



# TSN and Linux




**The Time Sensitive Networking (TSN) extensions allow real-time capable and deterministic communication via conventional Ethernet. As operating system for end devices very often Linux is used, which is extended by real-time capability especially in connection with the PRE-EMPT-RT patch. Therefore, Linux is a popular choice as operating system for those devices and switches using TSN.**


Still today, TSN networks are often developed with proprietary or special solution-based Linux approaches. The question arises: How far can TSN networks yet be realized with Mainline Linux? The basic functionalities such as time synchronization and traffic scheduling can already be implemented today. Yet, not all TSN-related standards have been fully implemented by now.

## TSN solutions

Linux in combination with the PREEMPT-RT Patch [1] or Xenomai [2] is a very popular and widely used operating system for industrial devices. Accordingly, TSN is an important use case for real time Linux. This includes applications ranging from Audio/Video broadcasting to industrial control systems utilizing fieldbuses. Traditionally, real time networking on Linux has been accomplished by bypassing the Linux kernel network stack completely. Such solutions include DPDK [3] or vendor specific proprietary implementations.

All these variants have in common that modifications must be made to the Linux kernel. This is true for the involved network drivers, the data and control plane from the application to the kernel as well as the configuration of a TSN network. This leads to the following problems:

-  Network interface exclusive for real time, no co-existence with best effort traffic
-  No reuse of any standard Linux interfaces and tools
-  Requirement of manufacture specific applications

-  Increased maintenance effort due to third party components

This leads to the question whether TSN networks can actually be realized with Mainline Linux? This question will be answered below.

## TSN with Mainline Linux

Exactly three components are required for TSN: Time synchronization, traffic scheduling and configuration. A common time base of devices and switches provides the foundation for deterministic network scheduling. Implementing the requirements of the applications and synchronizing all components in the network is part of the configuration.

## Time synchronization

All end devices and bridges require the same understanding of time. The real-time capability of the network ensures that each participant performs the right actions at the right time. This requires synchronization of the internal clocks between all participating partners. In TSN networks, this is done via Ethernet itself using the generalized Precision Time Protocol (gPTP). This protocol ensures an accuracy in the range below one microsecond.

The Linux kernel offers its own subsystem [4] for controlling PTP hardware clocks (PHC). The clocks themselves are displayed as POSIX clocks. Thus, you can use the standard interfaces for reading, setting and controlling the clocks. The PTP protocol itself is not part of the kernel, but is implemented in the user space. The Linux kernel provides hardware access to the clocks. The Linux PTP stack can be seen in Figure 1.



One of the most popular Linux User Space PTP Stacks is linuxptp[5]. Linuxptp implements the PTP protocol according to both IEEE 1588 and IEEE 802.1AS-2011 in the role of a terminal sta-

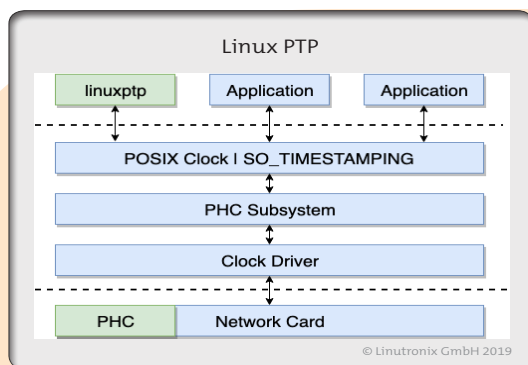


Fig. 1: Linux PTP stack

tion. In TSN networks, time synchronization is performed via 802.1AS. This standard is a subset of 1588 and offers extensions e.g. for synchronization via WiFi. TSN is currently planned to use the standard IEEE 802.1AS-2020, which is, at least for the most components, not yet supported in Mainline Linux.

PTP constantly performs peer delay measurements. Therefore, accurate timestamps for PTP messages are required. Linux supports this for MAC as well as PHY timestamping via the SO\_TIMESTAMPING [6] interface.

## Traffic scheduling

The determinism and thus the real-time capability in TSN networks are ensured by various shapers and schedulers. The available bandwidth is controlled and divided among the applications according to the requirements.

Traditionally, applications under Linux communicate with networks via the socket interface. The network stack handles packet handling, protocol implementations such as TCP/IP, UDP/IP or Raw Ethernet and hardware control. The network stack is configured by using schedulers, classes and filters. These mechanisms have been established and understood for years. Since TSN is an extension to Ethernet, the current Linux implementation is also based on these components. Accordingly, TSN integrates itself directly into the Linux network stack. The principle can be seen in Figure 2.

This principle allows both applications with real-time requirements and existing applications to coexist on a system without any special requirements and use the same interface. The Linux kernel controls access to the network hardware and ensures that packets are sent at the correct time. That is the Time Slot Management component. However, the kernel must be configured correctly. Because the requirements come from the application. The network stack is configured using Queuing Disciplines (Qdiscs).

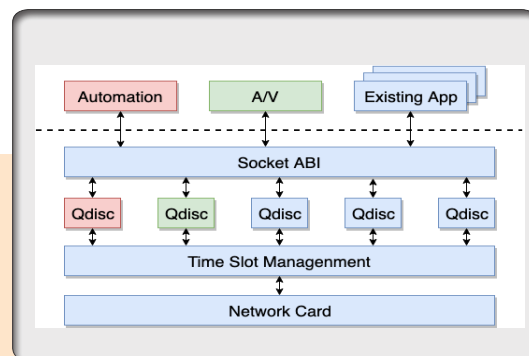


Fig. 2: Linux TSN implementation

A Qdisc corresponds to a packet scheduler. The scheduler decides the point of time when a packet is given to the network hardware or application. As part of the TSN implementation, various Qdiscs have been developed in recent years and integrated into the Mainline Linux kernel.

Applications for controlling industrial plants usually have to send packets in regular cycles and with as little jitter as possible. Modern network cards offer the possibility that the hardware itself sends frames at a certain time. Exactly for this scenario a Qdisc called Earliest Tx Time [8] was developed. The configuration is done via the traffic control subsystem and its associated user space tool tc. An example can be seen in Figure 3.

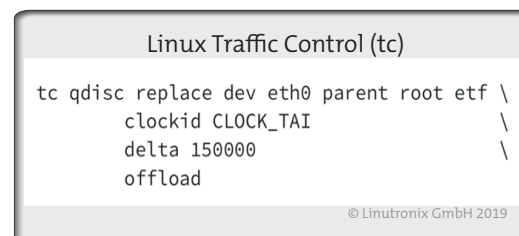


Fig.3: Linux traffic control

This tc call replaces the current Qdisc of the eth0 interface with ETF. The timestamps are specified in reference to the TAI clock. The application can specify a timestamp for each frame via a socket option in the user space (SO\_TXTIME). The offload parameter specifies that the hardware offloading of the network card should be used. If that is not available, a software implementation is used.

Traffic shaping is just as important as minimizing jitter. The TSN standards define several different mechanisms. Quite common especially in audio and video streaming environments is the limitation of traffic to a fixed bandwidth. For this purpose, the Linux kernel since version 4.15 contains the Credit Based Shaper [9], also implemented as Qdisc. An example configuration can be found in Figure 4.

The configuration parameters are taken from the 802.Qav standard.

Real-time capability is ensured not only by bandwidth limitation, but also by time slot

```
tc qdisc replace dev eth0 parent root cbs \
  locredit -1470 hicredit 30 sendslope -980000 \
  idleslope 20000 \
  offload 1
```

Fig.4: Linux CBS  
Fig.4: Linux CBS

management. The TSN standard 802.Qbv defines the Time Aware Shaper for this purpose. The Linux kernel already contains such an implementation, the so-called TAPRIO [10] Qdisc. That scheduler allows the subdivision of the bandwidth into cyclic repeating time slots. The Ethernet traffic is divided into different classes, and slots are assigned to them. An example cycle can be seen in Figure 5.

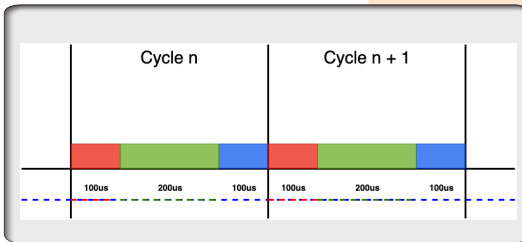


Fig.5: TSN time slot management

This cycle contains three time slots for three different traffic classes, e.g. for automation, streaming and general purpose applications. Each traffic class has a slot of 100 or 200 microseconds. After 400 microseconds, the cycle is repeated and the next one begins.

This example cycle can be configured for a network interface as follows:

```
tc qdisc replace dev eth0 parent root handle 100 taprio \
  num_tc 3 \
  map 2 2 1 0 2 2 2 2 2 2 2 2 2 2 2 2 \
  queues 1@0 1@1 2@2 \
  base-time 1570615200123456789 \
  sched-entry S 01 200000 \
  sched-entry S 02 400000 \
  sched-entry S 04 200000 \
  clockid CLOCK_TAI \
  flags 0x02
```

Fig.6: Linux taprio configuration

In this configuration the number of traffic classes is determined first. The assignment of packets in the user space to these traffic classes is done by the map parameter via the 16 socket priorities. TSN capable hardware usually has several Tx queues. The traffic classes are assigned to the queues based on the queues parameter. The actual cycle and the start time are created afterwards. The flags can be used to make configurations such as hardware offloading.

## Real Time Ethernet Communication

Apart from the time synchronization and traffic scheduling configuration, Linux has to support real time Ethernet communication for applications as well. The Linux network stack has been optimized for Layer 4 (TCP and UDP over IP) and throughput. Therefore, it is not well suited for deterministic and real time communication.

The introduction of the eXpress Data Path (XDP) [11] technology into the Linux kernel closed that gap. XDP is a in-kernel fast path which allows to process Ethernet frames before they are passed to the regular Linux network stack. Therefore, XDP makes use of eBPF [12]. This allows the applications to provide user logic in terms of eBPF programs. These programs decide how incoming Ethernet frames are handled. A new socket type (AF\_XDP) has been introduced. This allows to redirect Ethernet frames directly into the applications without further processing in the kernel. The memory management is moved from the network stack into the application which allows more deterministic handling. Furthermore, XDP sockets provide the ability to perform zero-copy which decreases frame processing times.



One significant advantage of XDP in contrast to traditional kernel bypass mechanisms is that XDP is well integrated into the kernel. Based on the decision made by the eBPF programs a frame can be either treated as real time data for an application or still moved to the Linux kernel for further processing. This allows for co-existence between general purpose and real time traffic on the same network interface at the same time

## Configuration

Time synchronization and traffic scheduling form the basis for TSN networks. But how are the schedules and individual parameters to be calculated and finally distributed to the terminal stations and switches? The relevant requirements come from the applications. The 802.1Qcc standard defines the management and protocols for TSN configuration. On the one hand there is the Central Network Controller (CNC), which determines the communication paths or cycles and configures the switches, on the other hand there is the Centralized User Configuration (CUC), which receives the requirements from the applications and passes them on to the CNC. The structure is shown in Figure 7

In addition to the centralised model, further approaches such as decentralised or mixed forms of the two approaches are conceivable. The protocols used for configuration include NETCONF, RESTCONF and Yang models.

The configuration is an orthogonal problem for a Linux system. The CUC and CNC as well as the necessary programs on the terminal stations or switches are just software from the point of view of the Linux kernel. Nevertheless, there is a requirement for open solutions. Examples for such open source TSN configuration software include:

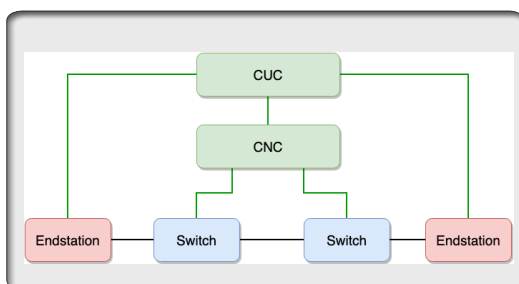


Fig.7: TSN configuration

- ControlTSN, Research Project  
<https://www.accesstsn.com/>
- OpenCUC: Prototype OPC/UA PubSub  
<https://github.com/openCUC/openCUC>
- OpenCNC: Prototype CNC

## Conclusion

The basic functions such as time synchronization, traffic scheduling as well as deterministic frame transmission and reception are possible today with a standard Linux system. The basic infrastructure has been created in the Linux kernel. This means that basic TSN networks can be realized with (mainline) Linux. Not all TSN standards are supported yet. Table 1 shows the current status and whether the standard is relevant for Linux at all.

Standard	Status	Linux relevant	Linux support
802.1AS-2011	published	yes	in parts
802.1AS-2020	published	yes	in parts
802.1Qav	published	yes	yes
802.1Qbv	published	yes	yes
802.1Qbu	published	yes	in progress
802.1Qbr	published	yes	in progress
802.1Qca	published	no	no
802.1Qcc	published	no	no
802.1Qch	published	yes	no
802.1Qci	published	yes	in parts
802.1QCB	published	yes	in progress

Table.2: TSN Standards

In summary, there is no fundamental problem that would make it impossible to implement these standards and the necessary Linux extensions. Accordingly, it is only a matter of time before all standards are supported in (mainline) Linux.

## Bibliography and Sources

- [1] „PREEMPT\_RT Wiki,“ [Online]. Available: <https://wiki.linuxfoundation.org/realtime/start>. [accessed 07 Sep 2019].
- [2] „Xenomai,“ [Online]. Available: <https://xenomai.org/>. [accessed 07 Sep 2019].
- [3] „Data Plane Development Kit“ [Online]. Available: <https://www.dpdk.org/>. [accessed 07 Sep 2019].
- [4] R. Cochran, „PTP (Precision Time Protocol) Documentation,“ [Online]. Available: <https://www.kernel.org/doc/Documentation/ptp/ptp.txt>. [accessed 07 Sep 2019].
- [5] R. Cochran, „The Linux PTP Project,“ [Online]. Available: <http://linuxptp.sourceforge.net/>. [accessed 07 Sep 2019].
- [6] P. Ohly, „The Linux Kernel Archives,“ [Online]. Available: <https://www.kernel.org/doc/Documentation/networking/timestamping.txt>. [accessed 06 Sep 2019].
- [7] T. Gleixner, „Evolution and current status of TSN in Linux,“ in TSN/A Konferenz, Stuttgart, 2018.
- [8] J. Sanchez-Palencia, „ETF (Earliest TxTime First) Documentation,“ [Online]. Available: <http://man7.org/linux/man-pages/man8/tc-ETF.8.html>. [accessed 06 Sep 2019].
- [9] V. C. Gomes, „CBS (Credit Based Shaper) Dokumentation,“ [Online]. Available: <http://man7.org/linux/man-pages/man8/tc-cbs.8.html>. [accessed 07 Sep 2019].
- [10] V. C. Gomes, „TAPRIO (Time Aware Priority Shaper) Dokumentation,“ [Online]. Available: <http://man7.org/linux/man-pages/man8/tc-taprio.8.html>. [accessed 07 Sep 2019].
- [11] „AF\_XDP,“ [Online]. Available: [https://www.kernel.org/doc/html/latest/networking/af\\_xdp.html](https://www.kernel.org/doc/html/latest/networking/af_xdp.html). [accessed 02 Sep 2022].
- [12] „BPF Documentation,“ [Online]. Available: <https://docs.kernel.org/bpf/index.html>. [accessed 02 Sep 2022].



Are you interested? Would you like to learn more about our products and solutions? Simply contact us via telephone or email.

### LINUTRONIX GMBH

Bahnhofstr. 3 | D-88690 Uhlidingen - Mühlhofen  
Phone +49 7556 25 999 0 | Fax +49 7556 25 999 99  
[sales@linutronix.de](mailto:sales@linutronix.de) | [www.linutronix.de](http://www.linutronix.de)

**LINUTRONIX**  
L I N U X F O R I N D U S T R Y