PTP from scratch

Milena Olech
Maciek Machnikowski

Agenda

- LinuxPTP
- netdevsim
- ptp_mock
- LinuxPTP in Netdevsim
- Limitations & Use-cases
- Demo

Required components (usually)

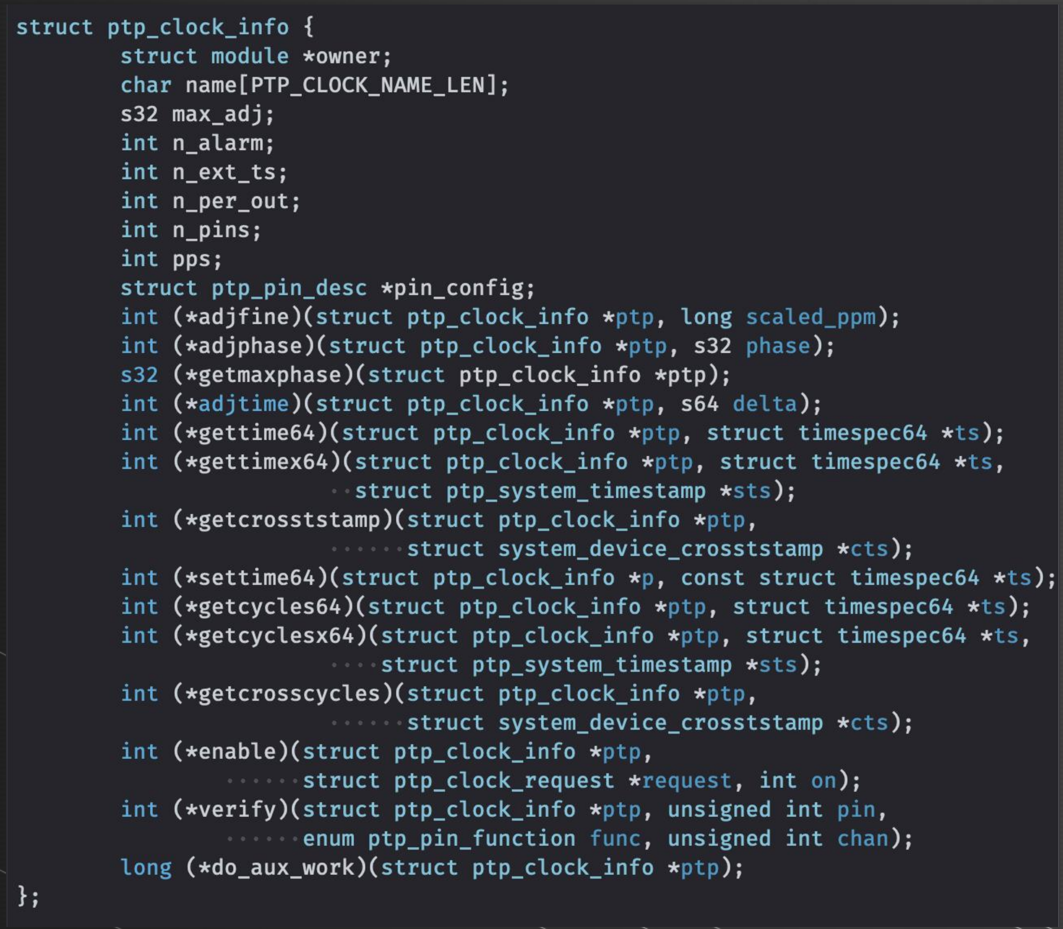
- Hardware clock
- Driver for the HW clock
- NIC with timestamping
- Timestamping mechanism generating
  - Tx timestamps
  - Rx timestamps

LinuxPTP

- Phc_ctl
  - modify the PHC (PTP HW Clock)
- Ts2phc
  - synchronize PHC to the external source
- Phc2sys
  - synchronize system time to the PHC
- Ptp4l
  - synchronizes two (or more) PHCs

LinuxPTP

- Phc_ctl
  - gettime, settime, adjtime, adjfine, max_adj
- Ts2phc
  - n_ext_ts, n_per_out, n_pins, enable, verify
  - settime, adjtime, adjfine, max_adj
- Phc2sys
  - gettime, getcrosststamp
- Ptp4l
  - gettime, settime, adjtime, adjfine + Tx/Rx timestamping



```
struct ptp_clock_info {
        struct module *owner;
        char name[PTP_CLOCK_NAME_LEN];
        s32 max_adj;
        int n_alarm;
        int n_ext_ts;
        int n_per_out;
        int n_pins;
        int pps;
        struct ptp_pin_desc *pin_config;
        int (*adjfine)(struct ptp_clock_info *ptp, long scaled_ppm);
        int (*adjphase)(struct ptp_clock_info *ptp, s32 phase);
        s32 (*getmaxphase)(struct ptp_clock_info *ptp);
        int (*adjtime)(struct ptp_clock_info *ptp, s64 delta);
        int (*gettime64)(struct ptp_clock_info *ptp, struct timespec64 *ts);
        int (*gettimex64)(struct ptp_clock_info *ptp, struct timespec64 *ts,
                          struct ptp_system_timestamp *sts);
        int (*getcrosststamp)(struct ptp_clock_info *ptp,
                              struct system_device_crosststamp *cts);
        int (*settime64)(struct ptp_clock_info *p, const struct timespec64 *ts);
        int (*getcycles64)(struct ptp_clock_info *ptp, struct timespec64 *ts);
        int (*getcyclesx64)(struct ptp_clock_info *ptp, struct timespec64 *ts,
                            struct ptp_system_timestamp *sts);
        int (*getcrosscycles)(struct ptp_clock_info *ptp,
                              struct system_device_crosststamp *cts);
        int (*enable)(struct ptp_clock_info *ptp,
                      struct ptp_clock_request *request, int on);
        int (*verify)(struct ptp_clock_info *ptp, unsigned int pin,
                      enum ptp_pin_function func, unsigned int chan);
        long (*do_aux_work)(struct ptp_clock_info *ptp);
};
```

netdevsim

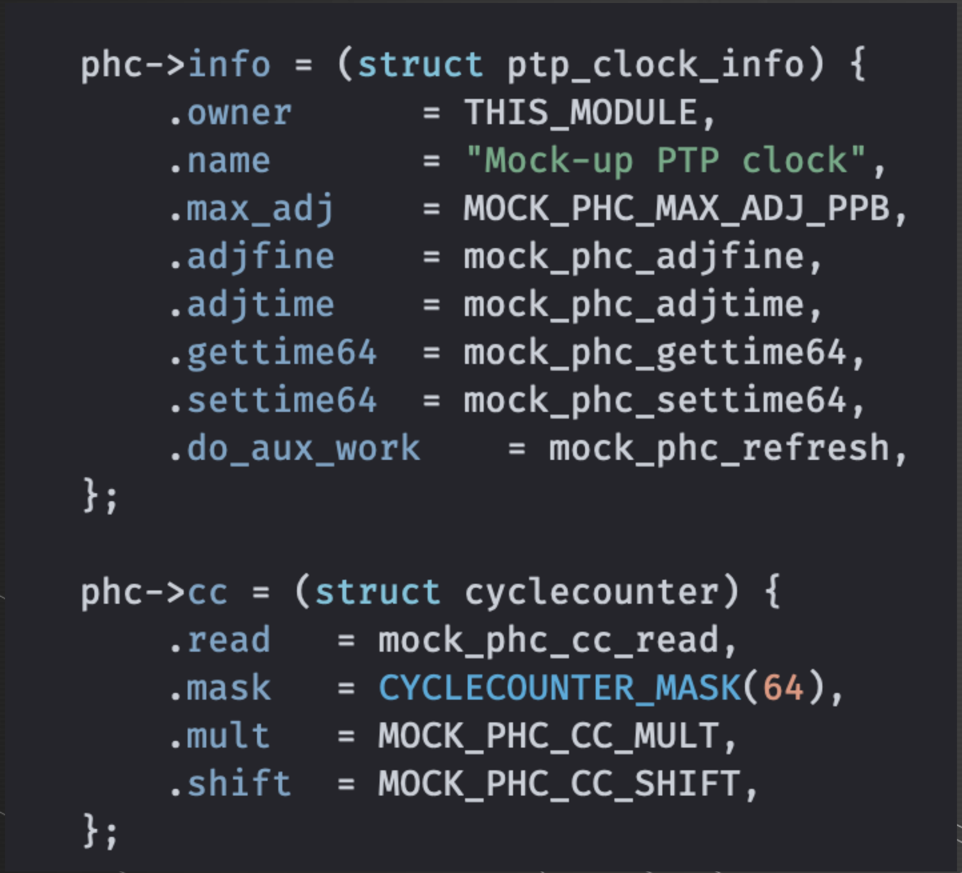
- **Simulated networking device**
- Used for testing APIs **without requiring capable hardware**
- Emulate different **hardware offloads**
- Recently implemented packet forwarding between instances

ptp_mock

- Common mock-up PTP Hardware Clock (PHC) driver
- Implements PTP Hardware Clock for virtual network devices
- Creates an object that **emulates the PTP clock**
- Emulates PHC using timecounters subsystem
  - Mathematical overlay over CLOCK_MONOTONIC_RAW

ptp_mock

- Allows
  - Setting virtual time
  - Reading virtual time
  - Changing virtual frequency



```
phc->info = (struct ptp_clock_info) {
    .owner      = THIS_MODULE,
    .name       = "Mock-up PTP clock",
    .max_adj    = MOCK_PHC_MAX_ADJ_PPB,
    .adjfine    = mock_phc_adjfine,
    .adjtime    = mock_phc_adjtime,
    .gettime64  = mock_phc_gettime64,
    .settime64  = mock_phc_settime64,
    .do_aux_work    = mock_phc_refresh,
};

phc->cc = (struct cyclecounter) {
    .read   = mock_phc_cc_read,
    .mask   = CYCLECOUNTER_MASK(64),
    .mult   = MOCK_PHC_CC_MULT,
    .shift  = MOCK_PHC_CC_SHIFT,
};
```

Testing a PHC driver

- phc_ctl
  - set - Set the PHC time
  - get - Get the PHC time
  - freq - Frequency adjust
  - Example
  - *phc_ctl /dev/ptp0 freq 100000000 set 0.0 wait 10.0 get*

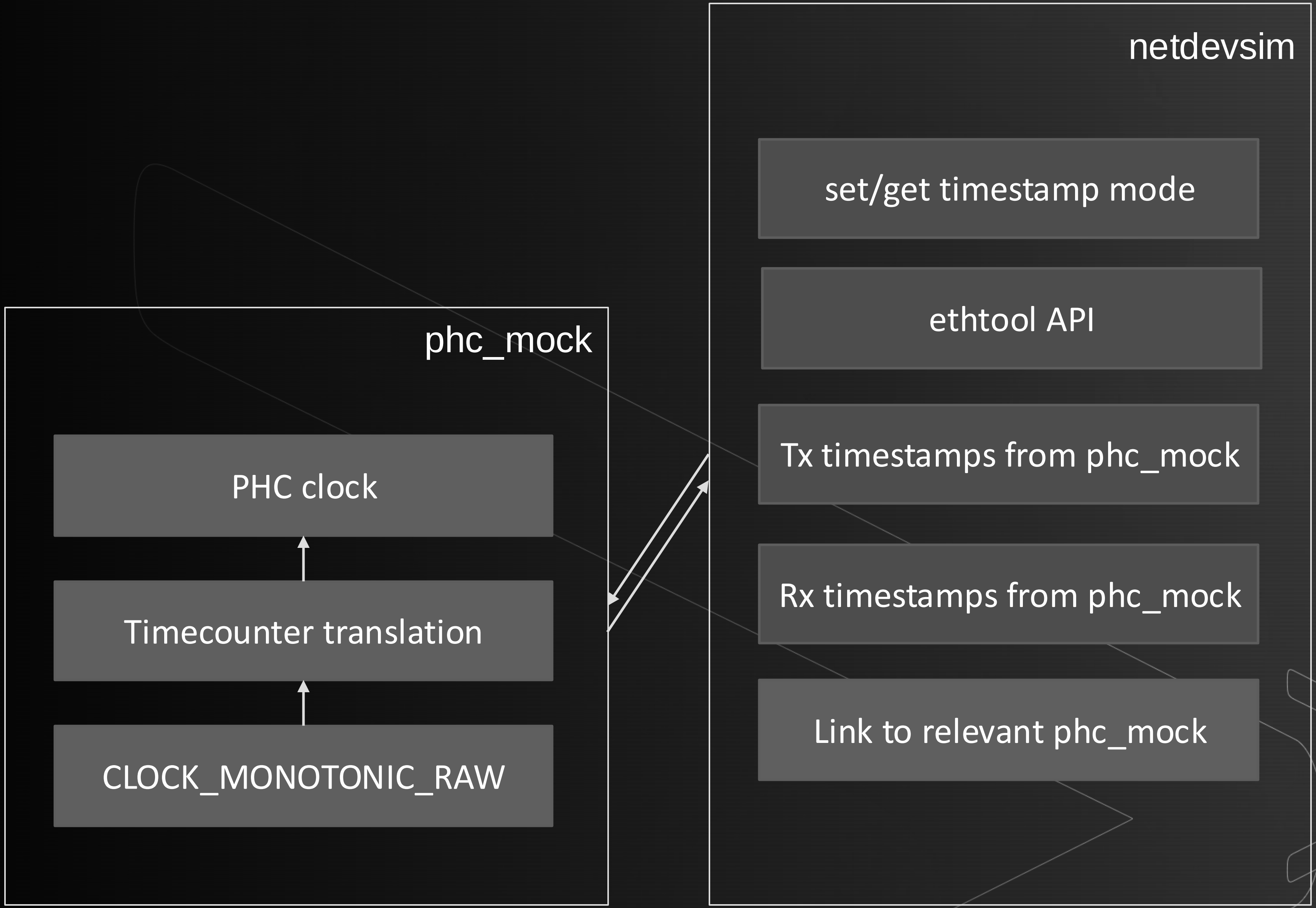
Required components (usually)

Hardware clock	CLOCK_MONOTONIC_RAW
Driver for the HW clock	ptp_mock
NIC driver	netdevsim
Timestamping mechanism generating	missing in netdevsim
Tx timestamps	
Rx timestamps	
IOCTL support	

What’s next

- We have the PTP Hardware Clock
- …and we can forward packets
- **What if we connect them and mock timestamping?**

Trust me! I’m an architect!



phc_mock

PHC clock

Timecounter translation

CLOCK_MONOTONIC_RAW

netdevsim

set/get timestamp mode

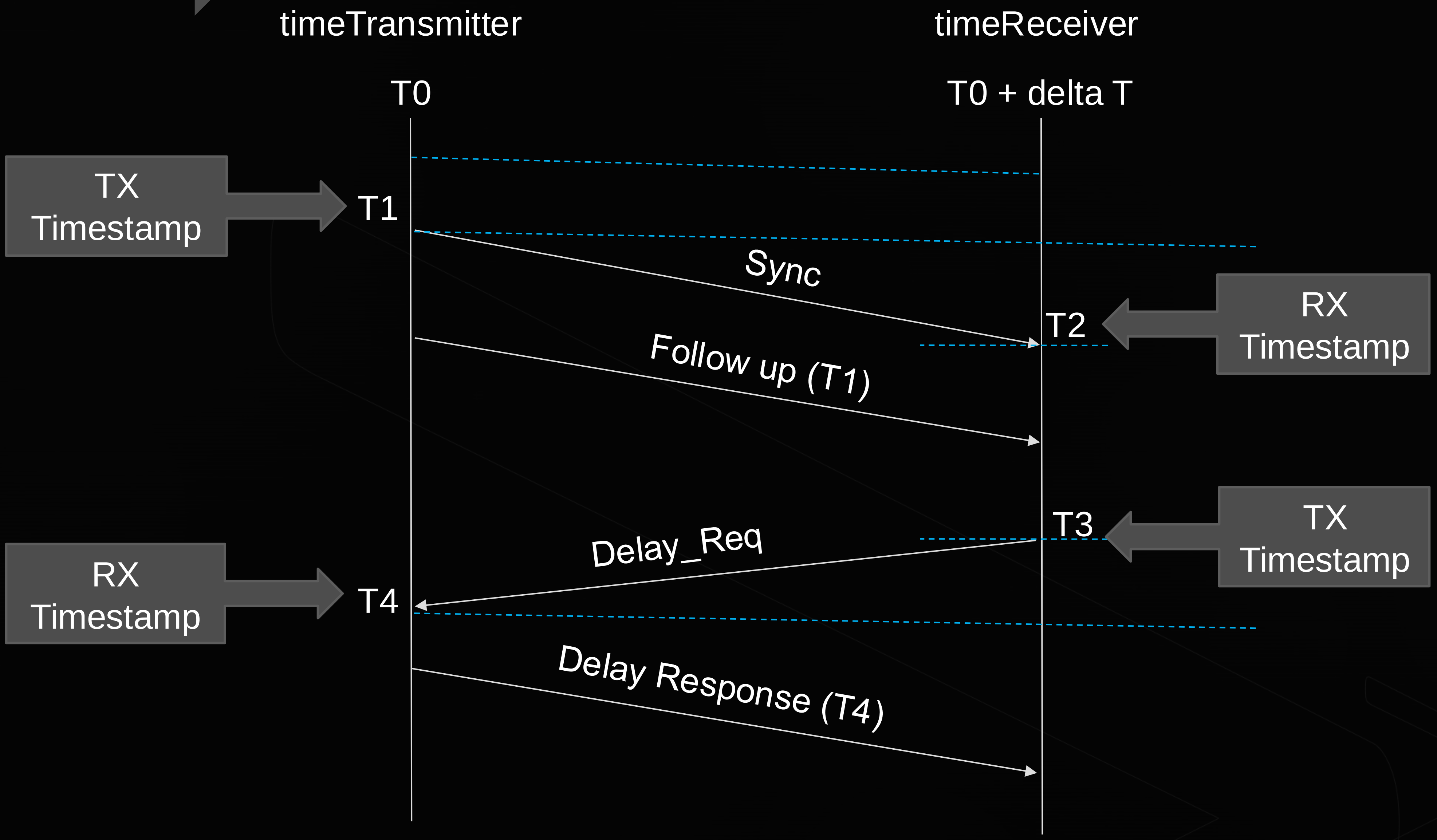
ethtool API

Tx timestamps from phc_mock

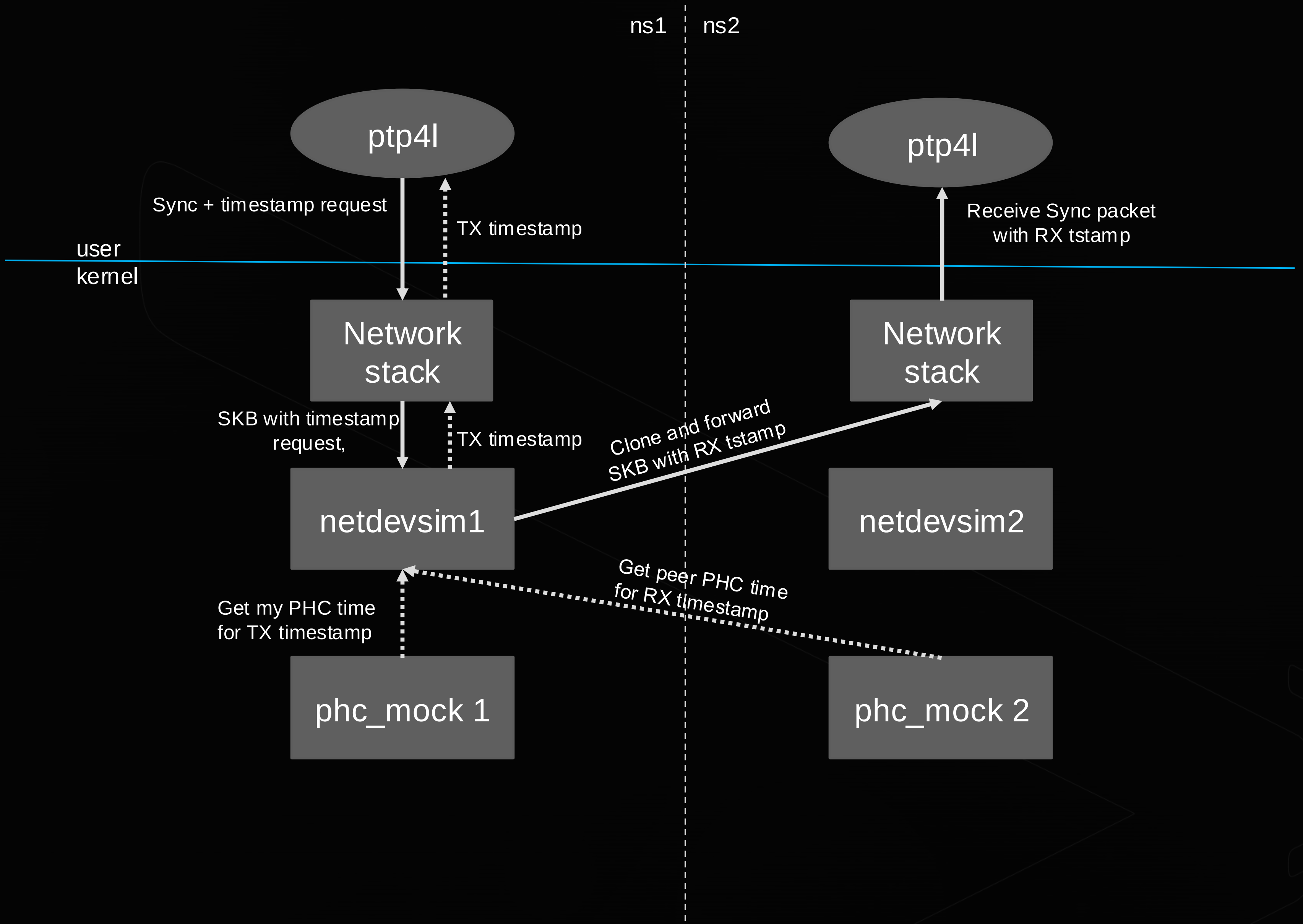
Rx timestamps from phc_mock

Link to relevant phc_mock

PTP


timeTransmitter
timeReceiver
T0
T0 + delta T
TX Timestamp
T1
Sync
T2
RX Timestamp
Follow up (T1)
T3
TX Timestamp
Delay_Req
RX Timestamp
T4
Delay Response (T4)

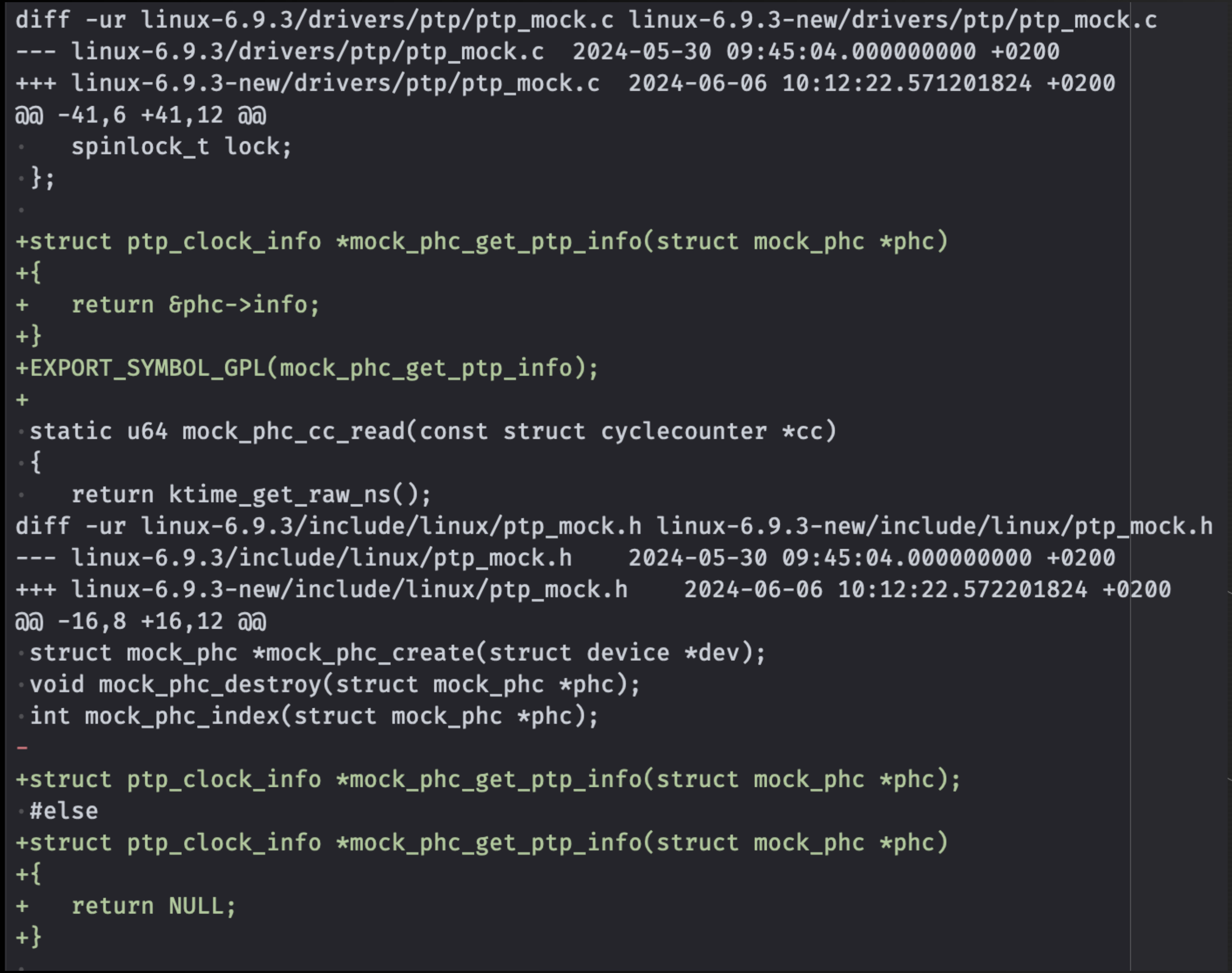
PTP implementation in netdevsim


ns1
ns2
ptp4l
ptp4l
Sync + timestamp request
TX timestamp
Receive Sync packet with RX tstamp
user
kernel
Network stack
Network stack
SKB with timestamp request,
TX timestamp
Clone and forward SKB with RX tstamp
netdevsim1
netdevsim2
Get peer PHC time for RX timestamp
Get my PHC time for TX timestamp
phc_mock 1
phc_mock 2

Step 1: Connect PHC to netdev

- Implement a proper connection between the PHC and the netdev
- ptp_mock needs to expose the phc->info structure
- Internal PTP APIs, such as gettime() require struct **ptp_clock_info,** which is not accessible for the netdevsim
- So far, the connection between netdevsim and ptp_mock was loose
    - netdevsim was not doing anything with the allocated clock

Step 1: Connect PHC to netdev

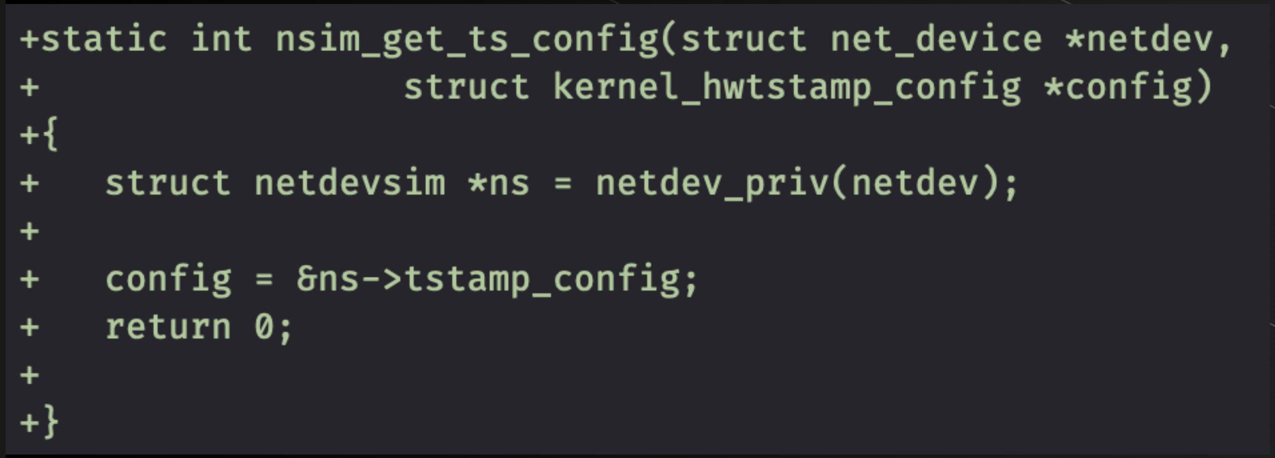


```
diff -ur linux-6.9.3/drivers/ptp/ptp_mock.c linux-6.9.3-new/drivers/ptp/ptp_mock.c
--- linux-6.9.3/drivers/ptp/ptp_mock.c  2024-05-30 09:45:04.000000000 +0200
+++ linux-6.9.3-new/drivers/ptp/ptp_mock.c  2024-06-06 10:12:22.571201824 +0200
@@ -41,6 +41,12 @@
    spinlock_t lock;
 };
 
+struct ptp_clock_info *mock_phc_get_ptp_info(struct mock_phc *phc)
+{
+   return &phc->info;
+}
+EXPORT_SYMBOL_GPL(mock_phc_get_ptp_info);
+
 static u64 mock_phc_cc_read(const struct cyclecounter *cc)
 {
    return ktime_get_raw_ns();
diff -ur linux-6.9.3/include/linux/ptp_mock.h linux-6.9.3-new/include/linux/ptp_mock.h
--- linux-6.9.3/include/linux/ptp_mock.h    2024-05-30 09:45:04.000000000 +0200
+++ linux-6.9.3-new/include/linux/ptp_mock.h    2024-06-06 10:12:22.572201824 +0200
@@ -16,8 +16,12 @@
 struct mock_phc *mock_phc_create(struct device *dev);
 void mock_phc_destroy(struct mock_phc *phc);
 int mock_phc_index(struct mock_phc *phc);
-
+struct ptp_clock_info *mock_phc_get_ptp_info(struct mock_phc *phc);
 #else
+struct ptp_clock_info *mock_phc_get_ptp_info(struct mock_phc *phc)
+{
+   return NULL;
+}
```

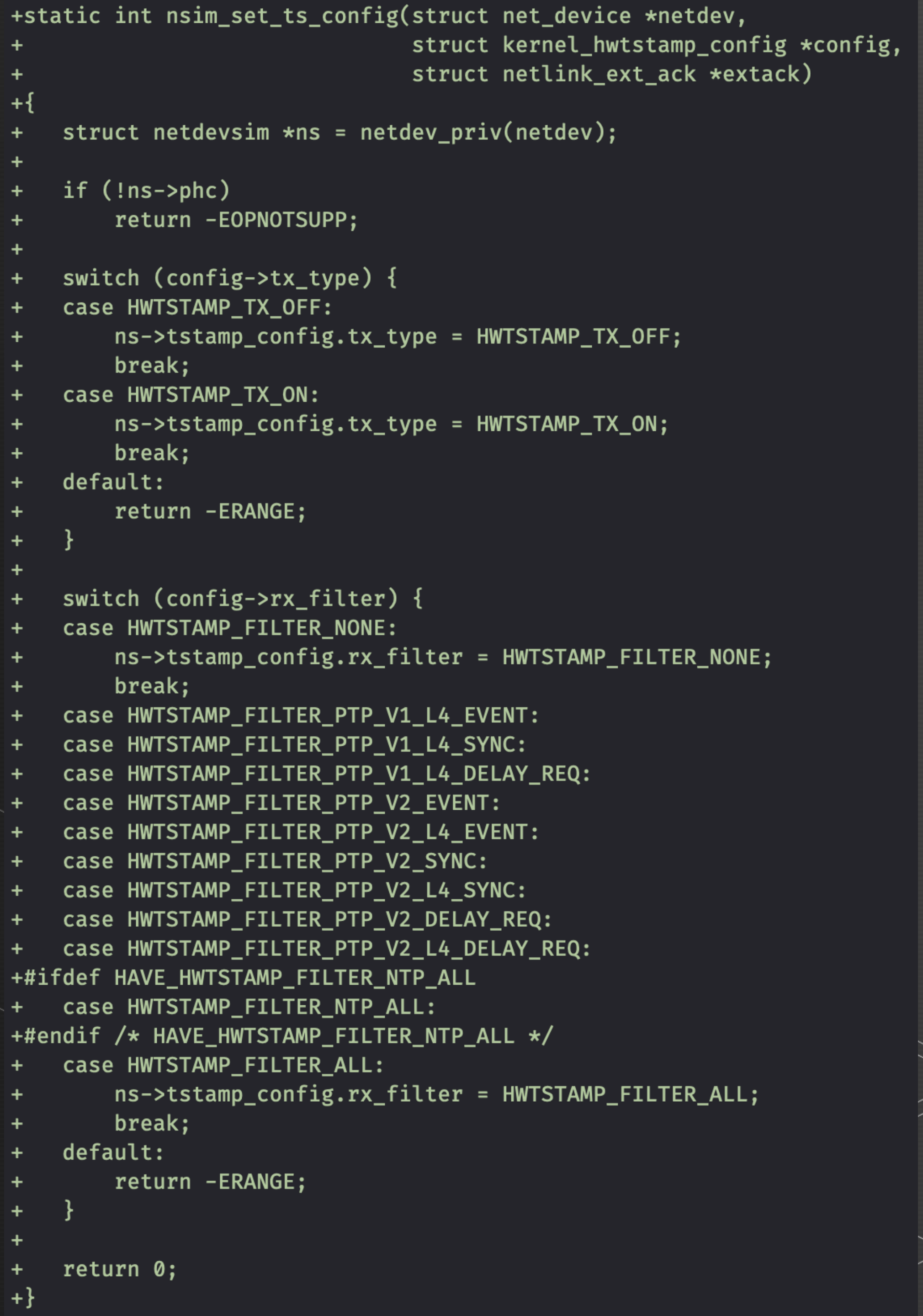
Step 2: set/get timestamp mode

- Set/get the configuration of timestamps
- Enable/disable timestamps
- Set HW timestamp filters
  - Fallback to FILTER_ALL if no filters are supported in the HW

Step 2: set/get timestamp mode

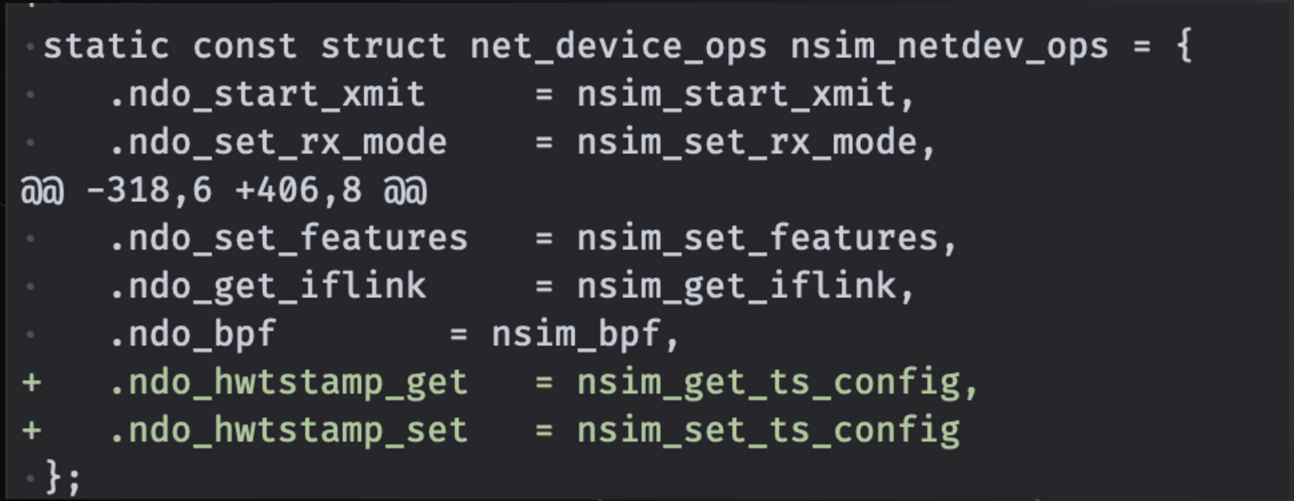


```
+static int nsim_get_ts_config(struct net_device *netdev,
+                             struct kernel_hwtstamp_config *config)
+{
+	struct netdevsim *ns = netdev_priv(netdev);
+
+	config = &ns->tstamp_config;
+	return 0;
+
+}
```



```
+static int nsim_set_ts_config(struct net_device *netdev,
+                             struct kernel_hwtstamp_config *config,
+                             struct netlink_ext_ack *extack)
+{
+	struct netdevsim *ns = netdev_priv(netdev);
+
+	if (!ns->phc)
+		return -EOPNOTSUPP;
+
+	switch (config->tx_type) {
+	case HWTSTAMP_TX_OFF:
+		ns->tstamp_config.tx_type = HWTSTAMP_TX_OFF;
+		break;
+	case HWTSTAMP_TX_ON:
+		ns->tstamp_config.tx_type = HWTSTAMP_TX_ON;
+		break;
+	default:
+		return -ERANGE;
+	}
+
+	switch (config->rx_filter) {
+	case HWTSTAMP_FILTER_NONE:
+		ns->tstamp_config.rx_filter = HWTSTAMP_FILTER_NONE;
+		break;
+	case HWTSTAMP_FILTER_PTP_V1_L4_EVENT:
+	case HWTSTAMP_FILTER_PTP_V1_L4_SYNC:
+	case HWTSTAMP_FILTER_PTP_V1_L4_DELAY_REQ:
+	case HWTSTAMP_FILTER_PTP_V2_EVENT:
+	case HWTSTAMP_FILTER_PTP_V2_L4_EVENT:
+	case HWTSTAMP_FILTER_PTP_V2_SYNC:
+	case HWTSTAMP_FILTER_PTP_V2_L4_SYNC:
+	case HWTSTAMP_FILTER_PTP_V2_DELAY_REQ:
+	case HWTSTAMP_FILTER_PTP_V2_L4_DELAY_REQ:
+#ifdef HAVE_HWTSTAMP_FILTER_NTP_ALL
+	case HWTSTAMP_FILTER_NTP_ALL:
+#endif /* HAVE_HWTSTAMP_FILTER_NTP_ALL */
+	case HWTSTAMP_FILTER_ALL:
+		ns->tstamp_config.rx_filter = HWTSTAMP_FILTER_ALL;
+		break;
+	default:
+		return -ERANGE;
+	}
+
+	return 0;
+}
```

Step 2: set/get timestamp mode

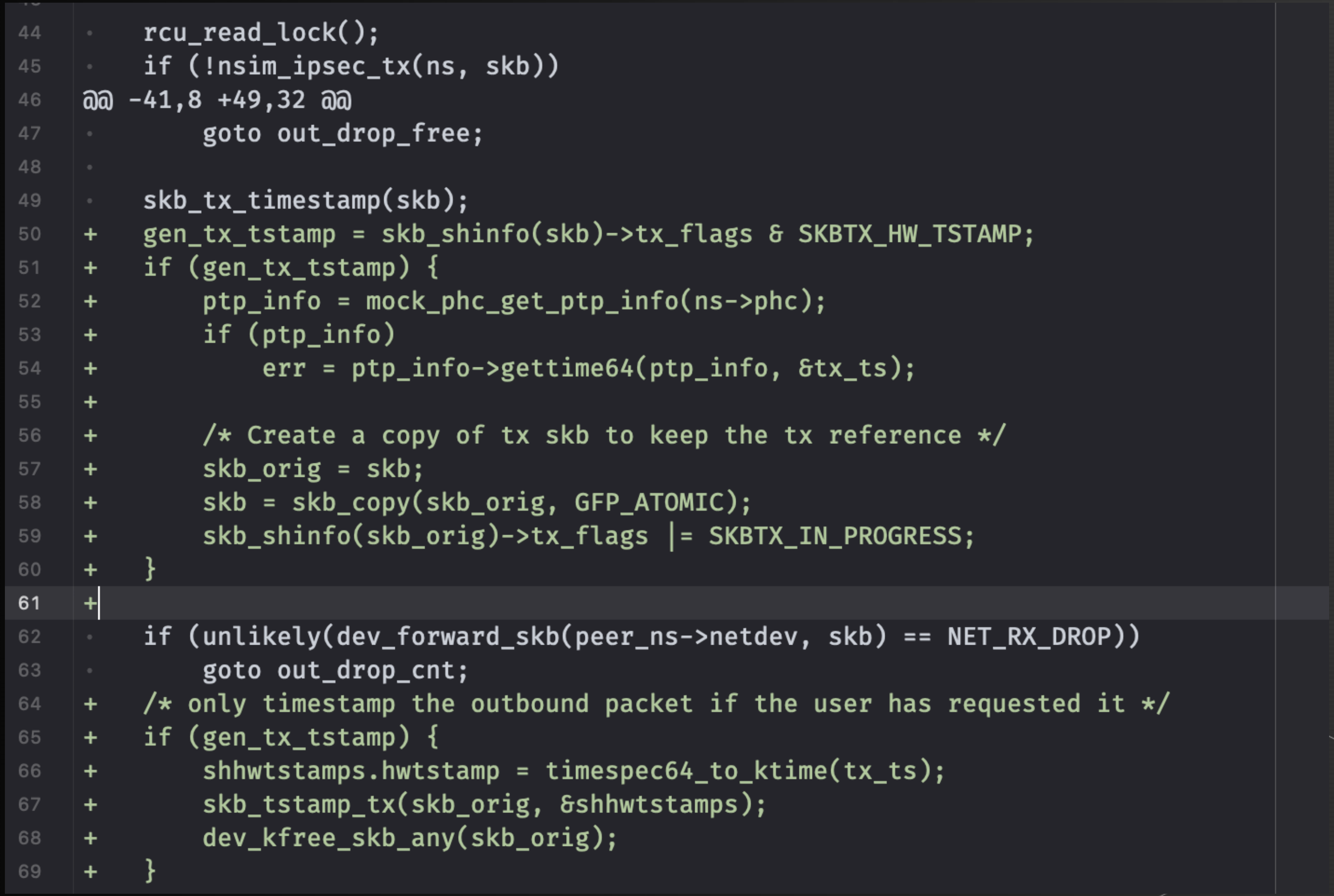


```diff
 static const struct net_device_ops nsim_netdev_ops = {
    .ndo_start_xmit     = nsim_start_xmit,
    .ndo_set_rx_mode    = nsim_set_rx_mode,
@@ -318,6 +406,8 @@
    .ndo_set_features   = nsim_set_features,
    .ndo_get_iflink     = nsim_get_iflink,
    .ndo_bpf        = nsim_bpf,
+   .ndo_hwtstamp_get   = nsim_get_ts_config,
+   .ndo_hwtstamp_set   = nsim_set_ts_config
 };
```

Step 3: Tx timestamping

- Transmit timestamps need to read the current time from the PHC allocated by the netdev
- But only when timestamping mode is enabled
- Need to keep the original SKB to return Tx timestamp back to the stack
- skb_tstamp_tx
- Only after that – release Tx SKB

Step 3: Tx timestamping



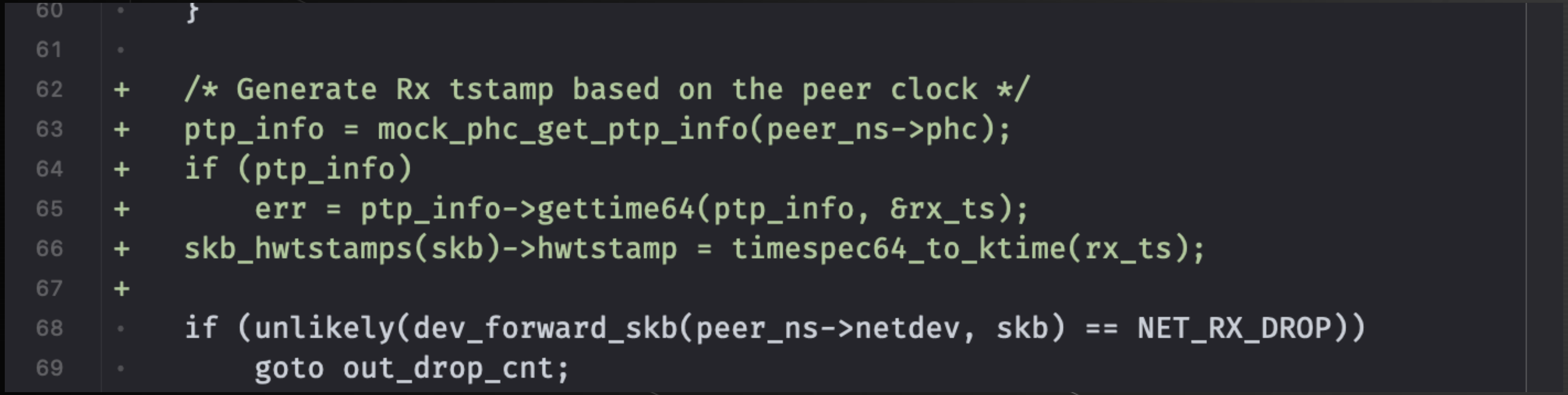
```
    rcu_read_lock();
    if (!nsim_ipsec_tx(ns, skb))
@@ -41,8 +49,32 @@
        goto out_drop_free;

    skb_tx_timestamp(skb);
+   gen_tx_tstamp = skb_shinfo(skb)->tx_flags & SKBTX_HW_TSTAMP;
+   if (gen_tx_tstamp) {
+       ptp_info = mock_phc_get_ptp_info(ns->phc);
+       if (ptp_info)
+           err = ptp_info->gettime64(ptp_info, &tx_ts);
+
+       /* Create a copy of tx skb to keep the tx reference */
+       skb_orig = skb;
+       skb = skb_copy(skb_orig, GFP_ATOMIC);
+       skb_shinfo(skb_orig)->tx_flags |= SKBTX_IN_PROGRESS;
+   }
+
    if (unlikely(dev_forward_skb(peer_ns->netdev, skb) == NET_RX_DROP))
        goto out_drop_cnt;
+   /* only timestamp the outbound packet if the user has requested it */
+   if (gen_tx_tstamp) {
+       shhwtstamps.hwtstamp = timespec64_to_ktime(tx_ts);
+       skb_tstamp_tx(skb_orig, &shhwtstamps);
+       dev_kfree_skb_any(skb_orig);
+   }
```

Step 4: Rx timestamping

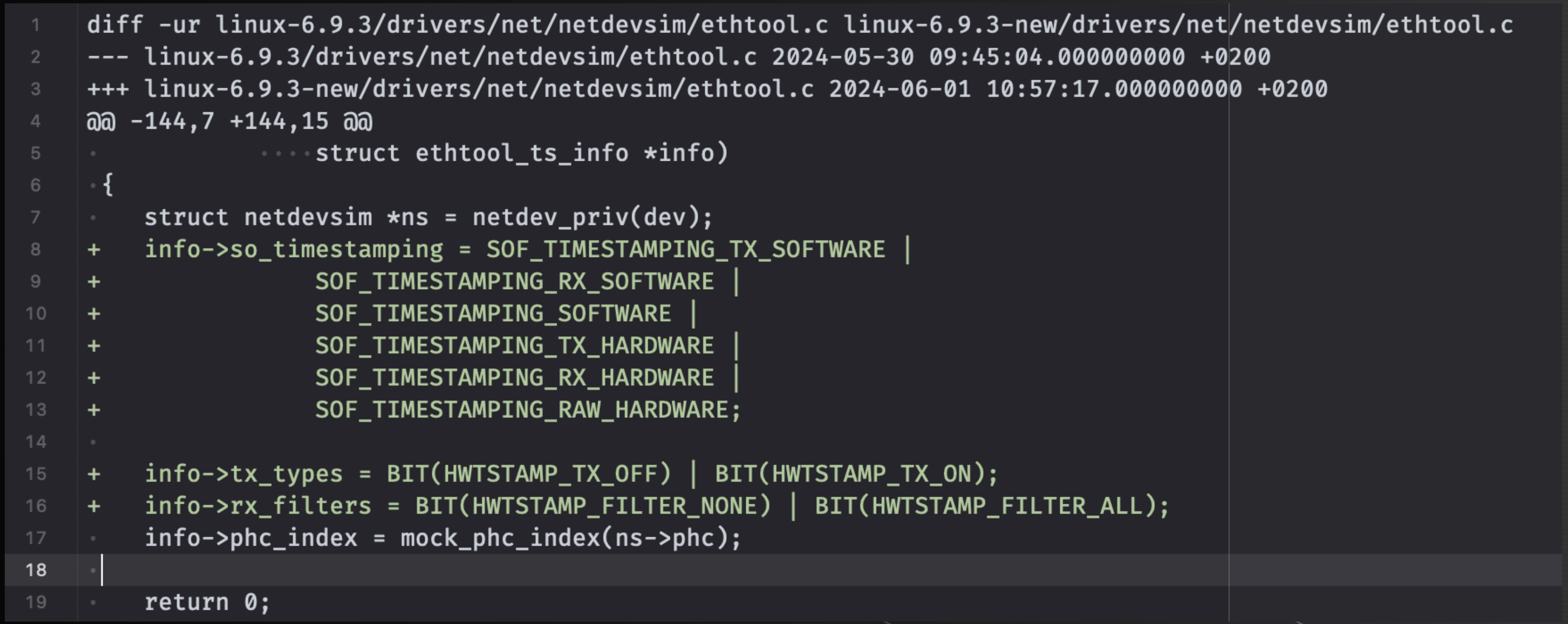
- Receive timestamp needs to read the time of the peer’s PHC
- Forward a copy of an SKB to pass it to the peer

Step 4: Rx timestamping



```
·   }
·
+   /* Generate Rx tstamp based on the peer clock */
+   ptp_info = mock_phc_get_ptp_info(peer_ns->phc);
+   if (ptp_info)
+       err = ptp_info->gettime64(ptp_info, &rx_ts);
+   skb_hwtstamps(skb)->hwtstamp = timespec64_to_ktime(rx_ts);
+
·   if (unlikely(dev_forward_skb(peer_ns->netdev, skb) == NET_RX_DROP))
·       goto out_drop_cnt;
```

Step 5: ethtool

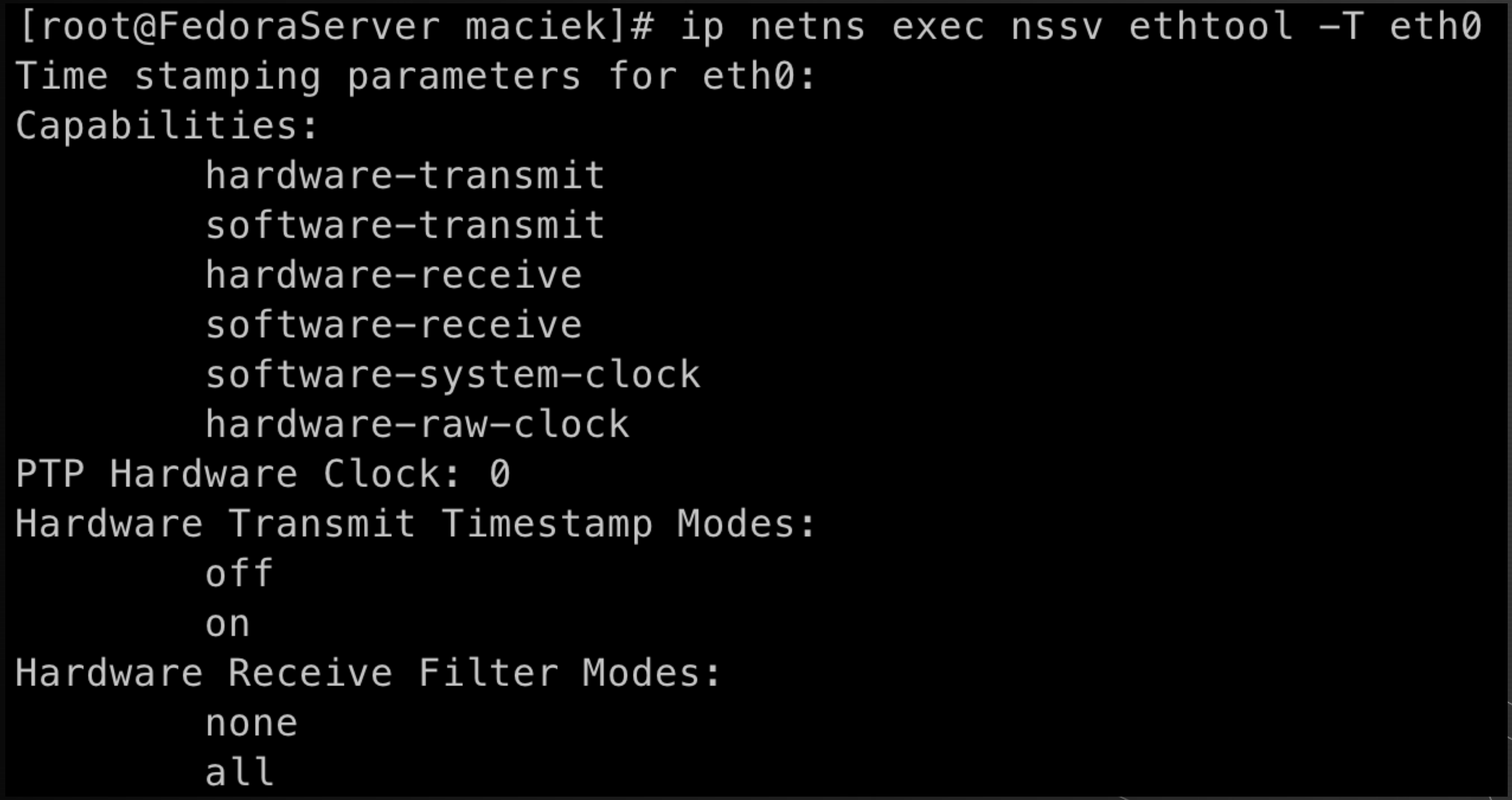


```
diff -ur linux-6.9.3/drivers/net/netdevsim/ethtool.c linux-6.9.3-new/drivers/net/netdevsim/ethtool.c
--- linux-6.9.3/drivers/net/netdevsim/ethtool.c 2024-05-30 09:45:04.000000000 +0200
+++ linux-6.9.3-new/drivers/net/netdevsim/ethtool.c 2024-06-01 10:57:17.000000000 +0200
@@ -144,7 +144,15 @@
                struct ethtool_ts_info *info)
 {
    struct netdevsim *ns = netdev_priv(dev);
+   info->so_timestamping = SOF_TIMESTAMPING_TX_SOFTWARE |
+               SOF_TIMESTAMPING_RX_SOFTWARE |
+               SOF_TIMESTAMPING_SOFTWARE |
+               SOF_TIMESTAMPING_TX_HARDWARE |
+               SOF_TIMESTAMPING_RX_HARDWARE |
+               SOF_TIMESTAMPING_RAW_HARDWARE;

+   info->tx_types = BIT(HWTSTAMP_TX_OFF) | BIT(HWTSTAMP_TX_ON);
+   info->rx_filters = BIT(HWTSTAMP_FILTER_NONE) | BIT(HWTSTAMP_FILTER_ALL);
    info->phc_index = mock_phc_index(ns->phc);

    return 0;
```

Step 5: ethtool



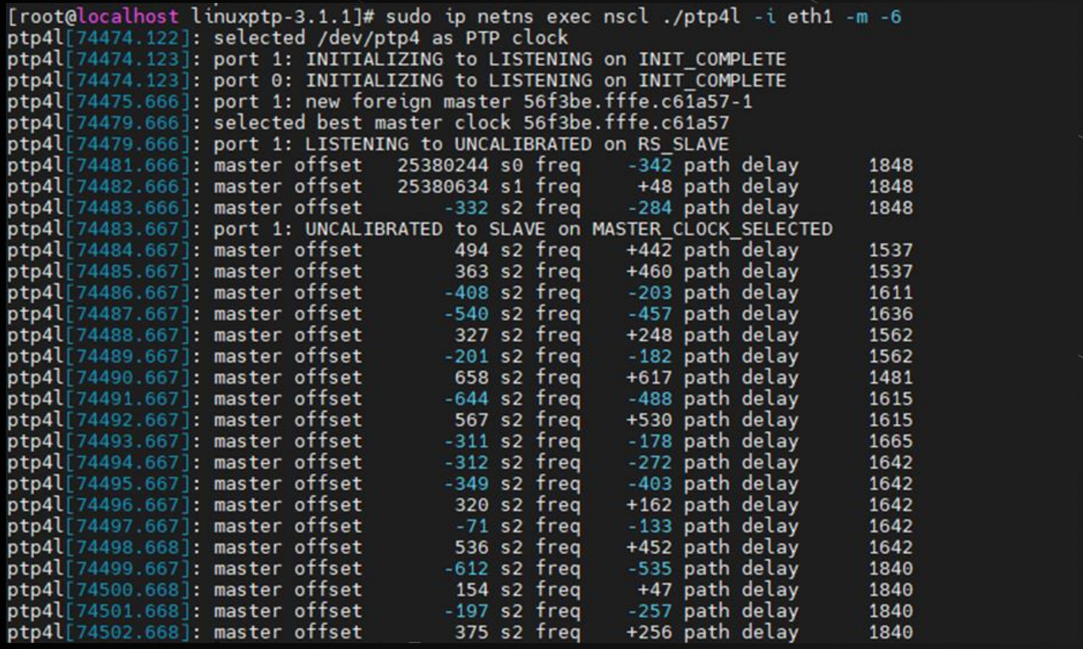
```
[root@FedoraServer maciek]# ip netns exec nssv ethtool -T eth0
Time stamping parameters for eth0:
Capabilities:
        hardware-transmit
        software-transmit
        hardware-receive
        software-receive
        software-system-clock
        hardware-raw-clock
PTP Hardware Clock: 0
Hardware Transmit Timestamp Modes:
        off
        on
Hardware Receive Filter Modes:
        none
        all
```

Running ptp4l

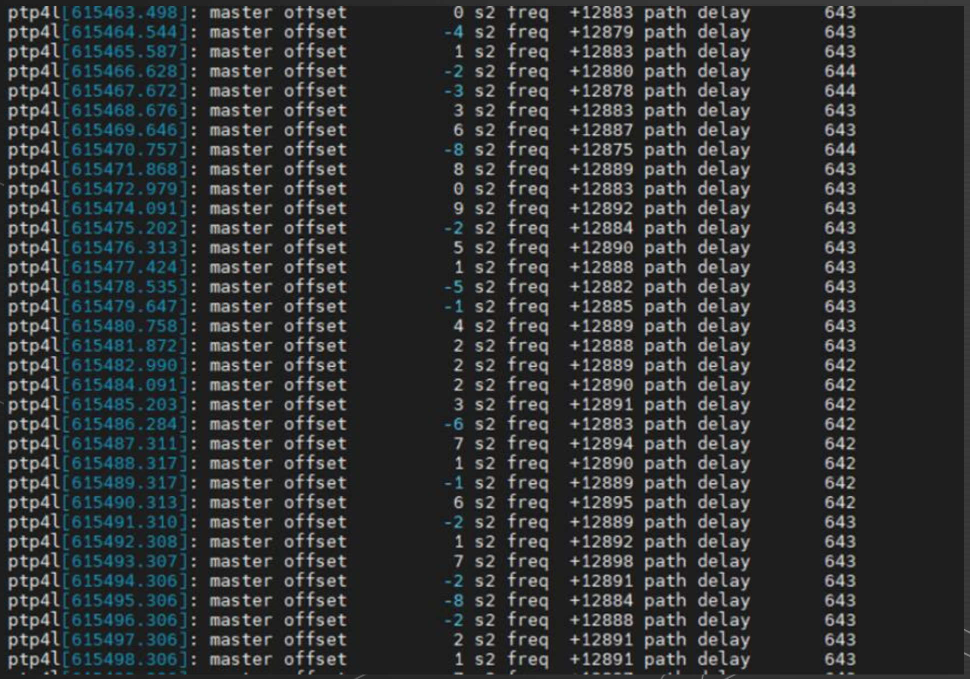
- Connected netdevsims require namespaces
- So we need to run ptp4l in namespaces
- Easy setup – reuse the peer.sh script
  - Creates two namespaces
  - And two netdevs
  - And connects them
  - Remove everything else that tries to clean-up or send data ☺

Limitations

- Low Timestamp quality
- ptp4l master offset ~300/400
- Traffic passed only when netdevsim interfaces are assigned to namespaces



```
[root@localhost linuxptp-3.1.1]# sudo ip netns exec nscl ./ptp4l -i eth1 -m -6
ptp4l[74474.122]: selected /dev/ptp4 as PTP clock
ptp4l[74474.123]: port 1: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[74474.123]: port 0: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[74475.666]: port 1: new foreign master 56f3be.fffe.c61a57-1
ptp4l[74479.666]: selected best master clock 56f3be.fffe.c61a57
ptp4l[74479.666]: port 1: LISTENING to UNCALIBRATED on RS_SLAVE
ptp4l[74481.666]: master offset   25380244 s0 freq    -342 path delay      1848
ptp4l[74482.666]: master offset   25380634 s1 freq     +48 path delay      1848
ptp4l[74483.666]: master offset       -332 s2 freq    -284 path delay      1848
ptp4l[74483.667]: port 1: UNCALIBRATED to SLAVE on MASTER_CLOCK_SELECTED
ptp4l[74484.667]: master offset        494 s2 freq    +442 path delay      1537
ptp4l[74485.667]: master offset        363 s2 freq    +460 path delay      1537
ptp4l[74486.667]: master offset       -408 s2 freq    -203 path delay      1611
ptp4l[74487.667]: master offset       -540 s2 freq    -457 path delay      1636
ptp4l[74488.667]: master offset        327 s2 freq    +248 path delay      1562
ptp4l[74489.667]: master offset       -201 s2 freq    -182 path delay      1562
ptp4l[74490.667]: master offset        658 s2 freq    +617 path delay      1481
ptp4l[74491.667]: master offset       -644 s2 freq    -488 path delay      1615
ptp4l[74492.667]: master offset        567 s2 freq    +530 path delay      1615
ptp4l[74493.667]: master offset       -311 s2 freq    -178 path delay      1665
ptp4l[74494.667]: master offset       -312 s2 freq    -272 path delay      1642
ptp4l[74495.667]: master offset       -349 s2 freq    -403 path delay      1642
ptp4l[74496.667]: master offset        320 s2 freq    +162 path delay      1642
ptp4l[74497.667]: master offset        -71 s2 freq    -133 path delay      1642
ptp4l[74498.668]: master offset        536 s2 freq    +452 path delay      1642
ptp4l[74499.667]: master offset       -612 s2 freq    -535 path delay      1840
ptp4l[74500.668]: master offset        154 s2 freq     +47 path delay      1840
ptp4l[74501.668]: master offset       -197 s2 freq    -257 path delay      1840
ptp4l[74502.668]: master offset        375 s2 freq    +256 path delay      1840
```



```
ptp4l[615463.498]: master offset          0 s2 freq  +12883 path delay       643
ptp4l[615464.544]: master offset         -4 s2 freq  +12879 path delay       643
ptp4l[615465.587]: master offset          1 s2 freq  +12883 path delay       643
ptp4l[615466.628]: master offset         -2 s2 freq  +12880 path delay       644
ptp4l[615467.672]: master offset         -3 s2 freq  +12878 path delay       644
ptp4l[615468.676]: master offset          3 s2 freq  +12883 path delay       643
ptp4l[615469.646]: master offset          6 s2 freq  +12887 path delay       643
ptp4l[615470.757]: master offset         -8 s2 freq  +12875 path delay       644
ptp4l[615471.868]: master offset          8 s2 freq  +12889 path delay       643
ptp4l[615472.979]: master offset          0 s2 freq  +12883 path delay       643
ptp4l[615474.091]: master offset          9 s2 freq  +12892 path delay       643
ptp4l[615475.202]: master offset         -2 s2 freq  +12884 path delay       643
ptp4l[615476.313]: master offset          5 s2 freq  +12890 path delay       643
ptp4l[615477.424]: master offset          1 s2 freq  +12888 path delay       643
ptp4l[615478.535]: master offset         -5 s2 freq  +12882 path delay       643
ptp4l[615479.647]: master offset         -1 s2 freq  +12885 path delay       643
ptp4l[615480.758]: master offset          4 s2 freq  +12889 path delay       643
ptp4l[615481.872]: master offset          2 s2 freq  +12888 path delay       643
ptp4l[615482.990]: master offset          2 s2 freq  +12889 path delay       642
ptp4l[615484.091]: master offset          2 s2 freq  +12890 path delay       642
ptp4l[615485.203]: master offset          3 s2 freq  +12891 path delay       642
ptp4l[615486.284]: master offset         -6 s2 freq  +12883 path delay       642
ptp4l[615487.311]: master offset          7 s2 freq  +12894 path delay       642
ptp4l[615488.317]: master offset          1 s2 freq  +12890 path delay       642
ptp4l[615489.317]: master offset         -1 s2 freq  +12889 path delay       642
ptp4l[615490.313]: master offset          6 s2 freq  +12895 path delay       642
ptp4l[615491.310]: master offset         -2 s2 freq  +12889 path delay       643
ptp4l[615492.308]: master offset          1 s2 freq  +12892 path delay       643
ptp4l[615493.307]: master offset          7 s2 freq  +12898 path delay       643
ptp4l[615494.306]: master offset         -2 s2 freq  +12891 path delay       643
ptp4l[615495.306]: master offset         -8 s2 freq  +12884 path delay       643
ptp4l[615496.306]: master offset         -2 s2 freq  +12888 path delay       643
ptp4l[615497.306]: master offset          2 s2 freq  +12891 path delay       643
ptp4l[615498.306]: master offset          1 s2 freq  +12891 path delay       643
```

Use-cases

- Netdevsim + PTP allows to **validate** PTP solutions **without HW access**
- Enable **LinuxPTP development** without HW acess
- Debugging
- Present **required kernel APIs**
- Kernel self-tests

Next steps

- Upstream
- Improve timestamp quality
  - With timecounters API and a single CLOCK_MONOTONIC_RAW sample

Demo

```
~ ❯
```