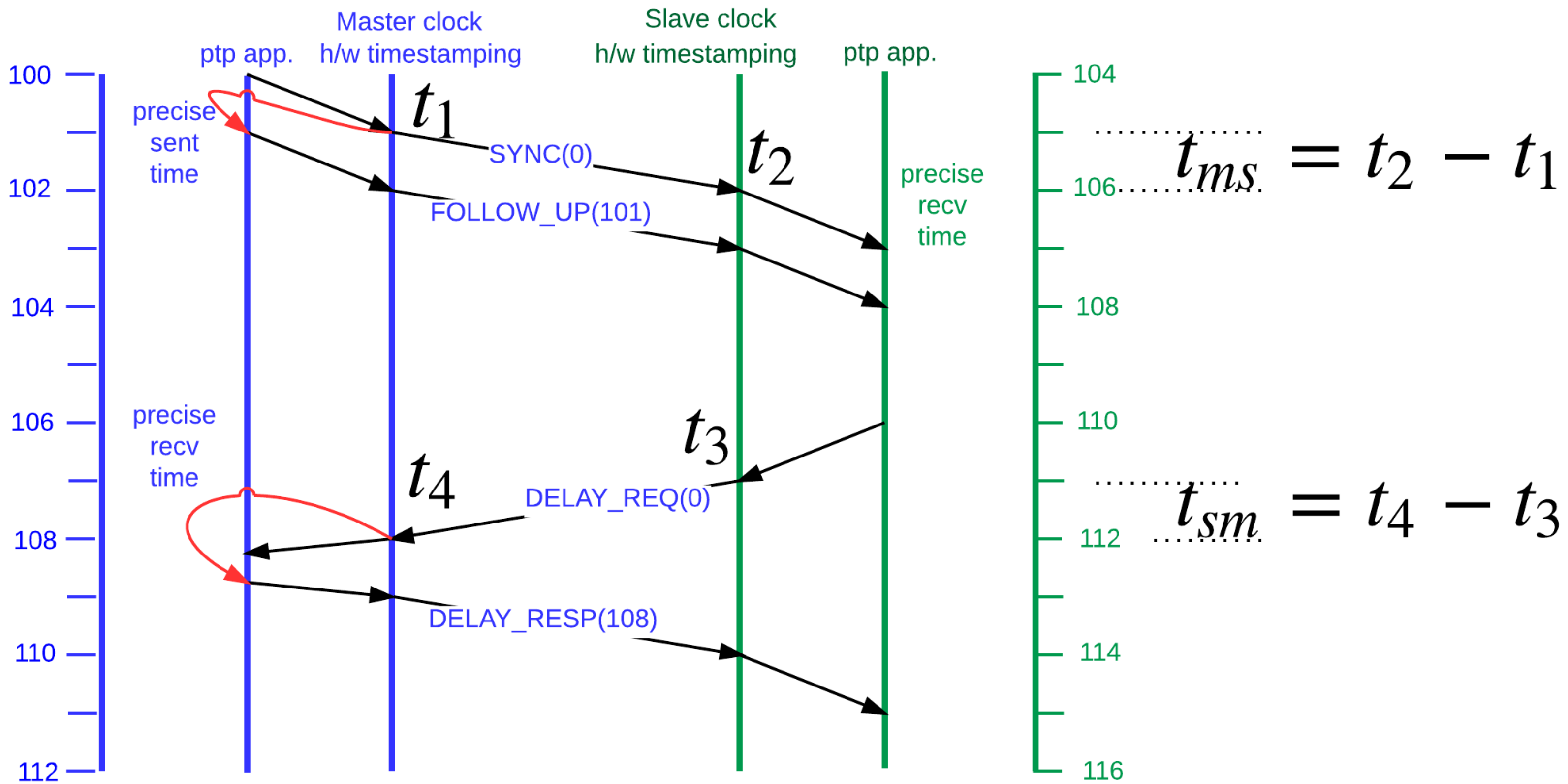
Offset and delay: ideal case



$$t_{ms} = t_2 - t_1 = delay + offset$$
$$t_{sm} = t_4 - t_3 = delay - offset$$
$$offset = (t_{ms} - t_{sm})/2$$
$$delay = (t_{ms} + t_{sm})/2$$

$$t_{ms} = 106 - 101 = 5$$
$$t_{sm} = 108 - 111 = -3$$
$$offset = (5 + 3)/2 = 4$$
$$delay = (5 - 3)/2 = 1$$

PTP message sequences



$$t_{ms} = t_2 - t_1 = delay + offset$$
$$t_{sm} = t_4 - t_3 = delay - offset$$
$$offset = (t_{ms} - t_{sm})/2$$
$$delay = (t_{ms} + t_{sm})/2$$

$$t_{ms} = 106 - 101 = 5$$
$$t_{sm} = 108 - 111 = -3$$
$$offset = (5 + 3)/2 = 4$$
$$delay = (5 - 3)/2 = 1$$

KERNEL H/W TIMESTAMPING SUPPORT

PTP HARDWARE CLOCKS (PHCS) IS IN LINUX KERNEL

- Support for obtaining timestamps from a PHC exists via the SO_TIMESTAMPING socket option
- It is integrated in kernel version 2.6.30
- SOF_TIMESTAMPING_RX_HARDWARE, SOF_TIMESTAMPING_TX_HARDWARE
- RX: h/w timestamp is set to `hwtstamp`
- TX: the outgoing packet is looped back to the socket's error queue with the h/w timestamp

```c
void ixgbe_ptp_rx_hwtstamp(struct ixgbe_adapter *adapter, struct sk_
{
        struct ixgbe_hw *hw = &adapter->hw;
        struct skb_shared_hwtstamps *shhwtstamps;
        u64 regval = 0, ns;
        u32 tsyncrxctl;
        unsigned long flags;

        tsyncrxctl = IXGBE_READ_REG(hw, IXGBE_TSYNCRXCTL);
        if (!(tsyncrxctl & IXGBE_TSYNCRXCTL_VALID))
                return;

        regval |= (u64)IXGBE_READ_REG(hw, IXGBE_RXSTMPL);
        regval |= (u64)IXGBE_READ_REG(hw, IXGBE_RXSTMPH) << 32;

        spin_lock_irqsave(&adapter->tmreg_lock, flags);
```

RX timestamping

```c
static void ixgbe_ptp_tx_hwtstamp(struct ixgbe_adapter *adapter)
{
        struct ixgbe_hw *hw = &adapter->hw;
        struct skb_shared_hwtstamps shhwtstamps;
        u64 regval = 0, ns;
        unsigned long flags;

        regval |= (u64)IXGBE_READ_REG(hw, IXGBE_TXSTMPL);
        regval |= (u64)IXGBE_READ_REG(hw, IXGBE_TXSTMPH) << 32;

        spin_lock_irqsave(&adapter->tmreg_lock, flags);
        ns = timecounter_cyc2time(&adapter->tc, regval);
        spin_unlock_irqrestore(&adapter->tmreg_lock, flags);

        memset(&shhwtstamps, 0, sizeof(shhwtstamps));
        shhwtstamps.hwtstamp = ns_to_ktime(ns);
```

TX timestamping

HOW TO USE IN A USER SPACE APP

- example: timestamping.c
- check NIC's timestamping capabilities using `ethtool`

```
#  ethtool -T eth0
Time stamping parameters for eth0:
Capabilities:
    hardware-transmit     (SOF_TIMESTAMPING_TX_HARDWARE)
    hardware-receive      (SOF_TIMESTAMPING_RX_HARDWARE)
    hardware-raw-clock    (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 0
Hardware Transmit Timestamp Modes:
    off                   (HWTSTAMP_TX_OFF)
    on                    (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
    none                  (HWTSTAMP_FILTER_NONE)
    all                   (HWTSTAMP_FILTER_ALL)
```

USING PTP APP. ON LINUX

PTPd

- http://ptpd.sourceforge.net
- BSD License
- runs on Linux, uClinux, FreeBSD, NetBSD

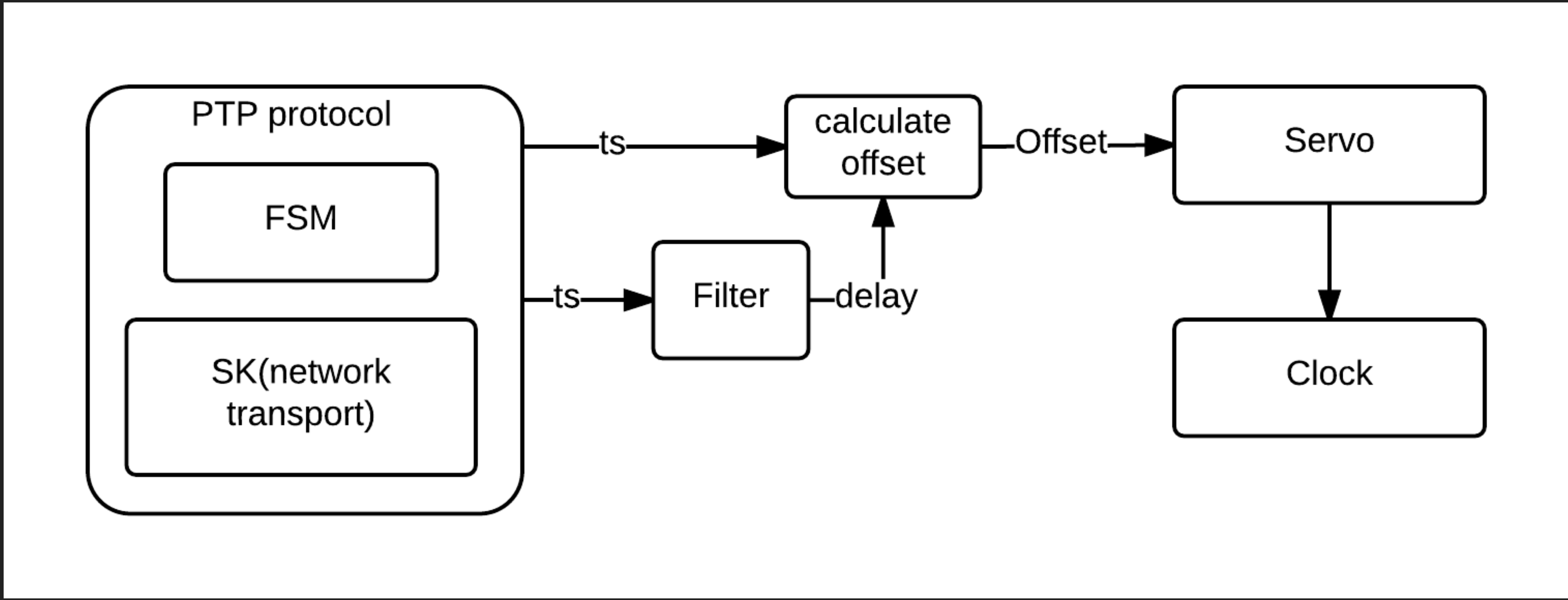
Linux PTP (ptp4l)

- http://linuxptp.sourceforge.net/
- GPL License
- Maintainer: Richard Cochran
- RedHat officially supports

PTP4L

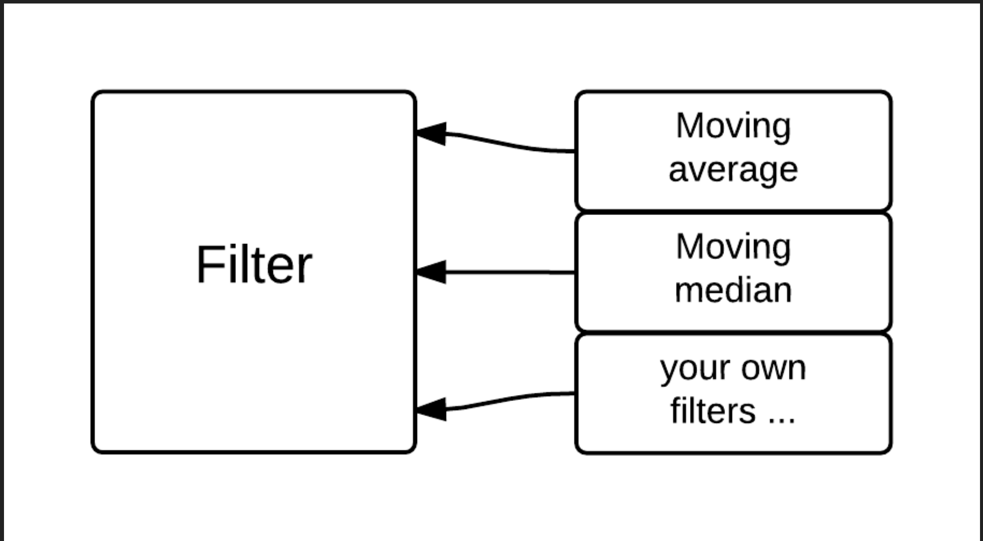
OVERVIEW

- ptp4l, pmc, phc2sys
- h/w, s/w timestamping



FILTER

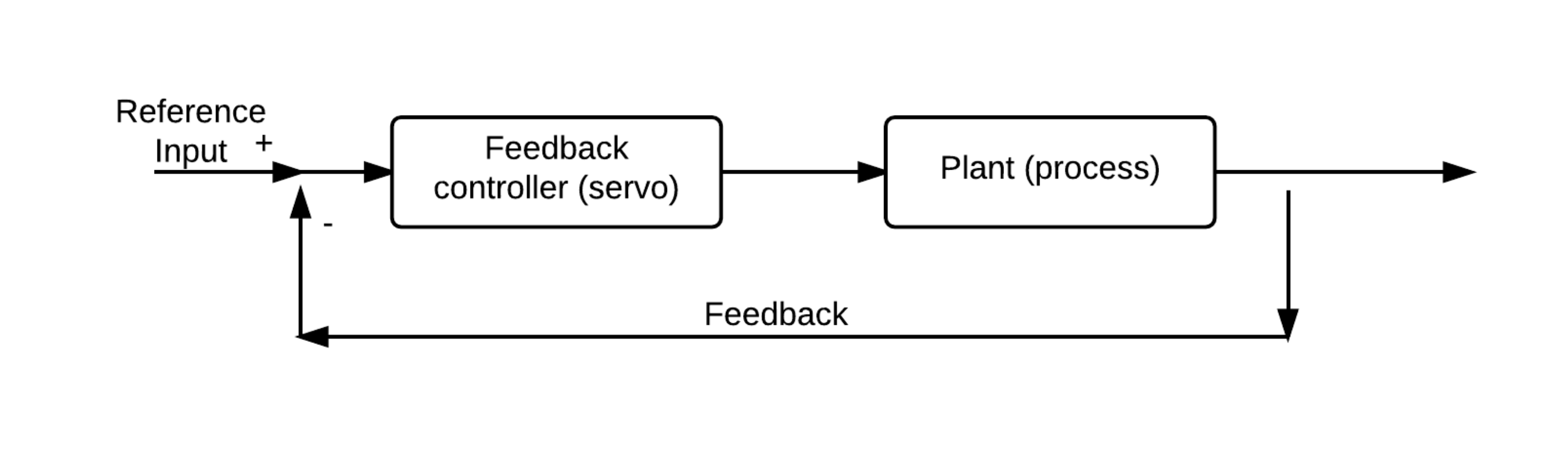
- different types of filter can be plugged in
- available filters: `moving avg`, `moving median`
- new type of filter can be added



BASICS OF FEEDBACK CONTROL

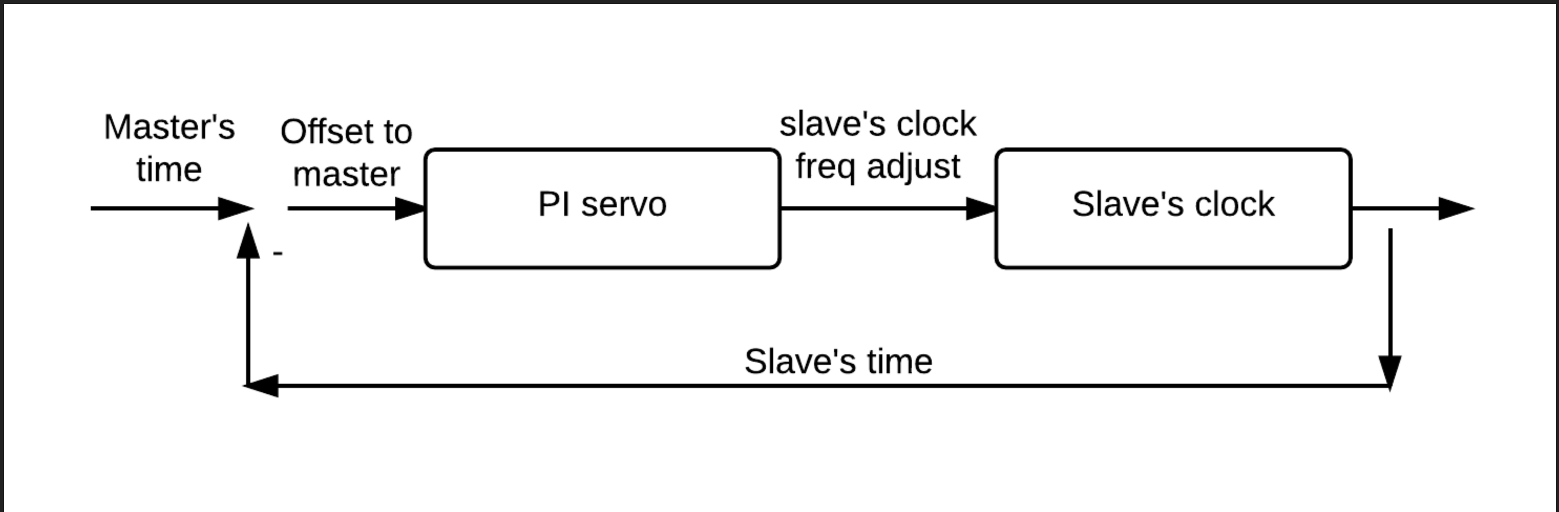
PID feedback controller (servo)

- P: proportional
- I: integral
- D: derivertive



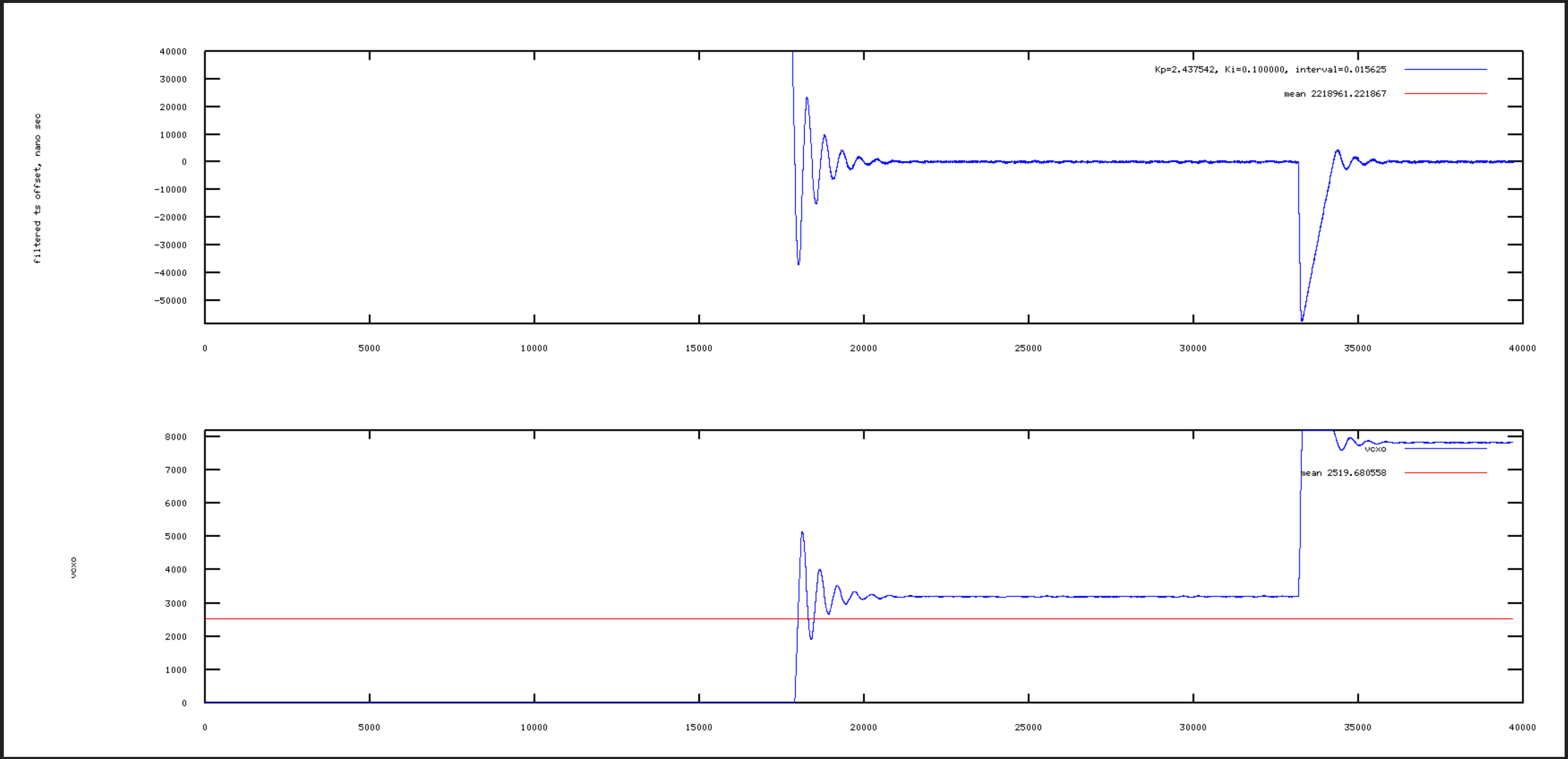
PTP4L'S SERVO

- PI controller is configured by default
- Optional Linear regression servo can be used
- Or you can add your own servo as well



Example: servo in action

- shows slave's `offset` and oscillator value
- changed master's clock freq twice, check if slave follows



TESTING ON 2 VIRTUAL MACHINES

- use VM to get familiar with PTP
- create 2 VMs and connect them
- use to fedora 20 kvm running on a linuxpc to test linuxptp
- use e1000 nic to use s/w timestamp
- disable firewall so that multicast can go through

MASTER VM

```
sudo ptp4l -S -A -l 7 -q -i ens3 -f /etc/ptp4l.conf -m

ptp4l[48.071]: PI servo: sync interval 1.000 kp 0.100 ki 0.001000
ptp4l[48.075]: port 1: INITIALIZING to LISTENING on INITIALIZE
ptp4l[48.076]: port 0: INITIALIZING to LISTENING on INITIALIZE
ptp4l[54.552]: port 1: announce timeout
ptp4l[54.552]: port 1: LISTENING to MASTER on ANNOUNCE_RECEIPT_TIMEO
ptp4l[54.553]: selected best master clock 525400.fffe.6fb9cf
ptp4l[54.553]: assuming the grand master role
ptp4l[54.554]: port 1: master tx announce timeout
ptp4l[54.554]: port 1: setting asCapable
ptp4l[55.553]: port 1: master sync timeout
ptp4l[56.553]: port 1: master sync timeout
ptp4l[56.554]: port 1: master tx announce timeout
```

pcap on the master clock

SLAVE VM

```
sudo ptp4l -S -A -l 7 -q -i ens3 -f /etc/ptp4l.conf -m -s

ptp4l[53.946]: PI servo: sync interval 1.000 kp 0.100 ki 0.001000
ptp4l[53.950]: port 1: INITIALIZING to LISTENING on INITIALIZE
ptp4l[53.951]: port 0: INITIALIZING to LISTENING on INITIALIZE
ptp4l[54.187]: port 1: setting asCapable
ptp4l[54.187]: port 1: new foreign master 525400.fffe.6fb9cf-1
ptp4l[58.188]: selected best master clock 525400.fffe.6fb9cf
ptp4l[58.188]: foreign master not using PTP timescale
ptp4l[58.189]: port 1: LISTENING to UNCALIBRATED on RS_SLAVE
ptp4l[59.703]: port 1: delay timeout
ptp4l[59.705]: path delay        201159     201159
ptp4l[59.972]: port 1: delay timeout
ptp4l[59.974]: path delay        215360     229562
ptp4l[60.188]: master offset  625645199 s0 freq  +18054 path delay
ptp4l[61.188]: master offset  625657327 s0 freq  +18054 path delay
```

pcap on the slave clock

PMC: PTP MANAGEMENT CLIENT

```
[labuser@fd-2 ~]$ sudo pmc -u -b 0 'GET CURRENT_DATA_SET'
sending: GET CURRENT_DATA_SET
    525400.fffe.5d19ce-0 seq 0 RESPONSE MANAGMENT CURRENT_DATA_SET
        stepsRemoved     1
        offsetFromMaster -10720.0
        meanPathDelay    200706.0
```

```
[labuser@fd-2 ~]$ sudo pmc -u -b 0 'GET TIME_STATUS_NP'
sending: GET TIME_STATUS_NP
    525400.fffe.5d19ce-0 seq 0 RESPONSE MANAGMENT TIME_STATUS_NP
        master_offset              10539
        ingress_time               1418085978021017613
        cumulativeScaledRateOffset +1.000000000
        scaledLastGmPhaseChange    0
        gmTimeBaseIndicator        0
        lastGmPhaseChange          0x0000'0000000000000000.0000
        gmPresent                  true
        gmIdentity                 525400.fffe.6fb9cf
```

PHC2SYS: SYNCHRONIZING THE CLOCKS

- to synchronize the system clock to the PTP hardware clock (PHC) on the NIC

```
./phc2sys -s /dev/ptp0  -w -l 6 -q -m
```

HOW TO BUILD

- kernel config

```
CONFIG_PTP_1588_CLOCK=y
CONFIG_PPS=y
// any other driver specif 1588 conf
```

- buildroot: ADI buildroot
- yocto: meta-oe

CONCLUSION

- with kernel's s/w, h/w timestamping support
- with open source user-space PTP applications
- IEEE 1588 PTP is available on Linux system

QUESTION ???

REFERENCES:

- lcjp14_ichikawa_0.pdf
- 2006_Conference_IEEE_1588_Tutorial.pdf
- WBNR_FTF12_NET_F0102.pdf
- Redhat: Configuring PTP Using ptp4l