“We’ve got realtime networking at home” – why many systems are moving to TSN so slowly

Johannes Zink

Pengutronix e.K.
Hildesheim, Germany
j.zink@pengutronix.de

Abstract

Since several years many companies have tried to deliver realtime data e.g. audio, video or industrial control commands over the network for a variety of applications. Often, the implementations failed to deliver the promised performance under circumstances such as heavily loaded networks or were incompatible with regular network protocols. While TSN fixes most of these issues, brownfield development and migration is a challenge. This paper will look into a variety of system design approaches, into promises kept and broken and into the consequences of different system design choices. It will also propose how to move towards modern implementations and discuss the role of the Linux kernel in this process.

Keywords

Realtime Networks, TSN, QoS, PTP, Resource Management, Implementation, Brownfield Deployment

Introduction

Even before TSN introduced bandwidth reservation and guaranteed bound end-to-end transmit latencies in Ethernet networks, manufacturers decided to move the transport layer of their realtime critical communication to Ethernet, mostly due to widespread availability and low cost of Ethernet hardware components.

Application Requirements for Realtime Networks

The most obvious use cases for distributing realtime data over networks include audio and video transmission and industrial control applications. Both have strong requirements with respect to clock synchronization, quality of service, end-to-end-latency, though aspects like network management (i.e. stream setup) may differ. While engineered network setups (i.e. setups with carefully designed topology, selected components and calculated bandwidth requirements) may be acceptable for industrial control, especially live audio and video transmission should work out-of-the-box with a wide variety of gear and topologies.

In general, the requirements can be analyzed with respect to

- time synchronization
- bounded transmission latency
- quality of service

Some applications may also require additional features such as physical layer redundancy.

Legacy Implementation Approaches

Most legacy implementations aim to improve determinism by segregating realtime from non-realtime traffic, while using as low as possible load on the realtime network segments. This way, no interfering traffic has to be taken into account when designing the network setup. Of course, these approaches do not integrate well into current converged network setups.

The most common implementations either split these domains into separate physical connections, or use VLANs. Additionally, some implementations employ QoS mechanisms such as DSCP in order to prioritize the more timing critical parts of their realtime traffic over the less critical parts and over in-band management traffic.

Since most applications do not require traffic to be routed, some implementations use non-standard Layer 3 and higher protocols instead of standard IP.

Time Synchronization

Many different time synchronization mechanisms have been deployed in the field. Nevertheless, most of them rely on variants of IEEE1588 PTP, though some use layer 2 implementations, mostly using FPGAs to implement daisy-chained Ethernet Links with clock recovery from the bitstream, exist.

For improving timing precision, transparent and boundary clocks are used in PTP, but since this usually requires support by the switch firmware, most legacy implementations, which do use only subsets of standard PTP, cannot take advantage of this possibility.

Bounded Transmission Latency

Most legacy implementations cannot guarantee bounded transmission latency, but often engineered networks massively overprovision bandwidth and usually try to avoid any interfering non-realtime traffic. Some implementations try to improve on this by using QoS mechanisms.

Quality of Service

Since there are no means of reserving bandwidth or transmission time slots in non-TSN ethernet, the only way of lowering end-to-end transmission latency and therefore lowering the risk of not delivering packets in time is to employ QoS mechanisms such as DSCP. These mechanisms will cause resources, namely bandwidth, to be scheduled preferably to realtime traffic. Since the underlying hardware may or may not support these means without signaling the possible lack of support, replacing parts of the network setup can result in deteriorated realtime performance.

Network Management

Most implementations use proprietary network management, especially for configuration of data streams and for resource management.

For most implementations, setup is performed before starting realtime operation. Often additional inband signaling is used for this, but most implementations switch off or at least reduce configuration options at run time, since this could interfere with the realtime operation.

Advantages and shortcomings of legacy implementations

Since the systems deployed in the field have been designed long before TSN or even AVB standards were available, they usually do not rely on additional features that go beyond standard ethernet operation. This allows to operate said systems even on legacy hardware, often without much specific hardware acceleration or support for modern standards such as TSN. While this does not seem as an advantage, it is sometimes regarded as such, since it often has lower requirements on networking equipment which results in lower installation cost. This comes of course with the obvious disadvantage of no realtime guarantees and therefore impeded operational stability, especially under load or unusual operating conditions. To account for this, and since retransmission is, due to timing constraints, usually not feasible, most implementations provide different coping mechanisms for packet or even link loss. Besides special constraints, like entering a safe state upon packet loss, e.g. in industrial control applications, most implementations use either redundant physical transmission paths with packet duplication at the sender and deduplication at the receiver, or transmit overlapping jitterbuffer and interpolation upon single lost packets. Nevertheless, the networks deployed are usually kept segregated and most of the time highly engineered, converged networks or ad-hoc setups without planning and testing are often strongly discouraged my manufacturers.

TSN – a general solution to all problems?

The TSN standard set provides solution approaches for most of the issues described. gPTP offers a very tight and precise way of clock synchronization, while traffic shaping and stream reservation provide a good solution for guaranteed bandwidth and guaranteed quality of service. Nevertheless, these standards only lay the groundwork for building tech stacks, and often existing protocols are only a mediocre fit for making well-generalized realtime capable solutions from legacy implementations. Also, TSN does not cover all requirements, e.g. only basic redundancy schemes can be implemented with the 802.1Qca and 802.1CB extensions. Nevertheless, TSN provides a standardized way for making ethernet traffic realtime capable under very generalized conditions.

Pushing TSN to the brownfield

As mentioned earlier, lots of highly specialized and creatively engineered solutions have been deployed in the field throughout the years. Often, these solutions were complex and expensive. While more recent technologies were iteratively pushed to the market, often adaption in the field was rather slow. With TSN, the same issue can be observed. Not only has availability for hardware support, which is required especially for gPTP and for traffic shaping, been limited for several years, but also the frequent changes in standards on the transition from AVB to TSN made it difficult not only for hardware vendors, but also for network stack to provide stable interfaces, APIs and documentation to developers building upon this technology. While these issues are now mostly solved thanks to mainline support for all important features in the network stack, moving application stacks and system design towards the new frontiers still proves to be difficult. Highly complex stacks that often require deep understanding of the respective standards are slowing development efforts, while incompatibilities to existing legacy hardware in the field block incremental migration with continuous upgrades, since the entire networked application can only be run at the lowest commonly supported set of standards, which often means that coexistence of TSN and legacy solutions is partially feasible, but interconnection between them is only very seldom possible.

Migration Strategies

So far, the most successful strategy for migration towards the technically superior TSN based approaches is to replace single subsystems and to implement gateways to legacy systems. While this comes at an extra cost, it provides a well-controllable and scalable approach and allows for partial replacement. While this might still be an interim solution and will still require new standards and tech stacks to be developed such that they can take better advantage of the realtime guarantees, it is a viable way for getting devices in the field and moving on.

How the Linux kernel can help move things forward

Since the Linux kernel not only implements the support required for TSN and also provides driver support for most modern processors and switches with hardware support, but also is a very open and approachable implementation, it is the ideal basis for implementing such gateway systems and bringing deterministic networking endpoints and bridges to the field. While the basic infrastructure is available, often the rest of the tech stack involved (i.e. user space daemons, management services) is still incomplete. Reference implementations and working minimal examples can make these stacks more approachable to application engineers, who often focus more on the application requirements than on details of network stacks and realtime standards. Open application stacks like open62541 or OpenAvnu provide a good starting point for application development based on deterministic networking. [1] [2]

While open source stacks provide an insight and an learning opportunities into the interdependencies of different standards and substandards, evaluation of system designs is often difficult due to the numerous parameters that have to be taken into account. Adding tooling for testing an implementation can lower the effort for evaluation of the system design significantly and adds a tremendous additional value.

Careful userspace API design and documentation will ease the implementation and improve the use of proper and stable interfaces. While some of the requirements will need to add DetNet (Deterministic Networking, a set of RFCs authored by the IETF DetNet Working Group with focus on deterministic datapaths that operate over layer 2 and layer 3 networks) on top of TSN, this approach will lay the groundwork for moving forward. [3]

Technical requirements in the field will require intermediate solutions, since standardization is a slowly moving process, but nevertheless moving towards the standardized solution should be encouraged as they become available.

References

1. open62541 github page, accessed September 6, 2022, https://github.com/open62541/open62541

2. OpenAvnu github page, accessed September 6, 2022, https://github.com/Avnu/OpenAvnu

3. IETF DetNet working group landing Page, accessed September 6, 2022, https://datatracker.ietf.org/wg/detnet/

Author Biography

Before joining the Pengutronix kernel team in 2022, Johannes worked as a software engineer for a manufacturer of professional audio equipment, where he co-authored several networking standards, and as a systems engineer at a manufacturer of industrial and mobile PLC systems, where he migrated several proprietary legacy networking systems to open source realtime solutions.