The existing interfaces for getting network packages time stamped are:

* SO_TIMESTAMP
  Generate time stamp for each incoming packet using the (not necessarily
  monotonous!) system time. Result is returned via recv_msg() in a
  control message as timeval (usec resolution).

* SO_TIMESTAMPNS
  Same time stamping mechanism as SO_TIMESTAMP, but returns result as
  timespec (nsec resolution).

* IP_MULTICAST_LOOP + SO_TIMESTAMP[NS]
  Only for multicasts: approximate send time stamp by receiving the looped
  packet and using its receive time stamp.

The following interface complements the existing ones: receive time
stamps can be generated and returned for arbitrary packets and much
closer to the point where the packet is really sent. Time stamps can
be generated in software (as before) or in hardware (if the hardware
has such a feature).

SO_TIMESTAMPING:

Instructs the socket layer which kind of information is wanted. The
parameter is an integer with some of the following bits set. Setting
other bits is an error and doesn't change the current state.

```
SOF_TIMESTAMPING_TX_HARDWARE:  try to obtain send time stamp in hardware
SOF_TIMESTAMPING_TX_SOFTWARE:  if SOF_TIMESTAMPING_TX_HARDWARE is off or
                               fails, then do it in software
SOF_TIMESTAMPING_RX_HARDWARE:  return the original, unmodified time stamp
                               as generated by the hardware
SOF_TIMESTAMPING_RX_SOFTWARE:  if SOF_TIMESTAMPING_RX_HARDWARE is off or
                               fails, then do it in software
SOF_TIMESTAMPING_RAW_HARDWARE: return original raw hardware time stamp
SOF_TIMESTAMPING_SYS_HARDWARE: return hardware time stamp transformed to
                               the system time base
SOF_TIMESTAMPING_SOFTWARE:     return system time stamp generated in
                               software
```

SOF_TIMESTAMPING_TX/RX determine how time stamps are generated.
SOF_TIMESTAMPING_RAW/SYS determine how they are reported in the
following control message:

```
struct scm_timestamping {
        struct timespec systime;
        struct timespec hwtimetrans;
        struct timespec hwtimeraw;
};
```

recvmsg() can be used to get this control message for regular incoming
packets. For send time stamps the outgoing packet is looped back to
the socket's error queue with the send time stamp(s) attached. It can
be received with recvmsg(flags=MSG_ERRQUEUE). The call returns the
original outgoing packet data including all headers preprended down to
and including the link layer, the scm_timestamping control message and

a sock_extended_err control message with ee_errno==ENOMSG and
ee_origin==SO_EE_ORIGIN_TIMESTAMPING. A socket with such a pending
bounced packet is ready for reading as far as select() is concerned.
If the outgoing packet has to be fragmented, then only the first
fragment is time stamped and returned to the sending socket.

All three values correspond to the same event in time, but were
generated in different ways. Each of these values may be empty (= all
zero), in which case no such value was available. If the application
is not interested in some of these values, they can be left blank to
avoid the potential overhead of calculating them.

systime is the value of the system time at that moment. This
corresponds to the value also returned via SO_TIMESTAMP[NS]. If the
time stamp was generated by hardware, then this field is
empty. Otherwise it is filled in if SOF_TIMESTAMPING_SOFTWARE is
set.

hwtimeraw is the original hardware time stamp. Filled in if
SOF_TIMESTAMPING_RAW_HARDWARE is set. No assumptions about its
relation to system time should be made.

hwtimetrans is the hardware time stamp transformed so that it
corresponds as good as possible to system time. This correlation is
not perfect; as a consequence, sorting packets received via different
NICs by their hwtimetrans may differ from the order in which they were
received. hwtimetrans may be non-monotonic even for the same NIC.
Filled in if SOF_TIMESTAMPING_SYS_HARDWARE is set. Requires support
by the network device and will be empty without that support.

SIOCSHWTSTAMP:

Hardware time stamping must also be initialized for each device driver
that is expected to do hardware time stamping. The parameter is defined in
/include/linux/net_tstamp.h as:

```
struct hwtstamp_config {
        int flags;      /* no flags defined right now, must be zero */
        int tx_type;    /* HWTSTAMP_TX_* */
        int rx_filter;  /* HWTSTAMP_FILTER_* */
};
```

Desired behavior is passed into the kernel and to a specific device by
calling ioctl(SIOCSHWTSTAMP) with a pointer to a struct ifreq whose
ifr_data points to a struct hwtstamp_config. The tx_type and
rx_filter are hints to the driver what it is expected to do. If
the requested fine-grained filtering for incoming packets is not
supported, the driver may time stamp more than just the requested types
of packets.

A driver which supports hardware time stamping shall update the struct
with the actual, possibly more permissive configuration. If the
requested packets cannot be time stamped, then nothing should be
changed and ERANGE shall be returned (in contrast to EINVAL, which
indicates that SIOCSHWTSTAMP is not supported at all).

Only a processes with admin rights may change the configuration. User space is responsible to ensure that multiple processes don't interfere with each other and that the settings are reset.

```
/* possible values for hwtstamp_config->tx_type */
enum {
        /*
         * no outgoing packet will need hardware time stamping;
         * should a packet arrive which asks for it, no hardware
         * time stamping will be done
         */
        HWTSTAMP_TX_OFF,

        /*
         * enables hardware time stamping for outgoing packets;
         * the sender of the packet decides which are to be
         * time stamped by setting SOF_TIMESTAMPING_TX_SOFTWARE
         * before sending the packet
         */
        HWTSTAMP_TX_ON,
};

/* possible values for hwtstamp_config->rx_filter */
enum {
        /* time stamp no incoming packet at all */
        HWTSTAMP_FILTER_NONE,

        /* time stamp any incoming packet */
        HWTSTAMP_FILTER_ALL,

        /* return value: time stamp all packets requested plus some others */
        HWTSTAMP_FILTER_SOME,

        /* PTP v1, UDP, any kind of event packet */
        HWTSTAMP_FILTER_PTP_V1_L4_EVENT,

        /* for the complete list of values, please check
         * the include file /include/linux/net_tstamp.h
         */
};
```

DEVICE IMPLEMENTATION

A driver which supports hardware time stamping must support the SIOCSHWTSTAMP ioctl and update the supplied struct hwtstamp_config with the actual values as described in the section on SIOCSHWTSTAMP.

Time stamps for received packets must be stored in the skb. To get a pointer to the shared time stamp structure of the skb call skb_hwtstamps(). Then set the time stamps in the structure:

```
struct skb_shared_hwtstamps {
        /* hardware time stamp transformed into duration
         * since arbitrary point in time
```

```
         */
        ktime_t hwtstamp;
        ktime_t syststamp; /* hwtstamp transformed to system time base */
};
```

Time stamps for outgoing packets are to be generated as follows:
- In hard_start_xmit(), check if skb_tx(skb)->hardware is set no-zero.
  If yes, then the driver is expected to do hardware time stamping.
- If this is possible for the skb and requested, then declare
  that the driver is doing the time stamping by setting the field
  skb_tx(skb)->in_progress non-zero. You might want to keep a pointer
  to the associated skb for the next step and not free the skb. A driver
  not supporting hardware time stamping doesn't do that. A driver must
  never touch sk_buff::tstamp! It is used to store software generated
  time stamps by the network subsystem.
- As soon as the driver has sent the packet and/or obtained a
  hardware time stamp for it, it passes the time stamp back by
  calling skb_hwtstamp_tx() with the original skb, the raw
  hardware time stamp. skb_hwtstamp_tx() clones the original skb and
  adds the timestamps, therefore the original skb has to be freed now.
  If obtaining the hardware time stamp somehow fails, then the driver
  should not fall back to software time stamping. The rationale is that
  this would occur at a later time in the processing pipeline than other
  software time stamping and therefore could lead to unexpected deltas
  between time stamps.
- If the driver did not call set skb_tx(skb)->in_progress, then
  dev_hard_start_xmit() checks whether software time stamping
  is wanted as fallback and potentially generates the time stamp.