LEARNING MADE EASY

Belden/Hirschmann
Special Edition

Time-Sensitive Networking

for dummies®
A Wiley Brand

Understand the *why* and *how* of TSN

Build and secure TSN networks

Use non-TSN Ethernet devices with TSN

Brought to you by:

BELDEN
SENDING ALL THE RIGHT SIGNALS

HIRSCHMANN
A BELDEN BRAND

Oliver Kleineberg
Axel Schneider

About Belden

Belden Inc. provides holistic communication solutions for mission-critical networks. These solutions include everything that is needed to transport messages securely between participants in a network. Belden solutions include connectors, cables, and active communication equipment, as well as communication software solutions such as routers, switches, firewalls, and network administration and security software.

About Hirschmann

Hirschmann Automation and Control GmbH is part of Belden's Industrial Solutions platform. Hirschmann designs and develops industrial Ethernet routers, switches, and firewalls that are used in mission-critical Industrial Internet of Things (IIoT) communication networks. Researchers and engineers from Belden's Hirschmann division were instrumental in contributing to the projects in the IEEE 802 working groups that are now known as "TSN — Time Sensitive Networking."

Time-Sensitive Networking

Belden/Hirschmann Special Edition

by Oliver Kleineberg and Axel Schneider

for dummies®
A Wiley Brand

Time-Sensitive Networking For Dummies®, Belden/Hirschmann Special Edition

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

ISBN 978-1-119-52791-6 (pbk), ISBN 978-1-119-52799-2 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Publisher's Acknowledgments

We're proud of this book and of the people who worked on it. Some of the people who helped bring this book to market include the following:

Development Editor: Joe Kraynak
Project Editor: Martin V. Minner
Editorial Manager: Rev Mengle
Executive Editor: Steve Hayes
Business Development Representative: Ashley Barth
Production Editor: Tamilmani Varadharaj

Table of Contents

Introduction

In highly automated systems, real-time communication is essential and sometimes vital. Imagine a self-driving car hesitating to brake for a pedestrian in its path or robots on an assembly line receiving delayed instructions from the computer that's synchronizing their movements.

Several real-time communication technologies, including EtherCAT, PROFINET IRT, and Sercos III, are used to ensure timely communications, but they have compatibility issues and offer limited, if any, support for future enhancements such as increased bandwidth.

Time-sensitive networking (TSN) overcomes these limitations to provide the following three essentials:

- Dependable real-time communication
- High bandwidth to accommodate the vast amount of sensor and background data that flows across automation networks
- Backward compatibility to Ethernet devices

About This Book

Time-Sensitive Networking For Dummies, Belden/Hirschmann Special Edition, introduces you to time-sensitive networking and explains the standards that make the magic happen. You also gain insight into the requirements of TSN network engineering and the benefits of using non-TSN Ethernet devices on a TSN network.

Although this book provides only a high-level overview, we hope it serves as a primer that sparks your interest and provides the general understanding required to start asking intelligent questions and digging deeper into this fascinating topic.

Foolish Assumptions

When writing this book, we made some foolish assumptions about you, one being that you'd be interested enough in TSN to

pick up this book. Another assumption is that you have a basic understanding of networking and specifically about how switched Ethernet networks work. If you know what an Ethernet frame looks like and have a very basic understanding of how Ethernet switches work, you're good to go.

If you aren't good to go, we recommend that you check out the Ethernet chapters in *Computer Networks* by Andrew S. Tanenbaum (Prentice Hall, 2010 or an earlier edition). If you're looking for more about Ethernet switching in particular, check out *The All New Switch Book* by Rich Seifert and Jim Edwards (John Wiley & Sons, Inc., 2008).

Icons Used in This Book

The margins of this book contain several helpful icons to draw your attention to special content:

TIP

We flag anything that can save you time and effort with this icon, so you don't miss out on the really good stuff.

TECHNICAL STUFF

To take a deeper dive into the technical details of a topic, look for this icon. This information isn't essential, but it may be interesting to those who are more technically inclined.

REMEMBER

This content is well worth remembering. No need to tattoo it on your forearm or write it on a yellow sticky note for your control console — just keep it in mind.

WARNING

Whoa! Before you take another step, check out the warning and proceed with caution. Heeding these warnings can help you avoid mistakes.

Beyond the Book

This is a short book — just 48 pages. If you reach the end and find yourself hungry for more technical details, visit `www.belden.com` and search for *TSN*.

IN THIS CHAPTER

- Transforming the automation pyramid into a pillar
- Recognizing the many advantages of TSN
- Seeing TSN in action in industrial and automotive networks

Chapter 1
Changing the Face of Automation Networks

The incredible progress in machine intelligence and robotics in recent years has driven progress in automation, especially industrial automation and automotive control. This chapter explores how automation networks are changing, describes the benefits that TSN offers these networks, and looks at TSN's role in industrial and automotive networks.

Tracing the Transition from Industrie 3.0 to 4.0

Industrial automation is currently in transition from what the German automation market refers to as Industrie 3.0 to Industrie 4.0. Industrie 4.0, or the *smart factory*, as it is sometimes called, is one part of the Industrial Internet of Things (IIoT). (*IIoT* is the combination of smart devices, advanced analytics, and machine learning to make production facilities more intelligent and dynamic.) This transition is commonly depicted as a shift from the automation pyramid to the automation pillar, as shown in Figure 1-1.

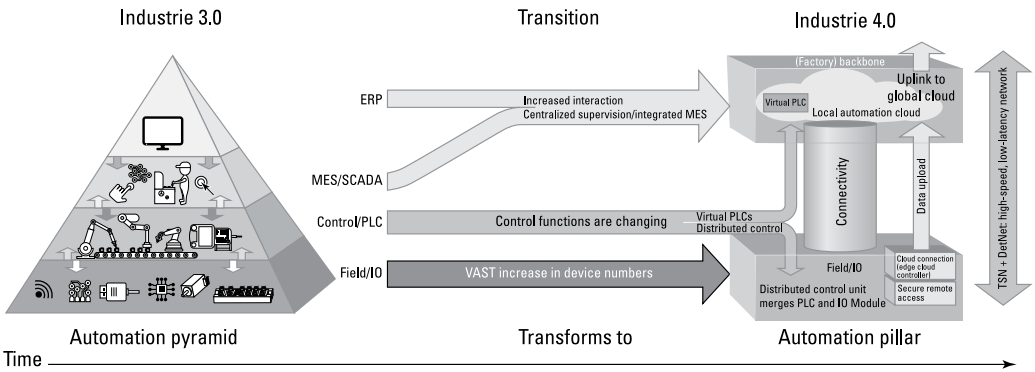


FIGURE 1-1: Moving from the automation pyramid to the automation pillar.

The pyramid model, which industries have followed for several decades, strictly separates functional layers from the factory shop floor (the field level) up to the management systems at the top. Real-time data communication is usually at the field level, where sensors and actuators are located, and between the field and controller levels.

The automation pillar in an IIoT (Industrie 4.0) production environment still has a field level on the factory shop floor, but the total number of sensors on the field level is drastically higher to allow a much tighter analysis and control of the manufacturing functions.

In the automation pillar, the controller level disappears. Some of the control functions move to the field level as distributed control units, which are used for extremely fast and reliable reactions, such as for safety functions. Other control units move into the management level (now called the *factory backbone*) as centralized control units (shown as "Virtual PLC") in Figure 1-1.

Virtual control functions or *virtual programmable logic controllers (PLCs)*, hosted in the local automation cloud, interact directly with the production process through the connectivity layer (described in the next paragraph). Virtualization of the PLCs provides maximum flexibility; for example, they can be added and removed, and computing power can be allocated to wherever it's needed most. Applications in the factory backbone don't even need to be located physically close to the field-level applications. They can be located anywhere: for example, in the IT department or even in a data center far from the factory, depending on the maximum end-to-end latency (delay) the applications allow. This flexibility in the control processes translates into flexibility in the production process.

Between the field level and factory backbone is a connectivity level. Both the field level and the connectivity level require high-speed, low-latency network performance. In addition, the connectivity layer carries lower-priority background traffic in a way that must not slow down the time-critical traffic. This is where TSN comes in.

Now that you're familiar with the components of the automation pillar, here's how it generally works:

1. Enterprise resource planning (ERP), the manufacturing execution system (MES), supervisory control, and data acquisition feed into the factory backbone to provide input for the production process.
2. Virtual PLCs provide centralized supervision and control over the production processes that occur on the field level, as time-sensitive control data is sent toward the field level devices.
3. Field level devices react based on the control data received from the PLCs and afterward return their feedback toward the PLCs. At the same time, sensor-generated data on the field level that isn't time-critical is sent upward through the connectivity layer and is aggregated in the local automation cloud and subjected to big-data analysis.
4. The feedback from the field level devices is passed to the PLCs, which compute and communicate changes back through the connectivity layer, to the machinery on the field level, to close the control loop.

REMEMBER

This process is the implementation of the "sensor to the cloud" vision; automation networks start at the sensor that's directly connected to the primary manufacturing process and finish within a cloud infrastructure service at the factory backbone. As described in Step 3, messages on the same network are either mission-critical or less urgent. The mission-critical control loop is used to operate the manufacturing process, and the less urgent sensor data is used to analyze and optimize the process. For further optimization, sensor data can be sent from the local cloud to a global cloud infrastructure. With this arrangement, companies can aggregate and analyze manufacturing data from different sites.

Recognizing the Need for TSN in Industrie 4.0

Think of the connectivity layer in the automation pillar as the information superhighway between the factory backbone and the field layer. Traffic consists of mission-critical data and less

urgent data. The connectivity layer needs to get all traffic where it's going, but mission-critical data is urgent; it needs to reach its destination on time.

When building a network to carry both urgent and non-urgent traffic, you have four options:

- Use TSN, which enables urgent and less urgent data to share the network connection, while preventing less urgent traffic from hindering the flow of the more urgent traffic.
- Build separate networks for the different applications — a high-cost option.
- Massively oversize the network infrastructure, a widely used but extremely expensive approach called *bandwidth overprovisioning*.
- Live with the traffic delays in mission-critical data, which usually isn't a viable option.

Of these four options, the clear choice is the first option — use TSN.

In the following sections, we dig deeper into the issues latency and jitter and provide additional insight into why TSN is the best approach for the connectivity layer.

Grasping the difference between latency and jitter

Before you wade into the deep end with TSN, understand the difference between latency and jitter:

- **Latency:** The time it takes data to travel from point A to point B
- **Jitter:** Any variation in latency

For example, suppose you *have to* make it to an appointment at a specific time. You go online, map the route, and find out that you need 30 minutes to drive there. You type the destination into your smartphone's GPS app, and it gives the same estimate: 30 minutes. In fact, you've driven there before, and it has always taken you 30 minutes. That's latency.

Thirty minutes before your appointment, you hop in your car and start driving to your appointment. Ten minutes later, you're stuck in traffic. You have no idea how long the delay will be. That's jitter. In any automation network (industrial or automotive), jitter occurs when low-priority traffic occupies a large amount of the available bandwidth, restricting the flow of high-priority data.

Compared to consistent latency, jitter is the bigger problem in the connectivity layer. To a certain degree, consistent latency can be compensated for in the control algorithms. Just as you can leave earlier to make an appointment, you can adjust when data is sent to ensure that it reaches its destination when it needs to be there. As long as the data is where it needs to be at that time, you have no problem.

Jitter, on the other hand, must be minimized to ensure consistent on-time delivery.

REMEMBER

A common misconception about real-time computing is that it always must be as fast as possible, happening right now like a live broadcast — or, in networking terms, as low-latency as possible. Real-time computing actually means that a system is subject to a *real-time constraint* or a promised deadline. Low latency doesn't matter as much as the data reaching its destination on time. Being on time doesn't necessarily mean being as fast as possible.

Directing all traffic across a single network with TSN

With TSN, all data travels the same information superhighway, with urgent data given high priority. It's sort of like an emergency vehicle lane or a bus lane on a highway except that TSN doesn't reserve traffic lanes because doing so would create inefficiencies. TSN directs traffic to maximize use of the available bandwidth.

Returning to the automation pillar, you should now have a clearer idea of how TSN enables the transition from Industrie 3.0 to 4.0. Urgent and non-urgent (background) traffic share the network infrastructure. For the control loops, high-priority traffic can travel unimpeded through the connectivity layer. At the same time, large amounts of sensor data can be moved up the pillar, through the connectivity layer into cloud infrastructure, where big-data analysis algorithms can be applied.

REMEMBER

TSN is the best option not only because it works best but also because it has a lower total cost of ownership (TCO). You may have the added cost of replacing existing switches with TSN switches, but that typically costs less than duplicating networks and maintaining the additional networks.

Putting TSN to Work in Automation Networks

In automation networks, TSN enables the convergence of numerous small, disconnected networks into one unified network structure. This new network can accommodate the requirements for real-time communication on a larger scale, while providing more transmission bandwidth for background data. You can see the benefits of network convergence at work in different markets. This section presents a few use cases.

Factory automation

In factory automation, network convergence enables distributed real-time control; large machinery and numerous robots can interact with each other more precisely and flexibly than previously possible. Organizations can enable applications, such as predictive maintenance, that require the analysis of substantial amounts of sensor data. A converged network from cloud to sensor also allows secure remote access from the Internet to the production machinery to perform maintenance and other tasks remotely.

Energy automation

In energy automation, for example, in electrical substations, TSN can be used to allow for time-critical data, such as sampled values from voltage and current, to travel through the network to the electrical protection equipment. TSN can also be used to improve the performance of important event notifications, Generic Object-Oriented Substation Events (GOOSE), when the GOOSE protocol uses the same network infrastructure used, for example, for sensor data or network surveillance.

Transportation applications

In transportation — for example, on train networks — convenience applications such as passenger entertainment can share a network with other applications such as passenger information or control functions that are not safety relevant. In turn, safety functions can be combined with other control functions on dedicated control networks.

Taking TSN for a Spin in Automotive In-Vehicle Networks

Modern automobiles are equipped with a massively increasing number of on-board electronic control units to cover novel functionality, such as autonomous driving. Increased functionality has driven even greater demand for physical connectivity and communication bandwidth.

Automotive bus systems, including FlexRay, Controller Area Network (CAN), and Media Oriented Systems Transport (MOST), struggle to keep up with this demand. They all rely on dedicated physical cabling, which adds complexity and weight. The additional weight reduces fuel economy and performance.

The main use case for in-vehicle automotive car and truck networks is around the convergence and replacement of the many different in-vehicle communication busses with deterministic Ethernet. The reason behind this is the weight and complexity of the cable harness in a modern automobile.

TSN to the rescue! TSN enables the convergence and replacement of many different in-vehicle communication busses to form a unified connectivity layer.

Ethernet has not yet replaced many of these bus systems, mainly because Ethernet cabling did not meet the electromagnetic immunity requirements of the in-vehicle car use case, but this has changed: The IEEE 802.3 working group has specified several physical layer chips (PHYs) that can meet these requirements. The introduction of PHYs has opened up the in-vehicle car market to Ethernet and TSN.

TSN, with its capability to merge traffic of different priorities, feedback-free, to a single cable, is ideally suited as an in-vehicle car backbone communication technology.

Car makers can use TSN in different ways depending on their architecture. For some manufacturers, TSN connects only the different application domains inside the vehicle, such as drive train, body control, and passenger entertainment, attaching to each domain via a gateway. Inside each domain, different technologies such as MOST or FlexRay are used.

In other cases, TSN is also used within the individual application domains and replaces the in-vehicle car bus network altogether. Some manufacturers are already stating that eventually, TSN will replace all in-vehicle bus systems, with the exception of diagnostic CAN.

IN THIS CHAPTER

- Recognizing the importance of time synchronization
- Making sure the trains are on time with TSN scheduling
- Preempting common causes of traffic delays

Chapter 2
Understanding What Makes TSN Tick

In Chapter 1, we explain the *what* and *why* of time-sensitive networking (TSN) — what it is and why it's used. In this chapter, we focus on the *how* of TSN — how it works and what makes it tick.

How TSN works is prescribed in a set of Institute of Electrical Electronics Engineers (IEEE) 802 Ethernet sub-standards, each of which defines the specifications for a distinct TSN function. By understanding the standards, you understand the TSN functions and get a clear picture of how TSN works.

In this chapter, we start with a bird's-eye view of the standards and then follow up with a deeper dive into each standard/function.

REMEMBER

IEEE 802 has always practiced the divide-and-conquer strategy: They break down every new technology into individual functions and then set specifications for each function to make the specs more manageable. They've taken the same approach to develop the TSN standards.

Piecing Together the TSN Puzzle

Imagine three pieces of a puzzle, each piece representing a related group of standards (see Table 2-1 and Figure 2-1):

- **Time synchronization** ensures that everything in the network agrees on what time it is. See the section "Synchronizing Time across the Network" for details.
- **Real-time scheduling** ensures the timely delivery of mission-critical data. See the section "Directing Traffic with Real-Time Scheduling" for details about real-time scheduling.
- **Communication stream reservation and configuration** controls and polices communication pathways and provides other support to ensure reliable communications. See the section "Maintaining a Reliable Communication Stream" for details.

With all the puzzle pieces assembled, TSN is born, ensuring reliable high-performance communication.

TABLE 2-1 TSN Functions and Standards

Function	Standard or Project ("P" denotes a project that is still running)	Status
Time synchronization	P802.1AS-Rev: Time synchronization for time-sensitive applications — Revision	In specification
	1588-2008 (PTP): Precision clock synchronization protocol	Finished
Real-time scheduling	802.1Qbv-2015: Enhancements for scheduled traffic	Finished
	802.1Qch-2017: Cyclic queueing and forwarding	Finished
	P802.1Qcr: Asynchronous traffic shaper	In specification
Real-time scheduling — support	802.1Qbu-2016: Frame preemption	Finished
	802.3br-2016: Interspersing express traffic	Finished

Function	Standard or Project ("P" denotes a project that is still running)	Status
Communication stream reservation and configuration	P802.1Qcc: Stream reservation protocol enhancements and performance improvements	In specification
	802.1Qci-2017: Per-stream filtering and policing	Finished
Communication stream reservation and configuration — fault tolerance	802.1Qca-2015: Path control and reservation	Finished
	802.1CB-2017: Frame replication and elimination for reliability	Finished

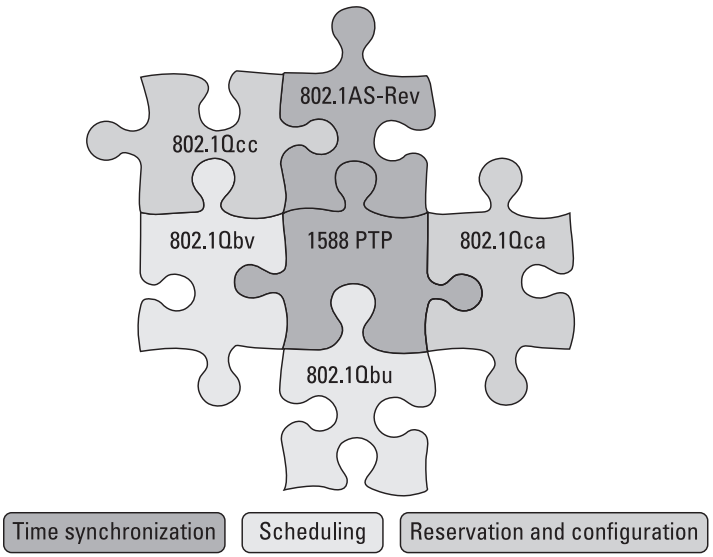


FIGURE 2-1: The TSN system.

REMEMBER

The TSN standards are a work in progress — all the key standards are finished, others are in progress (in specification), and some are on the to-do list (in preparation to start the specification). Table 2-1 presents the status of each standard during the writing of this book, but because new TSN standards are likely to be added to accommodate new use cases, the list of standards

likely will never be complete. However, this is a normal situation for the Ethernet technology: Ethernet and TSN comprise an ever-evolving and powerful ecosystem. The constant new development and Ethernet's commitment to remaining backward compatible are the main reasons Ethernet and TSN have been and continue to be so successful.

Now that you have a conceptual image of a TSN puzzle with all the pieces fitting nicely together, we're going scramble it a bit by pointing out that you can have TSN even if one or two pieces of the puzzle are missing. For example, you can use the Time-Aware Scheduler with or without frame preemption and vice versa. Implementing both significantly increases performance, but it isn't always necessary. Some devices operate just fine with only one of those functions.

REMEMBER

A device marketed as "TSN compatible" doesn't necessarily support all TSN standards. Carefully review the protocol implementation conformance statement that a device vendor is required to provide (for example, in a technical product brochure). If you're a TSN manufacturer, include in your protocol performance statement a detailed list of the TSN functions the device supports. The requirement to publish support for IEEE standards is nothing new; the same holds true for all Ethernet switches and IP routers.

TECHNICAL STUFF

IT'S COMPLICATED

Although you can usually approach TSN standards as though each is independent of the others, some standards or functions are interlinked. For example, the key functionality for TSN, the Time-Aware Scheduler, is described in Standard IEEE 802.1Qbv-2015, which is interlinked with the frame preemption functionality described in two different standards: IEEE 802.1Qbu-2016 for the interface toward the switch management and higher software layers, and IEEE 802.3br-2016 for the preemption function in the media access control (MAC) hardware.

Synchronizing Time across the Network

The term "time-sensitive" is a dead giveaway that time plays a major role in TSN. The importance of having all devices on a network agree to a set time makes sense. Just think about when you plan to meet a friend for lunch. If your clock shows 12:30 when your friend's clock is set to noon, assuming you're both "on time," you'll be a half hour early, or your friend will be a half hour late.

Likewise, with TSN, all devices and switches must be synchronized to ensure that everything arrives on time. For all devices in the network to coordinate their actions, such as scheduling Ethernet frame transmissions and opening and closing transmission gates, they must be operating on an agreed-upon time. Even the best scheduling schemes are ineffective if the devices aren't executing them in sync.

For time synchronization, TSN typically uses one of the following approaches, both of which require the use of clocks with accuracy in the nanosecond range:

- **Precision Time Protocol (PTP):** With this IEEE 1588-2008 standard, all compatible devices use the Best Master Clock Algorithm to identify the device with the most accurate clock, to which all other devices will be synchronized. This device then serves as the Grandmaster Clock.
- **An IEEE 1588 profile, such as IEEE 802.1AS-2011:** The PTP protocol is comprehensive and supports many synchronization and message transportation methods. Profiles can be used to narrow these options and to provide a crisp implementation for use cases that don't require all the options that IEEE 1588 offers. IEEE 802.1AS-2011 is such a profile. Because AS doesn't cover all requirements for industrial automation, the TSN standards include an update to this profile, P802.1AS-Rev.

Although time synchronization is essential, IEEE isn't very strict about which time sync protocol is used. But it's important that all devices are compatible and can synchronize to each other.

REMEMBER

Network time protocols, such as *the* Network Time Protocol (NTP), require that devices synchronize to a specific time of day: for example, the time provided by an atomic clock. However, TSN time protocols require only that devices choose the device in the network that's the most accurate timekeeper.

TIP

When purchasing TSN-enabled devices, consider the quality of their time synchronization — the tighter the synchronization, the better. Lower quality synchronization requires network designs that leave larger margins for error, which reduces TSN performance.

Directing Traffic with Real-Time Scheduling

TSN uses several scheduling mechanisms to carefully coordinate communications across the network. Strict adherence to the schedule ensures that the individual communication flows with various priorities don't interfere with one another. Interference would result in jitter, which would cause transmission delays in the Ethernet switch queues.

The IEEE 802.1Qbv-2015 Time-Aware Scheduler

The most important mechanism for coordinating communications across a network is the Time-Aware Scheduler, specified in the IEEE 802.1Qbv-2015 standard. The *Time-Aware Scheduler* helps to prevent bottlenecks in data transmission and minimizes the queueing effect in Ethernet switch transmission, which contributes to latency and jitter. To engage the Time-Aware Scheduler, the Ethernet switch must be able to identify frames with different priorities. This capability has been integrated into Ethernet since virtual local area networks (VLANs) were described in the IEEE 802.1Q standard in 1998.

The VLAN Ethernet frame format supports a three-bit field encoded into the frame header that denotes the Ethernet frame's priority. Using these three bits, Ethernet switches (even without TSN) can identify any of eight priority levels (0–7) ranging from low to high based on the urgency of the data being transmitted.

However, without the TSN Time-Aware Scheduler, Ethernet frames with low priority that are already in transmission could delay frames of the highest priority at every switch along the transmission path. Imagine all lanes of traffic backed up at an intersection. An emergency vehicle arrives, and even though it has the highest priority (the right of way), it can't move until the cars in front of it clear out of the way. Likewise, a backup of low-priority frames at an Ethernet switch could block the passage of higher priority frames.

To minimize latency and especially jitter, the transmission queues in the Ethernet switches ideally are empty when the high-priority traffic comes through — just as it's ideal for intersections to be clear when an emergency vehicle approaches.

To keep Ethernet switches clear for higher priority frames, the Time-Aware Scheduler takes the following approach (see Figure 2-2):

1. The Time-Aware Scheduler divides time into repeating segments of equal length, called *cycles* (Cycle n and Cycle n+1 in the figure).
2. Within each cycle are dedicated time slots for frames with different priority levels. Figure 2-2 shows two dedicated time slots in each cycle — Time Slot 1 and Time Slot 2.
3. The user can configure the cycle time and the time slots within the cycle according to the requirements of the applications that use the network to communicate.

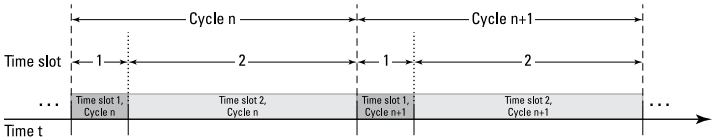


FIGURE 2-2: The Time-Aware Scheduler.

4. The user can also assign different priority levels to each time slot, as shown in Figure 2-3. Here, priority 3 frames have exclusive use of Time Slot 1, while priority frames 0–2 and 4–7 share Time Slot 2.

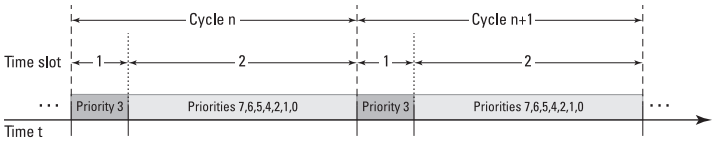


FIGURE 2-3: Priority levels are assigned to each time slot.

The Ethernet VLAN tag contains each frame's priority, which determines the time slot it uses. In this example (Figure 2-3), the Time-Aware Scheduler sends priority 3 frames to Time Slot 1, so they can bypass the rest of the traffic, which is assigned to Time Slot 2. If this were a highway, Time Slot 1 would be the emergency vehicle lane.

The reason for synchronized clocks in the switches and end devices now becomes more apparent. Because you don't have an extra emergency lane — just one Ethernet cable — you need the different priorities to access the cable at different times. This way, the time-critical traffic experiences a completely empty Ethernet cable (or lane) at the exact time it is sent. When that time arrives, the following happens: The switches open the transmission gates (time-aware gates) for the high-priority frames (emergency vehicles) all at the same time to ensure that the high priority traffic isn't interrupted along the way (see Figure 2-4). At the same time, the end devices need to know when the transmission gates are opening in the switches so they can time the transmission of high priority frames accordingly.

As shown in Figure 2-4, all Ethernet frames inside each switch are processed until they reach the transmission gates, where frames in each queue must wait until their respective transmission gate is opened, which in turn is determined by the gate control list, shown on the right in Figure 2-4. The gate control list is what the network user needs to configure for the switches to operate as intended and to open the correct gates at the correct point in time. In the figure, only the gate for the Priority 3 traffic queue is open. The gates for all other queues are closed. By temporarily closing the transmission gates for the other queues, Time-Aware Scheduler prevents that traffic from restricting the flow of priority 3 frames.

REMEMBER

Time-Aware Scheduler minimizes latency and jitter on Ethernet networks to what is physically possible. However, this performance comes at a price: Networks that use the Time-Aware Scheduler are no longer plug-and-play — they have to be configured.

Another reason you may not want to use Time-Aware Scheduler is that not all applications require the lowest latency and jitter. For situations in which the Time-Aware Scheduler doesn't seem to be the ideal option, you may want to consider other IEEE scheduling options presented in the next few sections. (You may also encounter situations in which using the other scheduling options in tandem with Time-Aware Scheduler makes sense.)

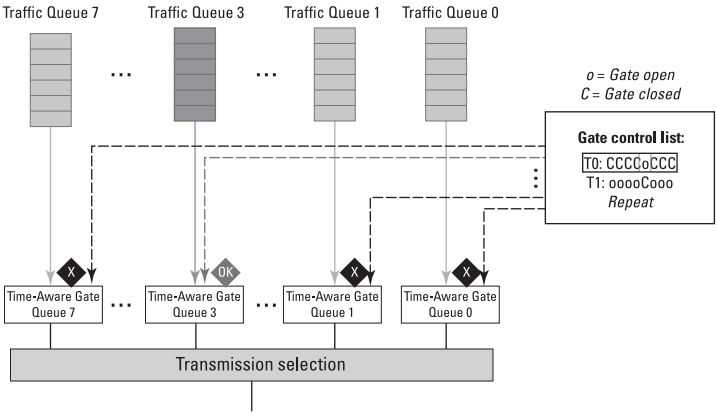


FIGURE 2-4: Allowing the flow of priority 3 frames.

The IEEE 802.1Qch-2017 cyclic queueing and forwarding

Some network applications can sustain a certain level of latency and jitter, which is usually the case if an application isn't physically fast enough to benefit from low latency on the network. You often encounter this in applications in the process or energy automation industry, where the sheer size and weight of the machinery prohibit low cycle times.

For these use cases, the IEEE 802.1 working group has specified the IEEE 802.1Qch-2017 cyclic queueing and forwarding standard. This approach provides less accuracy in latency and jitter than the Time-Aware Scheduler, but it requires much less configuration effort.

The Time-Aware Scheduler and cyclic queueing and forwarding are based on a common understanding of time and the definition of communication cycles. However, cyclic queueing and forwarding guarantees the transmission of frames for only one network hop per cycle. A *network hop* is a move from one Ethernet switch to the next (see Figure 2-5). In terms of the traffic analogy, cyclical

queueing and forwarding are like a guarantee that a vehicle would be stopped at each traffic light but have to wait there for only a set amount of time before getting a green light. So if the route had ten traffic lights with a 10ms wait at each light, the worst-case latency would be 100ms.

The benefit is that you don't need to configure time slots within the cycle. The disadvantage is that you get timing and delivery guarantees with the variance of a full cycle. If you know the number of network hops along the communication path and the length of the cycle, the computation of the arrival time window is relatively straightforward, but you're getting a window, not a specific time: Just multiply the number of hops or switches along your path with the cycle time. That's your worst-case delay. If your application can sustain operations with having a delay time window (and many applications can), then the IEEE 802.1Qch is a great way to use TSN without the need for complicated configuration and calculation.

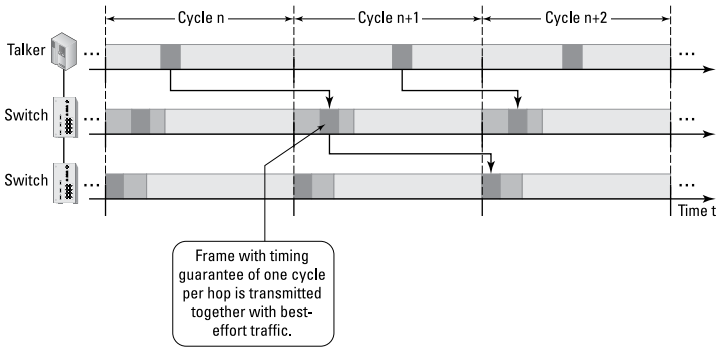


FIGURE 2-5: Cyclic queuing and forwarding.

Additional traffic schedulers and shapers

In addition to the Time-Aware Scheduler and cyclic queueing and forwarding, IEEE standards specify a number of traffic schedulers and shapers for different application scenarios. (A *traffic shaper* is a tool that directly influences data flows across a network. For example, Internet service providers often use traffic shapers to throttle network speeds for users who exceed their bandwidth limit.)

The IEEE 802.1Qav-2009 Credit-Based Shaper, for example, was designed as an Audio Video Bridging (AVB) standard before the work on TSN even started. The Credit-Based Shaper is designed to evenly spread out frames for audio and video transmission over time to avoid visible or audible hiccups at the receiving device. To do this, the shaper adds credits to audio or video frames as they wait in a queue and are delayed. The frames then "use" this credit to temporarily receive higher priority over the rest of the traffic to correct the delay and to catch up.


REMEMBER

A number of shapers and schedulers are in the works as we're writing this book, so the number of network traffic management tools will very likely increase in the years to come. More tools will enable TSN Ethernet to fulfill even more roles in new applications.

Combining TSN methods on a network

With so many methods to schedule and shape traffic, the question arises whether these methods can be used and combined on the same network. In most cases, the answer is yes, as long as the network devices support the use of more than one method at the same time. However, mixing and matching schedulers and shapers increases the device and network configuration effort considerably and typically requires the addition of automated network and device configuration systems. At a certain point, additional complexity becomes too much for human operators to handle on their own, requiring the addition of a suitable computerized control support mechanism.

Maintaining a Reliable Communication Stream

TSN is often referred to as *deterministic Ethernet,* which means the network is predictable. TSN synchronization and scheduling go a long way toward achieving deterministic Ethernet, but IEEE has additional standards to make scheduling more feasible and to maintain a reliable communication stream. These standards aren't necessarily related to synchronization and scheduling, but they meet different requirements for target applications, such as fault tolerance, and can protect against unwanted network traffic.

Keeping big frames from spilling over with guard bands

Transmission queues are notorious for being unpredictable. In some instances, a frame may be transmitted late in a cycle, and it may be too large to fit within the allotted time slot, as shown in the top half of Figure 2-6. In a situation like this, this low-priority frame spills over into the time slot reserved for a higher-priority item. To prevent that from happening, the Time-Aware Scheduler employs a guard band that blocks the large frame from entering the time slot, so the frame gets bumped to the beginning of the next cycle.

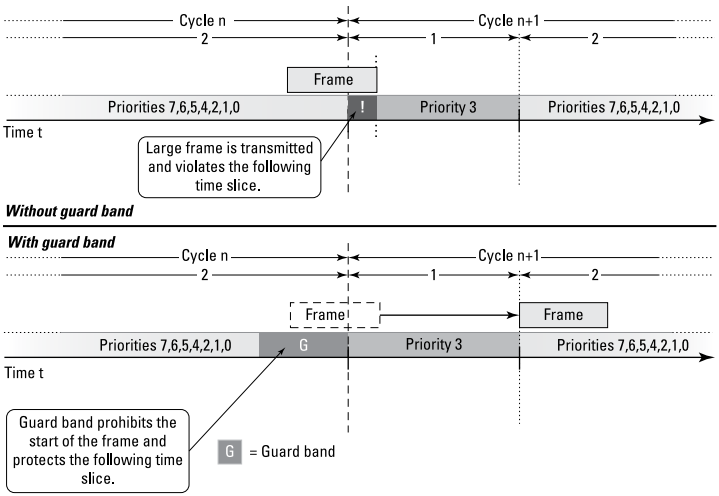


FIGURE 2-6: The guard band.

Guard bands are great (and essential) to protect latency guarantees for mission-critical frames. However, the guard band has a significant drawback: It blocks transmission of frames in the lower-priority queues, resulting in wasted bandwidth. This is comparable to closing down a road completely to ensure that a police car can travel without interruption — the police car gets through, but the time when the police car is not on the road cannot be used by other cars and goes to waste.

Minimizing bandwidth waste with frame preemption

Frame preemption involves splitting a frame into smaller packets, sometimes referred to as "framelets," which can be as small as 64 bytes. With frame preemption, you can reap the benefits of a guard band while minimizing the bandwidth loss by significantly reducing the size of the guard band, as shown in Figure 2-7.

With frame preemption, instead of preventing a switch from sending a frame, TSN allows most of the frame to be sent, blocking only the portion of it that won't fit in the time slot in the current cycle. Only that part of the frame is then bumped to the next cycle.

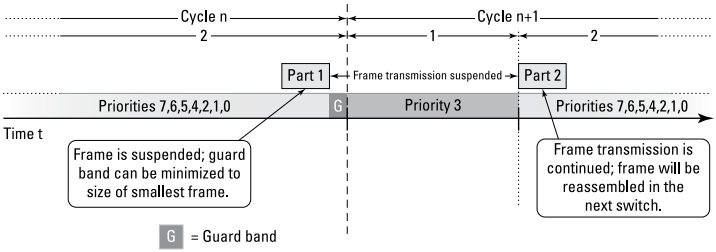


FIGURE 2-7: Frame preemption.

REMEMBER

Frame preemption requires at least two devices — one device that splits up the frame and a second device that reassembles it, so frame preemption can operate only on a physical link where both adjacent devices support it.

Using redundancy for fault tolerance

Networks take their communications seriously. Failure is not an option. Whether you're exchanging email, downloading files, or accessing web pages across a network, the transmissions must be reliable. This reliability is even more important for mission-critical communications on time-sensitive networks.

To prevent transmission failures, networks use various redundancy protocols, which all operate on the basis of the following principle: If one network cable or device fails, the redundancy protocol enables the network to automatically recover from the fault. Redundancy protocols work by using different secondary

transmission paths through the network to maintain the communication while the primary path is being repaired.

Redundancy protocols are generally grouped into the following two categories:

- **Seamless redundancy:** All network paths are used in parallel, so no disruption occurs if one path fails.
- **Non-seamless (failover) redundancy:** The protocol recovers the fault by switching from the primary path to the secondary path. Although this switch occurs quickly, it may result in a very brief disruption.

In this section, we explain these two types of redundancy protocols in the context of TSN.

TSN and seamless redundancy

Seamless redundancy is necessary to maintain continuous operation in the event of a fault: for example, in a vehicle that uses the network for critical functions such as autonomous driving, or in industrial automation with time-critical machine or functional safety requirements.

In such cases, TSN can use seamless redundancy mechanisms that were specified for use in special automation application fields, such as High-Availability Seamless Redundancy (HSR) or Parallel Redundancy Protocol (PRP). These are specified in IEC 62439-3. In addition, TSN can use IEEE 802.1CB-2017, which is part of the TSN standards and operates on a similar principle.

IEEE 802.1CB-2017, HSR, and PRP rely on the simultaneous transmission of frames over disjunctive transmission paths, as shown in Figure 2-8. Whenever a mission-critical frame is sent, it's duplicated and routed over at least two redundant paths. When the duplicate frames arrive at their destination, one is accepted and the other is dropped.

Seamless redundancy provides the highest level of fault tolerance possible in Ethernet networks. If one communication path fails, the other is able to sustain the application without interruption. HSR and PRP provide up to two redundant paths, while 802.1CB-2017 can provide more — as many as the physical network topology can support.

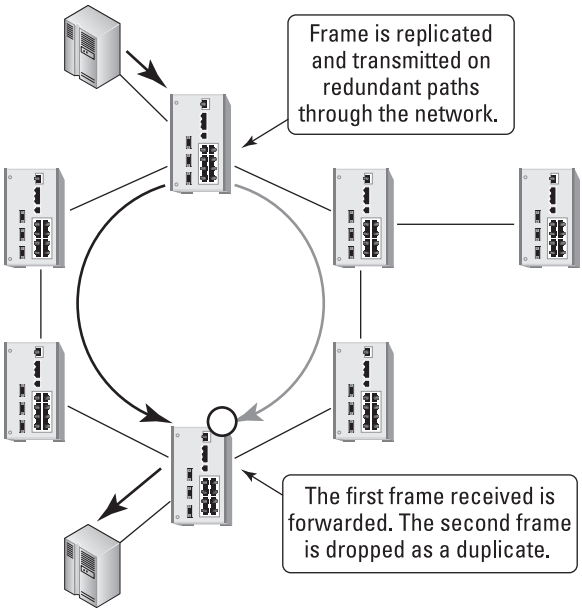


FIGURE 2-8: Seamless redundancy in action.

REMEMBER

With all paths in operation simultaneously, network engineering must ensure that the maximum end-to-end latency on the longest or slowest path stays below the maximum tolerated latency of the application. In other words, each path must be able to guarantee the required latency. If a path can't guarantee the required latency, it's of no use.

TSN and non-seamless redundancy

Not all TSN applications require seamless redundancy. Non-seamless redundancy mechanisms, such as the IEEE 802.1D-2004 Rapid Spanning Tree Protocol (RSTP) or the IEC 62439-2 Media Redundancy Protocol (MRP) have been used in industrial automation for many years. The only question is whether the application can function satisfactorily with the amount of latency and jitter that's created when the network has to switch communication paths in case of a fault.

In a TSN context, the network recovery time can be treated as additional latency and jitter, because the recovery time of the redundancy protocol delays the transmission of frames. To the end

device, this is visible as a variation in latency, as long as a possible complete loss of a frame is compensated by a retransmission.

If the recovery time of the redundancy protocol is lower than the maximum tolerated latency, the protocol can be used in a TSN context to protect the running application. With MRP, recovery times can be as low as 10 milliseconds.

Even when the redundancy protocol recovery time is higher than the tolerated maximum latency of the application, the redundancy protocol can still be of use. It can no longer support recovery where the application is not affected, because the latency cannot be guaranteed. However, the redundancy protocol can recover the basic network function after the fault has occurred. The main application will still be affected, of course, but restoring network function enables network management and monitoring tools to access all parts of the network again and immediately start working on repairs.

Reserving bandwidth using stream registration

Stream registration involves reserving bandwidth for any two devices on the network, which gives the communications between those two devices a higher priority. During an exchange between the two devices, one device is the talker, which publishes the stream, and the other is the listener, which receives the stream.

The switches in the network register the publication and reception of communication streams along the transmission paths. Part of the registration is the reservation of the required bandwidth for each stream to ensure that it can be transported safely. Registration ensures that the available network bandwidth is not overbooked. Switches can turn down the registration of new streams if the required bandwidth of a new stream would exceed the maximum available bandwidth on a device port.

Note that TSN can work with or without registered streams. Stream Registration dates back to the first set of AVB standards, where it was defined as the Stream Registration Protocol (SRP) in IEEE 802.1Qat-2010. IEEE P802.1Qcc extends the original SRP to include mechanisms for TSN streams as well.

IN THIS CHAPTER

- Knowing your options: Centralized, decentralized, and hybrid TSN
- Using TSN and non-TSN devices together on an Ethernet network
- Tightening security on a time-sensitive network

Chapter 3
Building and Securing Time-Sensitive Networks

Ethernet was originally developed as a plug-and-play high-speed cable network technology. Assuming all networking hardware and software was Ethernet-compatible, you connected all the devices you wanted to be on your local area network (LAN) to a central router using Ethernet cables, ran the network setup routine on each computer, and your Ethernet LAN was up and running. IEEE 802.11 made networking even easier with the introduction of wireless LAN with easy integration into Ethernet.

TSN Ethernet is no longer plug-and-play. TSN switches and end devices must be enabled and configured to closely synchronize communications between devices, especially for mission-critical data exchanges. Configuration includes enabling TSN mechanisms, such as time synchronization and the Time-Aware Scheduler (see Chapter 2) and then adjusting cycle times, time slots, frame priorities, and other parameters according to the application's requirements.

In this chapter, we tackle the topic of automated versus manual configuration, present the three different ways you can choose to structure a time-sensitive network, and then address two more specific issues — combining TSN and non-TSN devices and overcoming TSN-specific security issues.

REMEMBER

When you're dealing with nearly any large enterprise Ethernet network these days, the difference between plug-and-play Ethernet and non-plug-and-play TSN is barely an issue. With the introduction of VLANs and managed switches, mission-critical Ethernet installations have since drifted far from the plug-and-play approach. Most of today's enterprise and automation networks require engineering and configuration. TSN increases the configuration that's necessary to get a network running, but also brings lots of benefits to the table in return.

Choosing a Configuration Approach: Auto or Manual

Although manual setup and configuration of a small time-sensitive network is possible, configuration typically is mostly automated. The end devices announce their requirements, and all TSN switches and other devices between those end devices are configured accordingly through the following three-step process:

1. All TSN supported mechanisms in the network that the application requires are identified and enabled.
2. The sending device (talker) announces requirements about the data it's ready to send, such as required bandwidth and acceptable latency.
3. The receiving device (listener) calls for and receives the data in accordance with the talker's requirements.

REMEMBER

However, manual configuration is also an option. Just as human operators have always done with engineered networks, they can configure a small TSN: for example, for small automation networks, such as single automation cells. Manual configuration may also be practical for networks that are mostly static (those that can be engineered and configured once and will never change in their operational lifespan), such as critical functions in the design of an in-vehicle car network.

However, TSN was designed to cover much larger and more dynamic networks as well, such as IIoT networks. In networks such as these, TSN must account for movement of network participants, such as autonomous guided vehicles, workers, and robots. For these more complex applications, having a person

handle the configuration may not be practical, in which case a mechanism must be in place to automate the configuration process as devices join and leave the network and communication streams are established and torn down.

The configuration mechanism may provide central control, allow for decentralized control, or take the middle ground with a hybrid approach, depending on the TSN configuration model, as discussed in the following section.

Choosing a TSN Configuration Model

The first step in building a time-sensitive network is to choose the TSN configuration model you want to follow. You have three choices:

- Centralized
- Decentralized
- Hybrid

Centralized TSN

In the centralized TSN model (see Figure 3-1), the end devices announce their requirements to a central user configuration (CUC) instance, which can be done through an automation protocol such as Open Platform Communication Unified Architecture (OPC UA).

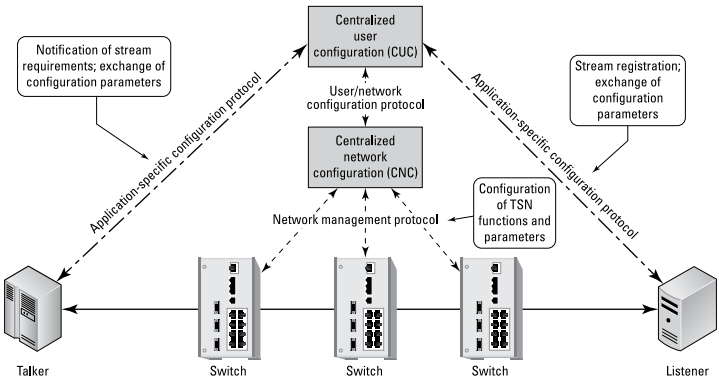


FIGURE 3-1: The centralized TSN model.

The CUC aggregates all information regarding the end devices and their communication requirements. A CUC can be envisioned, for example, as an automation support application that is running on a tablet, PC, or terminal. A second entity, called the centralized network configuration (CNC), assembles all information from the network infrastructure, such as the number of switches present, the physical and logical topology, the TSN mechanisms, and the available bandwidth. The CNC can be envisioned as a network support server or program.

Using the details collected about the communication requirements and the available network resources, the CUC computes a network configuration that accommodates all end devices. Assuming the CUC computes a feasible configuration, the CUC pushes the configuration to the network infrastructure through the CNC. If the CUC can't compute a network configuration that meets all of the application's TSN requirements, it announces the failure to the user, who can take corrective action.

The advantage of centralized TSN is that it supports large network infrastructures with dynamic device movement. Through the CUC and the visibility of all devices and all communication requirements, a user or device can tell immediately whether a network configuration is feasible or not, which is not the case with the decentralized model (discussed in the next section).

The disadvantage of the centralized model is that it requires dedicated infrastructure that houses the CNC and CUC. However, network management systems, such as Hirschmann Industrial HiVision, are commonplace in large industrial network infrastructure installations, so this disadvantage rarely restricts the use of centralized TSN in industrial automation applications.

Whether centralized TSN is feasible or even necessary for automotive car networks depends on the amount of computational power in an onboard control unit that a car manufacturer is willing to assign to this task and whether the car network design even requires central control. In such applications, decentralized TSN (discussed next) may be the better choice.

Decentralized TSN

In the decentralized TSN model (see Figure 3-2), the end devices transmit their requirements for sending and receiving TSN communication streams to the first TSN switch each is connected to.

This switch receives the information, evaluates it, and distributes it across the network through any other switches. Ultimately, all end devices are informed about the data streams offered in the network and the requests to receive certain streams.

When an offer and a request in the network match, the end devices and switches automatically establish the connection and process frames according to the parameters agreed upon by the two end devices.

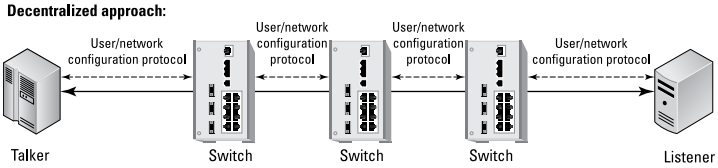


FIGURE 3-2: The decentralized TSN model.

This decentralized approach has the distinct advantage that the TSN switches require very little, if any, manual configuration. The decentralized model is more plug-and-play. In addition, the mechanisms for registration and deregistration are well proven, because they're based on the AVB Stream Registration Protocol. This protocol enables dynamic changes in the network, such as adding, removing, and moving devices, while the network is operational. Registration and deregistration occur automatically.

The drawback to decentralized TSN is that on larger networks, conflicts between the requirements of different devices and streams become more and more difficult to resolve. A decentralized network has no CUC or CNC to coordinate activities among the network devices.

Hybrid TSN


REMEMBER

When you're building a time-sensitive network, you don't have to choose one model to the exclusion of another. You can combine the centralized and decentralized TSN models. For example, you may find that centralized TSN is best for one part of the network, while decentralized TSN works better for other parts of the network.

While running both the centralized and decentralized model is possible, there is a third approach: By combining parts of the centralized and decentralized models, you essentially create a third option, a hybrid model, as shown in Figure 3-3.

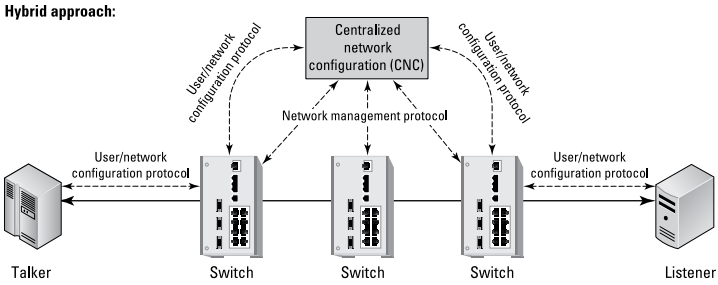


FIGURE 3-3: The hybrid TSN model.

The advantage of the hybrid model is that the end devices need to support only one configuration protocol (this is the same in the decentralized and hybrid models), although the user has the benefit of a centralized network view (CNC), just as with the centralized model.

Mixing TSN with Non-TSN Ethernet

TSN is the next step in the evolution of Ethernet, not a separate technology that grew up alongside it. As such, TSN is completely backward compatible with standard IEEE 802.1Q switched Ethernet. TSN-capable devices can be connected to non-TSN Ethernet devices and vice versa without the need for protocol translators or gateways. However, non-TSN Ethernet devices can't do TSN. You can still use non-TSN Ethernet devices on a network, but you can't use them to achieve *deterministic TSN communication* (guaranteed on-time delivery of data frames).

This section covers some of the benefits of using TSN on a network that isn't completely TSN compliant and explains what's involved in integrating vendor-specific industrial Ethernet solutions.

Recognizing TSN's benefit on a partially compliant network

With every new technology that involves changes in network infrastructure and end devices, the same question arises: What value does a new function have if certain devices on the network don't support it? This question is a good one because new protocols often have no value unless the network and all the devices on

it support the new protocol. A similar question is valid for TSN: Does a TSN-capable network add any value if the end devices don't support TSN?

With TSN, the answer can, to a certain degree, be answered yes, because a network of TSN devices can provide a certain level of determinism, even if the end devices don't support TSN standards, as shown in Figure 3-4. In this example, the network has two TSN switches that support the IEEE 802.1Qbv-2015 Time-Aware Scheduler. The two end devices (sender and receiver) are standard, non-TSN Ethernet devices.

Although this network can't guarantee a specific latency between each end device and the switch it's connected to, it can guarantee a specific latency between the two TSN switches and thus, the entire network, if all switches support TSN. As a result, transmissions through the network backbone are more robust and deterministic, which makes transmissions between the two end devices more robust and deterministic to some degree, as only the non-determinism in the end device links contributes to the overall transmission jitter.

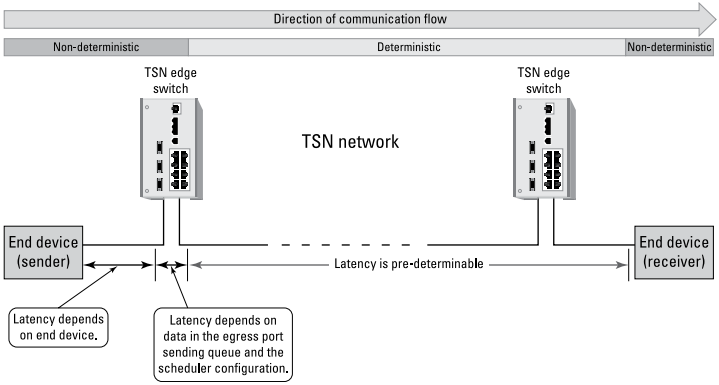


FIGURE 3-4: The benefits of TSN on a partially compliant network.

Integrating vendor-specific industrial Ethernet solutions

Before the finalization of TSN as a completely vendor-neutral real-time IEEE standard, many vendor-specific solutions were developed to cover the need for communicating time-critical data

across networks. Among them are well-established solutions such as PROFINET IRT, EtherCAT, and Sercos III. The availability and use of these vendor-specific solutions raise the following two questions:

- Will TSN replace these solutions?
- Can vendor-specific real-time Ethernet systems benefit from TSN?

The answer to the first question is no. TSN can't replace well-established vendor solutions because these vendor applications have their own configuration, parametrization, engineering, and application layer protocols that are outside the scope of TSN or any other IEEE 802 specification. IEEE 802 TSN is used to transport Ethernet frames. All higher protocol functionality, from configuration through human-machine interface to areas such as functional safety, remains in the hands of the application protocol.

The answer to the second questions is yes. Existing and new vendor-specific industrial Ethernet solutions can use TSN as transport technology to their benefit. All ecosystems can benefit from the technical advancements of the IEEE standards, such as higher bandwidth and support for new media types. In many cases, this is the direction in which vendor-specific systems are moving.

Addressing Security Issues

TSN doesn't specifically include cybersecurity mechanisms because security is outside the scope of basic Ethernet. Unfortunately, however, TSN presents new attack surfaces that were not present with Ethernet before. The largest attack surface is time synchronization. A successful attack on the time synchronization would be a very effective denial of service (DoS) because TSN can't operate without synchronized clocks on all devices.

Another aspect that makes cybersecurity slightly more complex in TSN networks is that many security mechanisms introduce additional latency and jitter into the network. For example, many industrial-grade firewall devices perform packet filtering through software on the device's central processing unit (CPU). This additional load on the CPU introduces jitter into the transmission path

because the performance of the packet filtering is dependent on the computational load of the CPU and the number of applied firewall rules. This jitter can be unacceptable for TSN, especially with cycle times in the low millisecond range or even lower.

Fortunately, methods are available to secure time-sensitive networks without compromising deterministic data transmission. This section describes two such methods.

WARNING

Make cyber security an integral part of the entire network design from the start. If security is implemented as an afterthought, it likely won't provide the protection necessary.

Segmenting traffic with zones and conduits

Securing a TSN network is just a matter of applying the correct technology at the right places in the network. General cyber security concepts such as zones and conduits, as specified in IEC 62443, can be used to help protect time-sensitive networks:

- **Zones:** A *zone* is a group of physical or logical devices that share common security requirements.
- **Conduits:** A *conduit* is a communication pathway that leads into or out of a zone.

Zones are separated by conduits, where traffic filtering and control are strictly enforced. Security can then be more relaxed within each zone to provide for unrestricted (or less restricted) data transmissions.

REMEMBER

When using conduits, keep the following important points in mind:

- Increasing security in conduits adds latency and jitter, so be sure to account for the latency requirements of the application.
- Communication zones can be designed around TSN streams with conduits at the border of the TSN streams.
- Conduits that carry TSN streams should resort to filtering technologies in hardware to ensure a minimal and predictable latency and jitter.

Filtering and policing traffic by stream

In addition to network security design that accommodates the requirements of TSN, the IEEE 802.1 working group has specified a security technology that protects against the most common, unintentional security threat in a network — the overloading of the network by a misbehaving device, sometimes referred to as the "babbling idiot problem."

To ensure that no device can overload a TSN network or even a single time slot, the TSN standard IEEE 802.1Qci-2017 specifies a set of supervision functions on ports of TSN switches called *per-stream filtering and policing*. Through these supervision functions, the switch can monitor whether an end device behaves correctly or is consuming inordinate bandwidth. In case of misbehavior, the switch can prune the traffic or shut down access for the misbehaving device completely.

IN THIS CHAPTER

- Recognizing that TSN is all about synchronizing activities
- Seeing the relationship with Ethernet before and after TSN was invented
- Addressing compatibility and security issues

Chapter 4
Ten TSN Takeaways

This book covers a lot of ground, from what TSN is and its applications in industry and in-vehicle automation, to how it works, and finally to the basics of building a time-sensitive network. All of this may leave you wondering: What do I really *need* to know about TSN?

This chapter highlights the key points about TSN — everything you should remember as you embark on your journey to discover more about this exciting technology and its applications in the real world.

TSN Is Ethernet Plus

TSN isn't so much a new technology as it is an improvement of an existing networking technology — Ethernet. Ethernet serves its purpose well as a standard for ensuring reliable data transmission across networks. If you send something across an Ethernet network, you can be confident that it will arrive at its destination.

The only trouble is that you can't be sure *precisely* when it will arrive. Data can get caught up in a queue, delaying its arrival. These delays are usually brief and don't significantly affect operations when all you're doing is opening a file or printing a

document. However, these delays make Ethernet impractical for applications that require real-time communications, such as in self-driving cars.

TSN changes all that by adding the timely delivery guarantees that were lacking in traditional Ethernet.

TSN Is Backward Compatible with Existing Ethernet

TSN continues the Ethernet tradition of remaining backward compatible with older technologies, which is good news for organizations that want to implement TSN on existing Ethernet networks. You won't need to install special gateways or protocol translators to make it work in an existing environment.

Timing Is Everything

TSN is all about tightly synchronizing activities across a network, so timing is crucial, as is reflected in the two most important TSN requirements:

- **Time synchronization:** All networked devices must agree on what time it is. Time of day doesn't matter, but the devices must be synchronized to a common time source.
- **Real-time scheduling:** Latencies must be guaranteed to ensure that data arrives on schedule. (*Latency* is the time required for data to travel from point of departure to its destination.)

REMEMBER

When selecting TSN devices, keep in mind that better clock synchronization results in more efficient TSN networks.

TSN Is Modular

TSN isn't a single standard; it's a collection of standards, including time synchronization, real-time communication, and communication stream reservation. This modularity enables

network engineers to choose and implement standards in ways that are most suitable for the network's intended application.


REMEMBER

A TSN-capable device doesn't have to support all TSN standards. Device manufacturers can implement the standards selectively, depending on the use case of the device, but they must declare which standards each device supports.

Configuration Is Key

Although Ethernet is designed to be plug-and-play, TSN is not. To reap the benefits of real-time communications, you must configure TSN devices to carefully coordinate their send and receive operations.

On smaller, static networks, manual configuration may be an option. On larger, more complex, or more dynamic networks, a mechanism is usually required to automate the configuration process.

TSN Is Ideal for Automation Applications

TSN is like the central nervous system in the human body, which enables the brain to carefully coordinate all bodily functions and movement often without the need for conscious thought. In a similar fashion, TSN enables timely communication among all sensors, actuators, and machinery, so they can coordinate their activities among one another or by way of a central control mechanism.

TSN's guaranteed timely delivery makes it ideal for automation applications, especially these two:

- Industrial automation
- In-vehicle automation, as in self-driving cars


REMEMBER

TSN's use is not restricted to automation applications. It can be beneficial in any application that requires reliable transmission of different priorities of traffic on a single network.

Guaranteed Latency Trumps Low Latency

TSN is often described as "low-latency" network communications, but latency is less of an issue than is jitter (variation in latency). What's important in TSN is that specific latencies be guaranteed to ensure that data is delivered precisely when it's scheduled to arrive — no sooner and no later. How long it takes for data to travel from point of departure to destination is less of a concern. As long as the latency is predictable and planned for (and not excessive), it's not a huge issue.

TSN Can Help Even If Some Devices Don't Support It

TSN can improve the overall performance of any Ethernet network even if one or more end devices don't support it. Any network connection between two TSN-compatible devices on a portion of the network may be configured to synchronize the devices and use real-time scheduling and other TSN standards to guarantee latencies between the devices. Although the entire Ethernet network won't be TSN optimized, it will benefit from more efficient communications over those optimized sections of the network.

TSN Introduces New Security Concerns

Because TSN relies so heavily on time synchronization and scheduling, it introduces unique security concerns, including the following:

- Malicious participants on the network might introduce additional data frames or change frame priorities to disrupt or delay mission-critical traffic.
- Security measures themselves might throw off timing by consuming extra CPU cycles necessary for ensuring timely processing of data frames.

However, existing security technologies and best practices can be used to secure TSN networks.

Existing Vendor-Specific Technologies May Benefit

Through minor changes in vendor-specific systems, for example, in master controllers, TSN can directly interact with vendor-specific, real-time Ethernet solutions. This flexibility enables the use of TSN to improve these systems without having to completely replace them.

HIRSCHMANN

A BELDEN BRAND

Be Ready for the Deterministic Data Transfer of the Future

- **Hirschmann is a pioneer** in TSN Industrial Ethernet and is constantly re-defining the limits of the technology
- **Hirschmann excels** in time synchronization. Hirschmann Switches are used in the most demanding applications that require precise timing
- **Hirschmann** network management software solutions enable the configuration and operation of modern IIoT networks

Speak with the expert to learn more about the TSN topic www.hirschmann.com/contact

Be certain. Belden.

Achieve dependable real-time communication

In highly automated systems, real-time communication is essential and sometimes vital. Several real-time communication technologies are used to ensure timely communications, but they have compatibility issues and offer limited support for future enhancements. *Time-sensitive networking (TSN)* overcomes these limitations by providing dependable real-time communication, high bandwidth, and backward compatibility to Ethernet devices. This book introduces TSN and explains how the magic happens.

Inside...

- Explore the standards behind TSN
- Coordinate communications with real-time scheduling
- Understand TSN configuration options
- Use TSN with non-TSN Ethernet devices
- Apply TSN in manufacturing automation
- Build automotive in-vehicle networks

BELDEN
SENDING ALL THE RIGHT SIGNALS

HIRSCHMANN
A BELDEN BRAND

Oliver Kleineberg is Global CTO for Industrial Networking at Hirschmann Automation and Control GmbH. **Axel Schneider** is Director of Research and Development for Industrial Networking at Hirschmann Automation and Control GmbH.

Go to Dummies.com®
for videos, step-by-step photos, how-to articles, or to shop!

for dummies®
A Wiley Brand

Also available as an e-book

ISBN: 978-1-119-52791-6

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.

Real-Time Syst (2007) 35:239–272
DOI 10.1007/s11241-007-9012-7

Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised

Robert I. Davis · Alan Burns · Reinder J. Bril · Johan J. Lukkien

Published online: 30 January 2007

Abstract Controller Area Network (CAN) is used extensively in automotive applications, with in excess of 400 million CAN enabled microcontrollers manufactured each year. In 1994 schedulability analysis was developed for CAN, showing how worst-case response times of CAN messages could be calculated and hence guarantees provided that message response times would not exceed their deadlines. This seminal research has been cited in over 200 subsequent papers and transferred to industry in the form of commercial CAN schedulability analysis tools. These tools have been used by a large number of major automotive manufacturers in the design of in-vehicle networks for a wide range of cars, millions of which have been manufactured during the last decade.

This paper shows that the original schedulability analysis given for CAN messages is flawed. It may provide guarantees for messages that will in fact miss their deadlines in the worst-case. This paper provides revised analysis resolving the problems with the original approach. Further, it highlights that the priority assignment policy, previously claimed to be optimal for CAN, is not in fact optimal and cites a method of obtaining an optimal priority ordering that is applicable to CAN. The paper discusses the possible impact on commercial CAN systems designed and developed using flawed schedulability analysis and makes recommendations for the revision of CAN schedulability analysis tools.

R. I. Davis (✉) · A. Burns
Real-Time Systems Research Group, Department of Computer Science, University of York, YO10 5DD, York, UK
e-mail: rob.davis@cs.york.ac.uk

A. Burns
e-mail: alan.burns@cs.york.ac.uk

R. J. Bril · J. J. Lukkien
Technische Universiteit Eindhoven (TU/e), Den Dolech 2, 5600 AZ Eindhoven, The Netherlands
e-mail: r.j.bril@tue.nl

J. J. Lukkien
e-mail: j.j.lukkien@tue.nl

Keywords Controller Area Network (CAN) · Fixed priority scheduling · Non-pre-emptive scheduling · Schedulability analysis · Response time analysis · Priority assignment policies

1 Introduction

1.1 Background

Controller Area Network (CAN) is a serial communications bus designed to provide simple, efficient and robust communications for in-vehicle networks. CAN was developed by Robert Bosch GmbH, beginning in 1983, and presented to a wider audience at the Society of Automotive Engineers (SAE) Congress in 1986—effectively the "birth of CAN". In 1987, the first CAN controller chips were released by Intel (82526) and Philips (82C200). In the early 1990s, Bosch submitted the CAN specification (Bosch, 1991) for standardisation, leading to publication of the first ISO standard for CAN (11898) in 1993 (ISO, 1993).

Mercedes was the first automotive manufacturer to deploy CAN in a production car, the 1991 S-class. By the mid 1990s, the complexity of automotive electronics was increasing rapidly. The number of networked Electronic Control Units (ECUs) in Mercedes, BMW, Audi and VW cars went from 5 or less at the beginning of the 1990s to around 40 at the turn of the millennium. With this explosion in complexity traditional point-to-point wiring became increasingly expensive to manufacture, install, and maintain due to the hundreds of separate connections and tens of kilograms of copper wire required. As a result CAN was rapidly adopted by the cost-conscious automotive industry, providing an effective solution to the problems posed by increasing vehicle electronics content. Following on from Mercedes, other manufacturers including Volvo, Saab, BMW, Volkswagen, Ford, Renault, PSA, Fiat and others all adopted CAN technology.

As a result of the wholesale adoption of CAN by the automotive industry, sales of CAN nodes (8, 16 and 32-bit microcontrollers with on-chip CAN peripherals) grew from just under 50 million in 1999 to over 340 million in 2003[1]—see Fig. 1.

By 2004, there were at least 15 silicon vendors manufacturing, in total, over 50 different microprocessor families with on-chip CAN capability.

Today almost all of the cars manufactured in Europe are equipped with at least one CAN bus. In the United States, the Environmental Protection Agency has mandated the use of CAN, for On Board Diagnostics, in all cars and light trucks sold in the US from model year 2008 onwards.

1.2 Automotive applications

In automotive applications, CAN is typically used to provide high speed networks (500 Kbits/s) connecting chassis and power-train ECUs, for example engine management and transmission control. It is also used for low speed networks (100 or 125 Kbits/s)

[1] Figures from the CAN in Automation (CiA) website www.can-cia.org

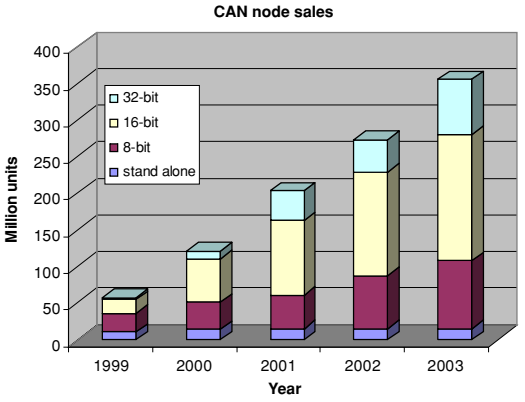


Fig. 1 Sales of microcontrollers with onchip CAN peripherals

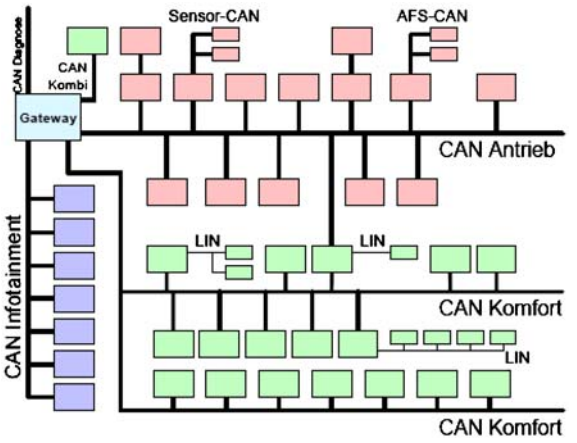


Fig. 2 VW Passat network architecture

connecting body and comfort electronics, for example door modules, seat modules and climate control. Data required by ECUs on different networks is typically gatewayed between the different CAN buses by a powerful ECU connected to both.

The network architecture of the VW Passat (Leohold, 2005) shown in Fig. 2, reproduced with permission from Nolte (2006), illustrates how a number of CAN buses are used to connect around 45 ECUs in that vehicle. Also shown in Fig. 2 are three Local Interconnect Networks (LIN). LIN is a complementary technology to CAN, and is used to provide inexpensive, low speed (20 Kbits/s) connectivity (LIN Consortium, 2003).

Table 1 summarises the requirements placed on in-vehicle networks for the BMW 7 Series. This is typical of automotive applications, where individual CAN buses are used to connect between 2 and 32 ECUs at bandwidths ranging from 100 to 500 Kbits/s (Frischkorn, 2005).

In automotive applications the *messages* sent on CAN are used to communicate state information, referred to as *signals*, between different ECUs. Examples of signals include: wheel speeds, oil and water temperature, engine rpm, gear selection, accelerator position, dashboard switch positions, climate control settings, window switch positions, fault codes, diagnostic information and so on. In a high-end vehicle, such as

Table 1 BMW 7 Series network requirements

	No. of ECUs	Bandwidth	No. of Messages	Cycle times
Body	14–30	100 Kbits/s	300	50 ms-2 s
Chassis	6–10	500 Kbits/s	180	10 ms-1 s
Power-train	3–6	500 Kbits/s	36	10 ms-10 s

the VW Phaeton, there can be more than 2500 distinct signals (Leohold, 2004), each effectively replacing what would, in a traditional point-to-point wiring loom, have been a separate wire.

Many of these signals have real-time constraints associated with them. For example, an ECU reads the position of a switch attached to the brake pedal. This ECU must send a message over the CAN network, carrying information (a signal) that the brakes have been applied. The ECU responsible for the rear light clusters receives the message, recognises the change in the value of the signal, and switches the brake lights on. All within a few tens of milliseconds of the brake pedal being pressed. Engine, transmission, and stability control systems typically place even tighter time constraints on signals, which may need to be sent as frequently as once every 5 milliseconds to meet their time constraints (Society of Automotive Engineers, 1993).

1.3 Research and real-time analysis

CAN is a serial data bus that supports priority based message arbitration and non-pre-emptive message transmission. In the early 1990s, a common misconception about CAN was that although the protocol was very good at transmitting the highest priority message with low latency, it was not possible to guarantee that less urgent signals, carried in lower priority messages, would meet their deadlines.

Tindell and Burns (1994) and Tindell et al. (1994b, 1995) showed how research into fixed priority pre-emptive scheduling for single processor systems could be adapted and applied to the scheduling of messages on CAN. This analysis provided a method of calculating the worst-case response times of all CAN messages. Using this analysis it became possible to engineer CAN based systems for timing correctness, providing guarantees that all messages, and the signals that they carry, would meet their deadlines.

Tindell's seminal research heavily influenced the design of on-chip CAN peripherals such as Motorola msCAN (Motorola, 1998) and has lead to a large body of work into schedulability theory and error models for CAN (Punnekkat et al., 2000; Nolte et al., 2002, 2003; Broster et al., 2002, 2005; Hansson et al., 2002; Broster and Burns, 2003), including at least two PhD theses (Broster, 2003; Nolte, 2006). Overall, this research into CAN scheduling has been cited in over 200[2] subsequent papers.

In 1995, Tindell's research was recognised by Volvo Car Corporation and successfully used in the configuration and analysis of the CAN buses for the forthcoming

[2] As of August 2006, reference (Tindell and Burns, 1994) has 78 citations, reference (Tindell et al., 1995) 199 citations and reference (Tindell et al., 1994b) 110 citations (Google Scholar).

Table 2 CAN messages highlighting flawed analysis

Message	Priority	Period	Deadline	TX time
A	1	2.5 ms	2.5 ms	1 ms
B	2	3.5 ms	3.25 ms	1 ms
C	3	3.5 ms	3.25 ms	1 ms

Volvo S80 (P23) (Casparsson et al., 1998). Following the success of this project, Volcano Communications Technologies AB[3] used Tindell's analysis as the basis of a commercial CAN schedulability analysis tool, called Volcano Network Architect.

Since 1998, these tools and the Volcano concept (Casparsson et al., 1998) have been used in the design and development of CAN networks and electronics systems for the Volvo XC90, S80, S/V/XC70, S60, V50 and S40 as well as many other cars from different manufacturers.

Prior to Tindell's work, low levels of bus utilization, up to 30 or 40%, were typical in automotive applications, with extensive testing required to obtain confidence that CAN messages would meet their deadlines. With the advent of a systematic approach based on schedulability analysis, CAN bus utilization could be increased to around 80% (DeMeis, 2005) whilst still guaranteeing that deadlines would be met.

1.4 Motivation

The design and development of many in vehicle Controller Area Networks relies on the schedulability analysis of CAN given by Tindell and Burns (1994) and Tindell et al. (1994b, 1995). In this section we show that this analysis is flawed. It may result in computed worst-case response times for messages that are optimistic, i.e. less than the response times that may actually occur. The set of CAN messages listed in Table 2 serve to highlight the problem with the existing schedulability analysis of CAN. As a simple example, we have assumed a 125 Kbit/s network with 3 messages, each of which carries 7 bytes of signal data. Assuming 11-bit identifiers and worst-case bit-stuffing, the maximum length of each message is 125 bits. The maximum transmission time of each message is therefore 1 ms.

The analysis method given by Tindell and Burns (1994) and Tindell et al. (1994b, 1995) calculates the worst-case response times of messages A, B and C as 2 ms, 3 ms and 3 ms respectively. Hence the system is deemed to be schedulable—the analysis supposedly guarantees that all of the messages will meet their deadlines in the worst case, despite the high bus utilisation of 97%.

Figure 3 illustrates the worst-case scenario for transmission of message C. We note that the first instance of this message is delayed by higher priority messages A and

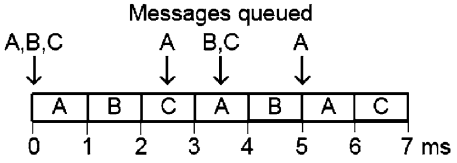


Fig. 3 Worst-case scenario for message C

[3] Volcano Communications Technologies AB was acquired by Mentor Graphics in May 2005.

B, leading to a response time of 3 ms—this is the "worst-case response time" calculated using existing CAN schedulability analysis methods. However, as message transmission is non-pre-emptive, the first transmission of message C has a knock on effect, delaying subsequent transmissions of higher priority messages A and B. Some of this higher priority interference is *pushed through* into the next period of message C leading to a longer response time for the second instance of message C.

At time $t = 7$ ms, the second instance of message C completes transmission with a response time of 3.5 ms. (Note at time $t = 7$ ms, there are no higher priority messages awaiting transmission and so there is no further push through interference that could delay subsequent instances of message C).

The actual worst-case response time for message C is 3.5 ms, which is greater than its deadline of 3.25 ms. The system is therefore unschedulable; contrary to the guarantees given by Tindell and Burns (1994) and Tindell et al. (1994b, 1995).

In fact, if the periods of messages B and C are shortened from 3.5 ms to 3.25 ms then the existing analysis results in unchanged worst-case response times, implying that the messages will still meet their deadlines. However, with these shorter periods the overall bus utilisation exceeds 100% and so the system cannot possibly be schedulable!

1.5 Related work

The schedulability analysis for CAN builds on previous research into fixed priority scheduling of tasks on single processor systems.

Lehoczky (1990) introduced the concept of a busy period and showed that if tasks have deadlines greater than their periods, referred to as *arbitrary deadlines*, then it is necessary to examine the response times of all invocations of a task falling within a busy period in order to determine the worst-case response time. Harbour et al. (1991) showed that if deadlines are less than or equal to periods, but priorities vary during execution, then again multiple invocations must be inspected to determine the worst-case response time. We note that non-pre-emptive scheduling is effectively a special case of pre-emptive scheduling with varying execution priority—as soon as a task starts to execute its priority is raised to the highest level. Tindell et al. (1994a) improved upon the work of Lehoczky (1990) providing a formulation for arbitrary deadline analysis based on a recurrence relation.

Building upon these earlier results, George et al. (1996) provided comprehensive schedulability analysis of non-pre-emptive fixed priority scheduling of single processor systems.

Bril (2006) refuted the analysis of fixed priority systems with deferred pre-emption given by Burns (1994), showing that this analysis may result in computed worst-case response times that are optimistic. The schedulability analysis for CAN given by Tindell and Burns (1994) and Tindell et al. (1994b, 1995) builds upon Burns (1994) and suffers from essentially the same flaw. A similar issue with work on pre-emption thresholds (Wang and Saksena, 1999) was first identified and corrected by Regehr (2002).

The revised schedulability analysis presented in this paper aims to provide an evolutionary improvement upon the analysis of CAN given by Tindell and Burns (1994) and Tindell et al. (1994b, 1995). To do so, it draws upon the analysis of Tindell

et al. (1994a) for fixed priority pre-emptive scheduling of systems with arbitrary deadlines and the analysis of George et al. (1996) for fixed priority non-pre-emptive systems.

A technical report (Bril et al., 2006a) and a workshop paper (Bril et al., 2006b) highlighted the problem for CAN but did not provide a specific in-depth solution. That is the purpose of this paper. A further technical report (Bril et al., 2006c) complements this paper, providing formal proofs of the worst-case response time of tasks under fixed priority scheduling with deferred pre-emption.

1.6 Organisation

The remainder of this paper is organised as follows: Section 2 describes the CAN protocol and terminology before outlining a suitable scheduling model and notation on which to base revised schedulability analysis. Section 3 provides new schedulability analysis for CAN, correcting the flaws in the existing approach. Section 4 discusses the system and message parameters needed for the flaws in the existing analysis to result in incorrect worst-case response times and hence misleading guarantees. Section 5 discusses the issue of optimal priority assignment for CAN messages. Section 6 summarises the implications of flaws in the existing analysis for commercial CAN applications. Finally, Section 7 concludes with a summary of the main contributions of this paper and recommendations for further research.

2 Controller area network

This section describes elements of the CAN protocol and characteristics of a system model that are needed to formulate a schedulability test. For a complete description of the CAN protocol, see the CAN specification version 2.0 (Bosch, 1991)

2.1 CAN protocol and terminology

CAN is an asynchronous multi-master serial data bus that uses Carrier Sense Multiple Access/Collision Resolution (CSMA/CR) to determine access.

CAN was designed as a simple and robust broadcast bus capable of operating at speeds of up to 1 Mbit/s. Message transfer over CAN is controlled by 4 different types of *frame*: Data frames, Remote Transmit Request (RTR) frames, Overload frames and Error frames.

The layout of a standard format data frame is shown in Fig. 4. Each CAN data frame is required to have a unique identifier. Identifiers may be 11-bit (standard format) or

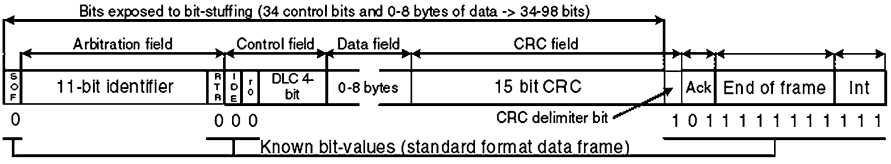


Fig. 4 Standard format data frame

29-bit (extended format). The identifier serves two purposes beyond simply identifying the message. First, the identifier is used as a priority to determine which message, among those contending for the bus, will be transmitted next. Second, the identifier may be used by receivers to filter out messages they are not interested in, and so reduce the load on the receiver's host microprocessor.

In this paper we are interested in the schedulability of data frames, with error frames also considered in Section 3.5. The schedulability analysis can however easily be extended to include RTR frames using the approach given by Tindell et al. (1995).

2.1.1 Priority based arbitration

The CAN physical layer supports two states termed *dominant* ('0') and *recessive* ('1'). If two or more CAN controllers are transmitting at the same time and at least one of them transmits a '0' then the value on the bus will be a '0'. This mechanism is used to control access to the bus and also to signal errors.

The CAN protocol calls for nodes to wait until a *bus idle period*[4] is detected before attempting to transmit. If two or more nodes start to transmit at the same time, then by monitoring each bit on the bus, each node can determine if it is transmitting the highest priority message (with a numerically lower identifier) and should continue or if it should stop transmitting and wait for the next bus idle period before trying again. As the message identifiers are unique, a node transmitting the last bit of the identifier field, without detecting a '0' bit that it did not transmit, must be transmitting the highest priority message that was ready for transmission at the start of arbitration. This node then continues to transmit the remainder of its message, all other nodes having backed off.

The requirement for a node to be able to overwrite a recessive bit, and the transmitting node detect this change, limits the combination of physical length and speed of a CAN bus. The duration of each bit must be sufficient for the signal to propagate the length of the network. This limits the maximum data rate to 1 Mbit/s for a network up to 40 m in length or to 125 Kbit/s for a 500 m long network.

The arbitration mechanism employed by CAN means that messages are sent as if all the nodes on the network shared a single global priority based queue. In effect messages are sent on the bus according to fixed priority non-pre-emptive scheduling.

The above high level description is a somewhat simplified view of the timing behaviour of CAN. CAN does not have a global concept of time, rather each CAN controller typically has its own clock which, within a tolerance specified by the protocol, may drift with respect to the clocks of other nodes. The CAN protocol therefore requires that nodes re-synchronise on each message transmission. Specifically, every node must synchronise to the leading edge of the *start of frame* bit caused by whichever node starts to transmit first.

Normally, CAN nodes are only allowed to start transmitting when the bus is idle. Thus, when the bus is idle beyond the 3-bit *inter-frame space* and a node starts to transmit a message beginning with the dominant start of frame bit ('0'), then all the other nodes synchronise on the leading edge of this bit and become *receivers*—i.e.

[4] A *bus idle period* is an interval of arbitrary length comprising only recessive bits and beginning with the last bit of the *inter-frame space*—the final 3-bit field shown in Fig. 4.

they are not permitted to transmit until the bus next becomes idle. In this case any message that becomes ready for transmission after the leading edge of the start of frame bit has to wait for the next bus idle period before it can enter into arbitration.

However, to avoid problems due to clock drift, the CAN protocol also specifies that, if a CAN node has a message ready for transmission and detects a dominant bit at the 3rd bit of the inter-frame space, it will interpret this as a start of frame bit, and with the next bit, start transmitting its own message with the first bit of the identifier without first transmitting a start of frame bit and without becoming a receiver.[5] Again the leading edge of the start of frame bit causes a synchronisation. This behaviour ensures that any messages that become ready for transmission, whilst another message is being sent on the bus, are entered into the next round of arbitration, irrespective of any within tolerance clock drift.

2.1.2 Error detection

CAN was designed as a robust and reliable form of communication for short messages. Each data frame carries between 0 and 8 bytes of payload data and has a 15-bit Cyclic Redundancy Check (CRC). The CRC is used by receiving nodes to check for errors in the transmitted message. If a node detects an error in the transmitted message, which may be a bit-stuffing error (see Section 2.1.3), a CRC error, a form error in the fixed part of the message, or an acknowledgement error, then it transmits an error flag. The error flag consists of 6 bits of the same polarity: '000000' if the node is in the error active state and '111111' if it is error passive. Transmission of an error flag typically causes other nodes to also detect an error, leading to transmission of further error flags.

Figure 5 illustrates CAN error frames, for further details see (Bosch, 1991; Punnekkat et al., 2000). The length of an error frame is between 17 and 31 bits. Hence each message transmission that is signalled as an error can lead to a maximum of 31 additional bits[6] of error recovery overhead plus re-transmission of the message itself.

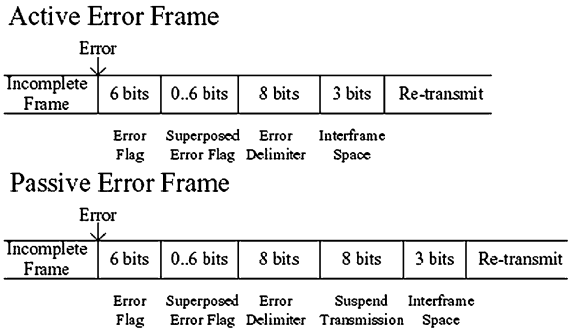


Fig. 5 CAN error frames

[5] See page 54 of the CAN Specification version 2.0 (Bosch, 1991).

[6] The analysis given by Tindell and Burns (1994) and Tindell et al. (1994b, 1995) uses 29 bits as the error recovery overhead as specified on page 8 of part A of the CAN specification 2.0 (Bosch, 1991) for standard identifiers only. We use 31 bits as specified on page 40 of the CAN specification 2.0 Part B (Bosch, 1991) for both standard and extended identifiers.

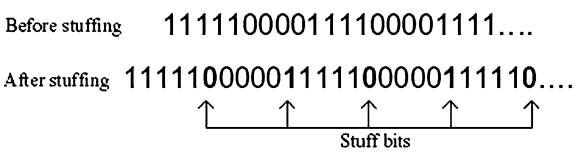


Fig. 6 Worst-case bit stuffing

2.1.3 Bit stuffing

As the bit patterns '000000' and '111111' are used to signal errors, it is essential that these bit patterns are avoided in the variable part of a transmitted message—see Fig. 4. The CAN protocol therefore requires that a bit of the opposite polarity is inserted by the transmitter whenever 5 bits of the same polarity are transmitted. This process is referred to as *bit-stuffing*, and is reversed by the receiver.

The worst-case scenario for bit-stuffing is shown in Fig. 6. Note that each stuff bit begins a sequence of 5 bits that is itself subject to bit stuffing.

Stuff bits increase the maximum transmission time of CAN messages. Including stuff bits and the inter-frame space, the maximum transmission time C_m, of a CAN message m containing s_m data bytes is given by:[7]

$$C_m = \left(g + 8s_m + 13 + \left\lfloor \frac{g + 8s_m - 1}{4} \right\rfloor \right) \tau_{\text{bit}} \tag{1}$$

where g is 34 for standard format (11-bit identifiers) or 54 for extended format (29-bit identifiers), $\lfloor a/b \rfloor$is notation for the *floor* function, which returns the largest integer less than or equal to a/b, and τ_{bit} is the transmission time for a single bit.

The formula given in Eq. (1) simplifies to:

$$C_m = (55 + 10s_m)\tau_{\text{bit}} \tag{2}$$

for 11-bit identifiers and

$$C_m = (80 + 10s_m)\tau_{\text{bit}} \tag{3}$$

for 29-bit identifiers.

2.2 Scheduling model

In this section we describe an appropriate system model and notation that can be used to analyse worst-case response times of messages on CAN and hence determine system schedulability.

The system is assumed to comprise a number of nodes (microprocessors) connected via CAN. Each node is assumed to be capable of ensuring that, at any given time when arbitration starts, the highest priority message queued at that node is entered into arbitration.

[7] This formula corrects a similar one by Tindell and Burns (1994) and Tindell et al. (1994b, 1995) which does not account for the fact that stuff bits are themselves also subject to bit stuffing.

The system is assumed to contain a static set of hard real-time messages, each statically assigned to a node on the network. Each message m has a fixed identifier and hence a unique priority. As priority uniquely identifies each message, in the remainder of this paper we will overload m to mean either message m or priority m as appropriate. Each message has a maximum number of data bytes s_m, and a maximum transmission time C_m, given by Eq. (1).

Each message is assumed to be queued by a software task, process or interrupt handler executing on the host microprocessor. This task is either invoked by, or polls for, the event and takes a bounded amount of time, between 0 and J_m, to queue the message ready for transmission. J_m is referred to as the *queuing jitter* of the message and is inherited from the overall response time of the task, including any polling delay.

The event that triggers queuing of the message is assumed to occur with a minimum inter-arrival time of T_m, referred to as the message *period*. This model supports events that occur strictly periodically with a period of T_m, events that occur sporadically with a minimum separation of T_m, and events that occur only once before the system is reset, in which case T_m is infinite.

Each message has a hard deadline D_m, corresponding to the maximum permitted time from occurrence of the initiating event to the end of successful transmission of the message, at which time the message data is assumed to be available on the receiving nodes that require it. Tasks on the receiving nodes may place different timing requirements on the data, however in such cases we assume that D_m is the tightest such time constraint.

The *worst-case response time* R_m, of a message is defined as the longest time from the initiating event occurring to the message being received by the nodes that require it.

A message is said to be *schedulable* if and only if its worst-case response time is less than or equal to its deadline ($R_m \leq D_m$). The system is schedulable if and only if all of the messages in the system are schedulable.

2.3 Practical implications of the model

Engineers wanting to use the analysis given in Section 3 to analyse CAN based systems must be careful to ensure that all of the assumptions of the above model hold for their system.

In particular, it is important that each CAN controller and device driver is capable of ensuring that, at any given time when arbitration starts, the highest priority message queued at that node is entered into arbitration. This behaviour is essential if message transmission is to take place as if there were a single global priority queue and for the analysis given in Section 3 to be applicable. As noted by Tindell and Burns (1994), the Philips 82C500 CAN controller cannot in general support this behaviour. Also the Intel 82527 CAN controller has a feature where messages are entered into arbitration in slot order rather than identifier order. In this case it is important that messages are allocated to slots in identifier order to preserve the correct priority based behaviour.

Many on-chip CAN controllers have multiple slots that can be allocated to either transmit or receive a specific message. For example some Motorola, National Semiconductor, Fujitsu and Hitachi on-chip CAN peripherals have 14, 15 or 16 such slots.

These slots typically have only a single buffer and therefore it is necessary to ensure that the previous instance of a message has been transmitted before any new data is written into the buffer, otherwise the previous message will be overwritten and lost. This behaviour provides an additional constraint on message transmission: the deadline of each message must be less than or equal to its period ($D_m \leq T_m$).

Recall that the worst-case response time of a message is from the occurrence of the initiating event to the end of successful message reception at the receiving nodes. As noted by Broster (2003), receiving nodes can access the message following the end of frame marker and before the 3-bit inter-frame space—see Fig. 4. The analysis given in the remainder of this paper is slightly pessimistic in that it includes the 3-bit inter-frame space in the computed worst-case response times. To remove this small degree of pessimism, it is valid to simply subtract $3\tau_{\text{bit}}$ from the computed response time values.

The time constraint of interest to engineers is the overall end-to-end response time from an initiating event occurring, such as the brake pedal switch closing, to the corresponding output response, for example the brake lights illuminating. The worst-case response time of a message represents only part of this overall end-to-end response time. There is an additional delay to consider, corresponding to the worst-case time between the message becoming available at the receiving node and the output being made. Processing of the signal information contained in a message is typically done either by a task that polls for the message or by an interrupt handler that is triggered by message reception. The worst-case response time of the receiving task or interrupt handler, including any polling delay, needs to be added to the worst-case response time of the message to determine the overall end-to-end response time.

The scheduling model assumed in this paper uses only one time domain, whilst CAN typically has a separate clock source for each node on the network. To ensure that the schedulability analysis for a real network does not produce optimistic results, it is necessary to take clock tolerances into account. This can be achieved by converting to real-time as follows: for message jitters and bit times on the bus the conversion to real-time should assume that the node clocks run as slowly as their tolerance allows. Similarly, message periods and deadlines derived from node clocks should be converted to real-time assuming that the node clocks run as quickly as their tolerance allows.

3 Response time analysis

Response time analysis for CAN aims to provide a method of calculating the worst-case response time of each message. These values can then be compared to the message deadlines to determine if the system is schedulable. Initially we provide analysis assuming no errors on the CAN bus. This analysis is then extended, in Section 3.5, to account for errors on the bus.

For systems complying with the scheduling model given in Section 2.2, CAN effectively implements fixed priority non-pre-emptive scheduling of messages. Following the analysis given by Tindell and Burns (1994) and Tindell et al. (1994b, 1995) the worst-case response time of a message can be viewed as being made up of three elements:

(i) The queuing jitter J_m, corresponding to the longest time between the initiating event and the message being queued, ready to be transmitted on the bus.
(ii) The queuing delay w_m, corresponding to the longest time that the message can remain in the CAN controller slot or device driver queue, before commencing successful transmission on the bus.
(iii) The transmission time C_m, corresponding to the longest time that the message can take to be transmitted.

The worst-case response time of message m is given by:

$$R_m = J_m + w_m + C_m \tag{4}$$

The queuing delay w_m, comprises two elements:

(i) *blocking* B_m, due to lower priority messages which may be in the process of being transmitted when message m is queued, and
(ii) *interference* due to higher priority messages which may win arbitration and be transmitted in preference to message m.

Given the behaviour of CAN described in the final two paragraphs of Section 2.1.1, the maximum amount of blocking occurs when a lower priority message starts transmission immediately before message m is queued and hence ready to be transmitted on the bus. Message m must wait until the bus is idle before it can be entered into arbitration. The maximum blocking time B_m, is given by:

$$B_m = \max_{k \in lp(m)} (C_k) \tag{5}$$

where *lp(m)* is the set of messages with lower priority than m.

The concept of a *busy period*, introduced by Lehoczky (1990), is fundamental in analysing worst-case response times. Modifying the definition of a busy period given by Harbour et al. (1991) to apply to CAN messages, a priority level-m busy period is defined as follows:

(i) It starts at some time t^s when a message of priority m or higher is queued ready for transmission and there are no messages of priority m or higher waiting to be transmitted that were queued strictly before time t^s.
(ii) It is a contiguous interval of time during which any message of priority lower than m is unable to start transmission and win arbitration.
(iii) It ends at the earliest time t^e when the bus becomes idle, ready for the next round of transmission and arbitration, yet there are no messages of priority m or higher waiting to be transmitted that were queued strictly before time t^e.

The key characteristic of a busy period is that all messages of priority m or higher, queued strictly before the end of the busy period, are transmitted during the busy period. These messages cannot therefore cause any interference on a subsequent instance of message m queued at or after the end of the busy period.

In mathematical terminology, busy periods can be viewed as right half-open intervals: $[t^s, t^e)$ where t^s is the start of the busy period and t^e the end of the busy period. Thus the end of one busy period may correspond to the start of another separate busy

period. This is in contrast to the simpler definition given by Lehoczky (1990), which unifies two adjacent busy periods, as we have defined them, and therefore sometimes results in analysis of more message instances than is strictly necessary.

The worst-case queuing delay for message m occurs for some instance of message m queued within a priority level-m busy period that starts immediately after the longest lower priority message begins transmission. This *maximal* busy period begins with a so-called *critical instant* (Liu and Layland, 1973) where message m is queued simultaneously with all higher priority messages and then each of these higher priority messages is subsequently queued again after the shortest possible time interval. In the remainder of this paper whenever we refer to a busy period we mean this maximum length busy period.

If more than one instance of message m is transmitted during a priority level-m busy period, then it is necessary to determine the response time of each instance, in order to find the overall worst-case response time of the message.

3.1 Basic analysis and stopping condition

Tindell and Burns (1994) and Tindell et al. (1994b, 1995) give the following equation for the worst-case queuing delay:

$$w_m = B_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m + J_k + \tau_{\text{bit}}}{T_k} \right\rceil C_k \tag{6}$$

where *hp(m)* is the set of messages with priorities higher than m, and $\lceil a/b \rceil$ is notation for the *ceiling* function which returns the smallest integer greater than or equal to a/b.

Although w_m appears on both sides of Eq. (6), as the right hand side is a monotonic non-decreasing function of w_m, the equation may be solved using the recurrence relation given below.

$$w_m^{n+1} = B_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{\text{bit}}}{T_k} \right\rceil C_k \tag{7}$$

A suitable starting value is $w_m^0 = B_m$. The recurrence relation iterates until, either $J_m + w_m^{n+1} + C_m > D_m$ in which case the message is not schedulable, or $w_m^{n+1} = w_m^n$ in which case the worst-case response time of the *first instance of the message in the busy period* is given by: $J_m + w_m^{n+1} + C_m$.

The flaw in the above analysis is that, given the constraint $D_m \leq T_m$, it implicitly assumes that if message m is schedulable, then the priority level-m busy period will end at or before T_m. We observe that with fixed priority pre-emptive scheduling this would always be the case, as on completion of transmission of message m no higher priority message could be awaiting transmission. However, with fixed priority non-pre-emptive scheduling, a higher priority message can be awaiting transmission when message m completes transmission, and thus the busy period can extend beyond T_m, as shown by the example in Section 1.4.

The length t_m, of the priority level-m busy period is given by the following recurrence relation, starting with an initial value of $t_m^0 = C_m$, and finishing when $t_m^{n+1} = t_m^n$:

$$t_m^{n+1} = B_m + \sum_{\forall k \in hep(m)} \left\lceil \frac{t_m^n + J_k}{T_k} \right\rceil C_k \tag{8}$$

where *hep* is the set of messages with priority higher than or equal to m. As the right hand side is a monotonic non-decreasing function of t_m, then the recurrence relation is guaranteed to converge provided that the bus utilisation U_m, for messages of priority m and higher, is less than 1:

$$U_m = \sum_{\forall k \in hep(m)} \frac{C_k}{T_k} \tag{9}$$

If $t_m \leq T_m - J_m$, then the busy period ends at or before the second instance of message m is queued. This means that only the first instance of the message is transmitted during the busy period. The existing analysis calculates the worst-case queuing time for this instance, via Eq. (7), and hence provides the correct worst-case response time in this case.

If $t_m > T_m - J_m$, then the existing analysis may give an optimistic worst-case response time, depending on whether the first, or a subsequent instance of message m has the longest response time.

We observe that the analysis presented in appendix A.2 of George et al. (1996) suggests that t_m is the smallest value that is a solution to Eq. (8), however this is not strictly correct. For the lowest priority message $B_m = 0$, and so $t_m = 0$ is trivially the smallest solution when all of the messages have zero jitter. We avoid this problem by using an initial value of $t_m^0 = C_m$.

3.2 Checking multiple instances

The number of instances Q_m, of message m that become ready for transmission before the end of the busy period is given by:

$$Q_m = \left\lceil \frac{t_m + J_m}{T_m} \right\rceil \tag{10}$$

To determine the worst-case response time of message m, it is necessary to calculate the response time of each of the Q_m instances. The maximum of these values then gives the worst-case response time.

In the following analysis, we use the index variable q to represent an instance of message m. The first instance in the busy period corresponds to $q = 0$, and the final instance to $q = Q_m - 1$. The longest time from the start of the busy period to instance q beginning successful transmission is given by:

$$w_m^{n+1}(q) = B_m + qC_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{\text{bit}}}{T_k} \right\rceil C_k \tag{11}$$

Table 3 CAN messages

Message	Priority	Period	Deadline	TX time
A	1	2.5 ms	2.5 ms	1 ms
B	2	3.5 ms	3.25 ms	1 ms
C	3	3.5 ms	3.25 ms	1 ms

The recurrence relation starts with a value of $w_m^0(q) = B_m + qC_m$, and ends when $w_m^{n+1}(q) = w_m^n(q)$, or when $J_m + w_m^{n+1}(q) - qT_m + C_m > D_m$ in which case the message is unschedulable. For values of $q > 0$ an efficient starting value is given by $w_m^0(q) = w_m(q-1) + C_m$.

The event initiating instance q of the message occurs at time $qT_m - J_m$ relative to the start of the busy period, so the response time of instance q is given by:

$$R_m(q) = J_m + w_m(q) - qT_m + C_m \tag{12}$$

The worst-case response time of message m is therefore:

$$R_m = \max_{q=0..Q_m-1}(R_m(q)) \tag{13}$$

We note that the analysis presented above is also applicable when messages have deadlines that are greater than their periods, so called arbitrary deadlines. However, if such timing characteristics are specified then the software device drivers or CAN controller hardware may need to be capable of buffering more than one instance of a message. N_m, the number of instances of each message that need to be buffered is bounded by:

$$N_m = \left\lceil \frac{R_m}{T_m} \right\rceil \tag{14}$$

We observe that the analysis presented by George et al. (1996) effectively uses $Q_m = \lfloor t_m/T_m \rfloor + 1$ rather than $Q_m = \lceil t_m/T_m \rceil$. This yields a value which is one too large when the length of the busy period plus jitter is an integer multiple of the message period. Although this does not give rise to problems, we prefer the more efficient formulation given by Eq. (10).

3.3 Example

In Section 1.4 we showed, with the aid of a simple example, how the existing analysis can provide optimistic worst-case response times and hence flawed guarantees that messages will meet their deadlines. We return to this example to illustrate how the analysis presented in this paper computes the correct worst-case response times. For ease of reference, the table of message parameters is repeated below.

Using the new analysis, the worst-case response time of message C ($m = 3$) is calculated as follows: As there are no lower priority messages, $B_3 = 0$. Starting with

a value of $t_3^0 = C_3 = 1$, the recurrence relation given by Eq. (8) iterates as follows: $t_3^1 = 3$, $t_3^2 = 4$, $t_3^3 = 6$, $t_3^4 = 7$, converging as $t_3^5 = t_3^4 = 7$. The length of the busy period is therefore 7.0 ms, and the number of instances of message C that need to be examined is given by Eq. (10):

$$Q_3 = \left\lceil \frac{7.0}{3.5} \right\rceil = 2$$

This tells us that there is the possibility that the existing analysis will calculate an optimistic worst-case response time. The value could, however, still be correct if the first instance of the message has the longest response time.

Calculation of the response time of the first instance proceeds using Eq. (11): $w_3^0(0) = 0$, $w_3^1(0) = 2$, converging as $w_3^2(0) = w_3^1(0) = 2$. Using Eq. (12), we have $R_3(0) = 3$, the same response time calculated by the existing analysis.

Moving on to the second instance, $w_3^0(1) = w_3(0) + C_m = 3$, $w_3^1(1) = 4$, $w_3^2(1) = 5$, $w_3^3(1) = 6$. At this point computation would normally stop as the response time, given by $J_3 + w_3(q) - qT_3 + C_3$, has reached 3.5 ms which is greater than the message deadline. However, if we continue iterating, assuming a longer deadline, then the recurrence relation converges on $w_3^4(1) = w_3^3(1) = 6$ and hence $R_3(1) = 3.5$ ms. The worst-case response time of message C is in fact 3.5 ms, as previously illustrated by Fig. 3 in Section 1.4.

3.4 Sufficient schedulability tests

The analysis given in Sections 3.1 and 3.2 corrects a significant flaw in the existing schedulability analysis for CAN. However, the schedulability test presented is more complex, potentially requiring the computation of multiple response times.

In this section we present two simpler but more pessimistic schedulability tests. These tests are "sufficient but not necessary". By "sufficient" we mean that all systems deemed to be schedulable by the tests are in fact schedulable, and by "not necessary" we mean that not all systems deemed to be unschedulable by the tests are in fact unschedulable.

The schedulability tests given in this section are only applicable given the constraint that message deadlines do not exceed their periods.

The response time of the first instance of a message in the busy period is given by Eq. (7). Assuming that this first instance completes transmission before its deadline and hence before the end of its period, then we have two possibilities to consider.

(i) If the busy period ends before the next instance of message m is queued, then Eq. (7) gives the correct worst-case response time.
(ii) Alternatively, if the busy period continues beyond the time at which the next instance of message m is queued, then we must also consider the response time of the second and any subsequent instances of message m, queued before the end of the busy period.

First, we derive an upper bound on the maximum length of the interval $[s_{m,q}, s_{m,q+1})$, between the times, $s_{m,q}$ and $s_{m,q+1}$, at which two arbitrary but consecutive instances,

q and $q+1$, of message m start transmission. We then show that this upper bound is also an upper bound on the queuing delay for instance $q+1$, and can therefore be used as the basis for a sufficient schedulability test.

We assume that:

(i) all $q+1$ instances fall within the same busy period,
(ii) the first q instances are schedulable—we will return to this point later.

We observe that at time $s_{m,q}$, when instance q starts transmission, there can be no other messages currently being transmitted, and no messages of higher priority than m awaiting transmission. Thus, an upper bound on the length of the time interval $[s_{m,q}, s_{m,q+1})$ can be found by making the potentially pessimistic assumption that all higher priority messages are queued just as instance q starts transmission. The smallest solution to Eq. (15) provides an upper bound on the length of this interval.

$$w_m^{n+1} = C_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{\text{bit}}}{T_k} \right\rceil C_k \tag{15}$$

Here, C_m is the transmission time of the qth instance of message m, which is transmitted first in the interval. The summation term represents the interference from higher priority messages, released during the interval, and sent before the $(q+1)$th instance of message m can start to be transmitted.

Given the assumption that the first q instances in the busy period are schedulable, and the constraint that $D_m \leq T_m$, then the start (and end) of transmission of the qth instance must happen before the end of its period, and hence before the $(q+1)$th instance is queued. This means that the queuing delay for the $(q+1)$th instance, as measured from the time at which it is queued to the start of its transmission, is less than the length of the interval $[s_{m,q}, s_{m,q+1})$. The queuing delay for the $(q+1)$th instance is therefore also bounded[8] by the solution to Eq. (15).

We now return to the assumption that the first q instances are schedulable. Schedulability of the $q=0$ instance can be determined using Eq. (7); whilst the schedulability of the second and all subsequent instances within the busy period can be determined, by induction, using Eq. (15).

To determine an upper bound on the queuing delay, a suitable starting value, for use in using Eq. (15), is $w_m^0 = C_m$. The recurrence relation iterates until, either $J_m + w_m^{n+1} + C_m > D_m$ in which case message m is deemed unschedulable, or $w_m^{n+1} = w_m^n$ in which case the second and subsequent instances of message m are schedulable, with an upper bound on their response times of $J_m + w_m^{n+1} + C_m$.

Intuitively, we might say that the second and subsequence instances of message m in the busy period are subject to blocking, of at most C_m, due to the previous instance of the same message.

[8] We observe that the queuing delay of the $(q+1)$th instance is in fact at least C_m less than the bound given by Eq. (15). This is because, to be schedulable, instance q must start transmission at least C_m before the end of its period. As accurate analysis is available, presented in Section 3.2, we do not pursue this potential improvement in the sufficient analysis further.

This result suggests a simple sufficient but not necessary schedulability test, formed by combining Eqs. (7) and (15) into a single equation—Eq. (16).

$$w_m^{n+1} = \max(B_m, C_m) + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{\text{bit}}}{T_k} \right\rceil C_k \tag{16}$$

We observe that the schedulability analysis embodied in Eq. (16) equates to assuming that an instance of message m can be subject to blocking; either of B_m, due to non-pre-emptive transmission of lower priority messages; or of C_m, due to the non-pre-emptive transmission of the previous instance of message m itself.

A further simplification is to assume that the blocking factor always takes its maximum possible value. This leads to a further sufficient but not necessary schedulability test:

$$w_m^{n+1} = B^{\text{MAX}} + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{\text{bit}}}{T_k} \right\rceil C_k \tag{17}$$

where B^{MAX} corresponds to the transmission time of the longest possible CAN message (8 data bytes), irrespective of the characteristics and priorities of the messages in the system.[9]

3.5 Error model

So far we have assumed that no errors occur on the CAN bus; however, as originally shown by Tindell and Burns (1994) and Tindell et al. (1994b, 1995), schedulability analysis of CAN may be extended to include an appropriate error model.

In this paper we consider only a very simple and general error model. We assume that the maximum number of errors present on the bus in some time interval t is given by the function $F(t)$. We assume no specific details about this function; save that it is a monotonic non-decreasing function of t. For a more detailed discussion of appropriate error models for CAN, see Punnekkat et al. (2000) and Broster et al. (2002, 2005).

We now modify the schedulability equations to account for the error recovery overhead. The worst-case impact of a single bit error is to cause transmission of an additional 31 bits of error recovery overhead plus re-transmission of the affected message. Only errors affecting message m or higher priority messages can delay message m from being successfully transmitted. The maximum additional delay caused by the error recovery mechanism is therefore given by:

$$E_m(t) = \left(31\tau_{\text{bit}} + \max_{k \in hep(m)} (C_k)\right) F(t) \tag{18}$$

[9] Tindell et al. (1995) state that the *blocking time on CAN is defined as the longest time that a message can take to be physically transmitted on "the bus"*. This simplified view provides a sufficient but not necessary schedulability test that corresponds to Eq. (17). However, later in Tindell et al. (1995), the blocking term is described as "*the longest time that any lower priority message can occupy the bus"*. This description, also in Tindell and Burns (1994) and Tindell et al. (1994b), results in a flawed schedulability test.

Revising Eq. (8) to compute the length of the busy period we have:

$$t_m^{n+1} = E_m\left(t_m^n\right) + B_m + \sum_{\forall k \in hep(m)} \left\lceil \frac{t_m^n + J_k}{T_k} \right\rceil C_k \tag{19}$$

Again an appropriate initial value is $t_m^0 = C_m$. Eq. (19) is guaranteed to converge on a solution provided that the utilisation U_m, including error recovery overhead, is less than 1.

As before, Eq. (10) can be used to compute the number of message instances that need to be examined to find the worst-case response time.

$$w_m^{n+1}(q) = E_m\left(w_m^n + C_m\right) + B_m + qC_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{\text{bit}}}{T_k} \right\rceil C_k \tag{20}$$

Eq. (20) extends Eq. (11) to account for the error recovery overhead. Note that as errors can impact the transmission of message m itself, the time interval considered in calculating the error recovery overhead includes the transmission time of message m as well as its queuing delay. Eqs. (20), (12) and (13) can be used together to compute the response time of each message instance q, and hence find the worst-case response time of each message m in the presence of errors at the maximum rate specified by the error model.

The sufficient schedulability tests given in Section 3.4 can be similarly modified via the addition of the term $E_m(w_m^n + C_m)$ to account for the error recovery overhead.

4 Discussion

In this section we consider various characteristics of CAN systems and discuss whether flaws in the existing analysis can result in erroneous guarantees under specific circumstances that are relevant to real-world systems.

We seek to answer the following questions.

1. Can the existing analysis give faulty guarantees to messages of any priority?
2. If the bus utilization is low, can the existing analysis still result in optimistic response times?
3. Do error models give sufficient engineering margin for error to account for the flaw in the analysis?
4. Does the omission of diagnostic messages during normal operation reduce interference/blocking enough to ensure that the deadlines of the remaining messages will be met?
5. Which message guarantees can we be sure are not at risk?

4.1 Priorities of messages at risk

We have found that the existing analysis gives the correct worst-case response times for the highest priority and the 2nd highest priority message. However; it can compute

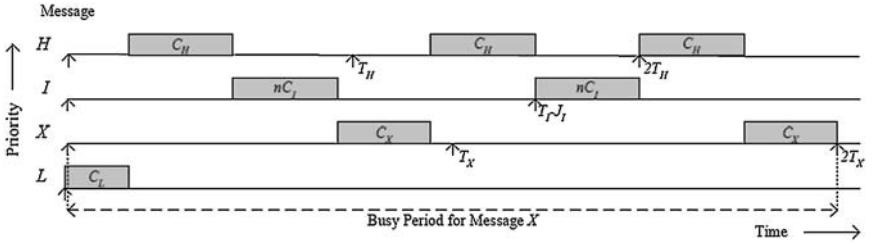


Fig. 7 Busy period for message X

incorrect worst-case response times for messages from the 3rd highest priority to the lowest priority.

This is illustrated by the example message set constructed below and depicted in Fig. 7. The example message set consists of;

(i) a high priority message H;
(ii) a group of n (where $n \geq 1$) intermediate priority messages, represented by I, which all have the same periods and transmission times;
(iii) a message X with a priority below those messages in group I,which highlights the flaw in the analysis and
(iv) a group of k (where $k \geq 0$) low priority messages represented by L, which all have the same transmission times.

The transmission times of the messages are C_H, C_I, C_X and C_L respectively, with the constraint that $C_X > C_L$.

The low priority messages L, are assumed to have very large periods and no jitter. These messages contribute only blocking to the response time of message X. (Note that if there are no lower priority messages, i.e. $k = 0$, then the example still holds with $C_L = 0$).

The period of message H is:

$$T_H = (C_L + 2C_H + 2nC_I + C_X)/2$$

The period of message X is:

$$T_X = (C_L + 3C_H + 2nC_I + 2C_X)/2$$

The period of the intermediate messages I, is assumed to be large ($T_I \gg 2T_X$); however the period less jitter for each intermediate message is:

$$T_I - J_I = C_L + 2C_H + nC_I + C_X$$

By contrast, messages H and X are assumed to have no jitter.

The busy period for message X is shown in Fig. 7. For simplicity, there is only one intermediate priority message shown in the diagram; however, the transmission time of this message is given as nC_I, representing the arbitrary number of intermediate messages that are considered.

We now show that under certain conditions message X exhibits the problem with the existing analysis. The length of the busy period for message X, given by Eq. (8), is:

$$t_X = C_L + 3C_H + 2nC_I + 2C_X = 2T_X$$

Hence, according to Eq. (10), there are two instances of message X in the busy period. We now compute the response times of these two instances. According to Eq. (11), and as $C_X > C_L$, the queuing delay of the first instance of message X is:

$$w_X(0) = C_L + C_H + nC_I$$

Similarly for the second instance:

$$w_X(1) = C_L + 3C_H + 2nC_I + C_X$$

According to Eq. (12), the response times of the two instances are:

$$R_X(0) = C_L + C_H + nC_I + C_X$$

and

$$R_X(1) = (C_L + 3C_H + 2nC_I + 2C_X)/2$$

Comparing the formulas for $R_X(0)$ and $R_X(1)$, then, provided that $C_H > C_L$, the response time of the second instance is greater than that of the first. Meaning that message X exposes the flaw in the existing analysis. (In fact, assuming that $D_X = T_X$, the second instance of message X is only just schedulable with $R_X = T_X$).

As we can choose an arbitrary number ($n \geq 1$) of intermediate priority messages, and similarly an arbitrary number ($k \geq 0$) of lower priority messages, message X may lie anywhere from the 3rd highest to the lowest priority in a set of messages with cardinality greater than or equal to 3. We conclude that any message from the lowest priority to the 3rd highest priority in a set of 3 or more messages can be given an optimistic response time and therefore a faulty guarantee by the existing analysis.

4.2 Breakdown utilisation

The example in Section 1.4 has a bus utilisation of 97%. It is interesting to ask if the existing analysis can yield optimistic worst-case response times for systems with much lower utilisation.

Returning to the example message set, constructed in Section 4.1, we now consider how low the utilisation of that message set can be.

To achieve the lowest possible utilisation, we need only consider the contribution from messages H and X; as the utilisation of both the intermediate messages I, and the low priority messages L, tends to zero when their periods are increased to an

Table 4 Utilisation of message sets breaking the existing analysis

Number of messages	Utilisation
3	45.5%
5	21.4%
10	9.2%
25	3.4%
100	0.82%

arbitrarily large value. We therefore have:

$$U = \frac{2C_X}{C_L + 3C_H + 2nC_I + 2C_X} + \frac{2C_H}{C_L + 2C_H + 2nC_I + C_X}$$

with the constraints that $C_H > C_L$and $C_X > C_L$.

The overall utilisation is minimised by choosing values of C_H and C_X as small as possible, and a value of C_I as large as possible. Given the constraints on CAN message sizes, the minimum occurs when we choose messages H and X to have zero data bytes, so $C_H = C_X = 55\tau_{\text{bit}}$, the intermediate messages to have 8 data bytes, so $C_I = 135\tau_{\text{bit}}$, and no lower priority messages, so $C_L = 0$.

We note that this message set is somewhat pathological, as all the intermediate priority messages have arbitrarily large periods/deadlines and correspondingly large queuing jitter. However, it does illustrate that in general the existing analysis breaks down at very low utilisation levels.

Table 4 provides an upper bound on this breakdown utilisation: the existing analysis is known to breakdown at these levels of utilisation, it may breakdown at still lower levels.

Whilst it is unlikely that real-world applications will have message configurations that replicate the pathological case discussed above, such systems may include messages with large amounts of queuing jitter. Typically these are 'gatewayed' messages that have inherited a large jitter from variability in the response time of a source message sent on another network. We conclude that, for applications characterised by non-zero queuing jitter, it is prudent to assume that there could be problems with the existing analysis, irrespective of overall bus utilisation.

In fact, for real-world CAN systems, characterised by messages with non-zero queuing jitter and consequently deadlines less than periods, overall bus utilisation is a poor indicator of system schedulability.

4.3 Margin for error

In Section 3.5 we saw how a generalised error model could be included in the revised schedulability analysis. Bit error rates on CAN are typically very low: 10^{-11} up to 10^{-6} depending on environmental conditions (Ferreira et al., 2004). However, errors do occur and it is therefore appropriate that any commercial application of CAN schedulability analysis should include at least a simple error model to account for sporadic errors on the bus. These errors are typically caused by external sources of Electromagnetic Interference (EMI) such as mobile phones, radar, radio transmitters, and lightning as well as other possible causes such as switch contacts, and shielding or wiring faults. As such errors are typically completely uncorrelated with message

transmission, it is reasonable to assume that any useful error model allows for the possibility of an error occurring at any given time, and hence the error function $F(t) \geq 1$ for a time interval of any length t.

Let us now consider the situation where the schedulability analysis given by Tindell and Burns (1994) and Tindell et al. (1994b, 1995) has been used along with an error model with $F(t) = 1$ to determine the schedulability of a system. The recurrence relation used by the existing analysis is given below:

$$w_m^{n+1} = B_m + E_m\left(w_m^n + C_m\right) + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{\text{bit}}}{T_k} \right\rceil C_k \tag{21}$$

Given that $F(t) \geq 1$, then from Eq. (18), the maximum additional delay to message m due to the error recovery mechanism is always longer than the transmission time of message m, i.e. $E_m(t) > C_m$. Substituting C_m for $E_m(t)$ in Eq. (21) gives:

$$w_m^{n+1} = B_m + C_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{\text{bit}}}{T_k} \right\rceil C_k \tag{22}$$

We note that as $E_m(t) > C_m$, the solution to Eq. (22) cannot be larger than the solution to Eq. (21).

Recall that Eq. (16) provides a correct sufficient but not necessary schedulability test for the case where there are no errors on the CAN bus. Comparing Eqs. (22) and (16), we observe that, as $\max(B_m, C_m) \leq B_m + C_m$, the solution to Eq. (16) cannot be larger than the solution to Eq. (22) and hence cannot be larger than the solution to Eq. (21). This means that if message m is deemed to be schedulable given the queuing delay computed by Eq. (21) for the case where there are errors on the bus, then it must also be schedulable given the queuing delay computed via Eq. (16) for the case where there are no errors on the bus.

This is an important result. It means that if the existing analysis showed that every message was schedulable in the presence of any reasonable error model, with $F(t) \geq 1$, then, despite the flaw in the existing analysis, every message is actually guaranteed to be schedulable when no errors are present. Put another way, the engineering margin for error provided by the error model is sufficient to account for the error in the analysis.

We observe however, that the robustness of systems analysed using the schedulability analysis given by Tindell and Burns (1994), and Tindell et al. (1994b, 1995) may not be all that was expected. Flaws in the existing analysis could lead to message configurations that will miss their deadlines in the presence of errors at a rate within the parameters of the specified error model, even though we can be sure that they will not miss their deadlines when no errors are present on the bus.

4.4 Message omission

Many CAN applications allow for 8 data byte diagnostic messages that are not transmitted during the normal mode of operation. These messages are transmitted only when the system is in diagnostic mode[10] and linked to service equipment. In this section we consider whether the omission of diagnostic messages provides sufficient

[10] Typically, all normal mode messages continue to be transmitted during diagnostic mode.

reduction in interference/blocking to ensure that messages do not miss their deadlines during normal operation, despite being given potentially optimistic worst-case response times by the existing analysis.

To answer this question, we consider a system that is deemed to be schedulable by the existing analysis. We assume that this system includes an 8 data byte diagnostics message Y, which is only transmitted when the system is in diagnostic mode. We note that as message Y has the maximum number of data bytes, its transmission time is equivalent to the largest possible blocking factor, so $C_Y = B^{\text{MAX}}$. The blocking factor for each message m of higher priority than Y, is therefore given by $B_m = B^{\text{MAX}}$, which means that the existing analysis based on Eq. (7) computes exactly the same worst-case response time for each higher priority message m, as the correct sufficient but not necessary schedulability analysis test based on Eq. (17). The existing analysis cannot therefore result in optimistic worst-case response times for messages of higher priority than Y.

For each message of lower priority than Y, the interference due to message Y is at least B^{MAX}. Comparing Eq. (7) and (17), we observe that the solution to Eq. (7), with diagnostic message Y included in the set of higher priority messages, is at least as large as the solution to Eq. (17) when message Y is excluded. This means that if a lower priority message m is deemed to be schedulable by the existing analysis when message Y is present, then it must also be schedulable according to the correct sufficient but not necessary schedulability analysis when message Y is omitted.

We conclude that the omission of a single maximum length message of arbitrary priority provides sufficient reduction in interference/blocking to ensure that the flaw in the existing analysis cannot lead to any of the remaining messages missing their deadlines.

4.5 Message guarantees not at risk

In this section we consider the circumstances under which the first instance of a message in the busy period is guaranteed to have the longest response time. Under these circumstances, despite its flaws, the existing analysis gives correct results.

Assuming that message deadlines do not exceed their periods, then Eq. (16) in Section 3.4 provides an upper bound on the queuing delay for the second and subsequent instances of message m in the busy period. Comparing Eqs. (7) and (16), we observe that if $B_m \geq C_m$, then the first instance of message m is guaranteed to have a response time at least as long as subsequent ones. From the definition of B_m given in Eq. (5), we conclude the following important result: the existing analysis gives the correct response time for any message where there exists at least one lower priority message with equal or longer transmission time/message length.

5 Priority assignment policies

The analysis presented in Section 3 is applicable irrespective of the priority ordering of CAN messages. However, choosing an appropriate priority ordering is important in obtaining a schedulable system, and in maximising robustness to errors.

Table 5 CAN messages highlighting non-optimal priority assignment

Message	Period	Deadline	Number of bits	TX time
A	3.0 ms	3.0 ms	135	1.08 ms
B	4.0 ms	4.0 ms	135	1.08 ms
C	4.5 ms	4.5 ms	65	0.52 ms

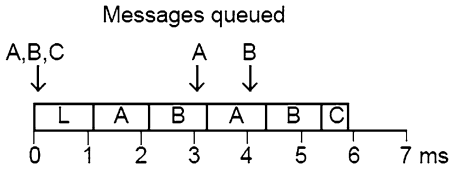


Fig. 8 Message response times with "optimal" priority assignment

Priority ordering is determined by a priority assignment policy. A priority assignment policy P is referred to as *optimal* if there are no systems that are schedulable using any other priority assignment policy that are not also schedulable using policy P.

Tindell and Burns (1994) and Tindell et al. (1995) claimed that *deadline monotonic* priority assignment (Leung and Whitehead, 1982) and "deadline minus jitter" or *(D-J)-monotonic* priority assignment (Zuhily, 2006) was optimal for CAN; however, whilst these policies are optimal for fixed priority pre-emptive scheduling, assuming deadlines no greater than periods, they are not optimal for fixed priority non-pre-emptive scheduling (George et al., 1996), and are therefore not optimal for CAN. This is illustrated by the following example using the set of messages given in Table 5.

This example assumes a 125 Kbit/s network and 11-bit identifiers. Messages A and B contain 8 data bytes and message C contains 1 data byte, giving transmission times of 1.08, 1.08 and 0.52 ms respectively, assuming worst-case bit stuffing. In addition, there are a number of lower priority messages, each containing 8 data bytes, which are also sent on the network; their transmission times are also 1.08 ms.

Setting message priorities in the order A—highest, then B, then C results in an unschedulable system. The worst-case response times of messages A and B are 2.16 ms and 3.24 ms respectively; however, in the worst case, message C does not even begin transmission before its deadline.

Figure 8 illustrates the long delays that message C is subject to before transmission. Messages A, B and C are assumed to be queued just too late to enter into arbitration at time $t = 0$ and hence the low priority message L is transmitted first.

The priority ordering A, B, C corresponds to both deadline monotonic and also (D-J)-monotonic priority ordering—as all the messages have zero queuing jitter. If these priority assignment policies are optimal then we should not be able to find another priority ordering which results in all of the deadlines being met; however, if we use the priority ordering A, C, B then the worst-case response times of the messages are: $R_A = 2.16$ ms, $R_C = 2.68$ ms and $R_B = 3.76$ ms, as illustrated in Fig. 9. With this priority ordering, all of the messages meet their deadlines.

The reason that the revised priority ordering results in a schedulable system is that giving the shortest message a higher priority enables all three messages to start transmission within 3 ms of being queued; hence none of them are subject to interference from a second instance of message A and subsequently a second instance of message

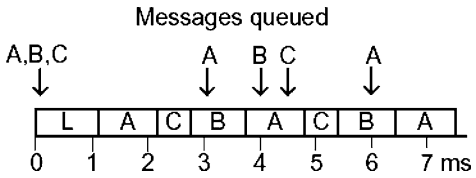


Fig. 9 Message response times with an alternative priority assignment

B. This example shows that the priority assignment policies assumed by Tindell and Burns (1994) and Tindell et al. (1995) to be optimal are not.

George et al. (1996) claimed that deadline monotonic priority assignment is optimal for non-pre-emptive systems with no jitter, provided that deadlines and execution times are in the same order i.e. $D_i < D_j$ implies $C_i \leq C_j$. The proof, given by George et al. (1996), assumes that *"as* $\forall i$, $D_i \leq T_i$ *the worst-case response time of any task is found in its first instance"*; however, this assumption is false, as we have seen with the simple example in Section 1.4, and so the proof is undermined. The theorem may or may not still be true.

George et al. (1996) also showed that the optimal priority assignment algorithm devised by Audsley (1991) is applicable to non-pre-emptive systems. In general, Audsley's algorithm is applicable provided that the worst-case response time of a message:

(i) does not depend upon the specific priority ordering of higher priority messages and,
(ii) does not get longer if the message is given a higher priority.

Inspection of the various equations presented in this paper shows that both of the above conditions hold. Neither the length of the queuing delay, nor the length of the busy period depends upon the specific priority order of higher priority messages. Similarly, although the blocking term can get larger with increased priority this is always counteracted by a decrease in interference that is at least as large; hence the length of the busy period and the length of the queuing delay cannot increase with increasing message priority. The optimal priority ordering of CAN messages can therefore be determined using Audsley's priority assignment algorithm, given below.

```
Optimal Priority Assignment Algorithm

for each priority level, lowest first
{
   for each unassigned message m
   {
       if m is schedulable at this priority
       {
           assign m this priority
           break (continue outer loop)
       }
   }
    return unschedulable
}
return schedulable
```

For n messages, Audsley's algorithm performs at most $n(n-1)/2$ schedulability tests and is guaranteed to find a schedulable priority assignment if one exists. It does not however specify an order in which messages should be tried at each priority level. This order heavily influences the priority assignment chosen if there is more than one ordering that is schedulable. In fact, a poor choice of initial ordering can result in a priority assignment that leaves the system only just schedulable. We therefore suggest that, as a useful heuristic, messages are tried at each priority level in (D-J) order, largest value of (D-J) first, with ties broken according to message length, longest first.

6 Implications and recommendations

In this section we discuss the implications of flaws in existing CAN schedulability analysis on commercial CAN schedulability analysis tools and deployed CAN applications.

6.1 CAN schedulability analysis tools

CAN schedulability analysis tools need to take account of the findings presented in this paper. This will involve checking, and if necessary updating, the analysis they employ; ensuring that it cannot provide optimistic worst-case response times and false guarantees.

The sufficient but not necessary schedulability tests given in Section 3.4 provide a "quick-fix" solution with minimal changes required to the existing analysis. These tests are however pessimistic and implementing the revised analysis given in Section 3 would potentially lead to a better technical solution.

Whilst "deadline minus jitter" or (D-J)-monotonic priority ordering is still a good heuristic to use, it is not necessarily the optimal priority assignment policy for CAN. Implementing priority ordering based upon Audsley's optimal priority assignment algorithm would ensure that a schedulable priority ordering is found whenever one exists.

6.2 Commercial CAN applications

System Designers configuring commercial CAN applications often take the engineering approach that all messages in the system should remain schedulable given the addition of any number of low priority messages that can be used for development and test purposes. Such analysis based on Tindell and Burns (1994) and Tindell et al. (1994b, 1995) would assume that every message is subject to the maximum blocking factor, as per the sufficient schedulability test given by Eq. (17). This schedulability test computes a correct upper bound on the actual response time of each message, and so provides a correct guarantee that the configured messages will meet their deadlines.

Given the flaws in the existing schedulability analysis, it would however be prudent for System Designers to check the precise details of the analysis used to compute worst-case response times for their systems. If the analysis used has the potential to compute

erroneous worst-case response times, then the feasibility of all the CAN configurations designed, developed and deployed using that analysis should be checked to ensure that they are in fact schedulable, and robust to errors at the rate specified by the prescribed error model.

6.3 Faults in deployed systems

Many deployed CAN systems, for example those in automotive applications, will have been analysed using the pragmatic engineering approach described in the previous section. The flaws in the existing analysis cannot lead to a problem with a deployed system in this case.

Many CAN applications allow for maximum length (8 data byte) diagnostic messages that are not transmitted during normal operation. Assuming that the existing analysis deemed the deployed system to be schedulable with these diagnostic messages present, then Section 4.4 showed that the omission of a single diagnostic message provides sufficient reduction in interference/blocking to ensure that the flaws in the existing analysis cannot lead to other messages missing their deadlines during normal operation.

In Section 4.5 we saw that the existing analysis gives the correct response time for any message where there is at least one lower priority message with equal or longer transmission time/message length. Many CAN applications use exclusively 8 data byte messages as a means of addressing the high ratio of overhead to useful data on CAN. In this case, the existing analysis is guaranteed to compute correct response times for all but the lowest priority message.

Even if a message has the potential to be given an erroneous worst-case response time by the existing analysis, then, unless that message is close to being unschedulable, the computed worst-case response time is still likely to be the true value. Even if an optimistic value is computed, then the true value may still be less than the message deadline. Finally, for a deadline miss to actually happen in a deployed system requires that the worst-case message phasing occurs, and at that point a number of messages take close to their maximum transmission times. This requires worst-case or near worst-case bit stuffing to occur which is, in itself, highly unlikely (Nolte et al., 2002).

Normal practice with commercial CAN configurations is to ensure that the schedulability analysis used includes provision for a plausible error model. In this case, Section 4.3 showed that such systems are guaranteed to be schedulable when no errors are present on the CAN bus provided that they were deemed to be schedulable in the presence of errors by the existing analysis.

We conclude that deadline misses in deployed CAN systems due to flaws in the existing analysis are extremely unlikely. Any such deadline failures are more likely to occur due to errors occurring on the bus at a higher rate than that accounted for by the error model.

We note that embedded CAN-based systems are built to be resilient to some messages missing their deadlines, and to much simpler forms of error such as wiring faults. CAN is not used, in its basic form, for safety critical systems due to known issues such as the “double receive” and “babbling idiot” problems (Rufino et al., 1998; Broster and Burns, 2003; Rufino, 2002).

7 Summary and conclusions

In this paper we highlighted a significant flaw in long-standing, highly cited, and widely used schedulability analysis of CAN. We showed how this flaw could lead to the computation of optimistic worst-case response times for CAN messages, broken guarantees, and deadline misses. This paper provides revised analysis that can be used to calculate correct worst-case response times for CAN.

In addition, we showed that:

1. The existing analysis can provide optimistic worst-case response times for messages from the 3rd highest priority to the lowest priority.
2. The existing analysis can lead to broken guarantees and hence deadline misses in systems with low bus utilisation.
3. Where an error model has been considered, the flaw in the existing analysis is not sufficient to lead to CAN configurations that will result in missed deadlines when no errors are present on the bus. The desired robustness to errors may not however be achieved.
4. The omission of a single maximum length diagnostic message, accounted for by the existing analysis, reduces interference/blocking enough to ensure that the deadlines of all the remaining messages are met during normal operation.
5. Despite its flaws, the existing analysis gives the correct response time for any message where there is at least one lower priority message with the same or longer transmission time/message length.

We discussed the implications of these results for commercial CAN systems developed using flawed analysis and provided two simple, sufficient schedulability tests enabling a "quick-fix" to be made to commercial CAN schedulability analysis tools.

Finally, we showed that neither deadline monotonic nor (D-J)-monotonic priority assignment is optimal for CAN. Audsley's priority assignment algorithm is however optimal for fixed priority non-pre-emptive systems and can be used to obtain a schedulable priority ordering for CAN whenever one exists.

7.1 Future work

A considerable body of academic work has grown up from Tindell's seminal analysis of CAN. The flaws in that original work may have partly undermined some of the subsequent research built upon it. Authors that have cited the original CAN analysis in their work are therefore encouraged to check the implications. In particular the academic work most likely to be affected is that which extends the original analysis and pushes system schedulability to its limits, for example work on error models.

7.2 Postscript

Volcano Network Architect (VNA) is a commercial CAN design and analysis tool originally developed by Volcano Communications Technologies AB, and now owned by Mentor Graphics Corporation. By 2004, over 20 million cars, with an average of 20 ECUs per car, had been programmed using this technology (Oswald, 2004).

The engineering team responsible for Volcano Network Architect was given early visibility of this paper, enabling them to check the validity of the schedulability analysis used in their commercial products.

The analysis provided by Volcano Network Architect was found to be sufficient: it assumes the longest possible blocking time irrespective of message priority (Horvath, 2006) and therefore computes an upper bound on response times, similar to that given by the sufficient schedulability tests described in Section 3.4.

Acknowledgments This work was partially funded by the UK EPSRC funded DIRC project, the EU funded FRESCOR project and the IST-004527 funded ARTIST 2 network of excellence on Embedded Systems Design.

References

Audsley NC (1991) Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS 164, Dept. Computer Science, University of York, UK

Bosch (1991) CAN Specification version 2.0. Robert Bosch GmbH, Postfach 30 02 40, D-70442 Stuttgart

Bril RJ (2006) Existing worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred pre-emption is too optimistic. CS-Report 06-05, Technische Universiteit Eindhoven (TU/e), The Netherlands

Bril RJ, Lukkien JJ, Davis RI, and Burns A (2006a) Message response time analysis for ideal Controller Area Network (CAN) refuted. CS-Report 06-19, Technische Universiteit Eindhoven (TU/e), The Netherlands.

Bril RJ, Lukkien JJ, Davis RI, Burns A (2006b) Message response time analysis for ideal Controller Area Network (CAN) refuted. In: Proceedings of the 5th International Workshop on Real-Time Networks (RTN'06)

Bril RJ, Lukkien JJ, Verhaegh WFJ (2006c) Worst-case response time analysis of real-time tasks under fixed priority scheduling with deferred preemption revisited. CS Report 06-34, Technische Universiteit Eindhoven (TU/e), The Netherlands

Broster I (2003) Flexibility in dependable communication. PhD Thesis, Department of Computer Science, University of York, UK

Broster I, Burns A, Rodríguez-Navas G (2002) Probabilistic analysis of CAN with Faults. In: Proceedings of the 23rd IEEE real-time systems symposium (RTSS'02), pp 269–278

Broster I, Burns A (2003) An analysable bus-guardian for event-triggered communication. In: Proceedings of the 24th real-time systems symposium. IEEE Computer Society Press, pp 410–419

Broster I, Burns A, Rodriguez-Navas G (2005) Timing analysis of real-time communication under electromagnetic interference. Real-Time Systems 30(1–2):55–81

Burns A (1994) Pre-emptive priority based scheduling: An appropriate engineering approach. In: S. Son (ed), Advances in Real-Time Systems, Prentice-Hall, pp 225–248

Casparsson L, Rajnak A, Tindell K, Malmberg P (1998/1) Volcano—a revolution in on-board communications. Volvo Technology Report

DeMeis R (2005) Cars sag under weighty wiring. Electronic Times, 10/24/2005

Ferreira J, Oliveira A, Fonseca P, Fonseca JA (2004) An experiment to assess bit error rate in CAN. In: Proceedings of 3rd international workshop of real-time networks (RTN2004). Cantania, Italy, pp 15–18

Frischkorn H-G (2005) Automotive architecture requirements. In: Proceedings of the summer school on architectural paradigms for dependable embedded systems. Vienna, Austria,. Vienna University of Technology, pp 45–74

George L, Rivierre N, Spuri M (1996) Pre-emptive and non-pre-emptive real-time uni-processor scheduling. Technical Report 2966, Institut National de Recherche et Informatique et en Automatique (INRIA), France

Hansson H, Nolte T, Norstrom C, Punnekkat S (2002) Integrating reliability and timing analysis of CAN-based Systems. IEEE Transactions on Industrial Electronics 49(6):1240–1250

Harbour MG, Klein MH, Lehoczky JP (1991) Fixed priority scheduling of periodic tasks with varying execution priority. In: Proceedings 12th IEEE real-time systems symposium. IEEE Computer Society Press, pp 116–128

Horvath I (2006) Private communication with the authors
ISO 11898-1 (1993) Road Vehicles—interchange of digital information—Controller Area Network (CAN) for high-speed communication. ISO Standard-11898, International Standards Organisation (ISO)
Lehoczky J (1990) Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: Proceedings 11th IEEE real-time systems symposium. IEEE Computer Society Press, pp 201–209
Leohold J (2004) Communication requirements for automotive systems. Keynote speech 5th IEEE international workshop on factory communication systems, Vienna, Austria, Vienna University of Technology
Leohold J (2005) Automotive system architecture. In: Proceedings of the summer school on architectural paradigms for dependable embedded systems. Vienna, Austria, Vienna University of Technology, pp 545–591
Leung JY-T, Whitehead J (1982) On the complexity of fixed-priority scheduling of periodic real-time tasks. Performance Evaluation 2(4):237–250
LIN Consortium (2003) LIN Protocol Specification, Revision 2.0. www.lin-subbus.org
Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM 20(1):46–61
Motorola Inc (1998) MSCAN Block Guide V03.01 Document No. SV12MSCANV3/D. FreeScale Semiconductor Inc. (Revised July 2004)
Nolte T, Hansson H, Norstrom C (2002) Minimizing CAN response-time analysis jitter by message manipulation. In: Proceedings 8th IEEE real-time and embedded technology and applications symposium (RTAS'02), pp 197–206
Nolte T, Hansson H, Norstrom C (2003) Probabilistic worst-case response-time analysis for the Controller Area Network. In: Proceedings of the 9th IEEE real-time and embedded technology and applications symposium (RTAS'03), pp 200–207
Nolte T (2006) Share-driven scheduling of embedded networks. PhD Thesis, Malardalen University Press
Oswald M (2004) Efficient automotive electronics. Automotive Engineer
Punnekkat S, Hansson H, Norstrom C (2000) Response time analysis under errors for CAN. In: Proceedings 6th real-time technology and applications symposium. IEEE Computer Society Press, pp 258–265
Regehr J (2002) Scheduling tasks with mixed pre-emption relations for robustness to timing faults. In: Proceedings 23rd real-time systems symposium. IEEE Computer Society Press, pp 315–326
Rufino J, Verissimo P, Arroz G, Almeida C Rodrigues L (1998) Fault-tolerant broadcasts in CAN. In Digest of Papers, The 28th IEEE international symposium on fault-tolerant computing (FTCS'98), pp 150–159
Rufino J (2002) Computational system for real-time distributed control. PhD-Thesis, Technical University of Lisbon, Instituto Superior
Society of Automotive Engineers (1993) Class C application requirement considerations, recommended practice, SAE Technical Report J2056/1
Tindell KW, Burns A (1994) Guaranteeing message latencies on Controller Area Network (CAN). In: Proceedings of 1st international CAN conference, pp 1–11
Tindell KW, Burns A, Wellings AJ (1994a) An extendible approach for analysing fixed priority hard real-time systems. Journal of Real-Time Systems 6(2):133–152
Tindell KW, Hansson H, Wellings AJ (1994b) Analysing real-time communications: Controller Area Network (CAN). In: Proceedings 15th real-time systems symposium (RTSS'94). IEEE Computer Society Press, pp 259–263
Tindell KW, Burns A, Wellings AJ (1995) Calculating Controller area network (CAN) message response times. Control Engineering Practice 3(8):1163–1169
Wang Y, Saksena M (1999) Scheduling fixed priority tasks with pre-emption threshold. In: Proceedings of the 6th international workshop on real-time computing systems and applications (RTCSA'99), pp 328–335
Zuhily A (2006) Optimality of (D-J)-monotonic priority assignment. Technical Report YCS404. Dept. of Computer Science, University of York, UK

Robert I. Davis received a DPhil in Computer Science from the University of York in 1995. Since then he has founded three start-up companies, all of which have succeeded in transferring real-time systems research into commercial product. At Northern Real-Time Technologies Ltd. (1995–1997) he was responsible for development of the Volcano CAN software library. At LiveDevices Ltd. (1997–2001) he was responsible for development of the Real-Time Architect suite of products, including an OSEK RTOS and schedulability analysis tools. In 2002, Robert returned to the University of York, and in 2004 he was involved in setting up a spin out company, Rapita Systems Ltd., aimed at transferring worst-case execution time analysis technology into industry. Robert is a member of the Real-Time Systems Research Group at the University of York, and a director of Rapita Systems Ltd. His research interests include scheduling algorithms and schedulability analysis for real-time systems.

Alan Burns is head of the Real-Time Systems Research Group at the University of York. His research interests cover a number of aspects of real-time systems including the assessment of languages for use in the real-time domain, distributed operating systems, the formal specification of scheduling algorithms and implementation strategies, and the design of dependable user interfaces to real-time applications. He has authored/co-authored over 370 papers and 10 books, with a large proportion of them concentrating on real-time systems and the Ada programming language. Professor Burns has been actively involved in the creation of the Ravenscar Profile, a subset of Ada's tasking model, designed to enable the analysis of real-time programs and their timing properties.

Reinder J. Bril received a B.Sc. and an M.Sc. (both with honours) from the University of Twente, and a Ph.D. from the Technische Universiteit Eindhoven, the Netherlands. He started his professional career in January 1984 at the Delft University of Technology. From May 1985 until August 2004, he was with Philips, and worked in both Philips Research as well as Philips' Business Units. He worked on various topics, including fault tolerance, formal specifications, software architecture analysis, and dynamic resource management, and in different application domains, e.g. high-volume electronics consumer products and (low volume) professional systems. In September 2004, he made a transfer back to the academic world, to the System Architecture and Networking (SAN) group of the Mathematics and Computer Science department of the Technische Universiteit Eindhoven. His main research interests are currently in the area of reservation-based resource management for networked embedded systems with real-time constraints.

Johan J. Lukkien has been head of the System Architecture and Networking Research group at Eindhoven University of Technology since 2002. He received an M.Sc. and a Ph.D. from Groningen University in the Netherlands. In 1991, he joined Eindhoven University, after two years leave at the California Institute of Technology. His research interests include the design and performance analysis of parallel and distributed systems. Until 2000 he was involved in large-scale simulations in physics and chemistry. Since 2000, his research focus has shifted to the application domain of networked resource-constrained embedded systems. Contributions of the SAN group are in the area of component-based middleware for resource-constrained devices, distributed co-ordination, Quality of Service in networked systems and schedulability analysis in real-time systems.

Real-time networks

Slides source: Prof. Luis Almeida

Outline

- Real-Time Networks
- Collapsed OSI Model
- Real-Time Network Examples
- Networked Real-Time Systems

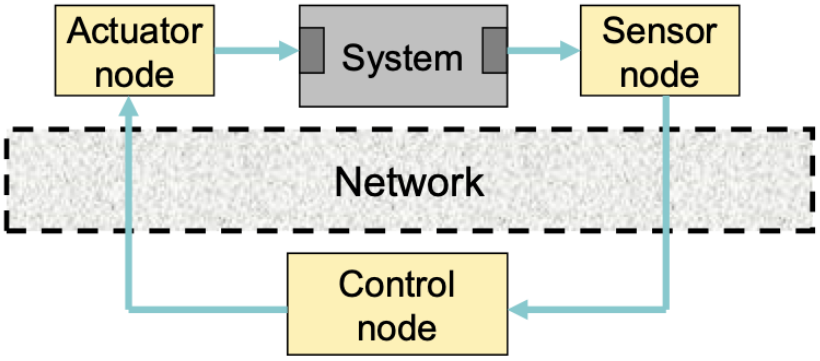
Background

Distributed architectures are common in many applications:

- Industrial automation
- Transportation systems (planes, cars, trucks, trains, …)
- Multimedia systems (surveillance, monitoring, video on demand, …)

In many cases with **critical timeliness** and **safety** requirements

Control loops are often closed over networks (= **networked control**)



Background

Motivations for distributed architectures:

- Processing closer to the data source/sink
  - "Intelligent" sensors and actuators
- Dependability
  - Error-containment within nodes
- Composability
  - System composition by integrating subsystems
- Scalability
  - Easy addition of new nodes with new or replicated functionality
- Maintainability
  - Modularity and simple node replacement
  - Simplification of the cabling

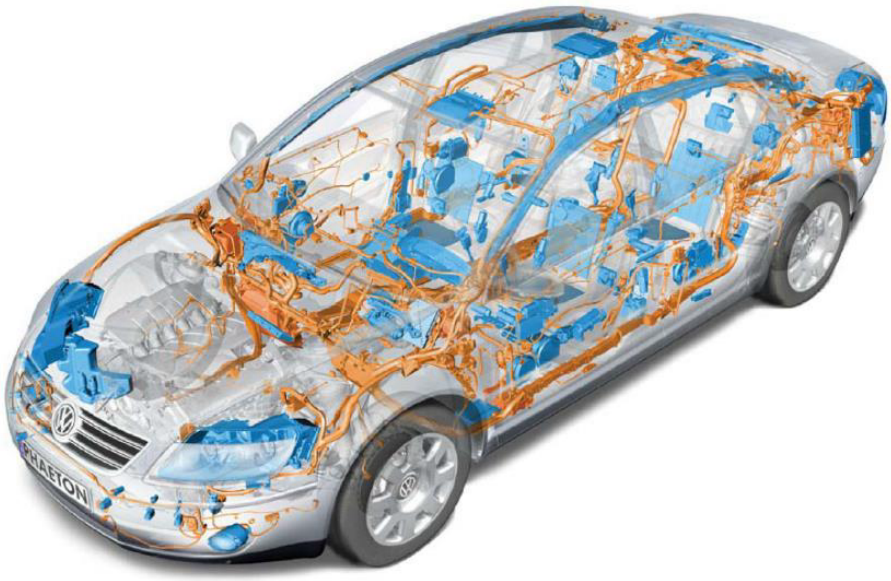
Background

Different networks with real-time capabilities are aimed at different application domains, e.g.

- ATINC629, SwiftNet, SAFEbus (avionics)
- WorldFIP, TCN (trains)
- CAN, TT-CAN, FlexRay, MOST (cars)
- ProfiBus, WorldFIP, P-Net, DeviceNet (automation)
- Firewire, USB (multimedia, general purpose)

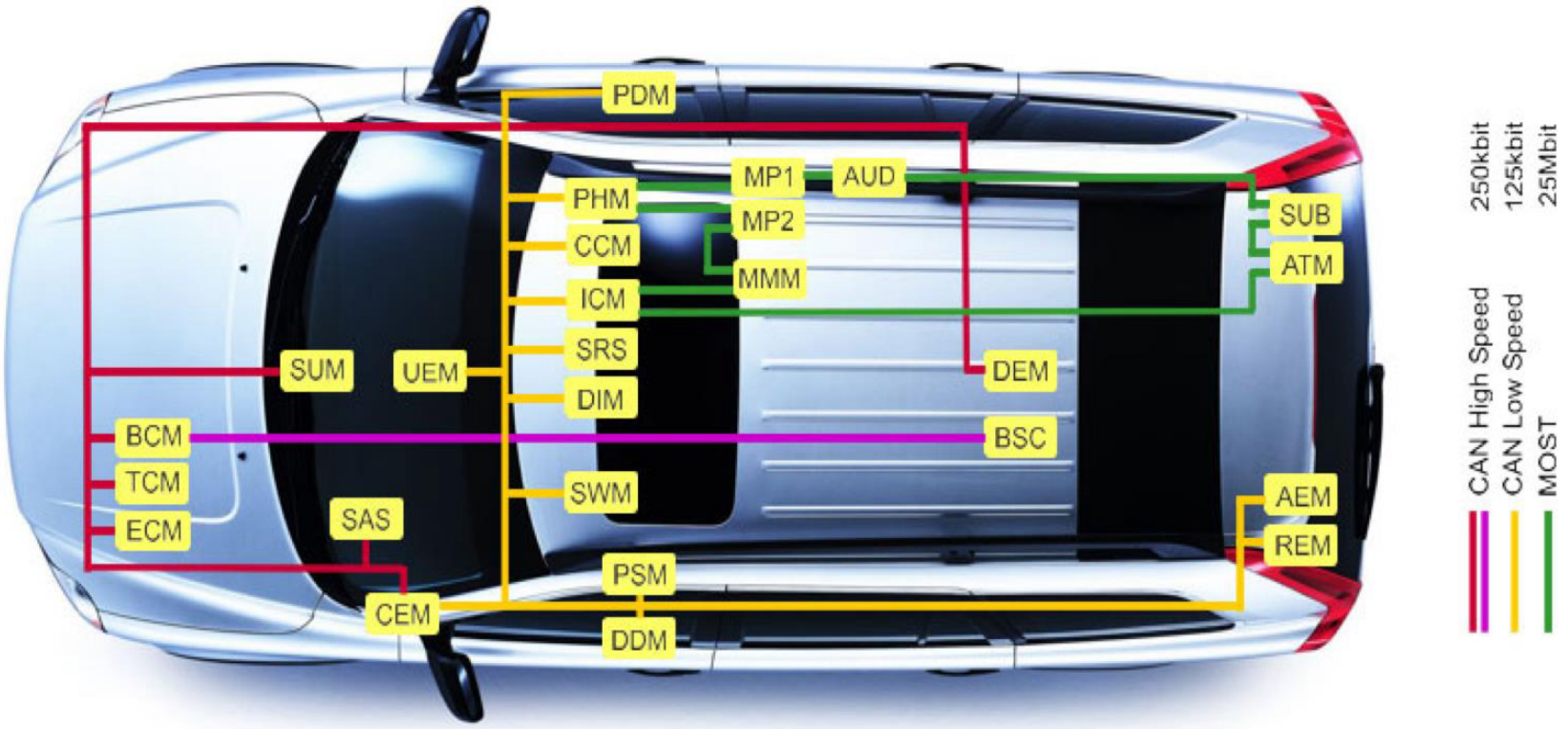
Example: VW Phaeton

- 11,136 electrical parts
- Total 61 ECUs (Electronic Controller Units = CPUs)
- 35 ECUs connected by 3 CAN buses sharing
  - 2500 signals
  - in 250 CAN messages
- Optical bus for high bandwidth infotainment data



The VW Phaeton
Adapted from (Loehold, WFCS2004)

Example: Volvo XC 90



Requirements

Typical requirements in real-time networks:

- Efficient transmission of **short data** (few bytes)
- **Periodic** transmission (control, monitoring) with **short periods** (ms), **low latency**, and **small jitter**
- Fast transmission (ms) of **aperiodic requests** (alarms, commands, …)
- Transmission of **non-real-time data** (configuration information, log data, …)
- Multicasting as well as unicasting (peer to peer)

Messages and Packets

- A **message** is a **unit of information** that should be transferred at a given time from a sender to one or more receivers
- Contains both **data** and **control information** that is relevant for the proper transmission of the data (e.g., sender, destination, checksum, …)
- Some networks automatically break large messages into smaller packets (**fragmentation/reassembly**)
- A **packet** is the smallest unit of information that is transmitted

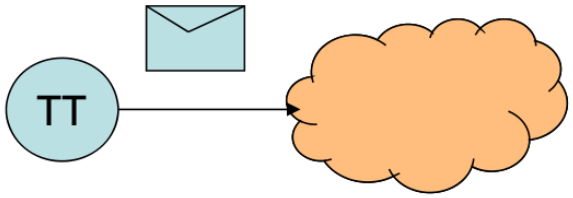
Real-Time Messages

- A real-time message is associated with a **deadline**
  - Soft or hard
- Real-time messages can have **event** or **state semantics:**
  - **Events** are perceived changes in the system state.
    All events are significant for the state consistency across sender and receiver.
  - **Event messages** must be queued at the receiver and removed upon reading. **Correct order** in delivery must be enforced.
  - **State messages** (containing state data) can be read many times and overwrite the values of the previous message concerning the same real-time entity.

State vs Event Semantics – Example

State message

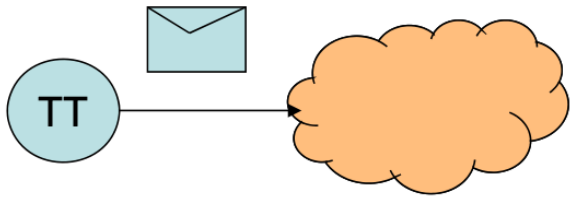
Temperature is 15°C



Temperature
sensor node

Event message

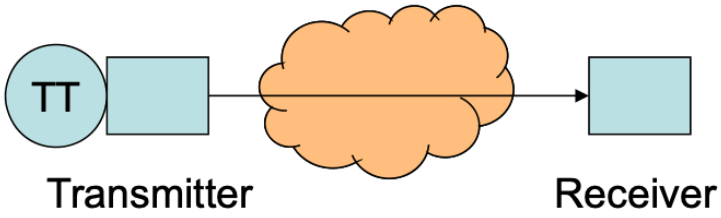
Temperature raised by 2°C

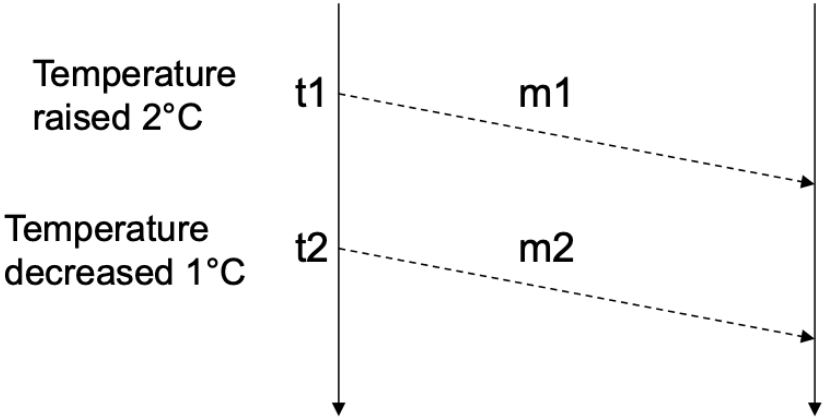


Temperature
sensor node

Event-Triggered Networks

- Messages are sent asynchronously, upon **events**
- Often, **event semantics** are used for the messages



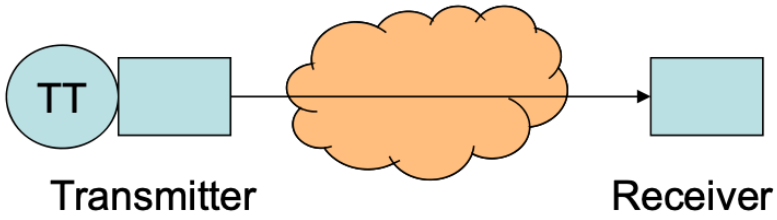


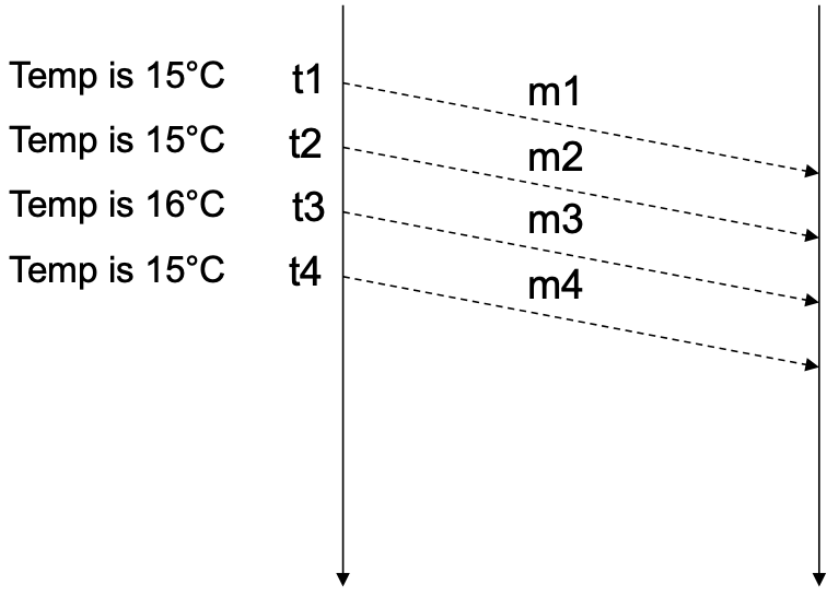
What if m1 is lost?

Time-Triggered Networks

- In time-triggered networks, there is a notion of **network time**
  - All clocks are globally synchronized
- Messages are sent at **predefined time instants,** typically using **state semantics**
  - Receivers have a **periodic refresh** of the system state
- The network load is predetermined

Time-Triggered Network





Losing a message gives inconsistency only until next message arrives

Event vs Time Triggered Networks

Time-triggered networks:

- More deterministic
  - All transmission instants are predefined
  - Fault-tolerance mechanisms are easier to design
- Less flexible in reacting to errors
  - Retransmissions often not possible because the schedule is fixed
  - A lost message is not recovered until the next period of the message stream
- Less flexible with respect to changes
  - Everything must be known at design time and very little can be changed dynamically (cp. static cyclic CPU scheduling)

Event vs Time Triggered Networks

Event-triggered networks:

- Lower level of determinism
  - Events can occur at any time (flexible)
  - MAC protocol may be needed
  - Harder to give hard real-time guarantees
- More complex fault-tolerance schemes
- More flexible with respect to transmission errors
  - Retransmissions can be carried out immediately

Example analysis: CAN

- Controller Area Network (CAN)
  - Created in the early 90s by Bosch, GmbH, for use in the automotive industry.
- Asynchronous bus access (CSMA)
  - Carrier-Sense Multiple Access (CSMA)
    - Set of protocols based on sensing bus inactivity before transmitting (asynchronous bus access)
    - There may be collisions
    - Upon collision, the nodes back off and retry later, according to some specific rule

CAN History

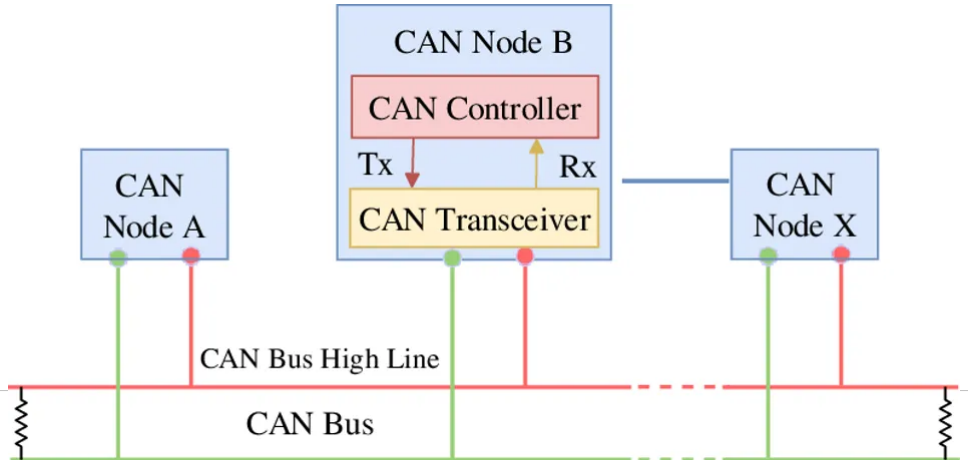
- CAN is used in nearly all cars sold today
  - Approx. 1 billion CAN enabled microcontrollers sold each year
  - Typical cars today have 20 – 30 ECUs inter-connected via 2 or more CAN buses
- Multiple networks
  - High speed" (500 Kbit/sec) network connecting chassis and power train ECUs
    - E.g. transmission control, engine management, ABS etc.
  - Low speed (100-125 Kbit/sec) network(s) connecting body and comfort electronics
    - E.g. door modules, seat modules, climate control etc.
  - Data required by ECUs on different networks
    - typically "gatewayed" between them via a powerful microprocessor connected to both

CAN use in vehicles

- CAN used to communicate *signals* between ECUs
  - Signals typically range from 1 to 16-bits of information
  - wheel speeds, oil and water temperature, battery voltage, engine rpm, gear selection, accelerator position, dashboard switch positions, climate control settings, window switch positions, fault codes, diagnostic information etc.
  - > 2,500 signals in a high-end vehicle
  - Multiple signals piggybacked into CAN messages to reduce overhead, but still 100's of CAN messages
- Real-time constraints on signal transmission
  - End-to-end deadlines in the range 10ms – 1sec
  - Example LED brake lights

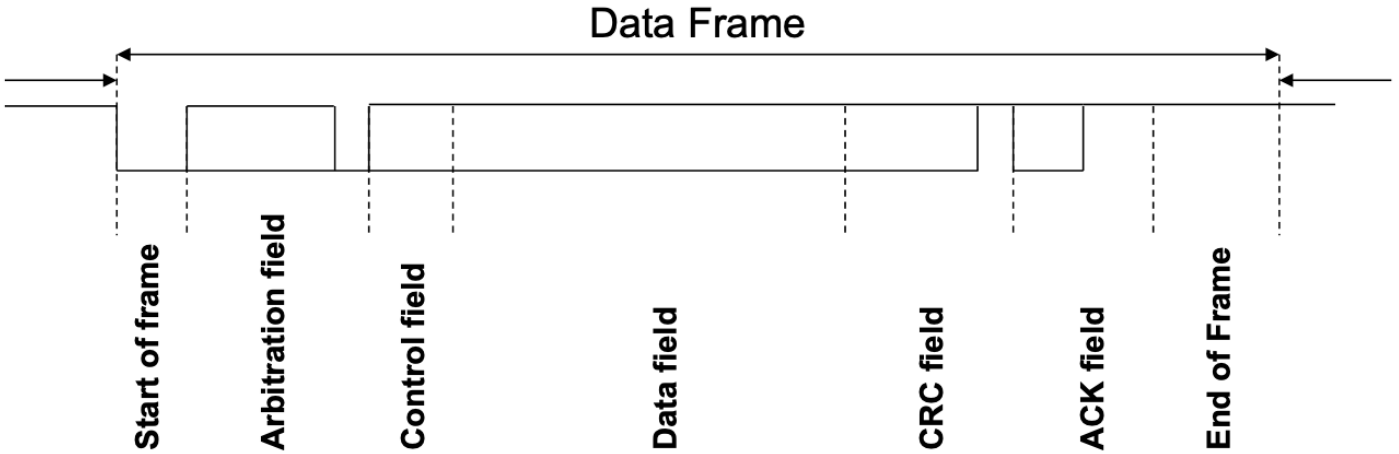
CAN Communication

- All nodes receive all frames
- The handling of the CAN bus communication within a node is done by a special CAN controller (card/chip)
- The CAN controller filters out frames not needed by the node
- Messages that are waiting to be sent are queued in a priority sorted list in the CAN controller



CAN Communication

- Messages are called frames
- A frame is tagged by an identifier
  - Indicates the contents of the frame (source addressing)
  - Used in the arbitration for prioritizing frames (frame with lowest identifier is selected to send in case of collision)
- The CAN physical layer behaves as a wired AND, i.e., if any node sends a logical 0, then all nodes receive 0



CAN Arbitration

- Frames start by sending the identifier field's **most significant bit first**
- While sending the identifier the frame is in **arbitration**
  - Other frames may be sent too
  - Need to find the highest priority frame
- If a node sends a 1 (recessive bit) but reads back a 0 (dominant bit) then it gives up and backs off
  - Means that a higher priority frame is being sent
  - Retries sending the frame when the bus is idle again

CAN Arbitration

Frame 1	1344	1
Frame 2	1306	1
Frame 3	1498	1
Bus		1

CAN Arbitration

Frame 1	1344	10
Frame 2	1306	10
Frame 3	1498	10
Bus		10

CAN Arbitration

Frame 1	1344	101
Frame 2	1306	101
Frame 3	1498	101
Bus		101

CAN Arbitration

Frame 1	1344	1010
Frame 2	1306	1010
~~**Frame 3**~~	~~1498~~	~~1011~~
Bus		1010

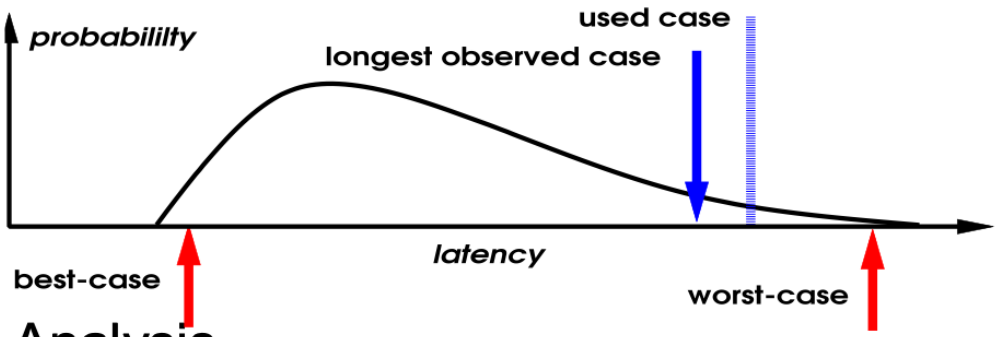
CAN Arbitration

~~**Frame 1**~~	~~1344~~	~~10101~~
Frame 2	1306	10100011010
~~**Frame 3**~~	~~1498~~	~~1011~~
Bus		10100011010

1306

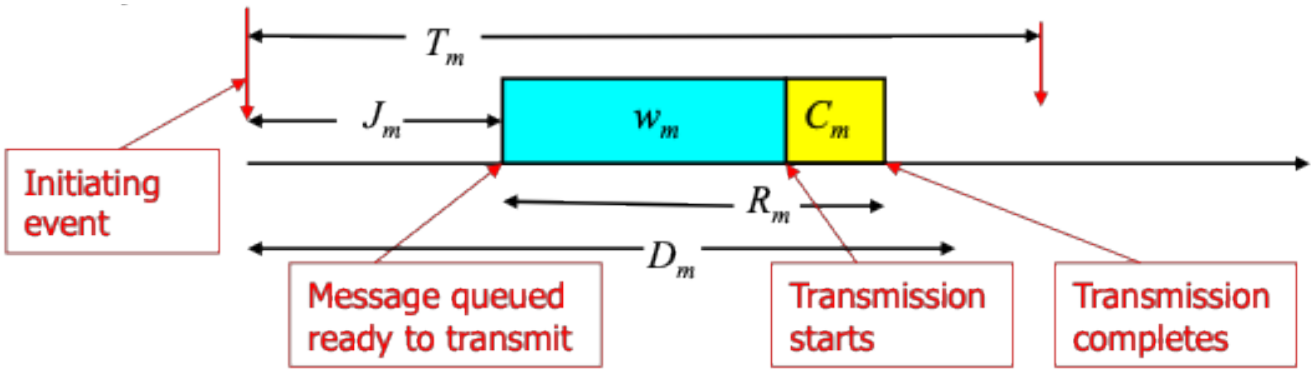
CAN and Hard Real-Time

- Need to bound the worst-case latency (end-to-end delay)
- Option 1: Testing



- Option 2: Analysis
  - Bus = shared resource (cp. CPU)
  - Frame = job (invocation of a task)
  - Fixed priority scheduling theory can be applied
    - Blocking factor from transmission of lower-priority frame

CAN Schedulability analysis: Model



- Each CAN message has a:
  - Unique priority *m* (identifier)
  - Maximum transmission time C_m
  - Minimum inter-arrival time or period T_m
  - Deadline $D_m <= T_m$
  - Maximum queuing jitter J_m
  - Transmission deadline $E_m = D_m - J_m$
- Compute:
  - Worst-case queuing delay w_m
  - Worst-case response time $R_m = w_m + C_m$
  - Compare with transmission deadline $R_m \leq E_m$

Commercial CAN Analysis Tools

- Volcano Network Architect
  - Commercial CAN schedulability analysis product

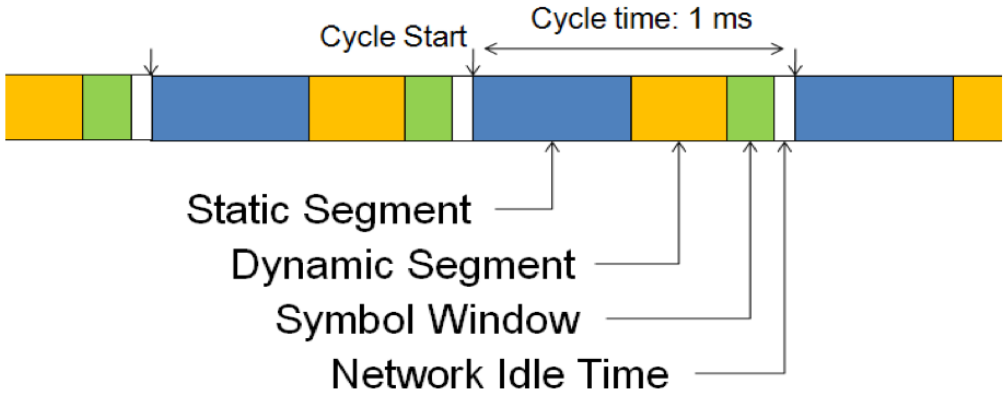
  - Uses the simple sufficient schedulability test #2, assuming maximum blocking factor irrespective of message priorities / number of data bytes

$$w_m^{n+1} = B^{MAX} + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{bit}}{T_k} \right\rceil C_k$$

  - Slightly pessimistic but correct upper bound on message worst-case response times
  - Used to analyse CAN systems for Volvo S80, S/V/XC 70, S40, V50, XC90 and many other cars from other manufacturers including Jaguar, Land Rover, Mazda, SAIC and others

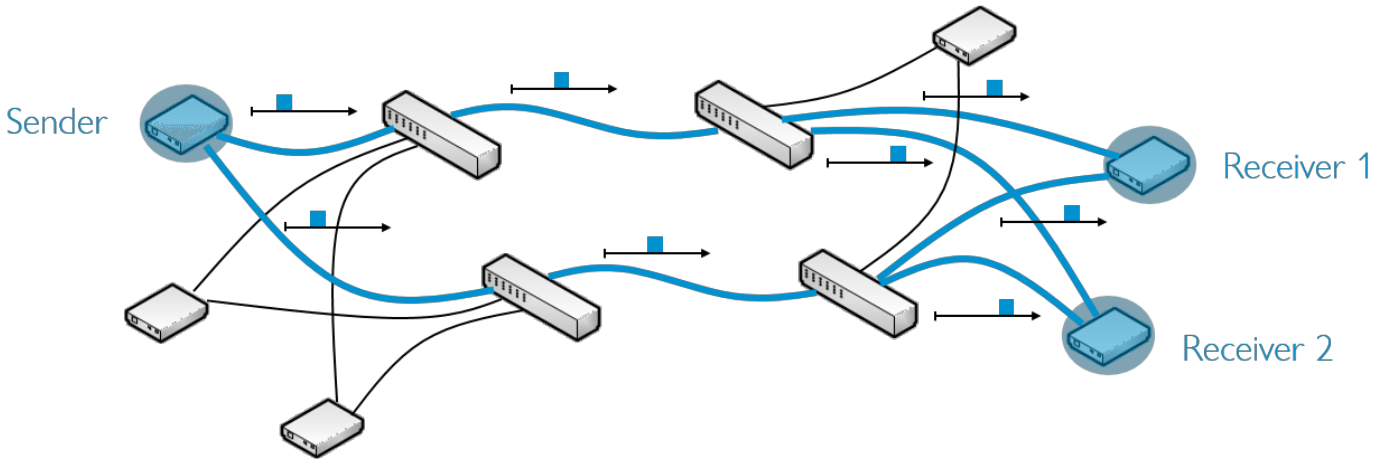
FlexRay

- TTP competitor
- Developed by European car manufacturers
- Combines time-driven and event-driven communication
- Event-driven communication allowed in special slots in the TDMA structure

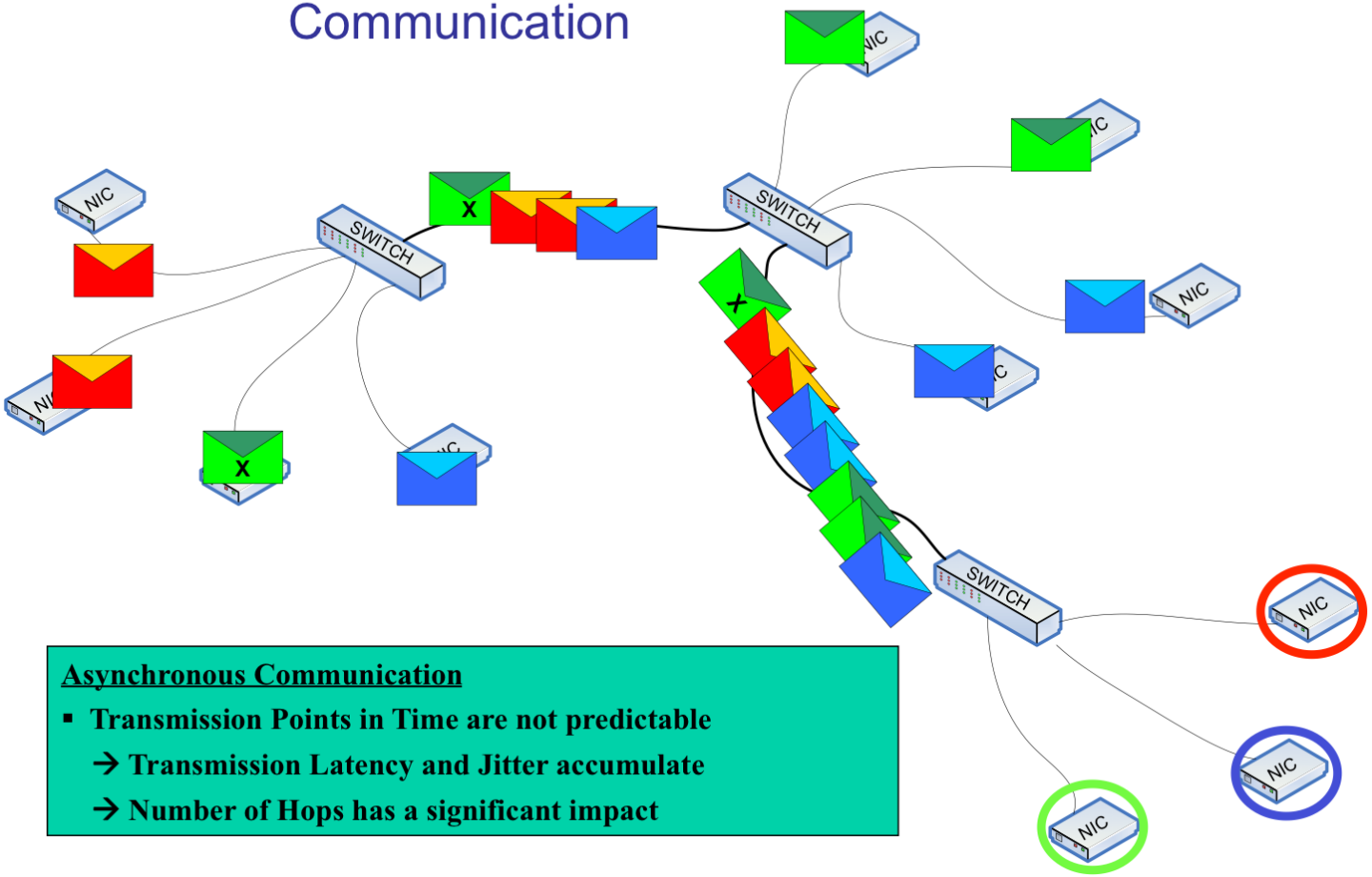


Switched networks (Ethernet)

- **End systems send and receive messages (also called hosts, end points or end stations)**
- End systems are connected via full-duplex multi-speed **links** and **switches** (bridges)
  The correct technical term for a switch according to the IEEE 802.1 standards is “bridge”
  Sometimes Switch and End system are integrated
- Links connect via **ports** to end systems and switches
- Messages are sent as streams with multiple frames per stream; communication is from one sender to one or multiple receivers (streams with multiple receivers are called “multicast”)



Ethernet = Unsynchronized Communication



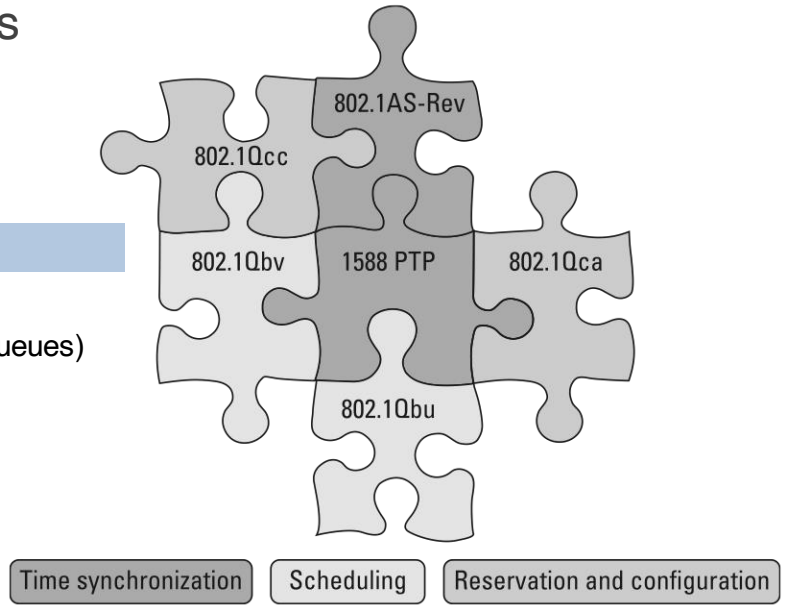
Asynchronous Communication

- **Transmission Points in Time are not predictable**
  - → **Transmission Latency and Jitter accumulate**
  - → **Number of Hops has a significant impact**

Time-Sensitive Network (TSN)

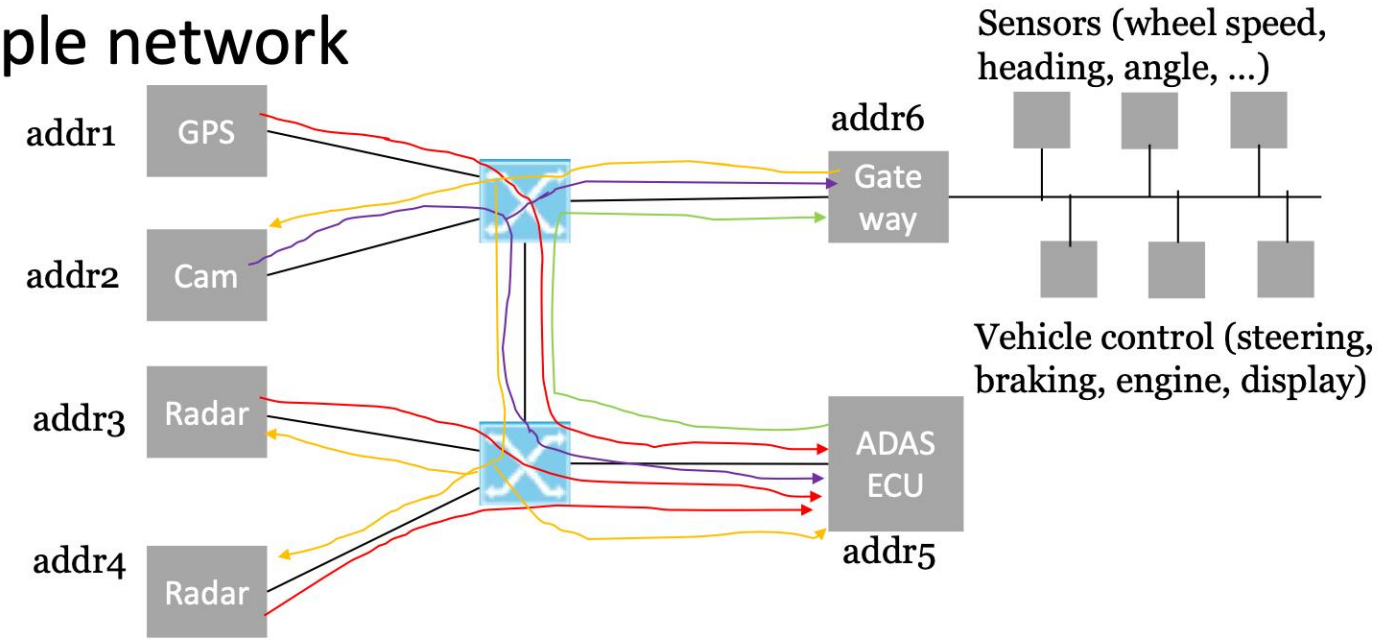
IEEE TSN task group - collection of sub-standards that enhance 802 Ethernet with real-time capabilities

Standard	Description
802.1AS	Timing & Synchronization
802.1Qbv	Enhancements for Scheduled Traffic (Timed Gates for Egress Queues)
802.1Qbu	Frame Preemption
802.1Qca	Path Control and Reservation
802.1Qcc	Central Configuration Management
802.1Qci	Per-Stream Time-based Ingress Filtering and Policing
802.1CB	Redundancy, Frame Replication & Elimination



TSN in vehicles

Example network



Streams:

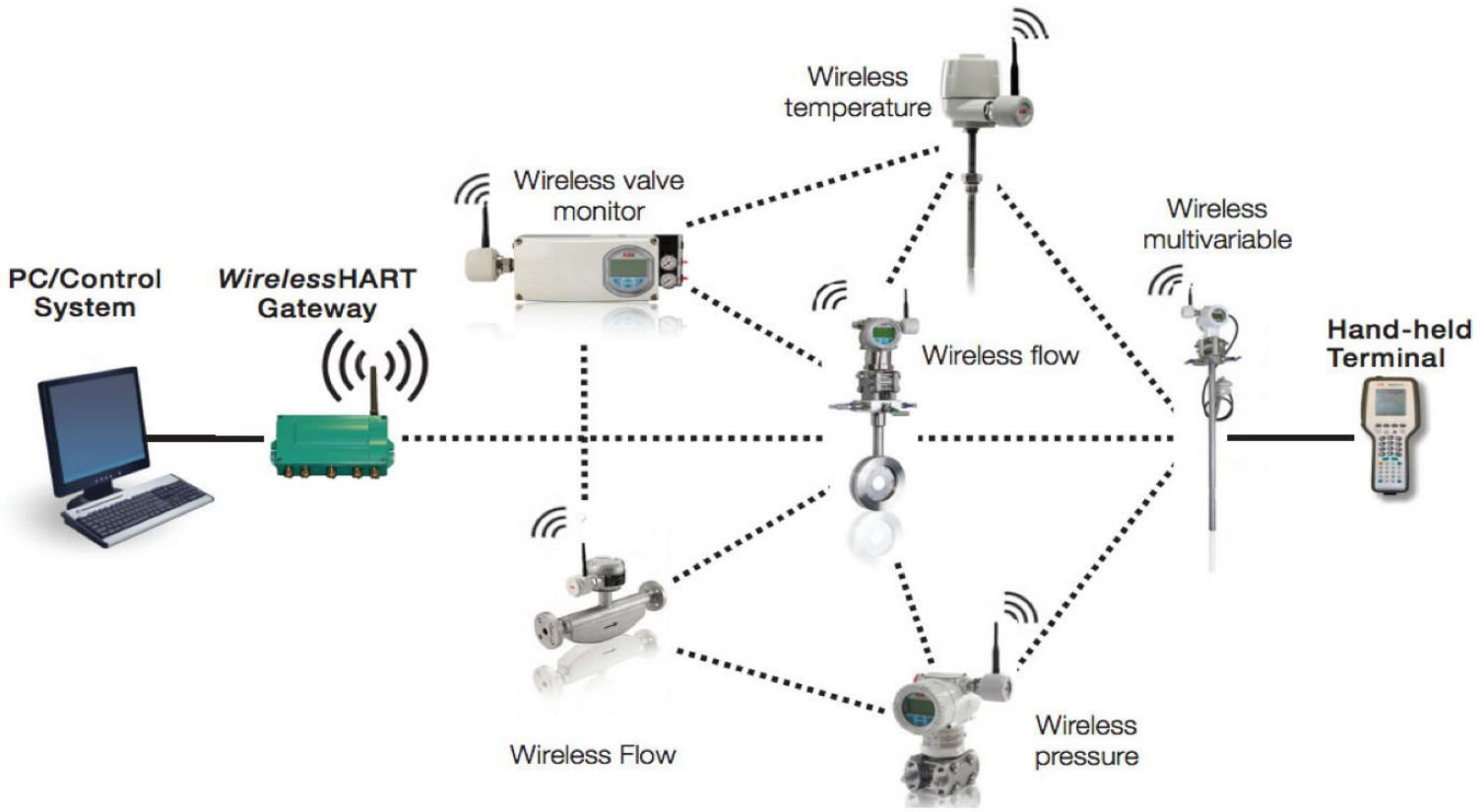
- **GPS and Radars to ADAS ECU**
- **Cam to ADAS ECU and Vehicle domain**
- **Vehicle sensors to Cam, Radars, and ADAS ECU**
- **ADAS ECU to Vehicle domain (actuation and corrected GPS)**

Wireless Networked Control

Emerging technology. Potential advantages:

- Flexibility
  - Placement: moving parts, mobile units, outdoor, ...
  - Commissioning and maintenance
- Cost
  - Less cables, fewer connectors, less wear and tear
  - Reduced design and installation costs
- New applications

Example: Wireless Industrial Automation



[ABB product sheet for Wireless HART]

Example: Car Trains

[Volvo Cars Newsletter: ”Fordonståg en möjlighet på vanliga motorvägar”]

Networks for Wireless Control

- ISM networks (licence free)
  - IEEE 802.11 (WLAN)
  - IEEE 802.15.1 (Bluetooth)
  - IEEE 802.15.4 (ZigBee)
- Mobile/cellular networks
  - GSM
  - 3G
  - 4G/LTE
  - Device to Device (D2D)

Protocols for Wireless Industrial Automation

Two protocols, both based on IEEE 802.15.4:

- Wireless Hart
  - Products since 2008 (ABB, Siemens, ...)
  - International standard (IEC 62591-1), 2010
- ISA 100 Wireless
  - Products since 2009 (Honeywell, GE, ...)
  - ANSI standard 2011
  - International standard (IEC 62734), 2014

Challenges in Networked Control

- **Long/variable delays**
- **Dropped packets**
- Safety
  - Must ensure safe error handling and safe shut downs
- Security

Above challenges even greater for wireless control systems

- Radio channel affected by the environment, disturbing nodes
- Routing in multi-hop networks

Time-Sensitive Networking (TSN)

Dependable communication in distributed cyber-physical systems

Prof. Paul Pop, paupo@dtu.dk
DTU Compute

Sources:
Prof. Soheil Samii, Linköping University, Sweden
János Farkas, Principal Researcher, Ericsson Research

Outline

- Motivation
- Switched network basics: Ethernet
- Time-Sensitive Networking (TSN): Introduction
- Traffic Shaping
  - Audio-Video Bridging (AVB)
  - Time-Aware Shaper (TAS)
  - Asynchronous Traffic Shaping (ATS)
- Dependable communication: Redundancy
- Configuring a TSN network

Motivation

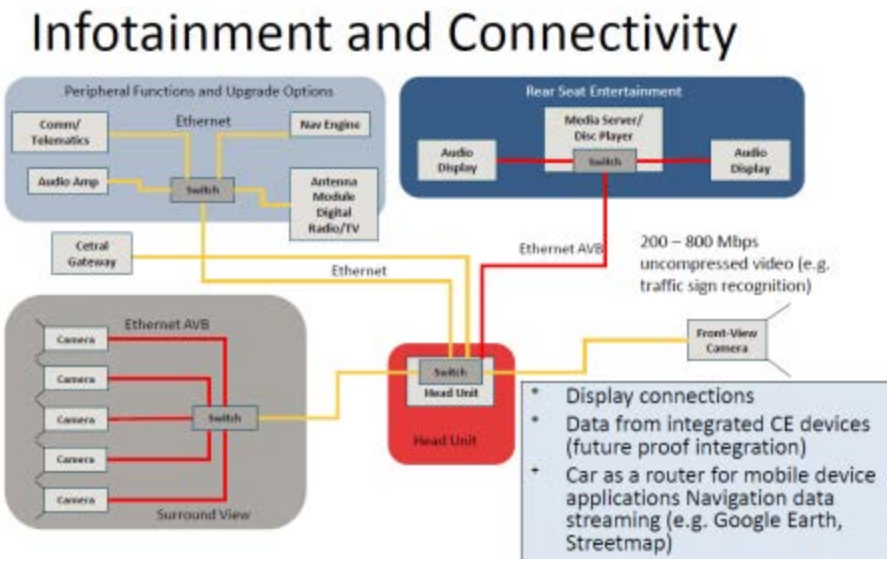
Time-Sensitive Networking (TSN)

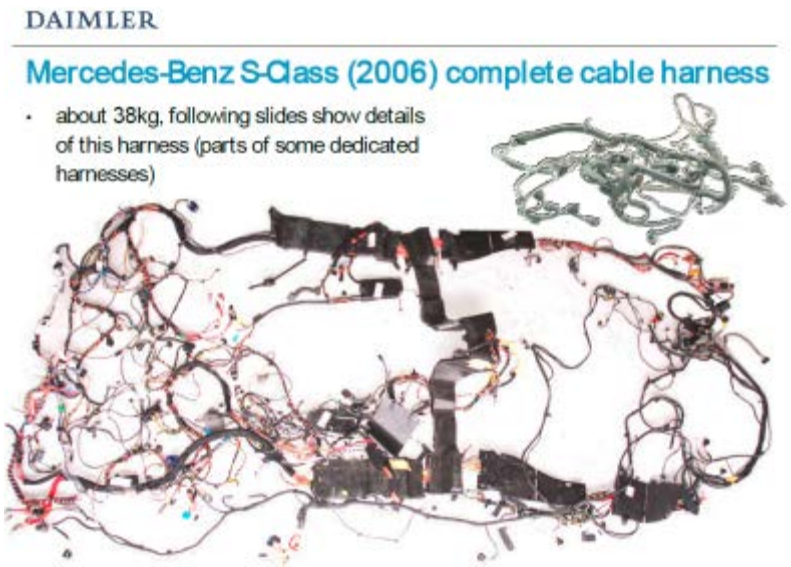
- In cyber-physical systems, real-time communication is vital:
  - Self-driving car hesitating to brake for a pedestrian in its path
  - Robots on an assembly line receiving delayed instructions from the computer that's synchronizing their movements

- **IEEE 802.1 Time-Sensitive Networking (TSN)** Task Group proposes sub-standards that extend Ethernet for safety-critical and real-time communication:
  - Dependable real-time communication
  - High bandwidth to accommodate the vast amount of sensor and background data that flows across automation networks
  - Backward compatibility to Ethernet devices

Automotive

- Emerging Markets: Vehicular Networks
  - Reduced Wiring Harness → Reduced weight and cabling costs
  - Reduce overall costs by using standardized chips
  - Reduce risks of binding to one silicon/solution vendor
  - Unified solution for different application areas (e.g. Infotainment, Power Train, Driver Assistance, …)

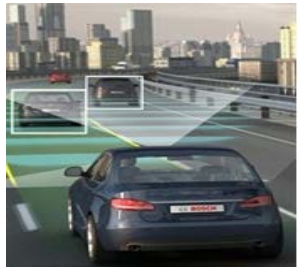




Picture Sources: IEEE 802.3 RTPGE SG

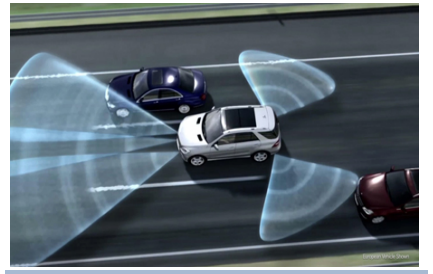
Automotive: increased functionality

Improvements on Sophisticated Automotive Advanced Drivers Assistance Systems (ADAS) like:



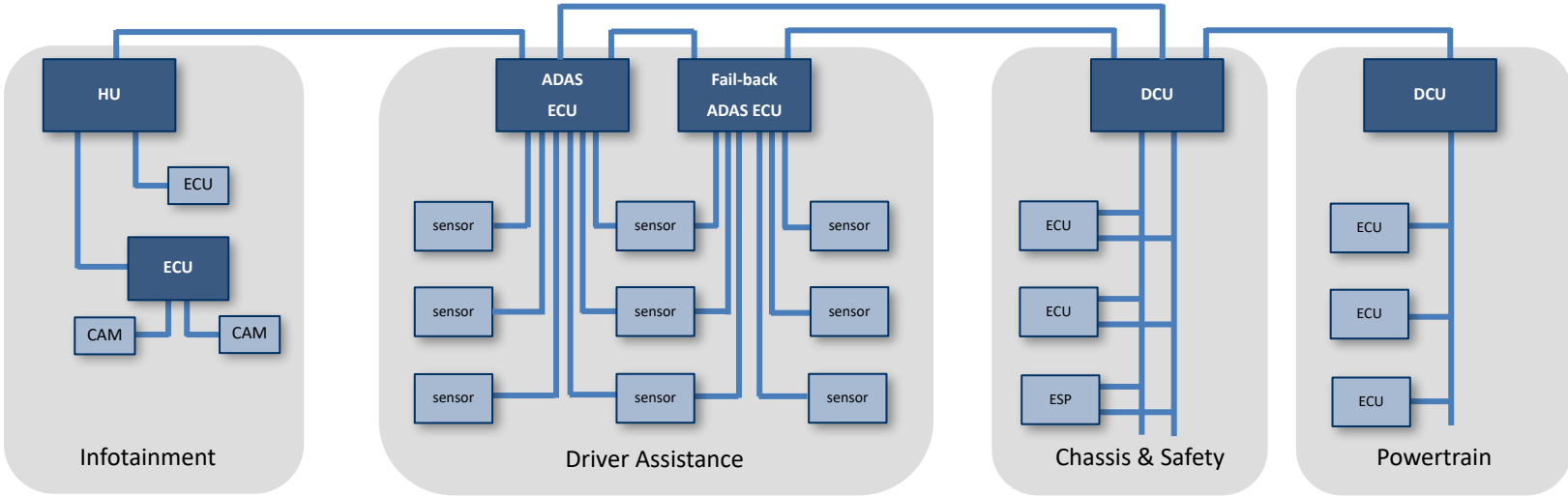
Adaptive Cruise Control

Lane Departure Assist



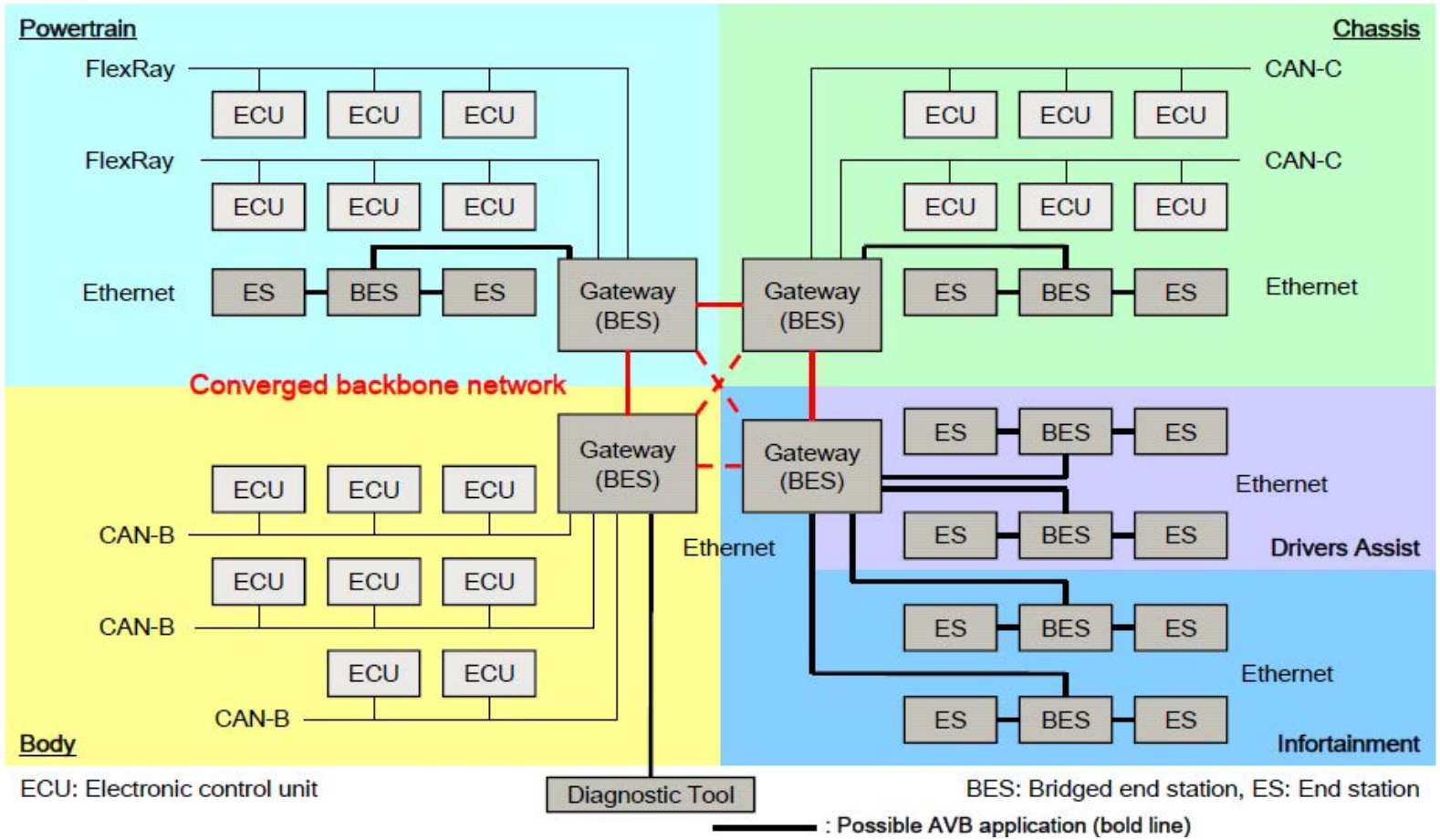
Blind Spot Assist

Brake Assist



TSN in Automotive

- An example converged backbone network for the domain architecture

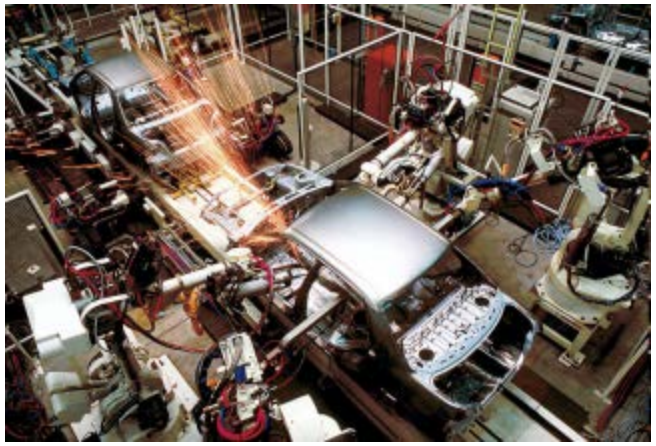


One possible application example of a future vehicular network

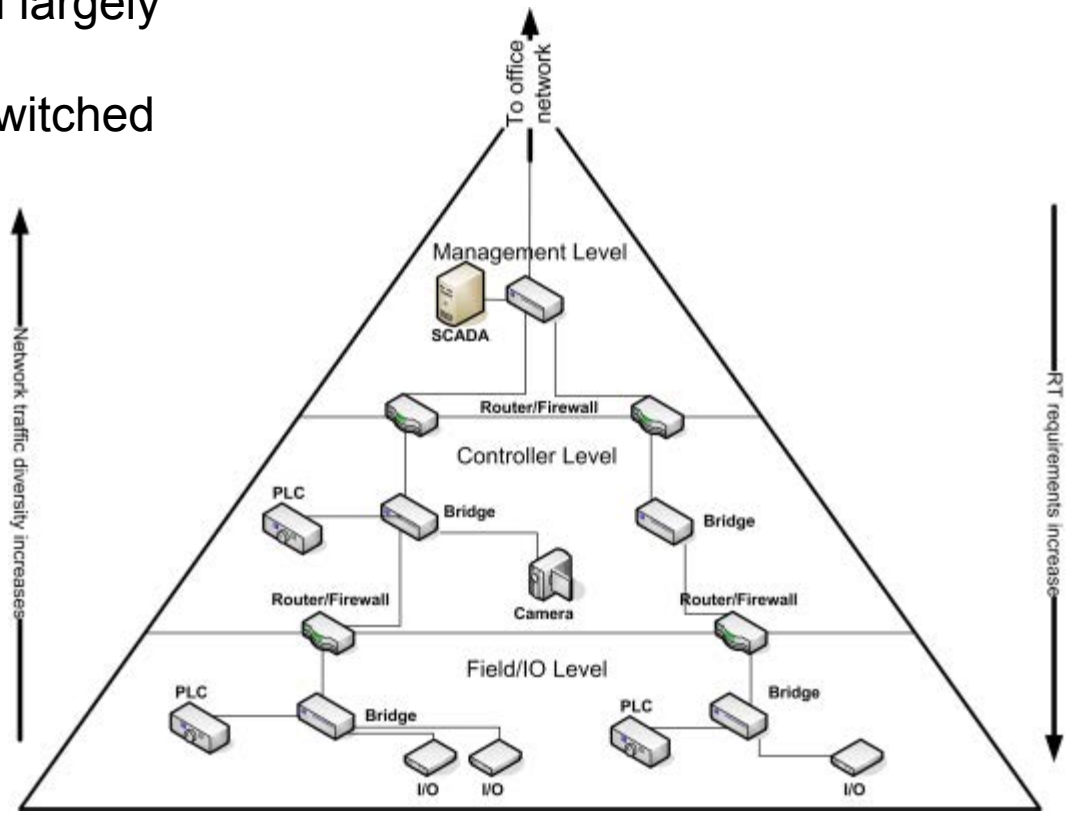
Source: IEEE 802.1 AVB TG presentation

TSN in Industrial Automation

- Early adopters outside IT: Industrial Automation (~1990s)
  - Higher Bandwidth than Fieldbusses
    (legacy automation network technologies, e.g. Profibus, Interbus, …)
  - Convergence with IT services
  - Widely available silicon could largely be re-used
  - Micro-Segmentation / Fully switched networks introduced first “deterministic Ethernet“
  - Easy fibre adoption

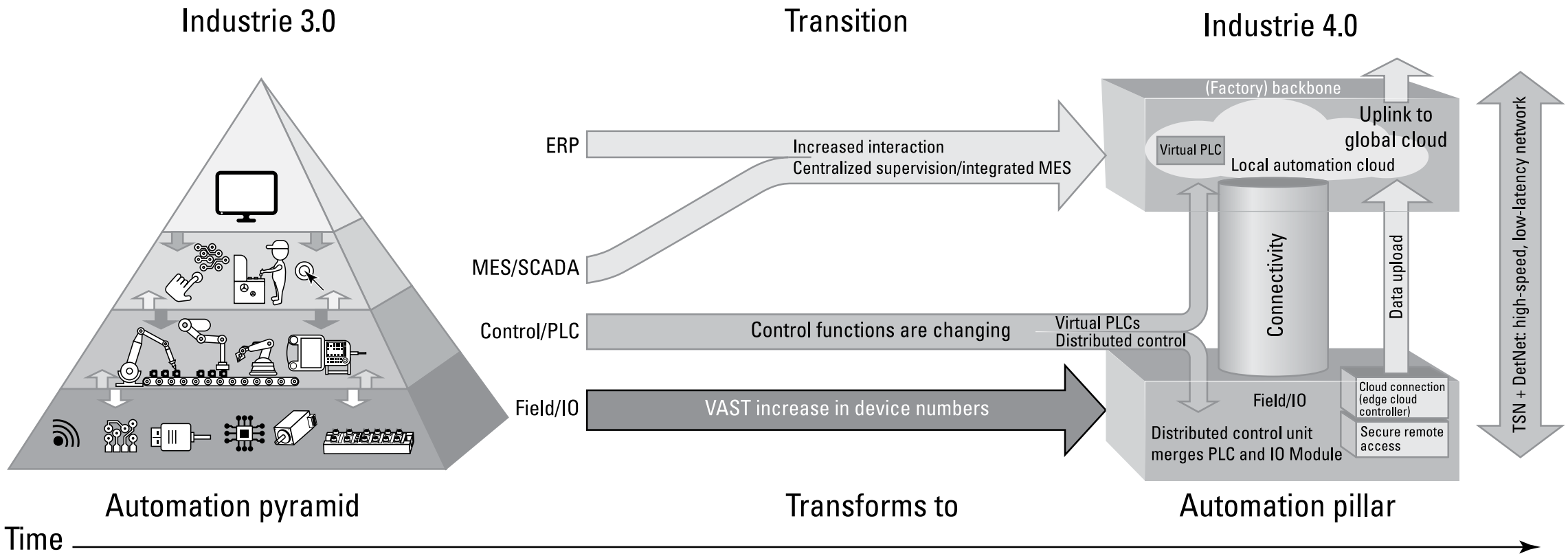


Manufacturing shop floor



Automation Pyramid

TSN in Industry 4.0



TSN: Other applications

- Emerging Markets: Mission-critical networking
  - Emerges out of Industrial Automation, massively broadening the scope
  - Requirements (far) beyond standard IT equipment relating to determinism in time and protocol behavior
  - Often used as transparent communication channel for End-to-End Safety Communication
  - Risk for Life and Limb if the system fails – High requirements to overall network, protocol and device robustness

…  …

Power Utility Automation

Traffic Control Systems …

Transportation …

TSN in Railways & Trains

- Ethernet in trains has applications in customer information and also infotainment

- Another application area lies in train control networks and video surveillance…
- …as well as passenger counters and detectors on the automatic train doors

TSN for Motion Control

Wind turbine: Synchronized rotor blade control actuators

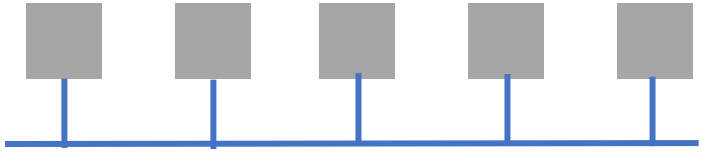
Printing machine: Large number of synchronized axles

Applications where robots and humans closely interact:

- Robot-assisted manufacturing
- Robot-assisted surgery
- Robotic prostheses
- …

Ethernet

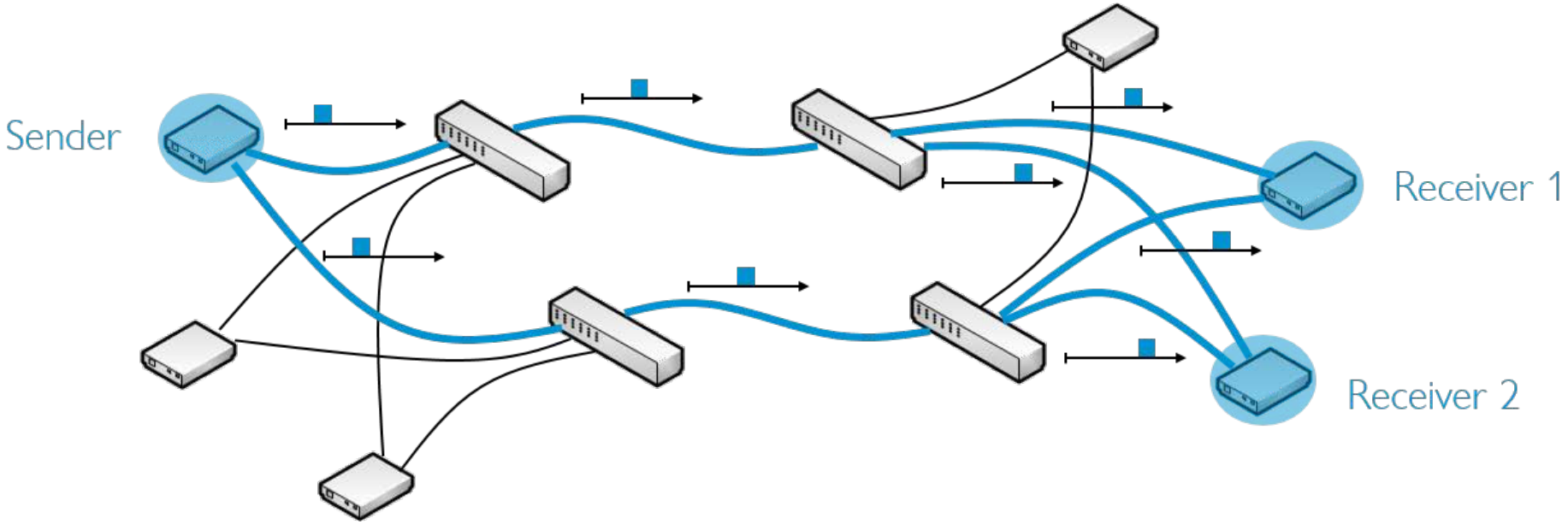
Ethernet CSMA/CD



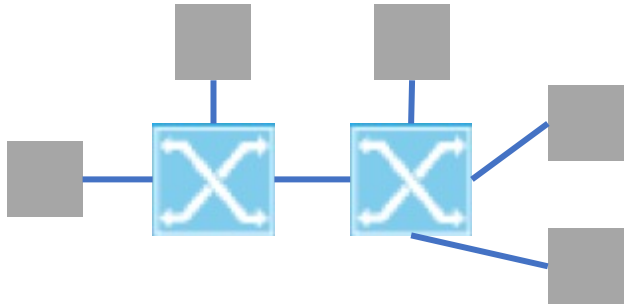
- Created in the mid 70s
- Ethernet bus: shared media access
- 46 to 1500 byte payloads; 10 Mbps to 10 Gbps
- Node transmits as soon as medium is free
- Collisions can occur during the interval of one slot after start of transmission (512 bits)
- Jamming signal (32 bits) is sent in case of collisions
- Retransmission after random amount of time
- Outdated
- Not appropriate for real-time systems

Switched network

- **End systems** send and receive messages (also called hosts, end points or end stations)
- End systems are connected via full-duplex multi-speed **links** and **switches** (bridges)
  The correct technical term for a switch according to the IEEE 802.1 standards is “bridge”
  Sometimes Switch and End system are integrated
- Links connect via **ports** to end systems and switches
- Messages are sent as streams with multiple frames per stream; communication is from one sender to one or multiple receivers (streams with multiple receivers are called “multicast”)

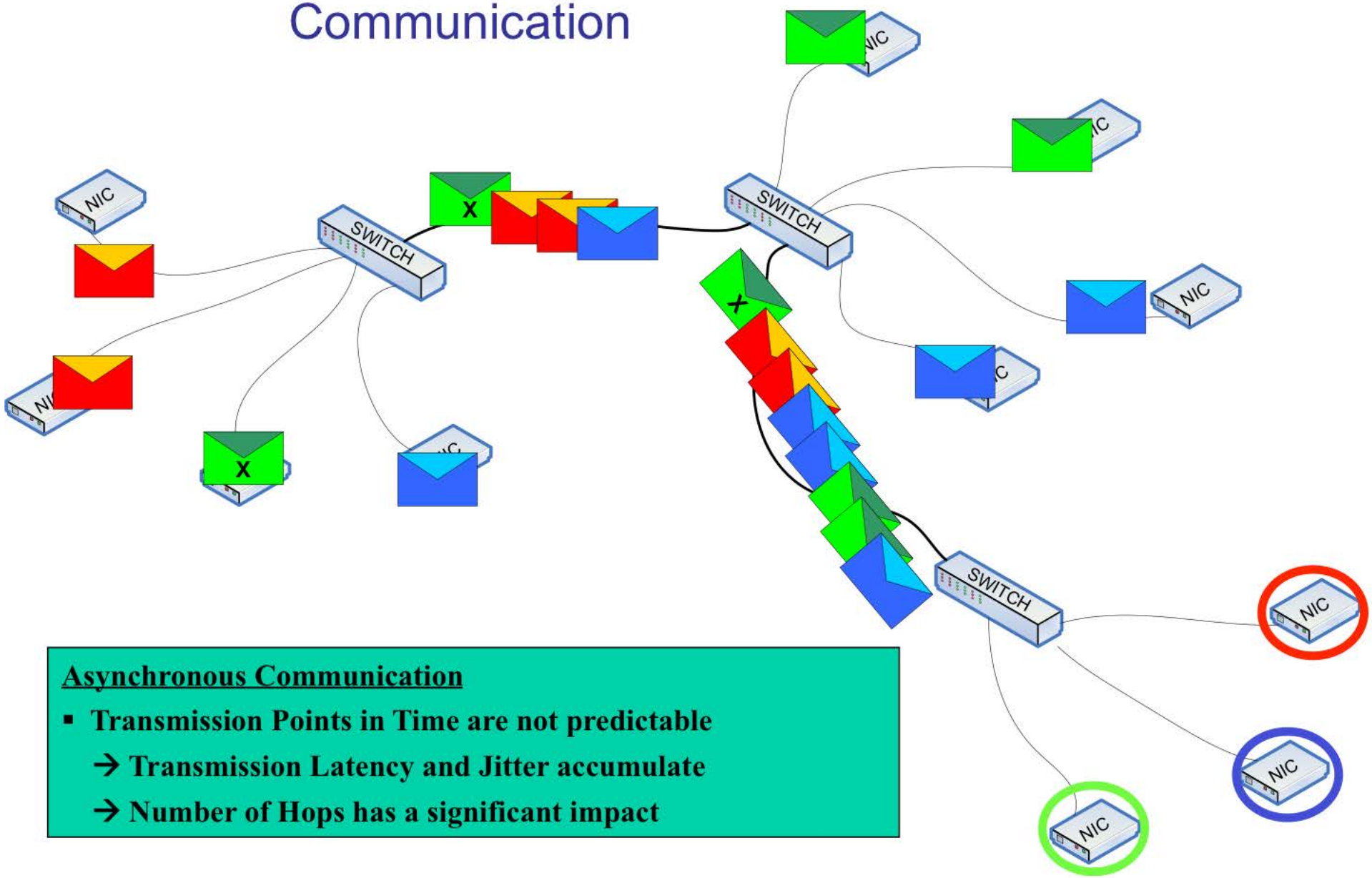


Switched Ethernet



- IEEE 802.1D and 802.1p. Now all in 802.1Q with Virtual LANs (VLANs) and priority queues
- Better but still gaps for real-time and safety-critical systems:
  - Priority inversions in FIFO queues
  - Interference through shared memory
  - Additional forwarding delay with address learning and flooding
  - Delays vary with switch technology
  - “Plug and Play” has been the mindset: lots of protocols that were not primarily designed for real-time applications (long delays and nondeterminism)

Ethernet = Unsynchronized Communication

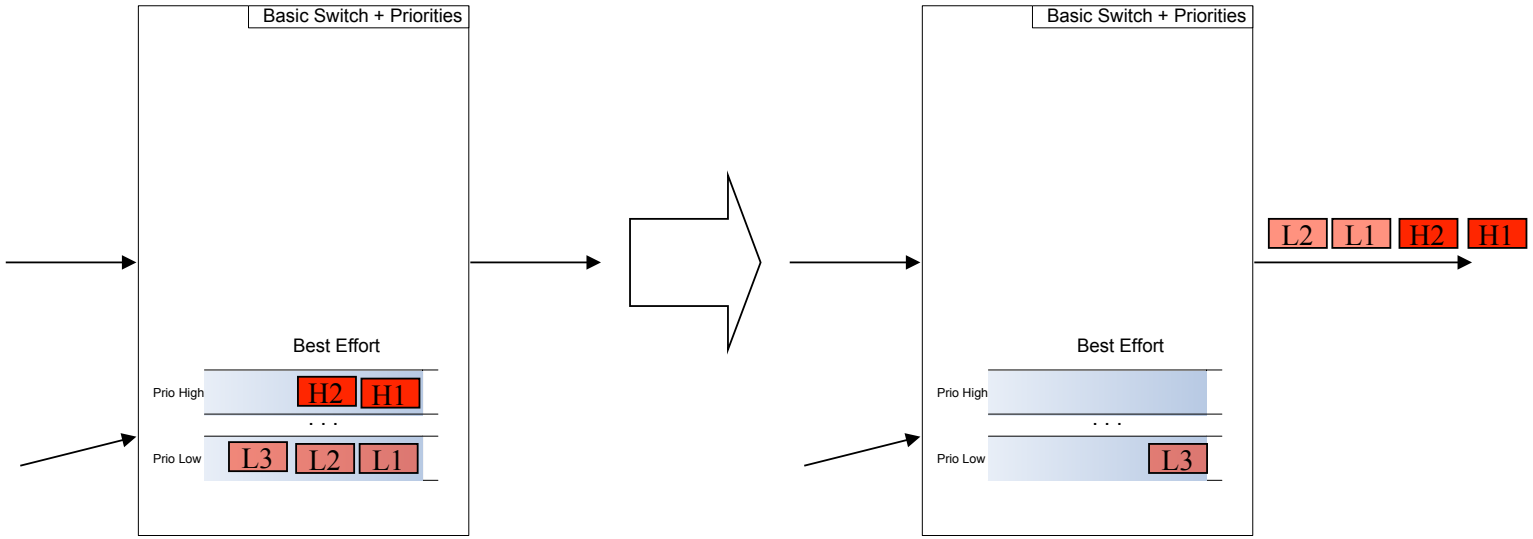


Asynchronous Communication

- **Transmission Points in Time are not predictable**
  - → **Transmission Latency and Jitter accumulate**
  - → **Number of Hops has a significant impact**

Priorities

•Frames with a high priority can overtake frames with a lower priority.



Problems with priorities:

- High priority frames may “starve” low priority frames.
- Too many high priority frames:
  → performance of high priority frames becomes insufficient.

Ethernet frame format

EtherType examples
0x0800 (IPv4), 0x86DD (IPv6), 0x88F7 (PTP), 0x88E5 (MACsec)

802.3 Ethernet packet and frame structure

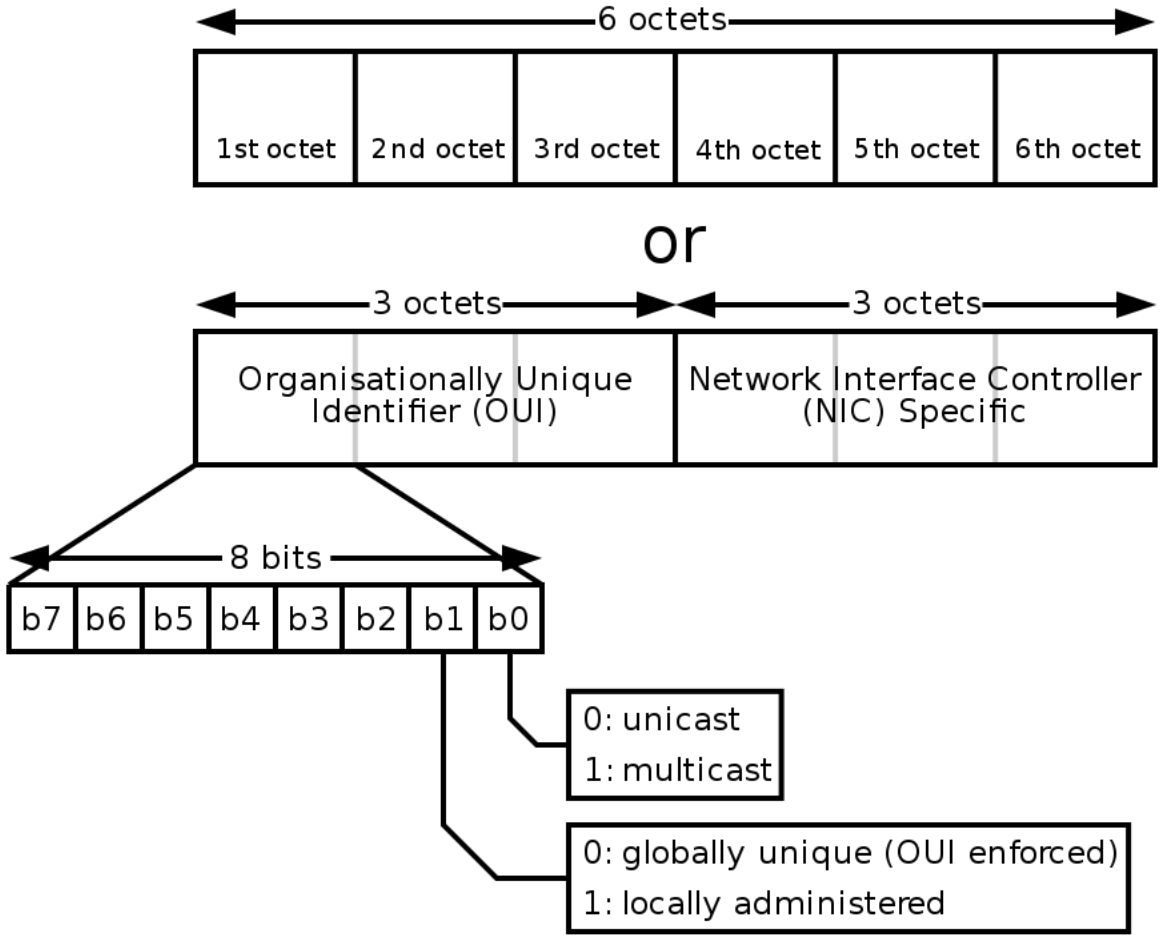
Layer	Preamble	Start of frame delimiter	MAC destination	MAC source	802.1Q tag (optional)	Ethertype (Ethernet II) or length (IEEE 802.3)	Payload	Frame check sequence (32-bit CRC)	Interpacket gap
	7 octets	1 octet	6 octets	6 octets	(4 octets)	2 octets	46-1500 octets	4 octets	12 octets
Layer 2 Ethernet frame			← 64–1522 octets →						
Layer 1 Ethernet packet & IPG	← 72–1530 octets →								← 12 octets →

802.1Q tag format

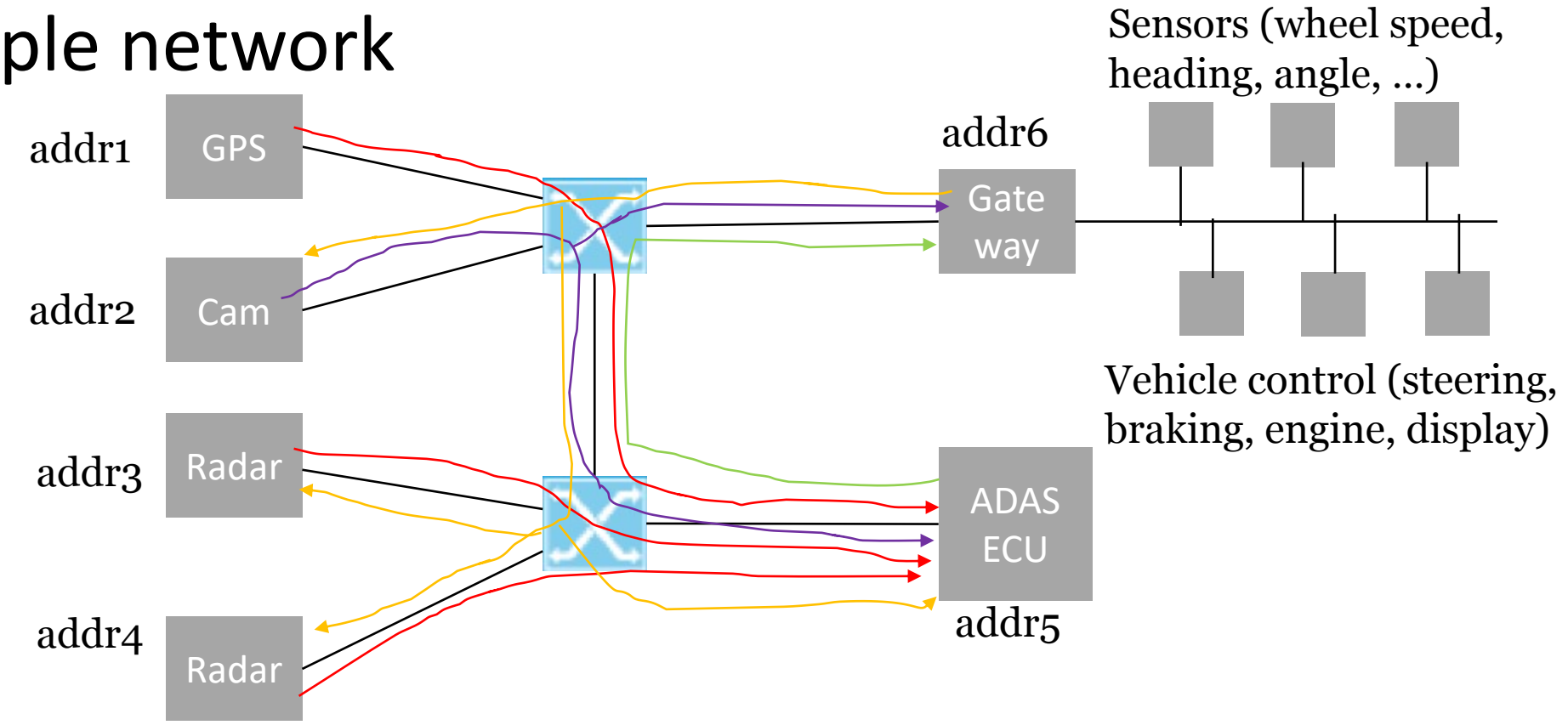
16 bits	3 bits	1 bit	12 bits
TPID	TCI		
	PCP	DEI	VID

- Unicast frames (Direct addressing of the (only) receiver)
- Multicast frames (when there are multiple receivers)
  - Some reserved for Layer 2 protocols (e.g., Spanning Tree Protocol and Precision Time Protocol)
  - AVB and TSN streams are typically multicast
- Broadcast frames (FF:FF:FF:FF:FF:FF)
  - Flood frame on all ports

MAC address


6 octets
1st octet
2nd octet
3rd octet
4th octet
5th octet
6th octet
or
3 octets
3 octets
Organisationally Unique Identifier (OUI)
Network Interface Controller (NIC) Specific
8 bits
b7
b6
b5
b4
b3
b2
b1
b0
0: unicast
1: multicast
0: globally unique (OUI enforced)
1: locally administered

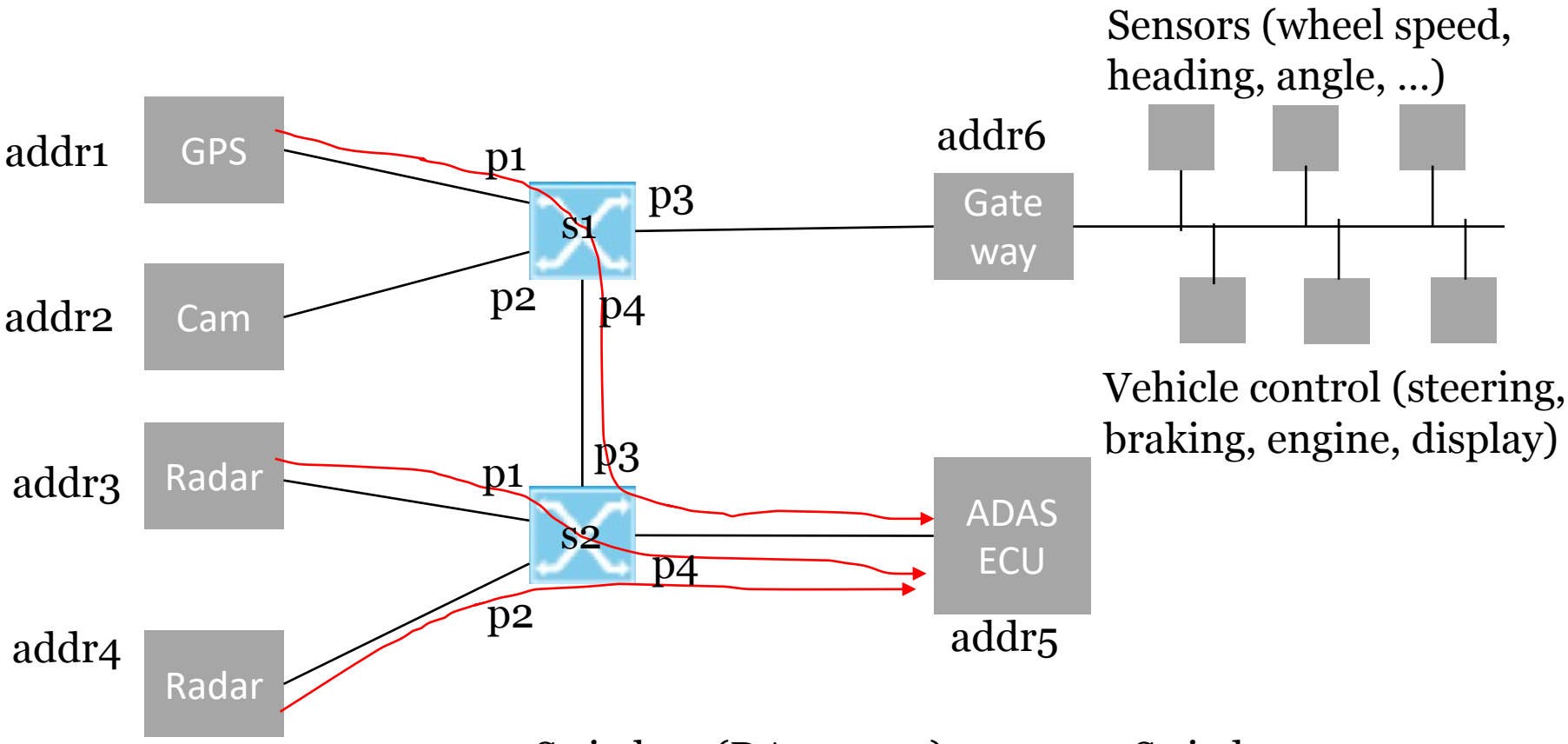
Example network



Streams:

- **GPS and Radars to ADAS ECU**
- **Cam to ADAS ECU and Vehicle domain**
- **Vehicle sensors to Cam, Radars, and ADAS ECU**
- **ADAS ECU to Vehicle domain (actuation and corrected GPS)**

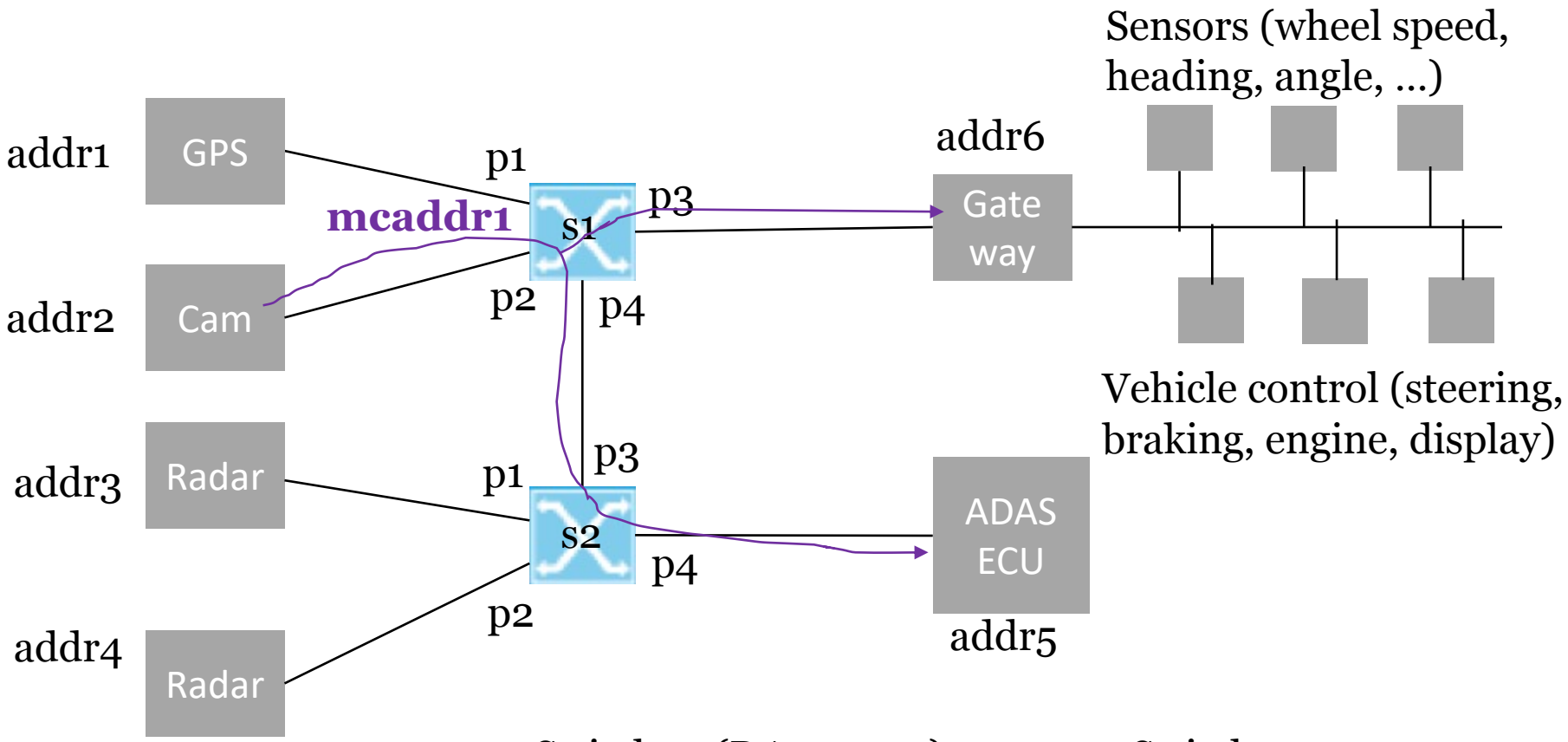
Forwarding tables



Switch s1 (DA -> port):
addr5 -> p4

Switch s2:
addr5 -> p4

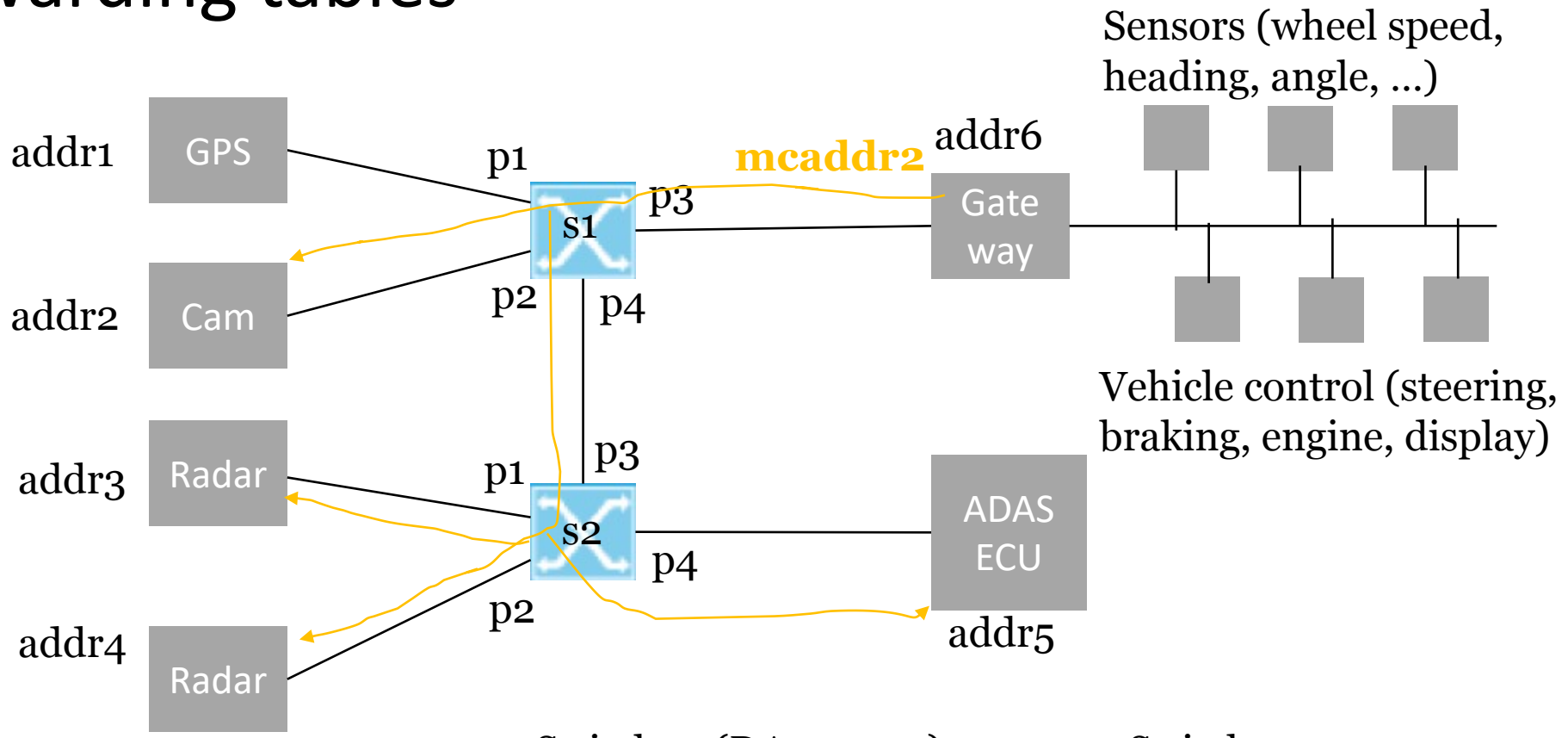
Forwarding tables



Switch s1 (DA -> port):
addr5 -> p4
mcaddr1 -> p3, p4

Switch s2:
addr5 -> p4
mcaddr1 -> p4

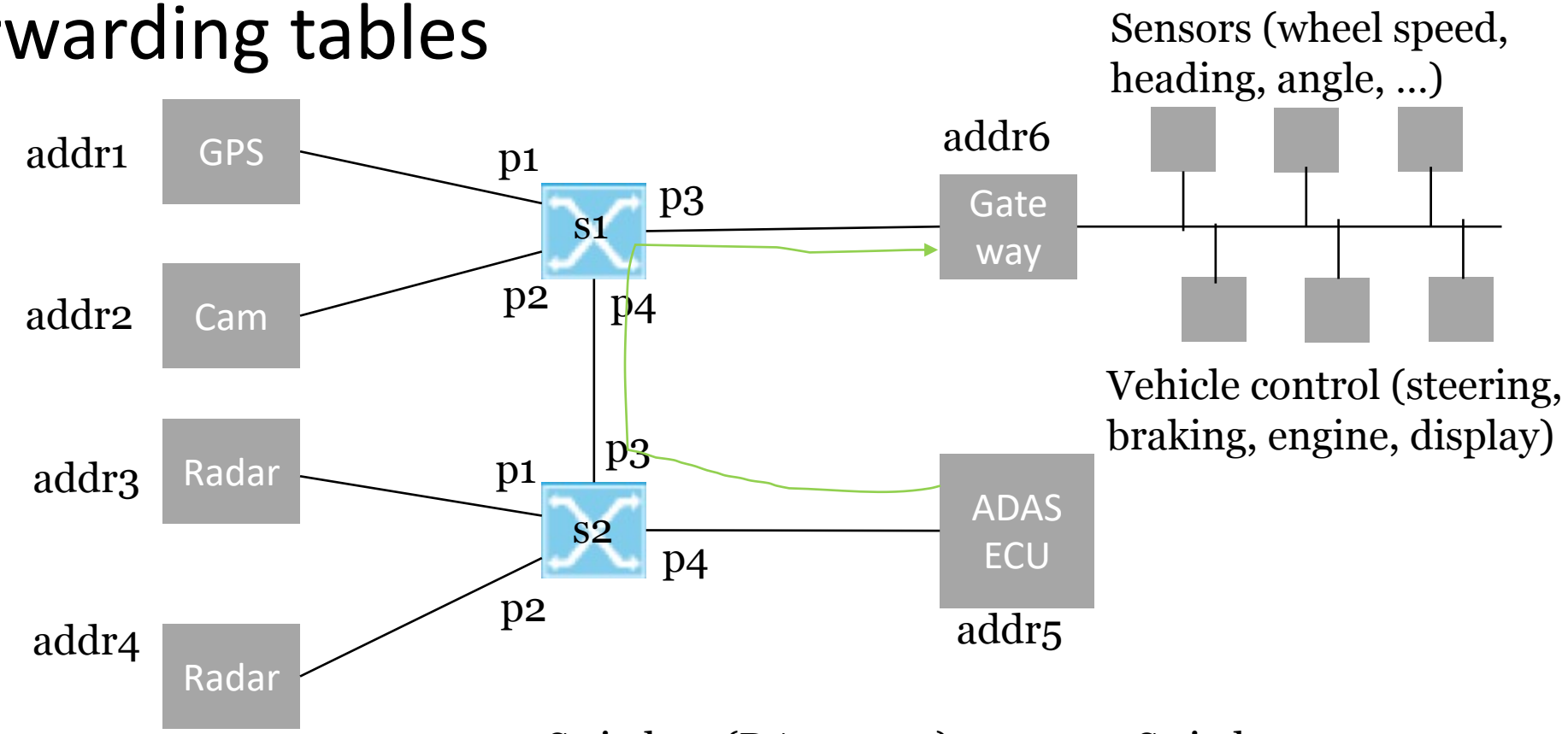
Forwarding tables



Switch s1 (DA -> port):
addr5 -> p4
mcaddr1 -> p3, p4
mcaddr2 -> p2, p4

Switch s2:
addr5 -> p4
mcaddr1 -> p4
mcaddr2 -> p1, p2, p4

Forwarding tables



Switch s1 (DA -> port):
addr5 -> p4
mcaddr1 -> p3, p4
mcaddr2 -> p2, p4
addr6 -> p3

Switch s2:
addr5 -> p4
mcaddr1 -> p4
mcaddr2 -> p1, p2, p4
addr6 -> p3

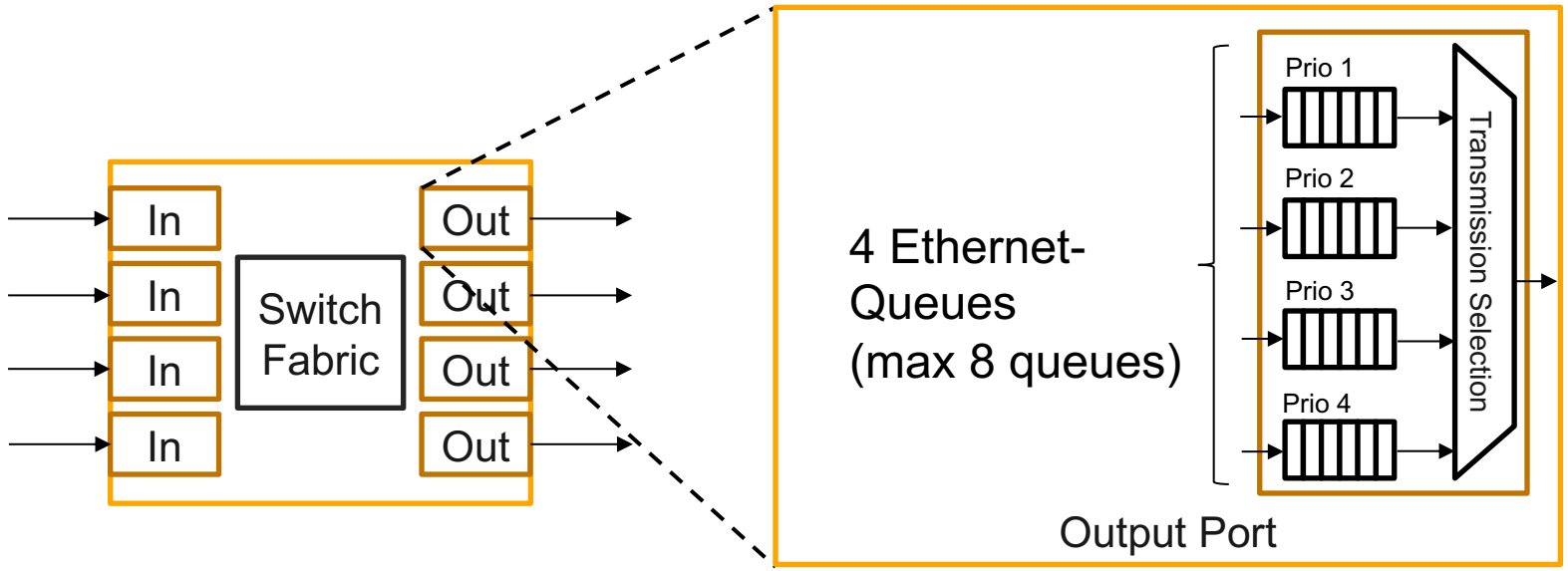
Table can also be a mapping from DA and VLAN ID to ports

Configuration of forwarding tables

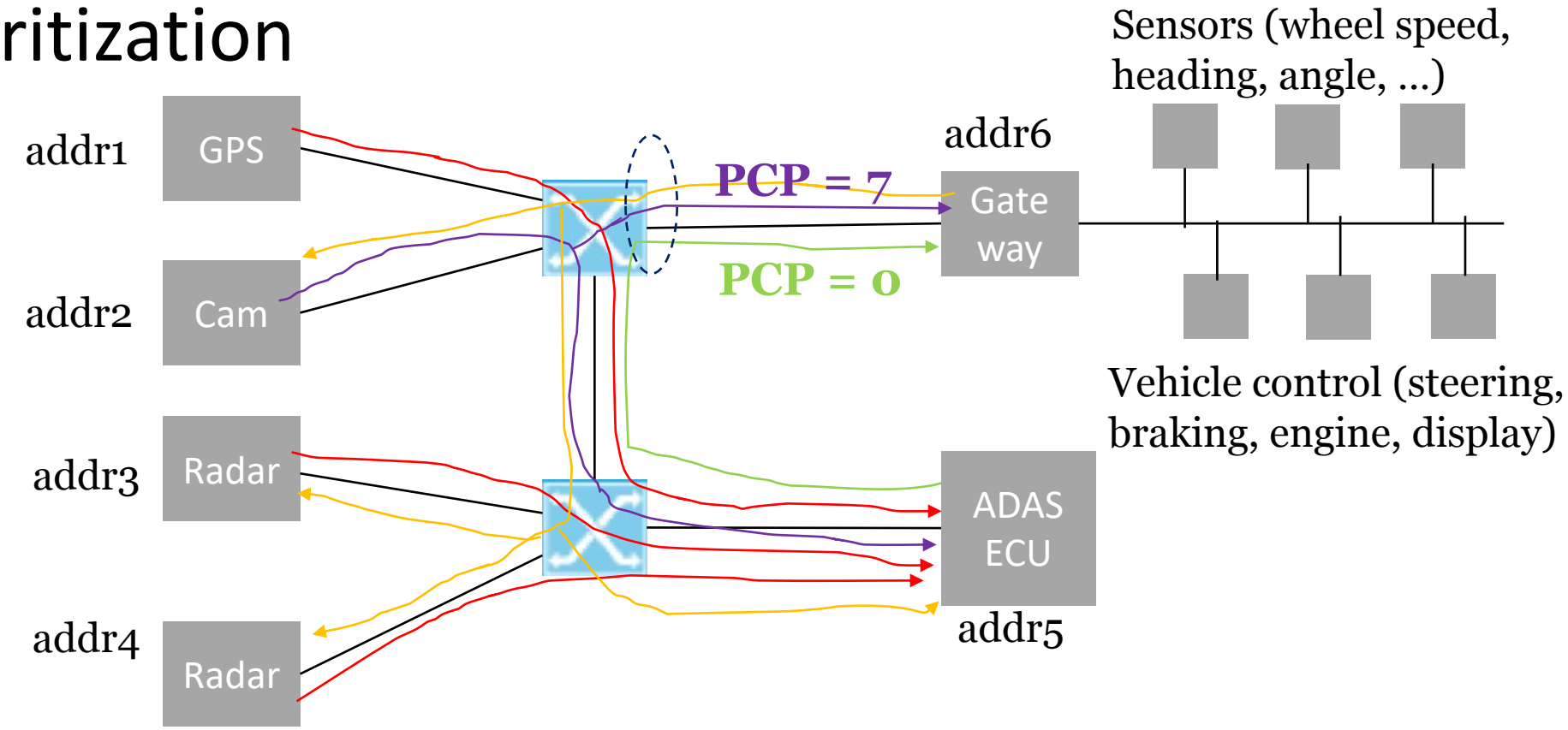
- For plug & play networks:
  - Flooding
  - Learning based on mapping ingress ports to source addresses (unicast)
  - Multiple stream registration protocol (multicast)
- Engineered networks (e.g., automotive, avionics)
  - Static configuration of forwarding tables
  - Protocols for learning are switched off (also helps to reduce risks for cybersecurity attacks)

Ethernet switch (“bridge” in IEEE 802.1)

- Frame carries priority
- Frames are assigned to egress ports (forwarding)
- Priorities are mapped to egress queues
- FIFO queues



Prioritization

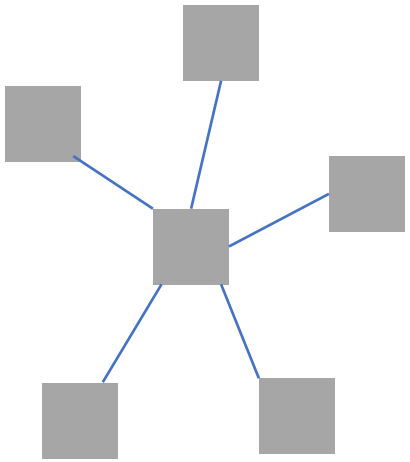


Streams:

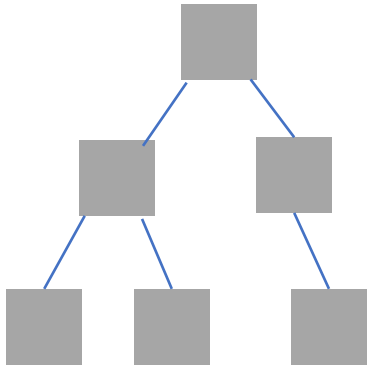
- **GPS and Radars to ADAS ECU**
- **Cam to ADAS ECU and Vehicle domain**
- **Vehicle sensors to Cam, Radars, and ADAS ECU**
- **ADAS ECU to Vehicle domain (actuation and corrected GPS)**

Different topologies

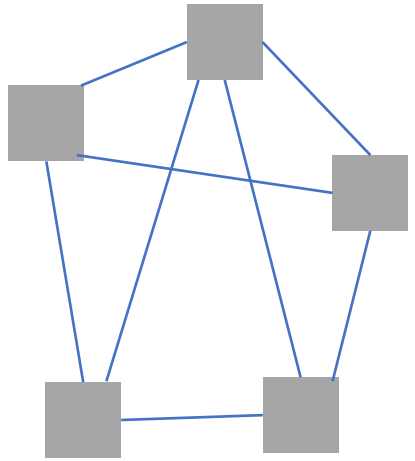
Star



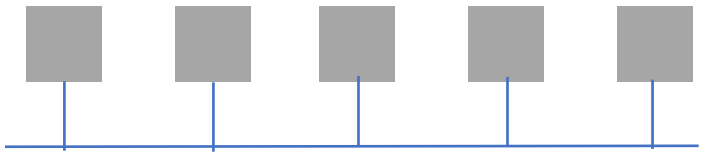
Tree



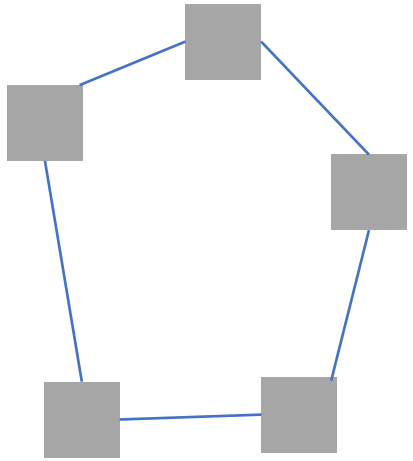
Mesh



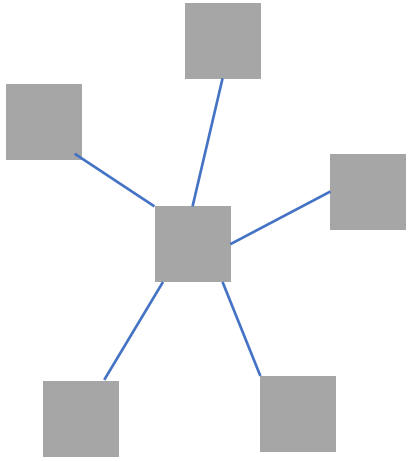
Bus

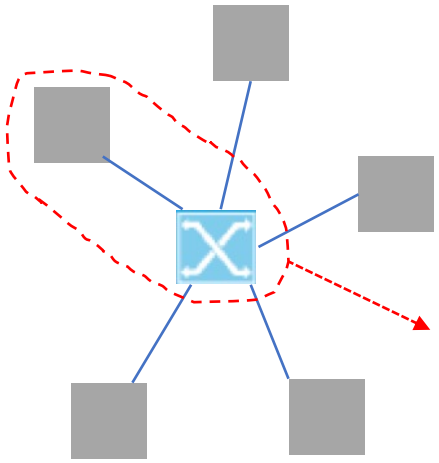


Ring



Star topology



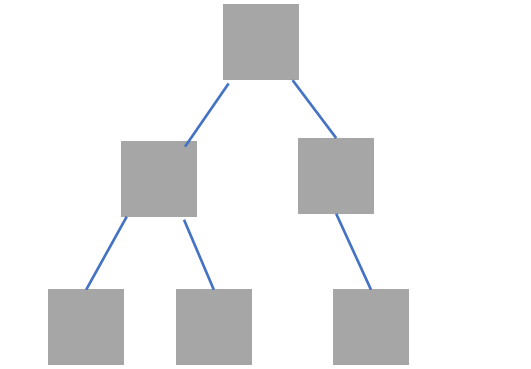


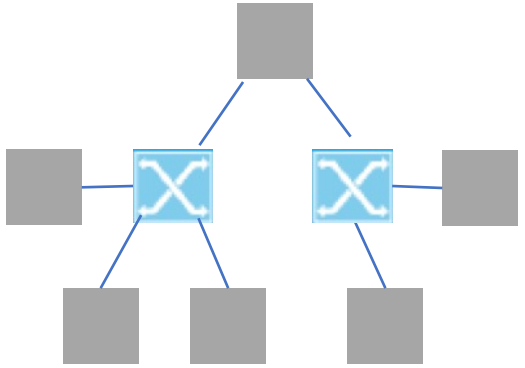
Some observations:

- One hop communication to all (low latency)
- Low cost
- Switch failure disconnects all communication
- Long wires, depending on physical location of endpoints

- Can be integrated in one module with MII, no PHY
- Integration is product and topology dependent

Tree topology

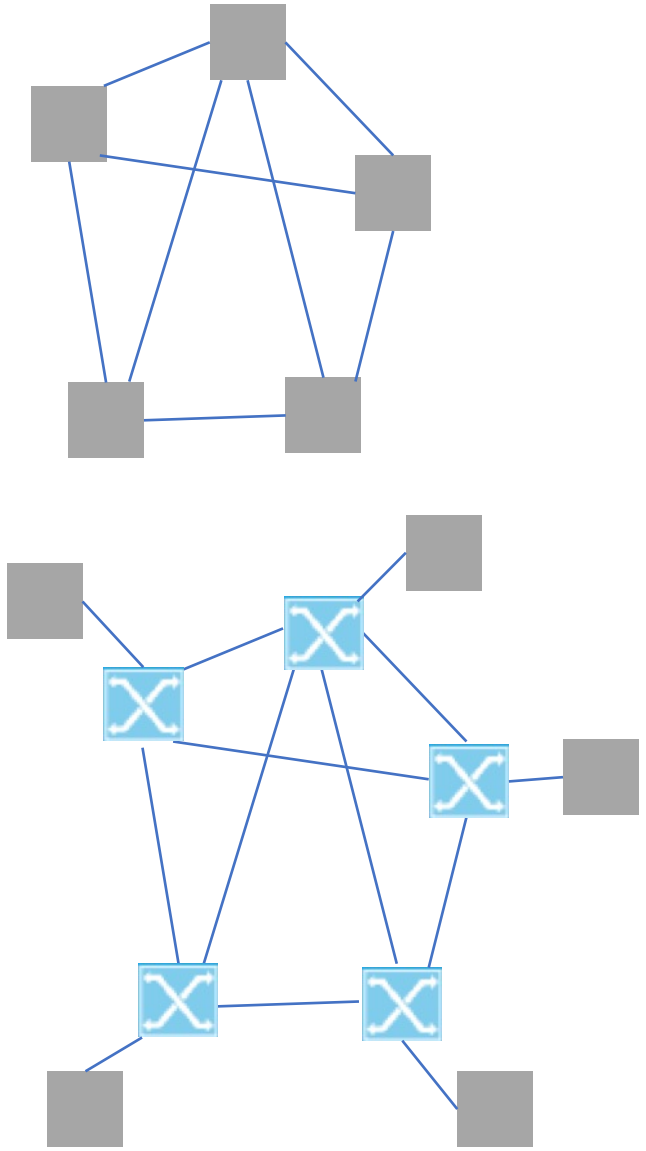




Some observations:

- Multiple hops between some nodes
- Switch failure disconnects several nodes
- Good for networks where communication mainly goes in one direction
- Segmented left and right side

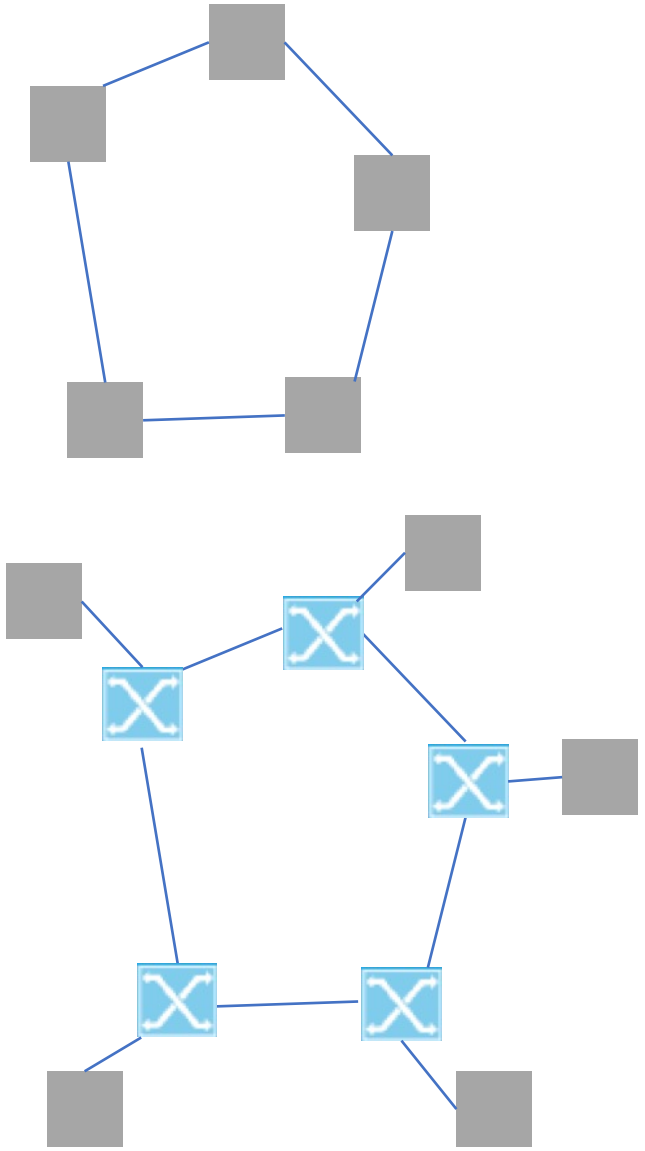
Mesh topology



Some observations:

- Multiple hops between some nodes
- Reduced impact of one switch failure
- Multiple communication paths (easier to meet timing and redundancy needs)
- High cost due to many switches

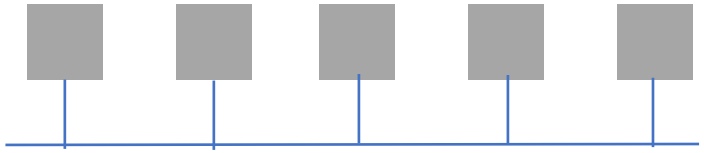
Ring topology



Some observations:

- Latency does not scale well with number of nodes
- Redundancy at lower cost than mesh solution
- Still many switches, but of lower port count
- Switch failure “only” disconnects one node

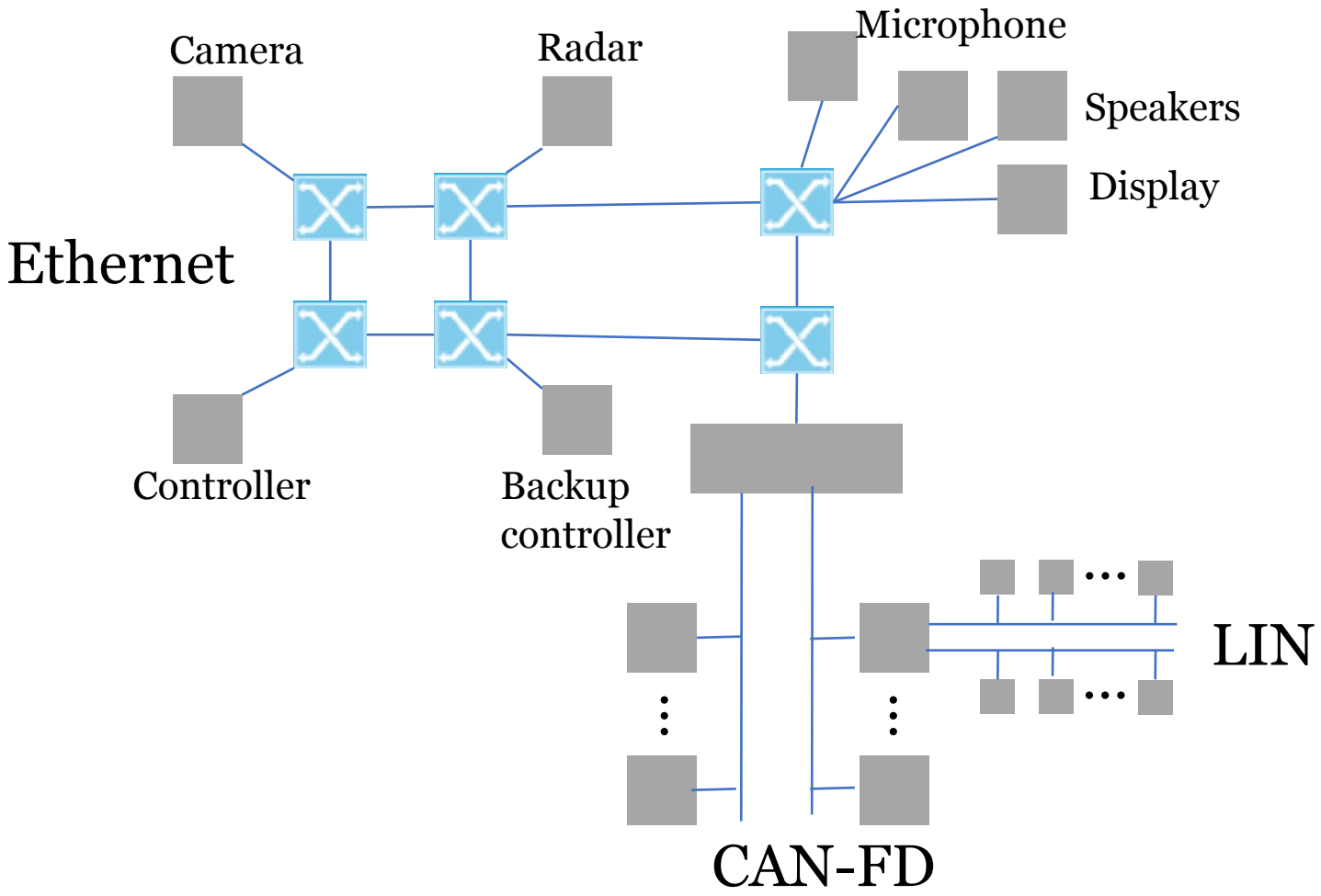
Bus topology



Some observations:

- 10BASE-T1S, IEEE Std 802.3cg
- Low cost (no switches)
- Single broadcast domain
- Shared media access
- 10BASE-T1S, IEEE Std 802.3cg, specifies PLCA (Physical Layer Collision Avoidance)
  - Master node sends beacon indicating new cycle
  - Nodes are allocated bandwidth during cycle in a round robin manner

Network is usually a mix of topologies and technologies



Adding bandwidth is not enough

- Support for real-time and safety-critical applications
- Several industries with similar communication requirements at OSI Layer 2 and above
  - Professional audio/video
  - Industrial automation
  - Automotive
  - Telecom
- Standardization development in IEEE 802.1
  - First, AVB. Now, TSN
  - Multi-hop, switched topologies make certain problems more complicated than bus-based real-time communication

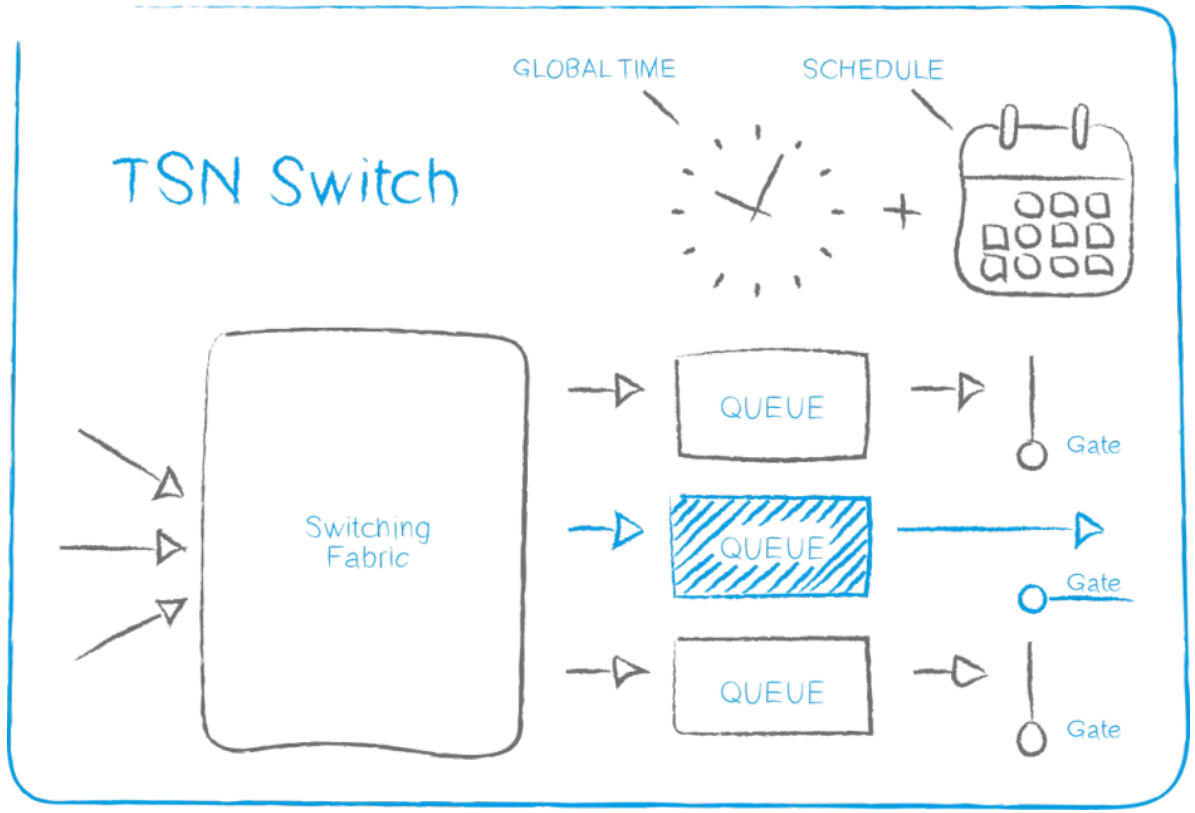
What was still missing?

- Guaranteed lower latencies; searching for the lowest possible latency
- Error detection and isolation
- Redundant communication
- Redundant clock synchronization

TSN introduction

What is a Time-Sensitive Network (TSN)

Time Sensitive Networking covers a set of Ethernet sub-standards currently defined in the IEEE 802.1 TSN task group

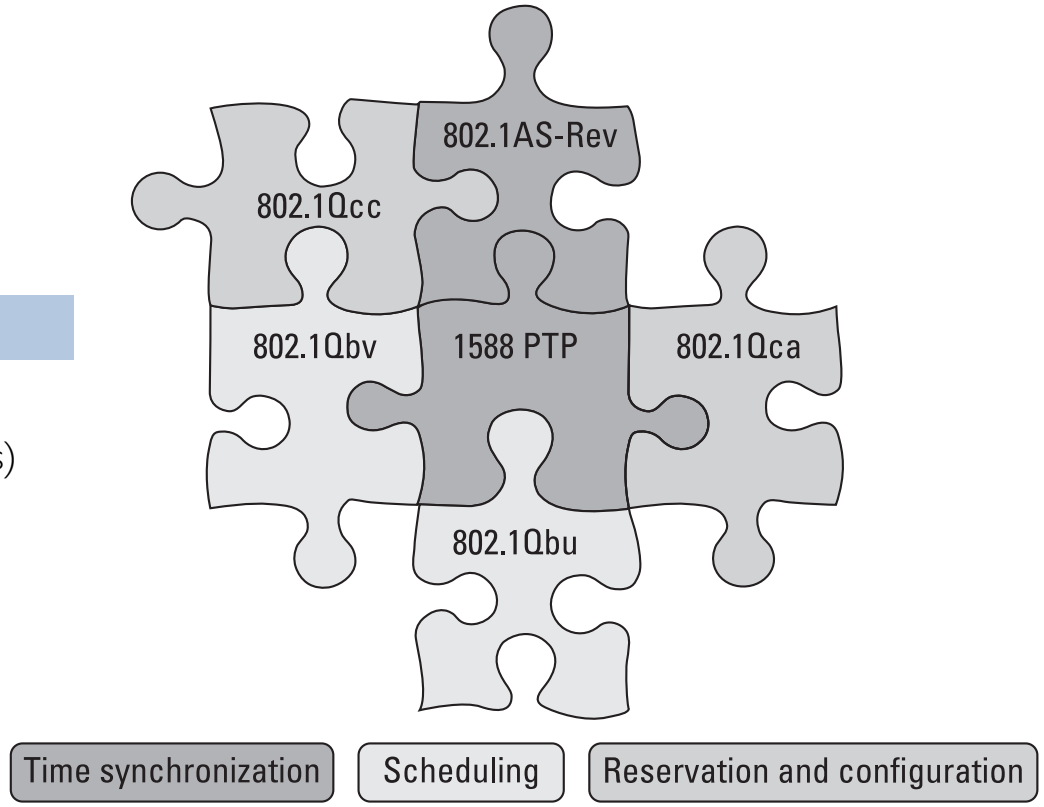


- At the heart of TSN are mechanisms that provide **time synchronization** for networked devices and **scheduled** forwarding of defined traffic flows through the network.
- Through time synchronization and scheduling, TSN delivers deterministic communication over standard Ethernet, thereby enabling the **convergence** of critical control traffic with data traffic over one infrastructure without the need for gateways or proprietary solutions.

Time-Sensitive Network (TSN)

IEEE TSN task group - collection of sub-standards that enhance 802 Ethernet with real-time capabilities

Standard	Description
802.1ASrev	Timing & Synchronization
802.1Qbv	Enhancements for Scheduled Traffic (Timed Gates for Egress Queues)
802.1Qbu	Frame Preemption
802.1Qca	Path Control and Reservation
802.1Qcc	Central Configuration Management
802.1Qci	Per-Stream Time-based Ingress Filtering and Policing
802.1CB	Redundancy, Frame Replication & Elimination



New TSN standards

- TSN: Group renamed itself from “Audio/Video Bridging” to “Time Sensitive Networking”

- Guaranteed lower latencies; searching for the lowest possible latency
  - 802.1Qbv: A priori defined, pre-scheduled communication
  - 802.1Qbu: Frame preemption
  - 802.3br: Preemption required changes in MAC
  - P802.1Qcr: Asynchronous traffic shaping

New TSN standards

- Error detection and containment
  - 802.Qci-2017: Monitoring of pre-defined “contract” between a data flow and the network
  - Meaning of “Contract” depends on the scheduling policy
- Redundant communication
  - 802.1CB-2017: Frame replication and elimination
- Redundant clock synchronization
  - 802.1AS-2020: updates to the clock synchronization standard to support backup masters and multiple time domains

Security standards

- 802.1 has developed three main standards for secure Ethernet communication

- 802.1AE (“MACsec”): integrity and, optionally, privacy, is ensured by symmetric key crypto (128 and 256 bit keys are supported)

- 802.1X:
  - Port authentication
  - Key agreement protocol

Security standards

- 802.1AR (Secure Device Identity):
  - Device identity based on public key crypto (elliptic curves)
  - Public Key Infrastructure and certificates

- These standards, in particular MACsec and Secure Device Identity, require parallel crypto logic on chip
  - Challenging in terms of cost, power, and size
  - Automotive industry has started to look at how to secure Ethernet communication and which portions of the 802.1 security standards apply

Many nonstandard solutions before TSN (still in use)

- Ethernet POWERLINK (Master/Slave protocol; open source)
- EtherCAT (2-port switch in each node; cut-through forwarding
  - Beckhoff automation (IEC 61158)
- PROFINET (TDMA)
  - Siemens
- AFDX (static configuration, traffic shaping) (Airbus)
- TTEthernet by TTTech (TDMA at message level, and rate-constrained traffic class).

TSN Profiles
for Various Application Areas

An IEEE 802.1 TSN Profile specification

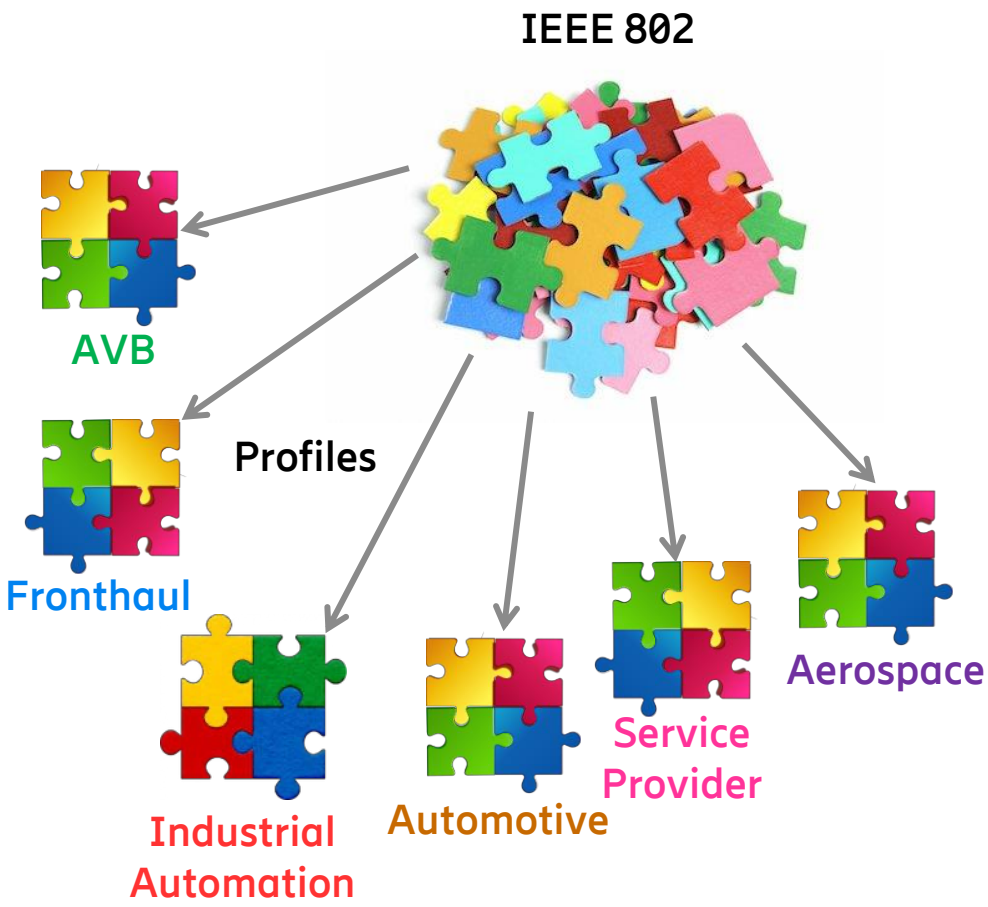
- Selects features, options, defaults, protocols, and procedures

Published IEEE 802.1 TSN profile standards:

- IEEE Std 802.1BA for Audio-Video Bridging (AVB) networks
- IEEE Std 802.1CM TSN for Fronthaul
- IEEE Std 802.1CMde Amendment on enhancements

Ongoing IEEE 802.1 TSN profile projects:

- IEC/IEEE 60802 TSN Profile for Industrial Automation
- P802.1DG TSN Profile for Automotive In-Vehicle Ethernet Communications
- P802.1DF TSN Profile for Service Provider Networks
- P802.1DP / AS6675 TSN Profile for Aerospace onboard Ethernet



Traffic Shaping

Motivation for traffic shaping

- A basic best-effort Ethernet network provides variable service (varying packet delivery latency and sometimes packet get dropped)
- Why does it happen?
  - Overbooking / Overloading. Sources are pushing data that is greater than the capacity of the network elements (links, buffers)
  - Traffic jams due to bursts of traffic from sources
  - Traffic piles up at network “intersections” (i.e., switches/bridges) to go out on the same port

Prioritization

- Prioritize Traffic: Handle packets carrying deterministic traffic in high priority output queues
  - Implement a range of output queues from low- to high-priority on each output port.
  - Prioritized packets are sent even when lower-priority packets are waiting.
  - A burst of high-priority traffic interrupts lower-priority traffic.
  - Good for high-priority traffic. ***Bad/unfair*** to lower-priority traffic.

Contracts and reservations

- Use the privilege of prioritization according to an agreed “contract”
  - Each prioritized source agrees to a contract (aka, a “reservation”) with the network that provides service acceptable to that source. This agreement could be made dynamically or be “engineered” into the network ahead of time.
  - Reservation ensures that prioritized traffic won’t overwhelm best-effort (low-priority) traffic.
  - Traffic shaping is at the heart of a reservation

Shaping approaches

- Credit Based Shaper (CBS): Provides a smooth stream of packets within a maximum data rate
  - Also called FQTSS (Forwarding and Queuing for Time Sensitive Streams) in 802.1Qav, now part of 802.1Q-2014
- 802.1Qbv: Time Aware Shaper (TAS): Provides access to queue at specified times
- 802.1Qcr: Asynchronous Traffic Shaper (ATS): Provides immediately delivery of packets, up to a specified burst size, and within a maximum data rate

Traffic Shaping

Audio/Video Bridging (AVB)

Audio/Video Bridging (AVB)

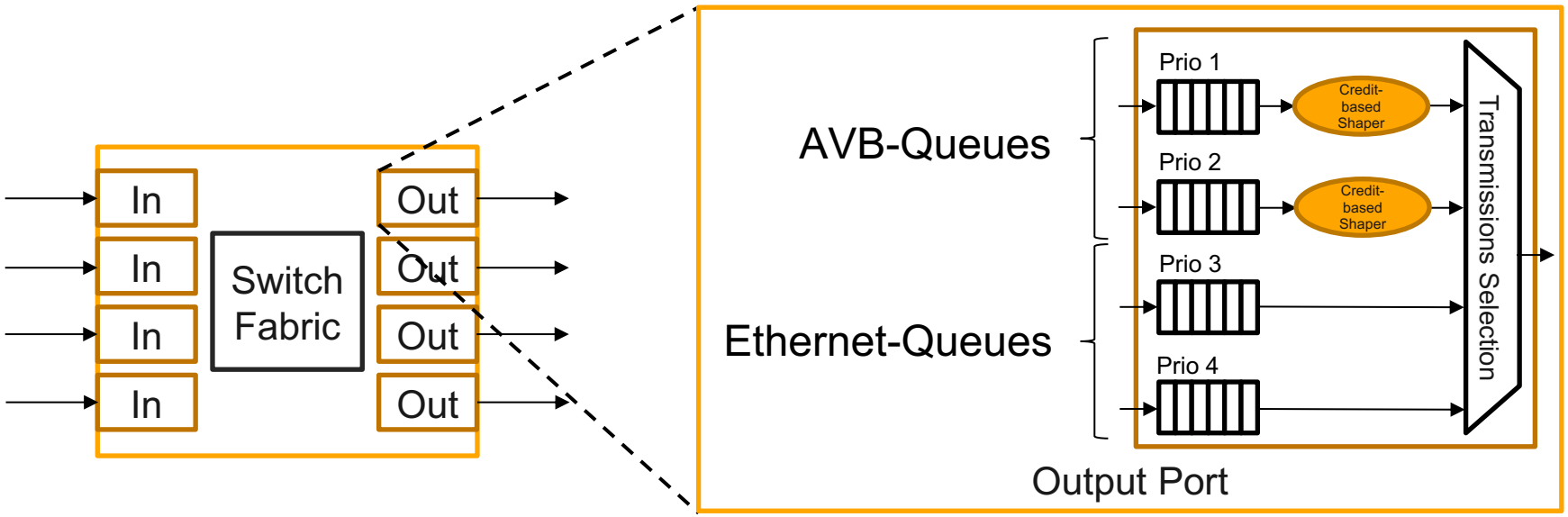
- Main drivers:
  - synchronized audio and video applications on Ethernet
  - Plug and play
- IEEE 802.1AS: Plug and play Clock synchronization
- Amendments to IEEE 802.1Q:
  - Stream reservation protocol (admission control)
  - Credit-based traffic shaping (guaranteed bandwidth across priority levels; no starvation; zero congestion loss / no dropped packets)
- IEEE 802.1BA: Umbrella document

First approach: AVB

- 2 ms maximum delay
  - the maximum delay between a musician doing “something” and hearing that same “something” is 10 ms
  - the transit time of sound from monitor speakers to the musician, plus digital signal processing (DSP) delays, plus mixer delays, plus more DSP delays uses up 8 ms
  - network gets 2 ms
- maximum synchronization error less than 10 microseconds

Quality of Service

- Credit based shaper delays messages to avoid bursts
  - To avoid buffer overflow and message loss
  - To guarantee some bandwidth to lower priority traffic



AVB credit-based shaper

- Space out the high priority stream frames as far as possible
- The spaced-out traffic prevents the formation of long bursts of high priority traffic, which typically arise in traffic environments with high bandwidth streams
- Bursts are responsible for significant QoS reductions of lower priority traffic classes
  - Can completely block the transmission of the lower priority traffic for the transmission time of the high priority burst
  - Increases maximum latency of this traffic and thereby also the memory demands in the bridges and end stations.

AVB credit-based shaper

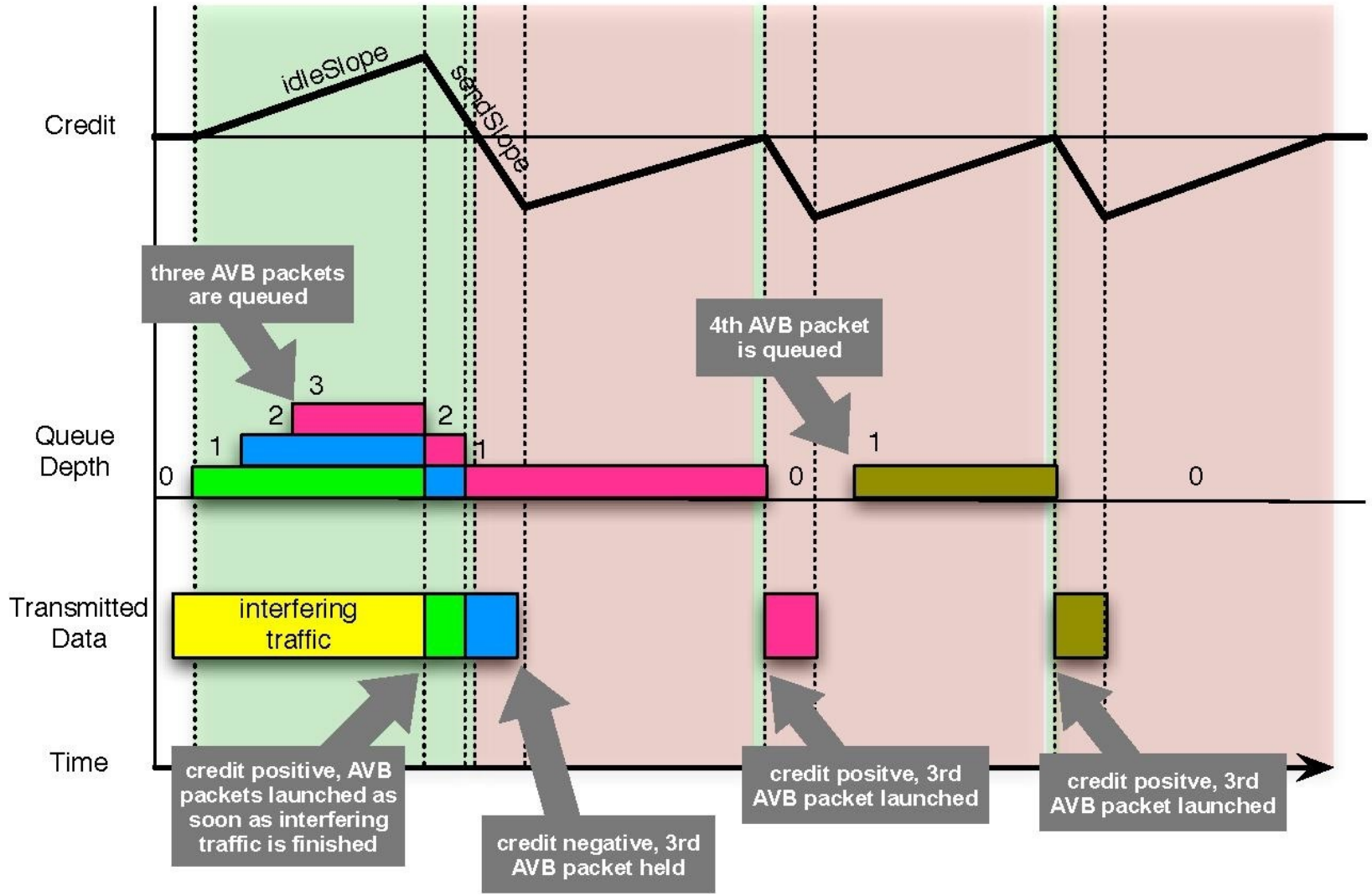
- Long bursts increase the interference time between high priority stream frames from different streams (which arrive from different ports) inside a bridge.
  - This increases the maximum latency of high priority stream frames and again the memory requirements in bridges
- Another task of the shaper is to enforce the bandwidth reservations. This enforces, on the one hand, that every AVB stream is limited to its reserved bandwidth in the talker, and, on the other hand, that the overall AVB stream bandwidth of each port (in talker and bridges) is limited to the reserved amount

Operation of credit-based shaper

$$\text{idleSlope} = \frac{\text{reservedBytes}}{\text{classMeasurementInterval}}$$

$$= \text{reservedBandwidth}.$$

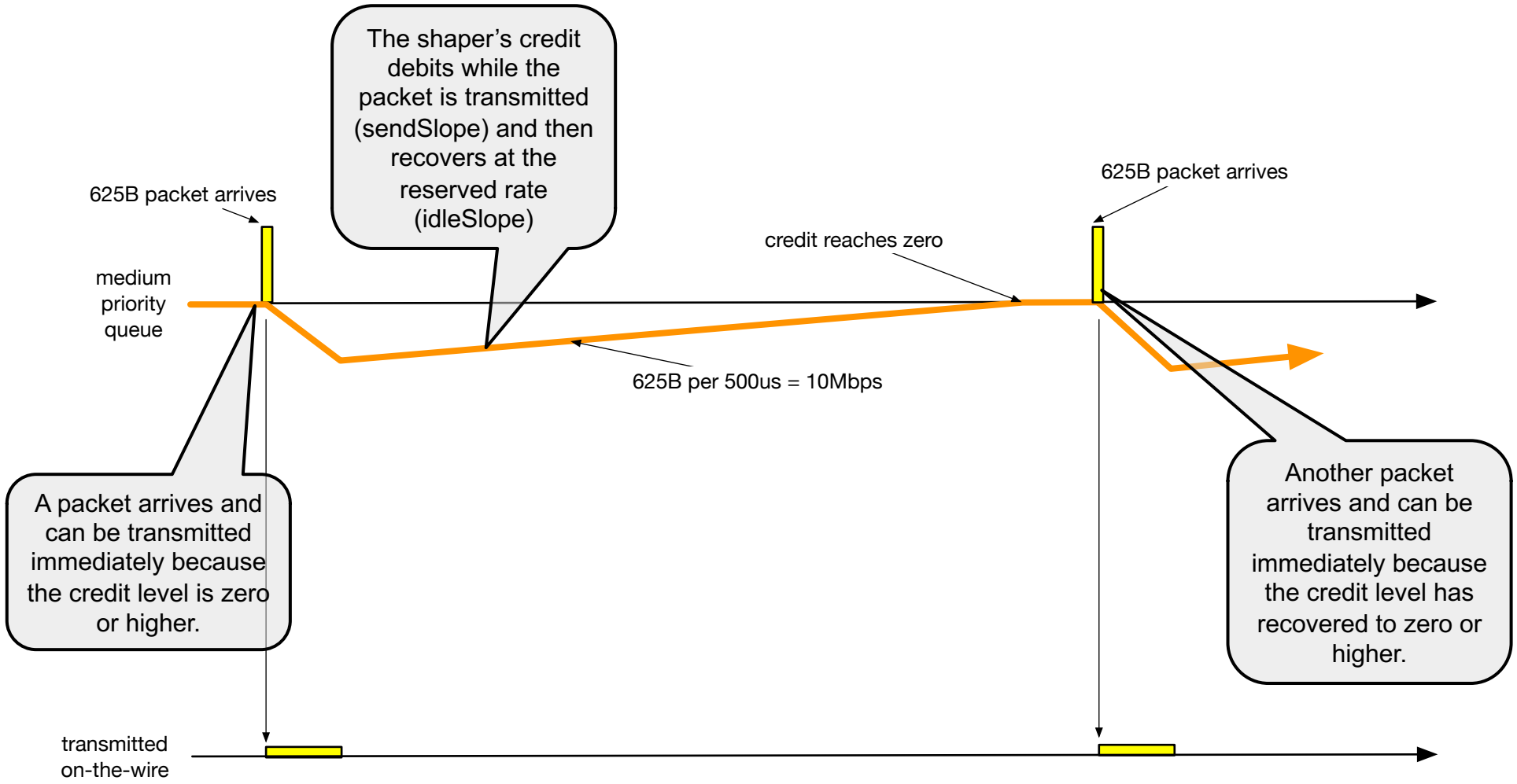
$$\text{sendSlope} = \text{idleSlope} - \text{portTransmitRate}$$



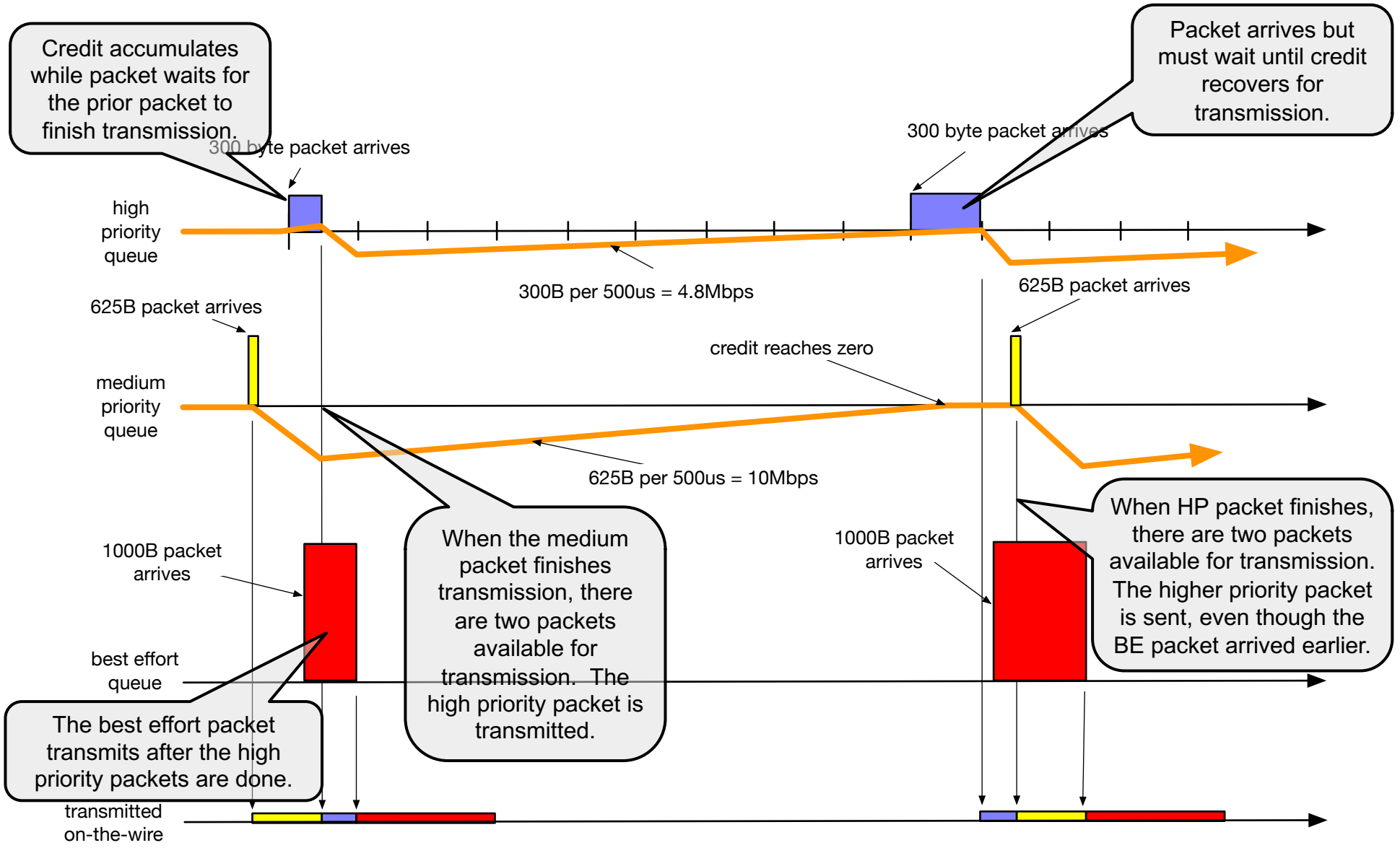
Credit calculation rules

- If there is positive credit but no frame to transmit, the credit is set to zero
- During the transmission of a frame, the credit is reduced with the send slope.
- If the credit is negative and no frame is in transmission, the credit is accumulated with the idle slope until zero credit is reached.
- If there is a frame in the queue that cannot be transmitted because another frame is in transmission, the credit is accumulated with the idle slope. In this case, the credit is not limited to zero.

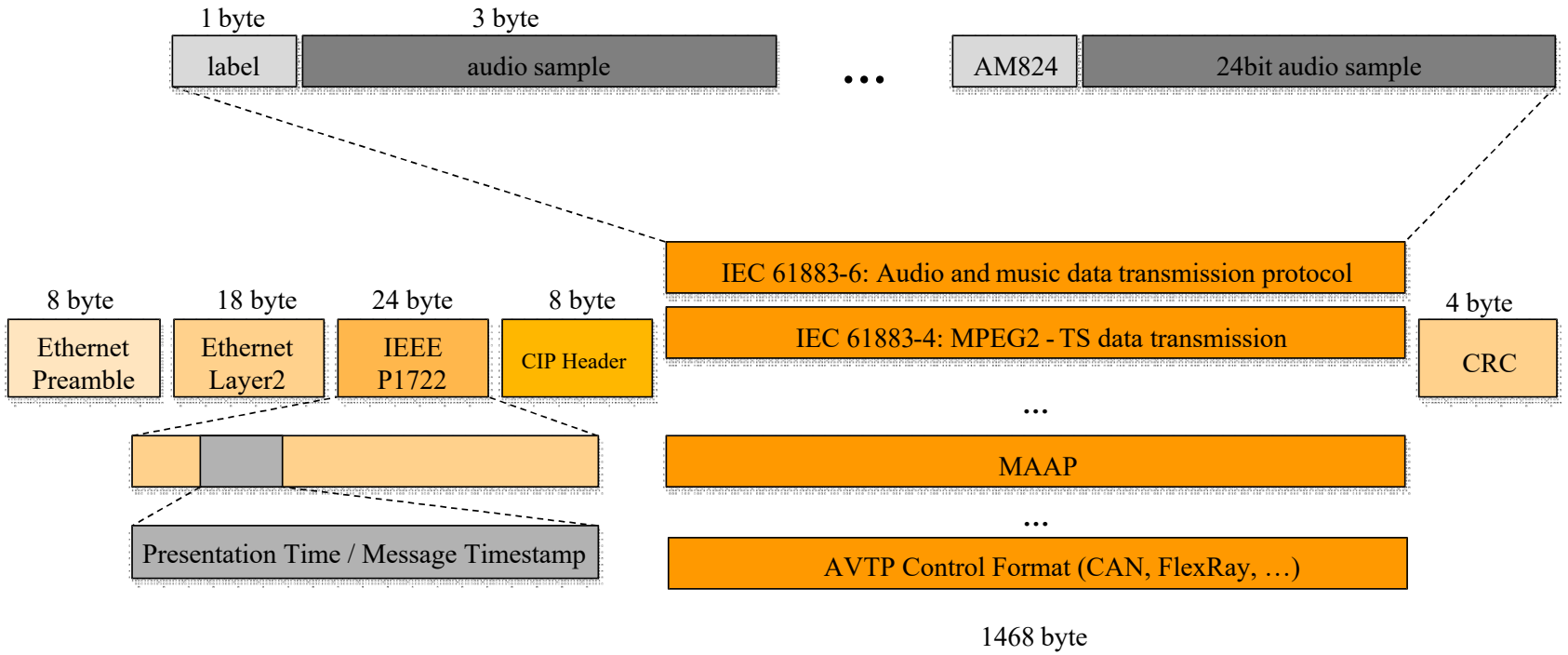
Credit-based shaping (802.1Q-2014 §34)



CBS with multiple queues



IEEE 1722: AVTP (AVB Transport Protocol)

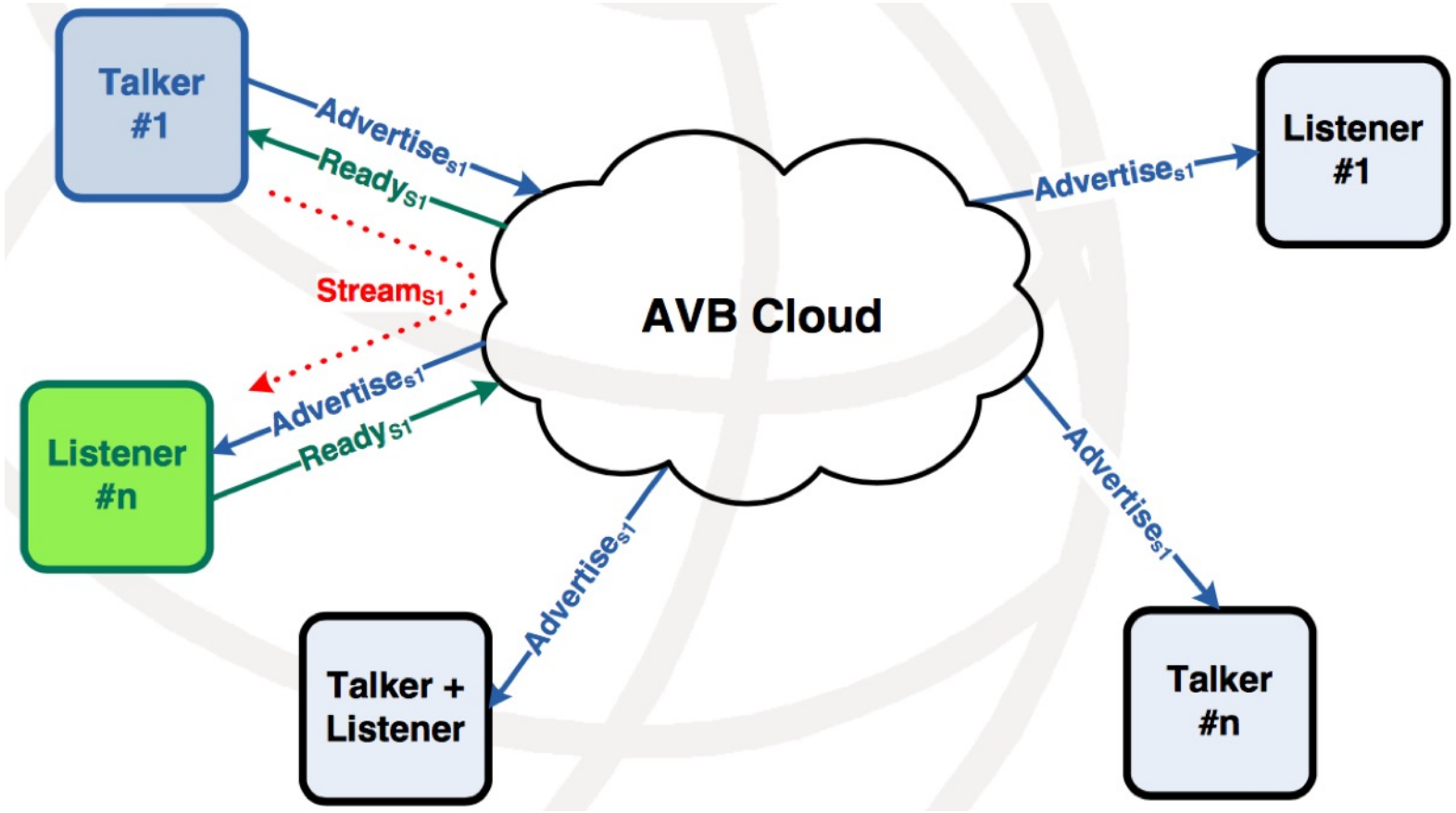


- AVTP control format added in 2016
- EtherType: 0x22F0

Stream reservation protocols

- 802.1Qat (now rolled into 802.1Q-2014 and later revisions)
- One of the core protocols of AVB
- Allows sources (talkers) to advertise streams to sinks/users (listeners) through the network
- Also allows to withdraw
- Gives end stations the tool to automatically configure the network to deliver content to the right users
- Multiple Stream Registration Protocol (MSRP)
- Multiple VLAN Registration Protocol (MVRP)
- MSRP and MVRP are in turn based on the Multiple Registration Protocol (MRP)

SRP Advertise and Ready frames



Talker advertise message format

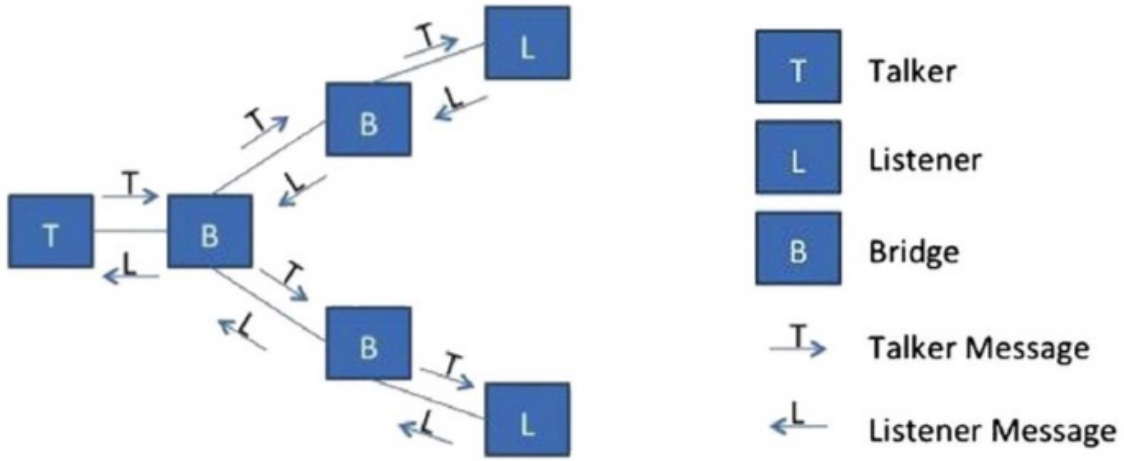
- stream ID (MAC address associated with the talker plus a 16 bit ID)
- stream DA
- VLAN ID
- priority (determines traffic class)
- rank (emergency or nonemergency)
- traffic specification (TSpec): max frame size; maximum number of frames per class interval
- accumulated latency

Forwarding of stream announce

- Talker send advertise message
- Each switch/bridge evaluates whether reservation can be made
  - whether sufficient bandwidth is available on each port
  - whether sufficient memory is available to guarantee no packet loss
  - reservation is not made; only when receiving listener message
  - forwards the talker message, after updating the accumulated the hop count

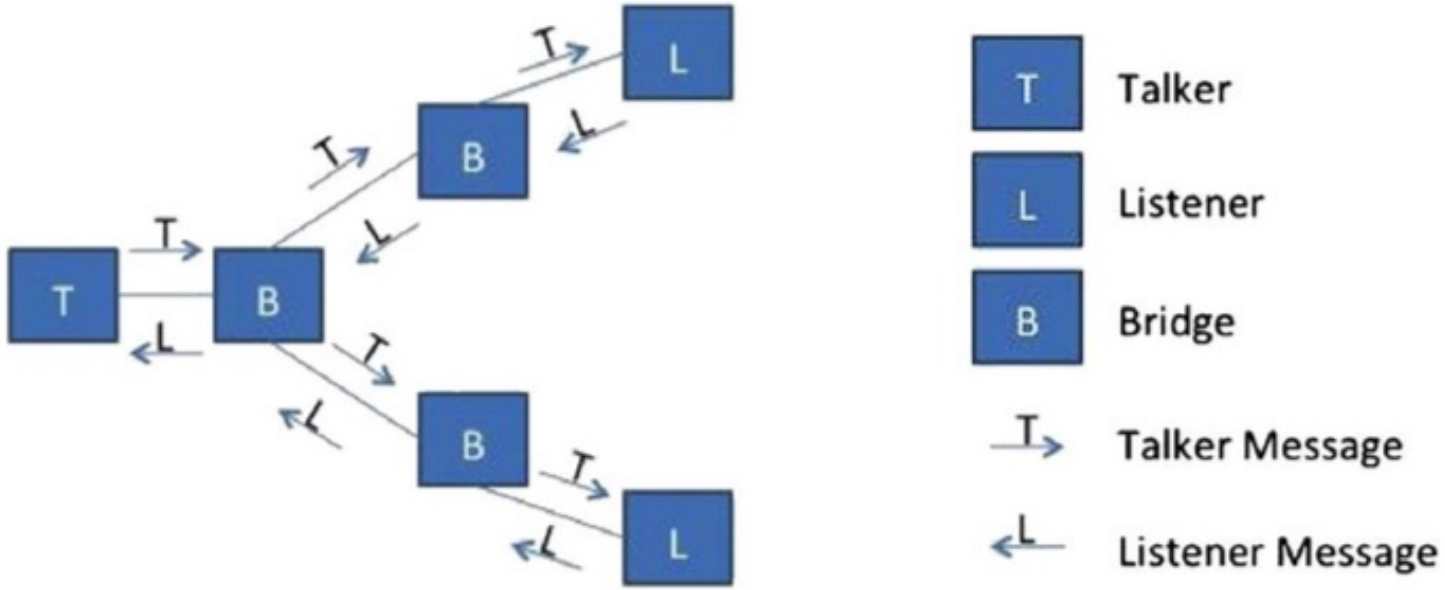
Reservation failures

- If any device on the path from talker to listener determines that the stream cannot be supported, it changes the type of the message from talker advertise to talker failed
- Then adds additional information to the message
  - bridge ID where the failure occurred;
  - reservation failure code to identify the reason for the failure.
  - Allows network engineer to pinpoint the location of the issue



Listeners

- Listeners send listener message if they want the stream
- Listener communicates the status of the stream by sending either a listener ready if it received a talker advertise or a listener asking failed if it received a talker failed



Reservations made

- When bridges receive a listener ready (or ready failed) message for a valid stream on a given port, they make a reservation on that port
  - update the bandwidth on the traffic shaper for the queue

$$\text{idleSlope} = \frac{\text{reservedBytes}}{\text{classMeasurementInterval}}$$

$$= \text{reservedBandwidth.}$$

  - update available bandwidth for the given port
  - adding the port to the forwarding entry for the stream DA

Listener messages propagated back

- Listener message propagated back toward the talker
- Talker receives a listener ready message, it may begin transmitting
- If talker receives ready failed, it knows that at least one listener has requested the stream but the corresponding reservation could not be created

For engineered (static) networks

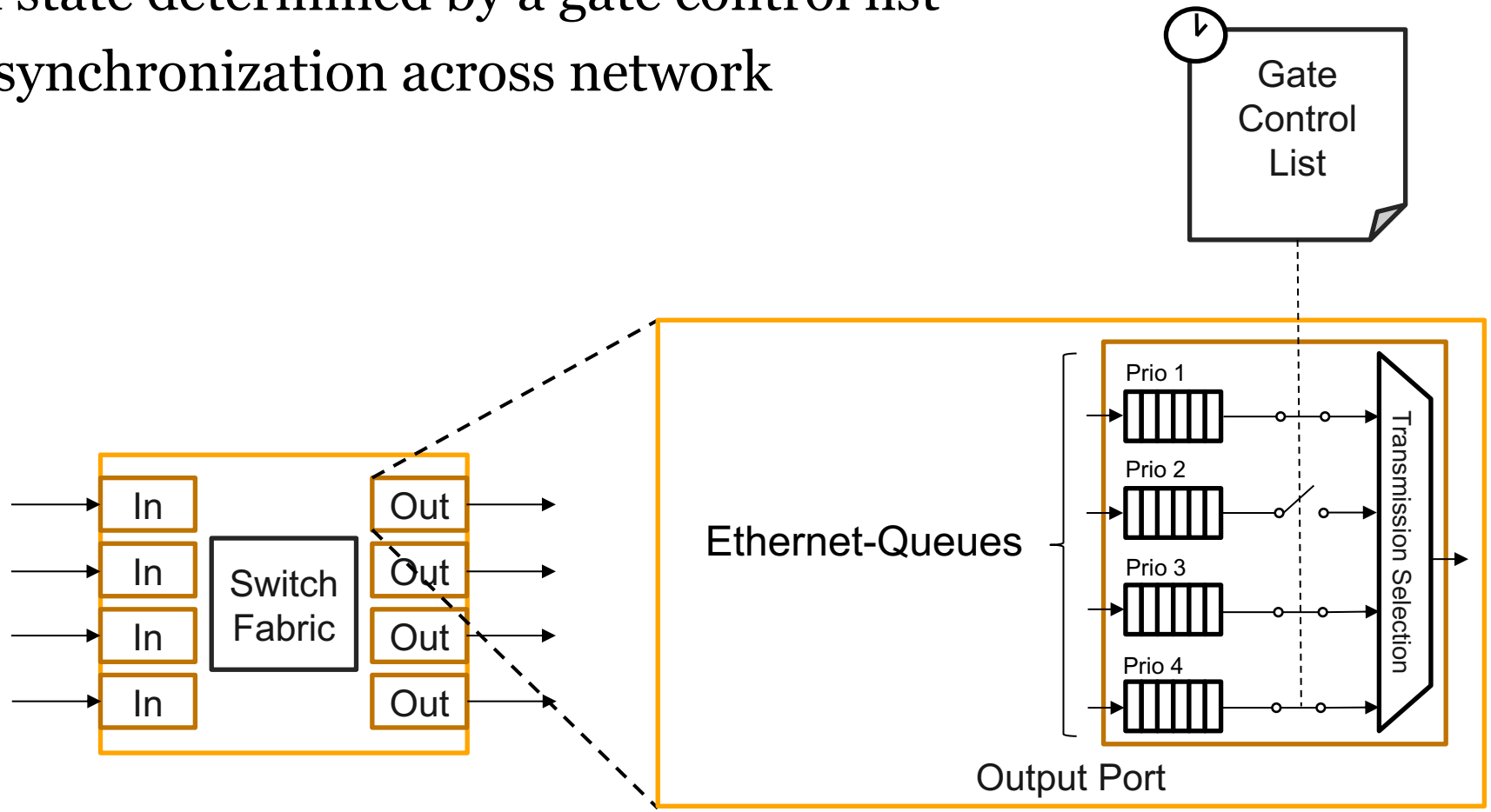
- Use SRP to establish data paths and bandwidth reservations once
  - Then program components with the resulting configuration
- “Manual” static configuration or network design tool

Traffic Shaping

Time-Aware Shaper (TAS)

IEEE 802.1Qbv: Time-Aware Shaper (TAS)

- Time gate on queue
- Open or closed state determined by a gate control list
- Requires time synchronization across network

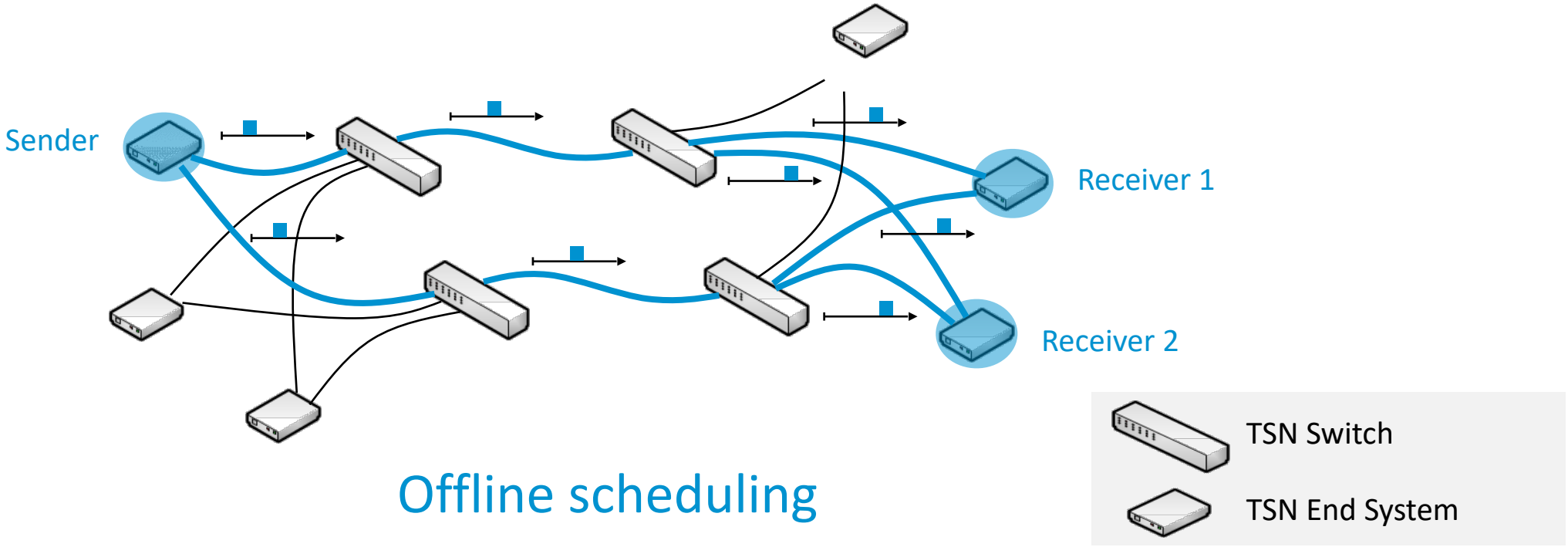


TSN schedule for TAS (Qbv)

The **TSN (Qbv) schedule** defines open and close events for the Gate Control List (GCL) in each output port of every TSN device in the network

The schedule is built off-line taking into account the maximum end-to-end latency, frame length, as well as constraints derived from resources and physical limitations.

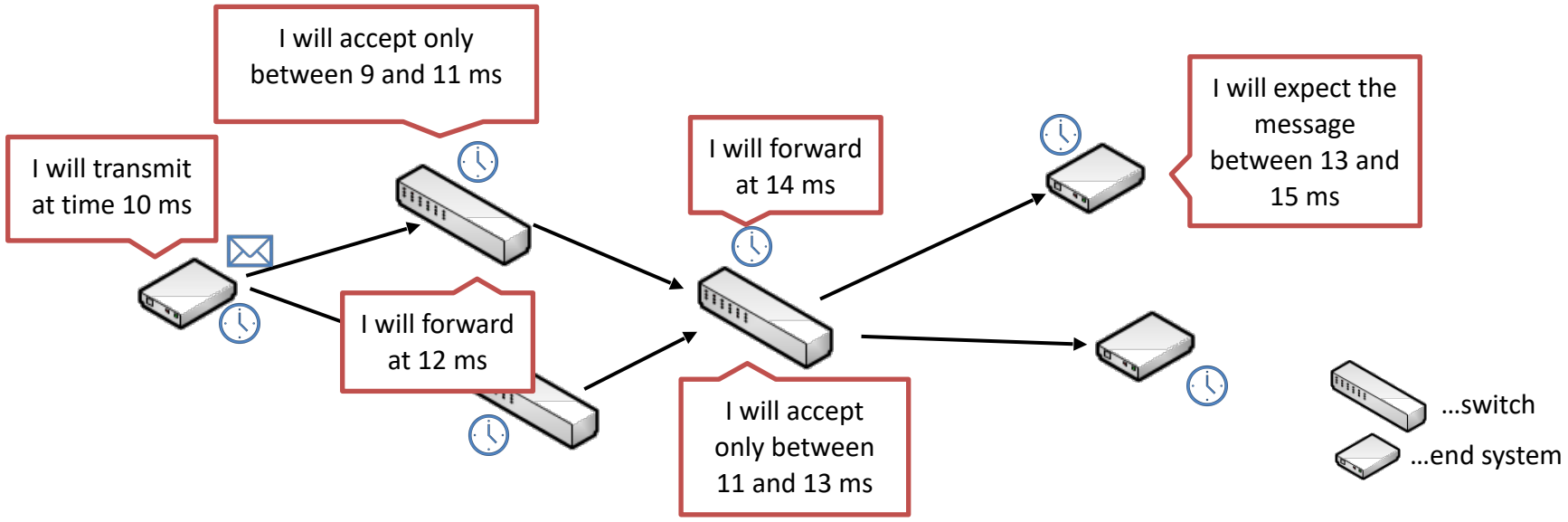
Advantage: the worst-case delays (latency) and the jitters can be minimized via the way the GCL is built



Offline scheduling

Network view

- Time-triggered communication & timing checks in a network with forwarding:
  global schedules & delays apply



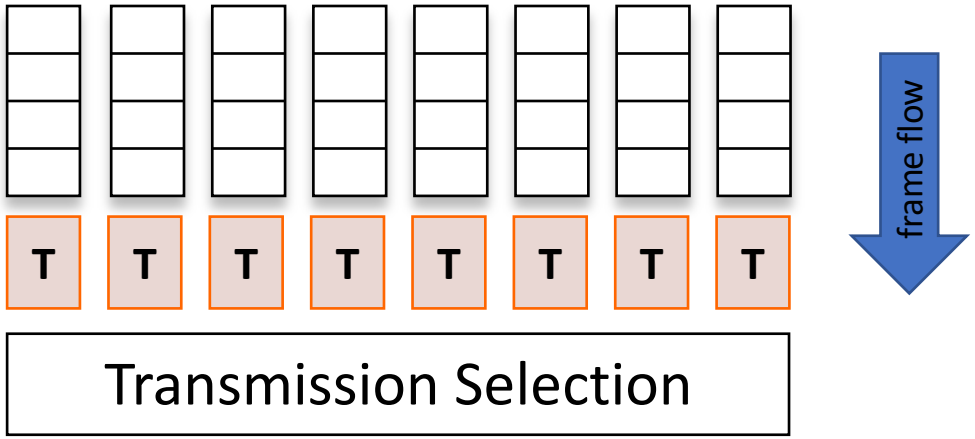
Clock Synchronization

- Time-sensitive communication is built on two principles
  - Synchronized global notion of time
  - Communication schedule
    - Traditionally this is calculated and distributed offline as part of a device configuration

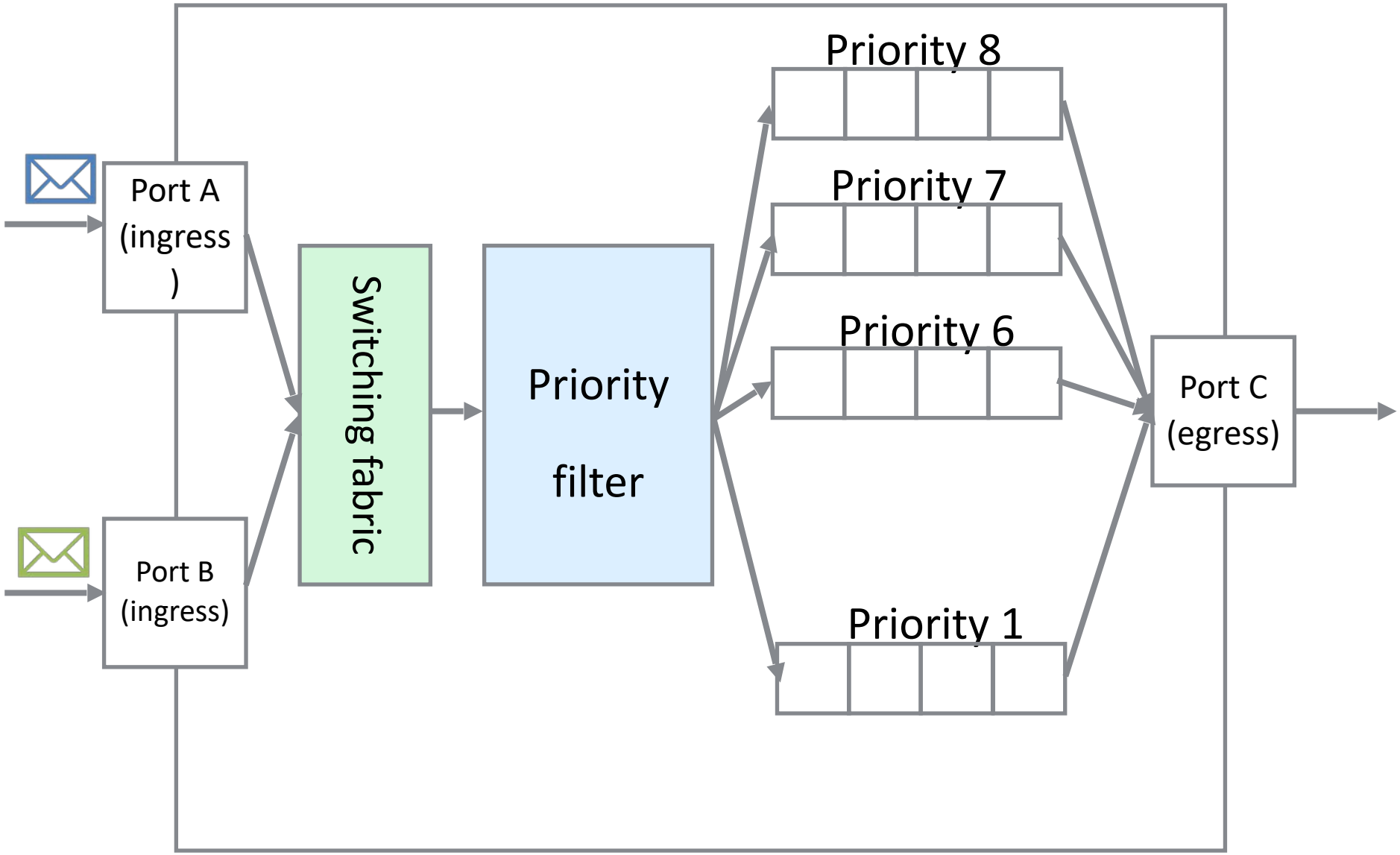
- Synchronization protocols synchronize the local clocks
  - Three important synchronization protocols are:
    - SAE AS6802
    - IEEE 1588
    - IEEE 802.1AS

Scheduled traffic

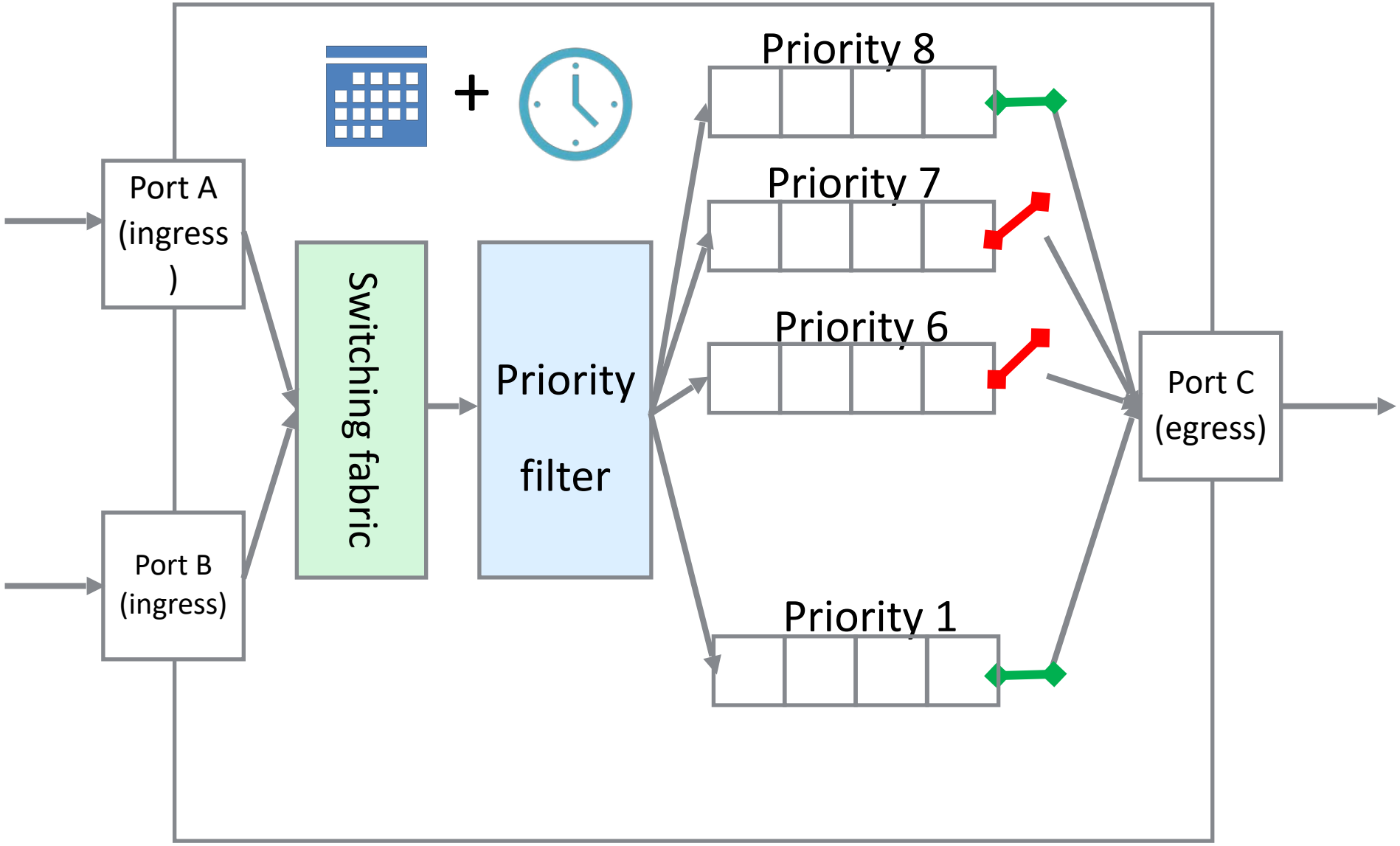
- Reduces latency variation for Constant Bit Rate (CBR) streams, which are periodic with known timing
- Time-based control/programming of the up to 8 bridge queues (802.1Qbv)
- Time-gated queues
- Gate: Open or Closed
- Periodically repeated time-schedule
- Time synchronization is needed



Priority switch



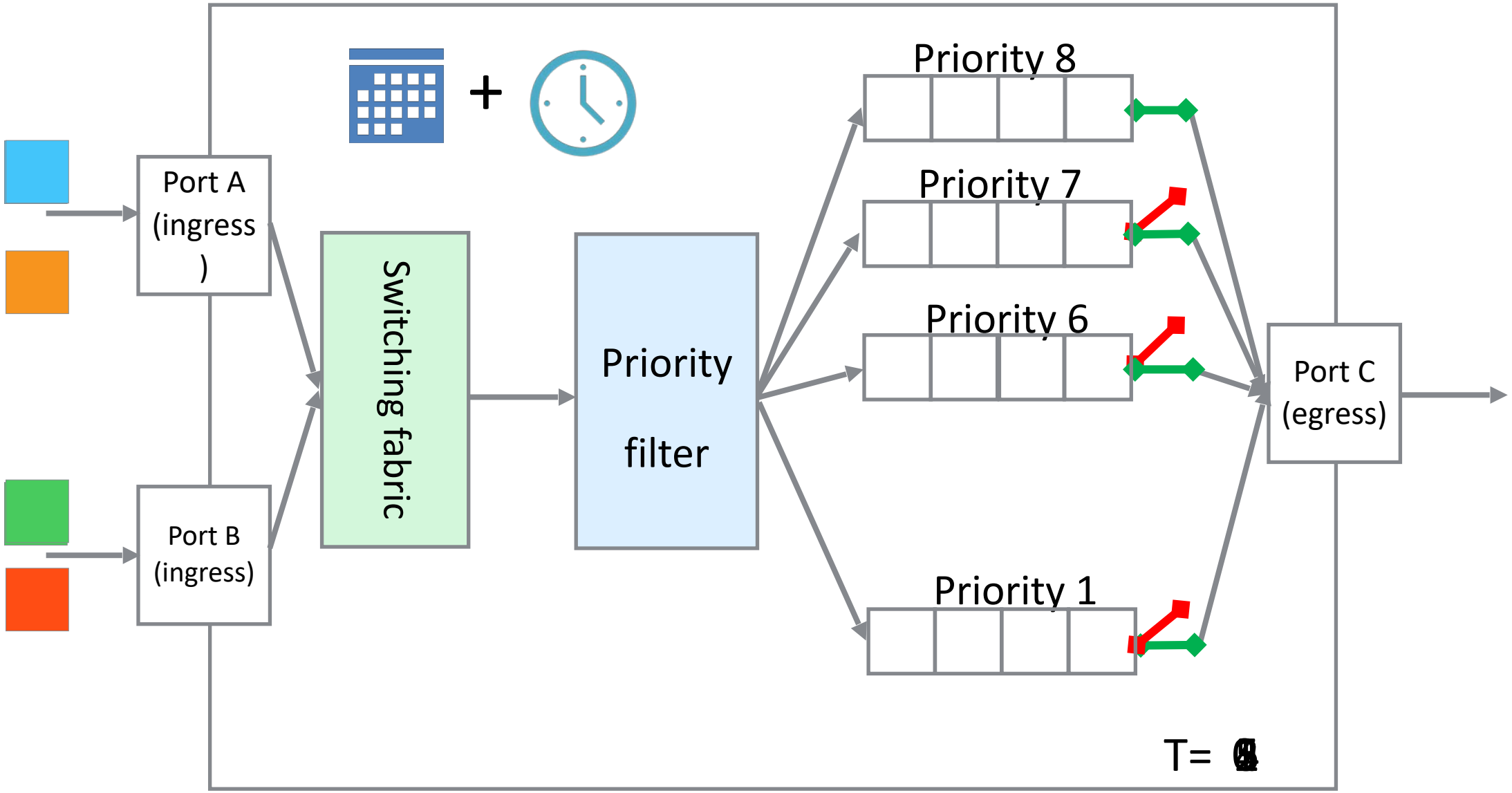
TSN Time-Aware Scheduler (TAS) Switch



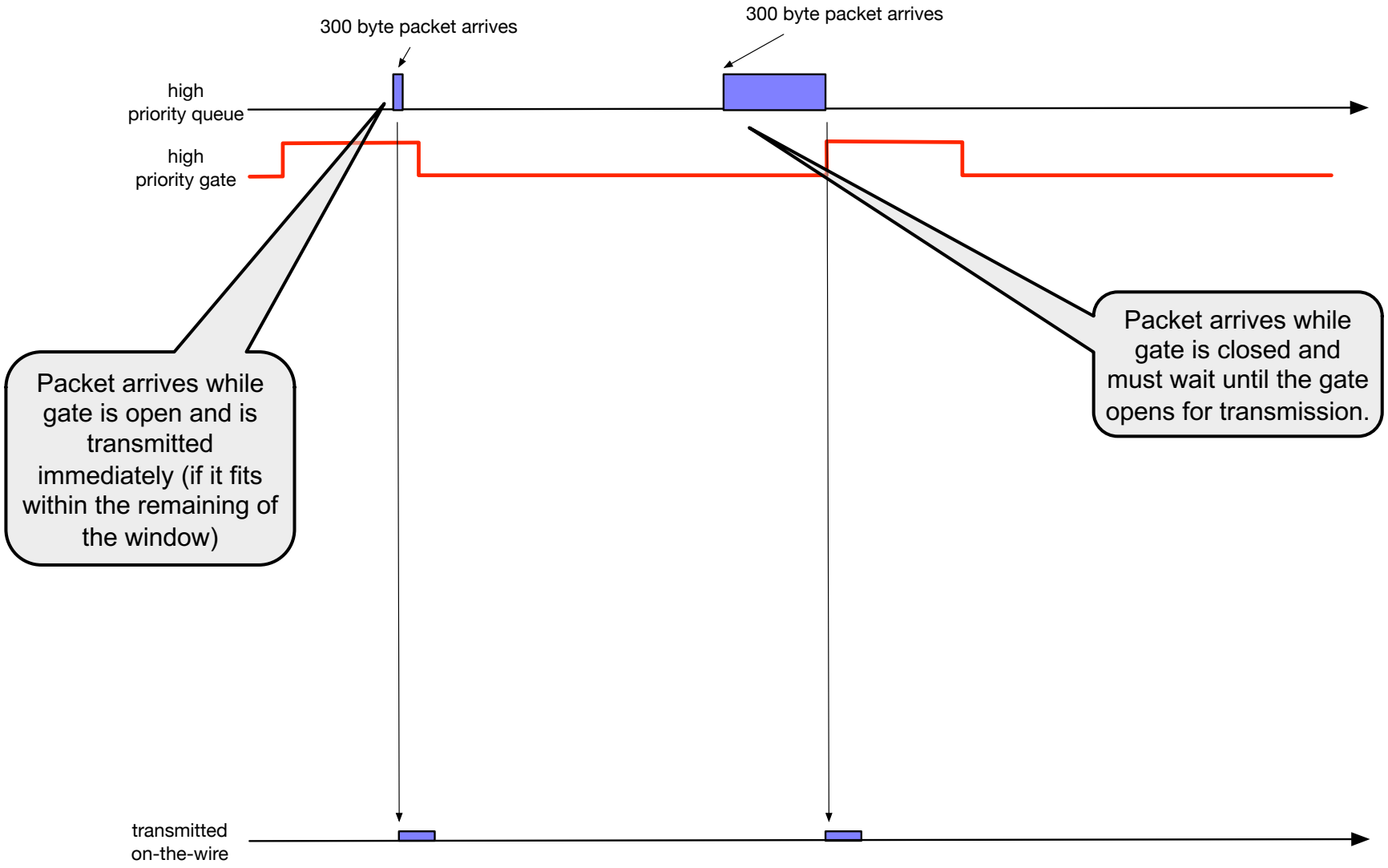
t=0 occoccoc
t=1 oocoooccc
t=5 ccccccccco
t=7 ococcccc
……

Gate **C**ontrol **L**ist (GCL)

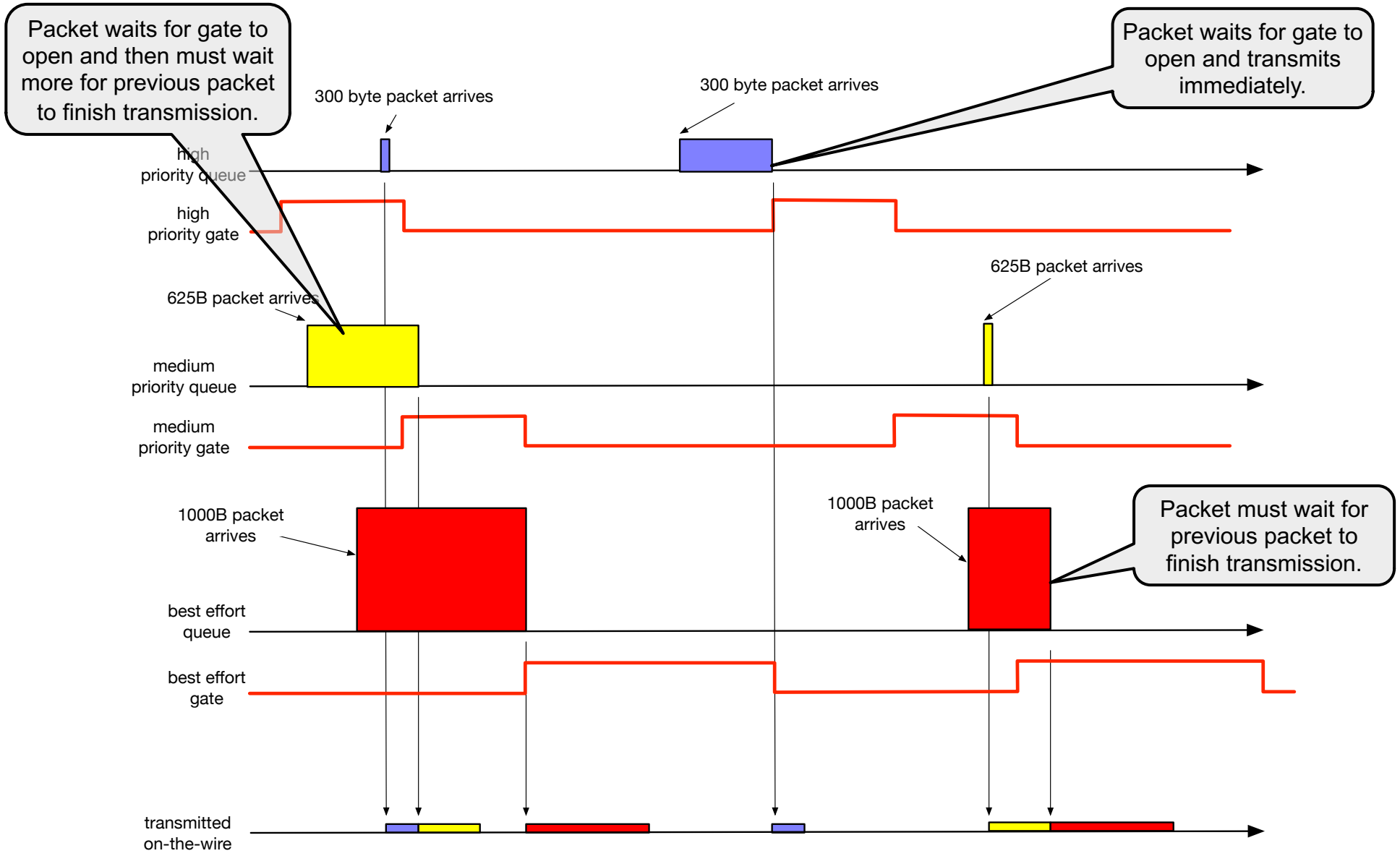
TSN Time-Aware Scheduler Switch



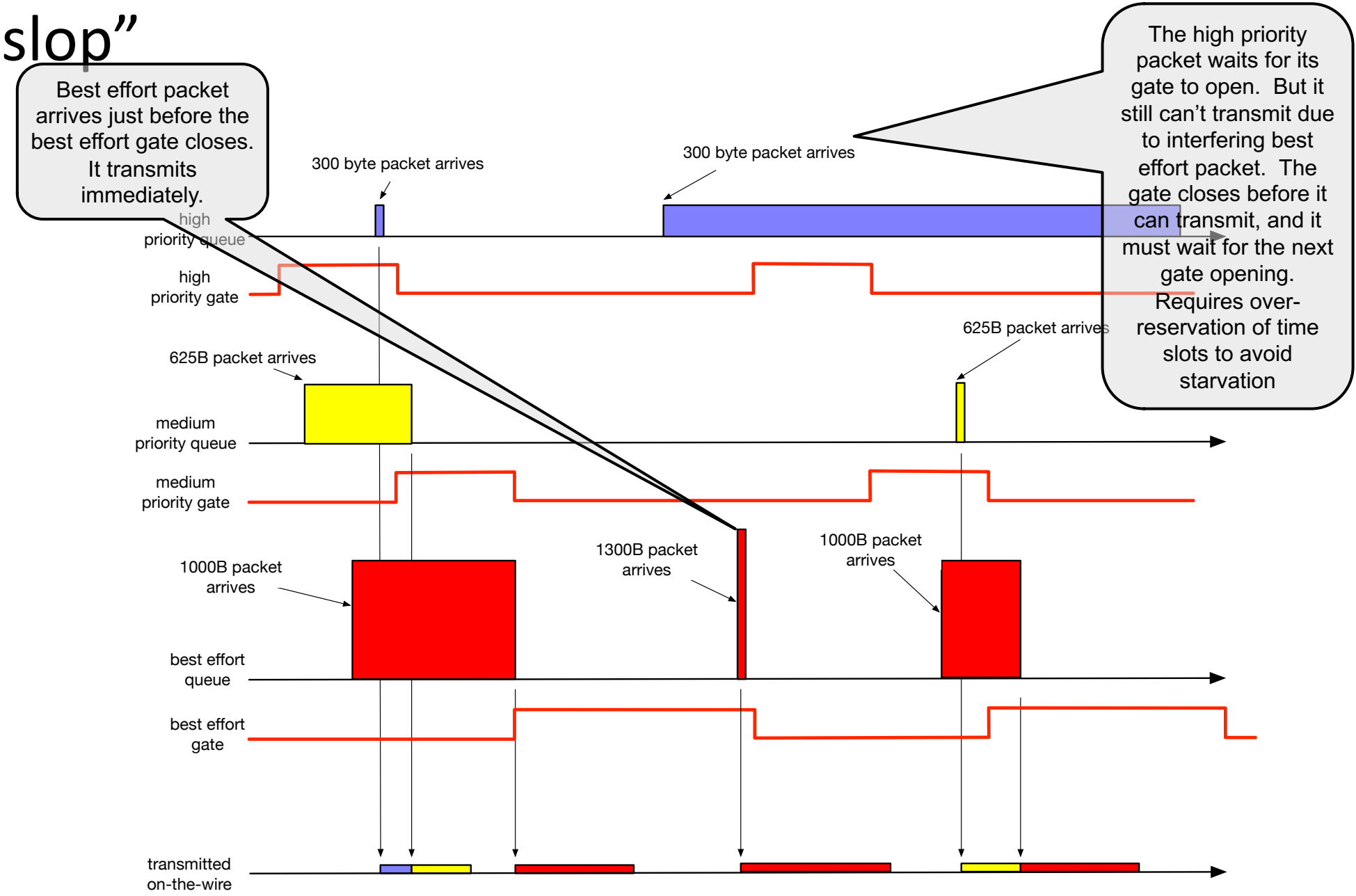
TAS – one queue



TAS – multiple queues



TAS - ”slot slop”



802.1Qbv TAS observations

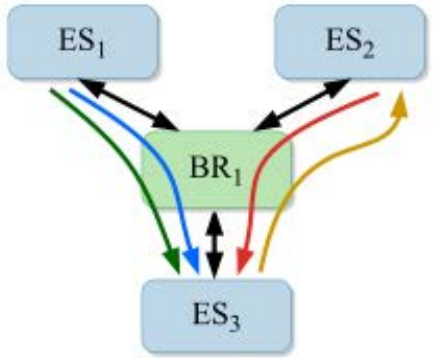
- Pre-defined time access to queues
- Suitable for highly engineered networks
- Suitable for carrying streams with common and regular structure (e.g., sensors that send small packets at very regular periodicity)

802.1Qbv TAS observations

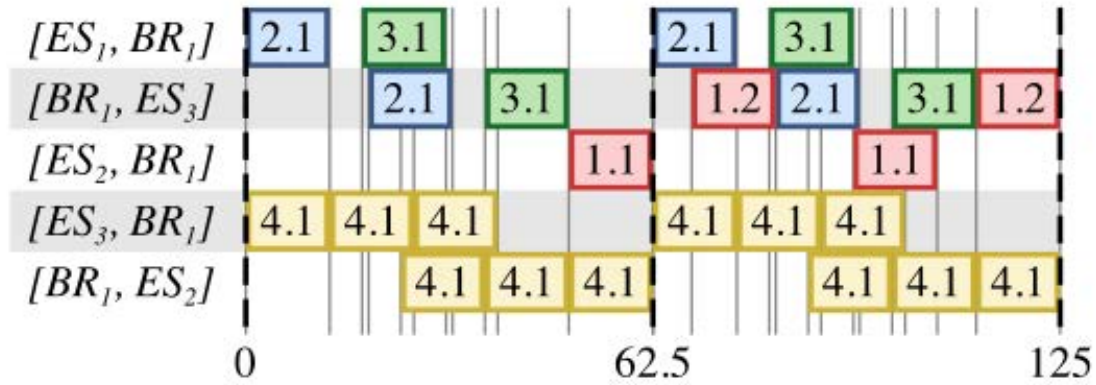
- Engineering the network can be difficult: depending on stream makeup, queue scheduling can be difficult to optimize or create
- Careful engineering to maximize efficiency and avoid “slot slop”
  - Add “guard band” to time slots
  - Eliminate non-engineered traffic
  - Use MACs with frame preemption capability (802.1Qbu-2016 / 802.3br)
- Likely need to synchronize software execution on nodes to avoid missing open Qbv windows

Building the schedules (GCLs) for TAS

Input:
- Network topology:
  - end systems, switches, links.
- TT flows:
  - route, period, data size, deadline.



Output:
- schedule:
  - GCL for each egress port.
  - Assignment of frames to egress port queues.



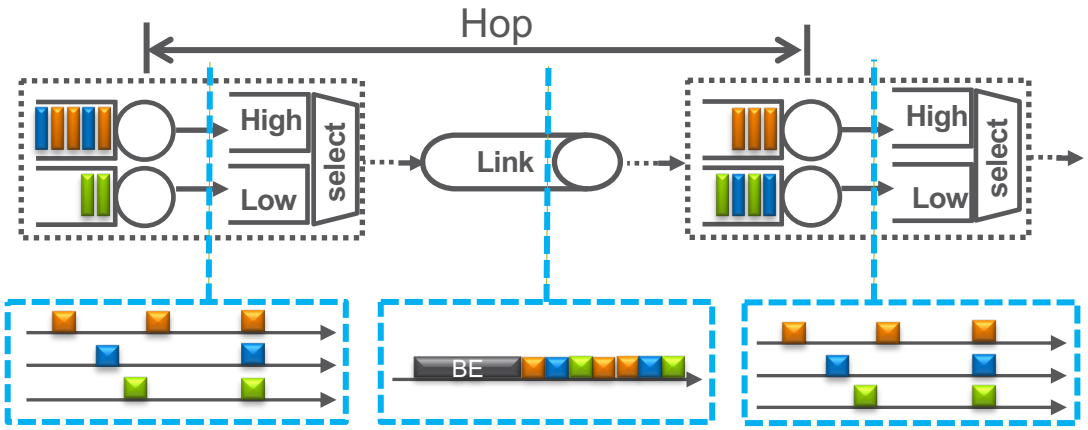
[1] S. S. Craciunas, R. Serna Oliver, and M. Chmelík. Scheduling real-time communication in IEEE 802.1Qbv Time Sensitive Networks. In *24th International Conference on Real-Time Networks and Systems.* IEEE, 2016.

Traffic Shaping

Asynchronous Traffic Shaping (ATS)

802.1Qcr-2020: Asynchronous traffic shaping (ATS)

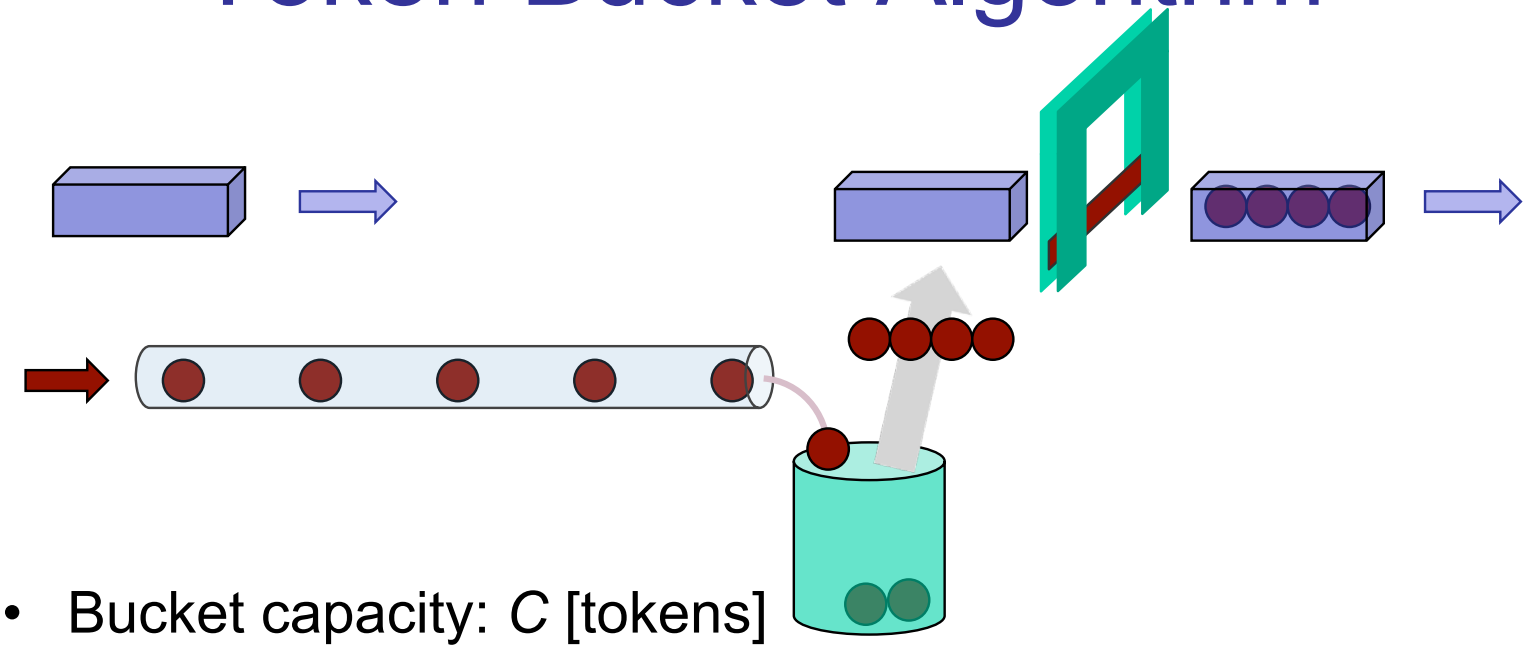
- Zero congestion loss without time synchronization
- Asynchronous Traffic Shaping (P802.1Qcr ATS)
  - Smoothen traffic patterns by re-shaping per hop
  - Prioritize urgent traffic over relaxed traffic



ATS

- ATS is based on the token bucket algorithm
- Shaper has a token bucket that fills with tokens at a committed bit rate, until it reaches the maximum capacity
- Packets arrive at random times and with random size
- Shaper releases packets for transmission scheduling when the bucket holds tokens greater than or equal to the size of the packet, followed by ~~incrementing the bucket size~~ decreasing the number of tokens
- If there is an insufficient number of tokens to release a packet for transmission scheduling after a maximum residence time has elapsed, the shaper discards the packet

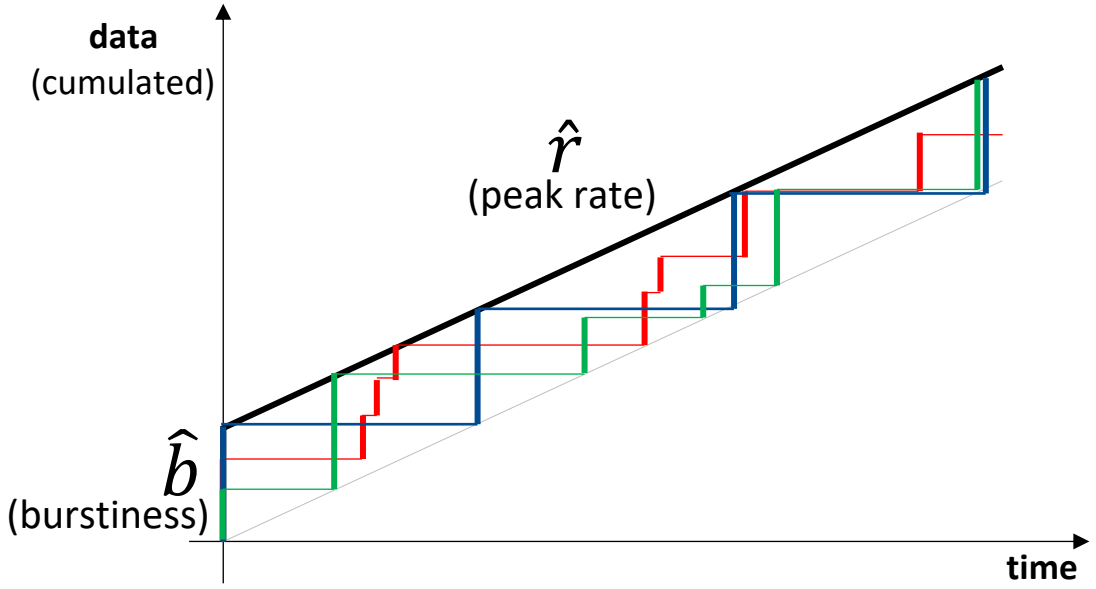
Token Bucket Algorithm



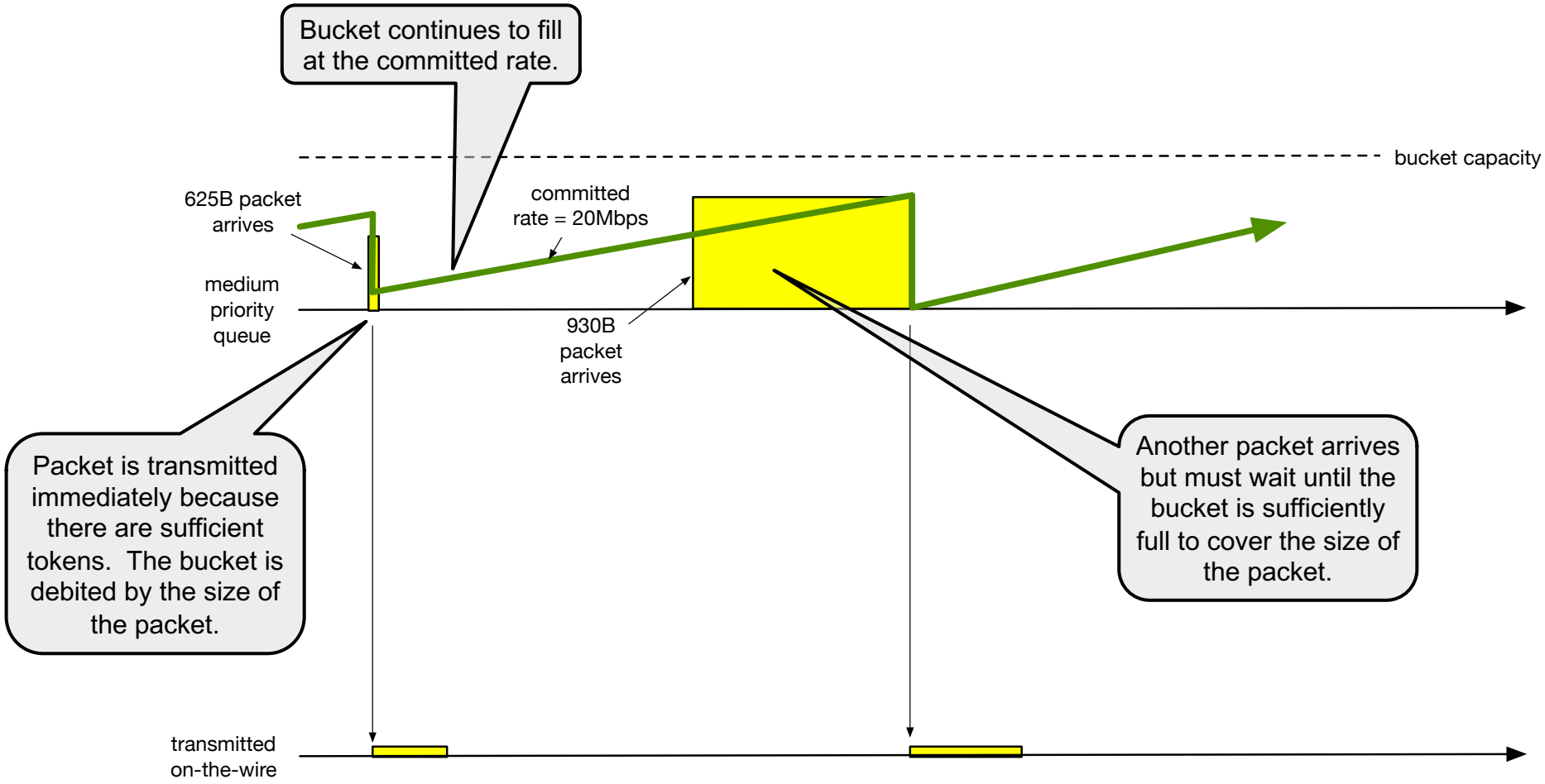
- Bucket capacity: *C* [tokens]
- Token arrival rate: *r* [tokens per second]
- When a packet of *n* bytes arrives, *n* tokens are removed from the bucket and the packet is sent
- If fewer than *n* tokens available, no token is removed

ATS stream traffic model

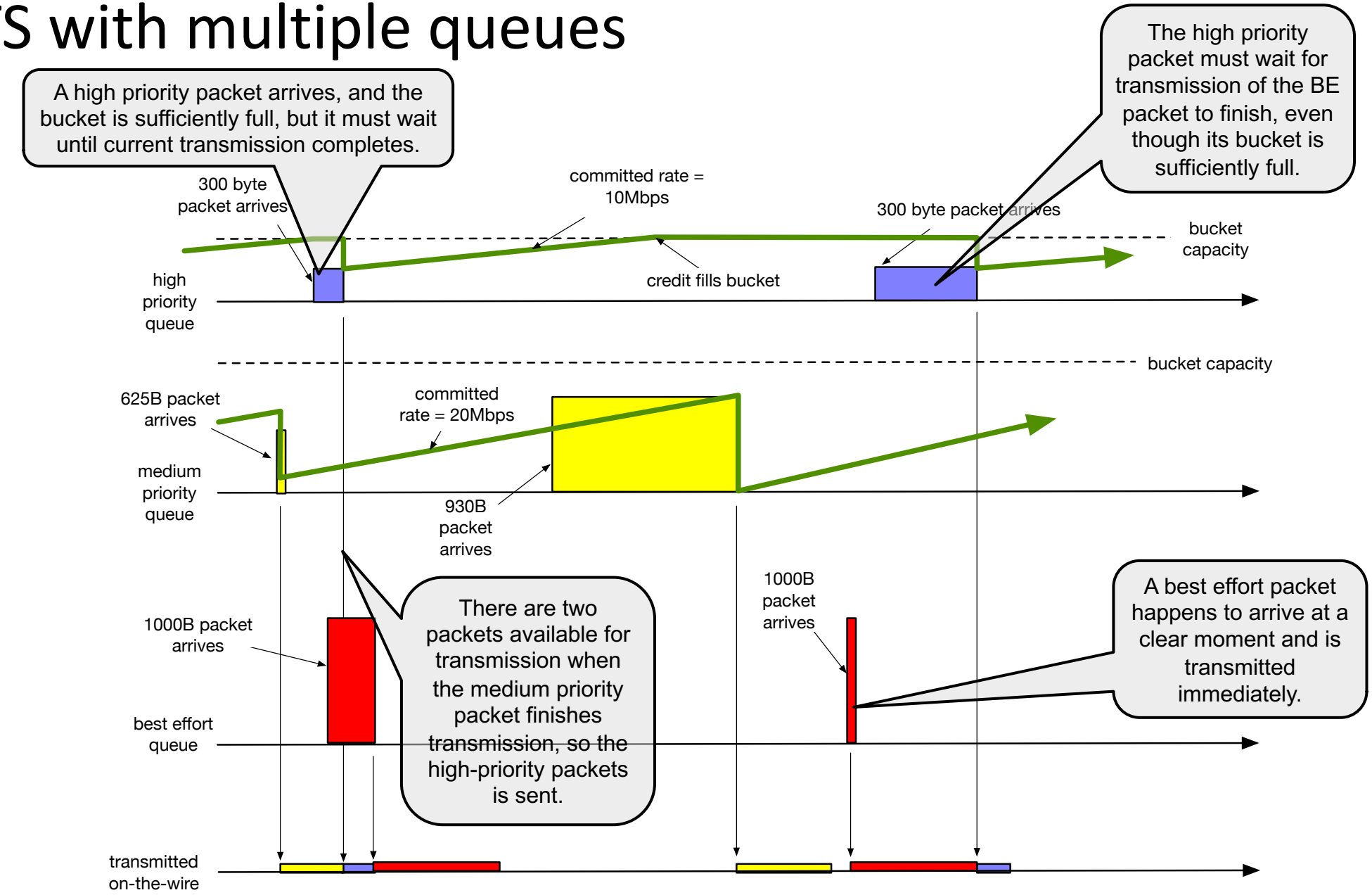
- Each ATS stream is characterized by two parameters, the burstiness b and the rate r.
  The burstiness corresponds to the capacity of the bucket and the rate corresponds to the arrival rate.
- A stream is a sequence of frames, similar to how a task is a sequence of jobs.
- Each colored line in the figure is a stream characterized by its burstiness (b) and rate (r), and each step within that line represents a frame being sent over time.



ATS operation

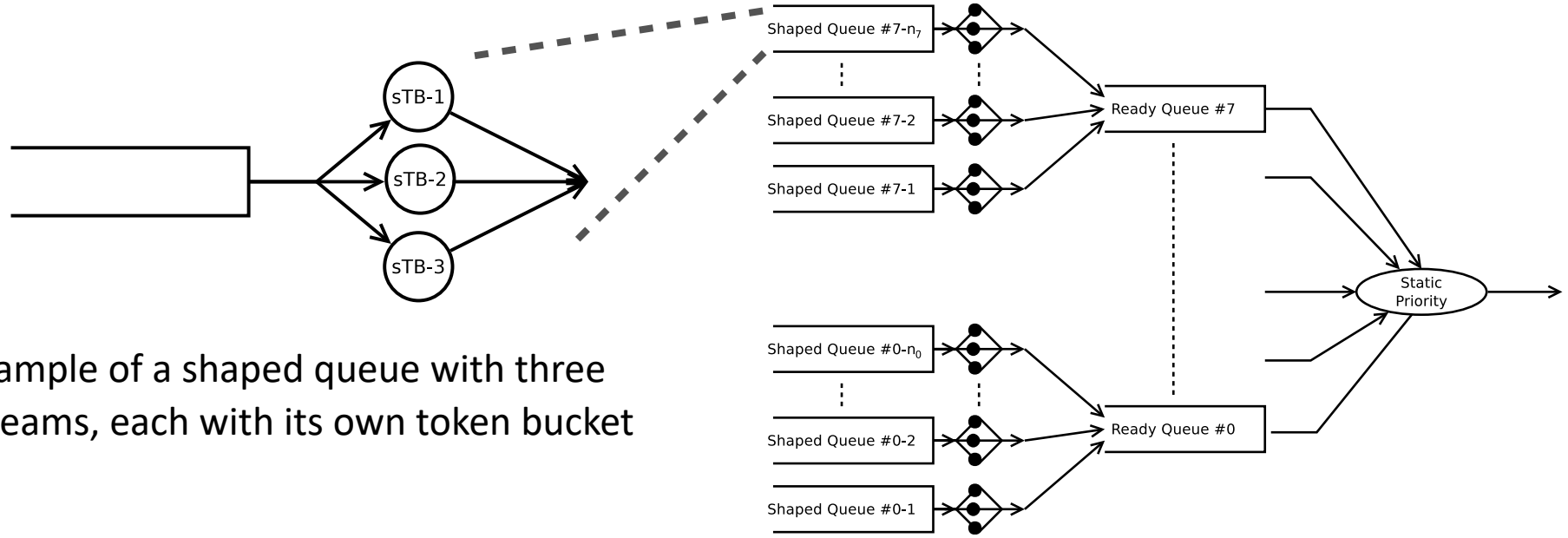


ATS with multiple queues



ATS port with 8 “ready” queues

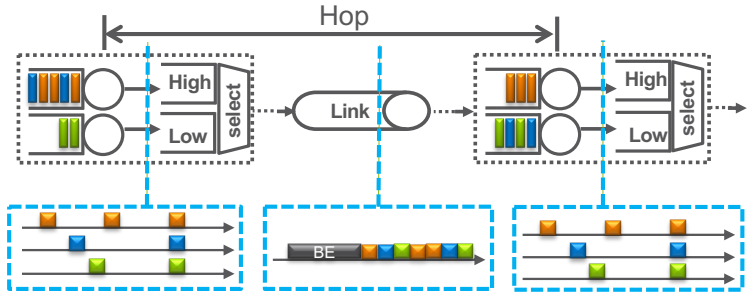
- Two types of queues:
  - **Shaped queues**: Store frames that need reshaping
    - Each stream in a shaped queue is regulated by its own token bucket algorithm
    - Token bucket shaping controls when frames can move from shaped to ready queue
  - Ready queues: Collect shaped frames for transmission; they work based on priorities
    - Multiple shaped queues feed into a single ready queue



Example of a shaped queue with three streams, each with its own token bucket

ATS Properties and Queue Assignment Rules

- Key Properties:
  - The worst-case delay a frame experiences at a hop is determined by the traffic arrival pattern and the shaper configuration. With ATS, this delay is not increased by the reshaping process itself.
  - Ensures **temporal composability**:
    - Temporal composability means the ability to analyze and predict timing behavior of a system by examining its components separately and then combining the results.
    - Worst-case delays can be calculated independently for each hop along a stream's path. The end-to-end worst-case delay is simply the **sum of these per-hop delays**.



- To preserve these properties, we need to follow three **Queue Assignment Rules** (QAR):
  - **QAR1:** Frames from different input ports must use separate shaped queues
  - **QAR2:** Frames with different priority levels from the same input port must use separate shaped queues
  - **QAR3:** Frames to be transmitted with different priority levels must use separate shaped queues

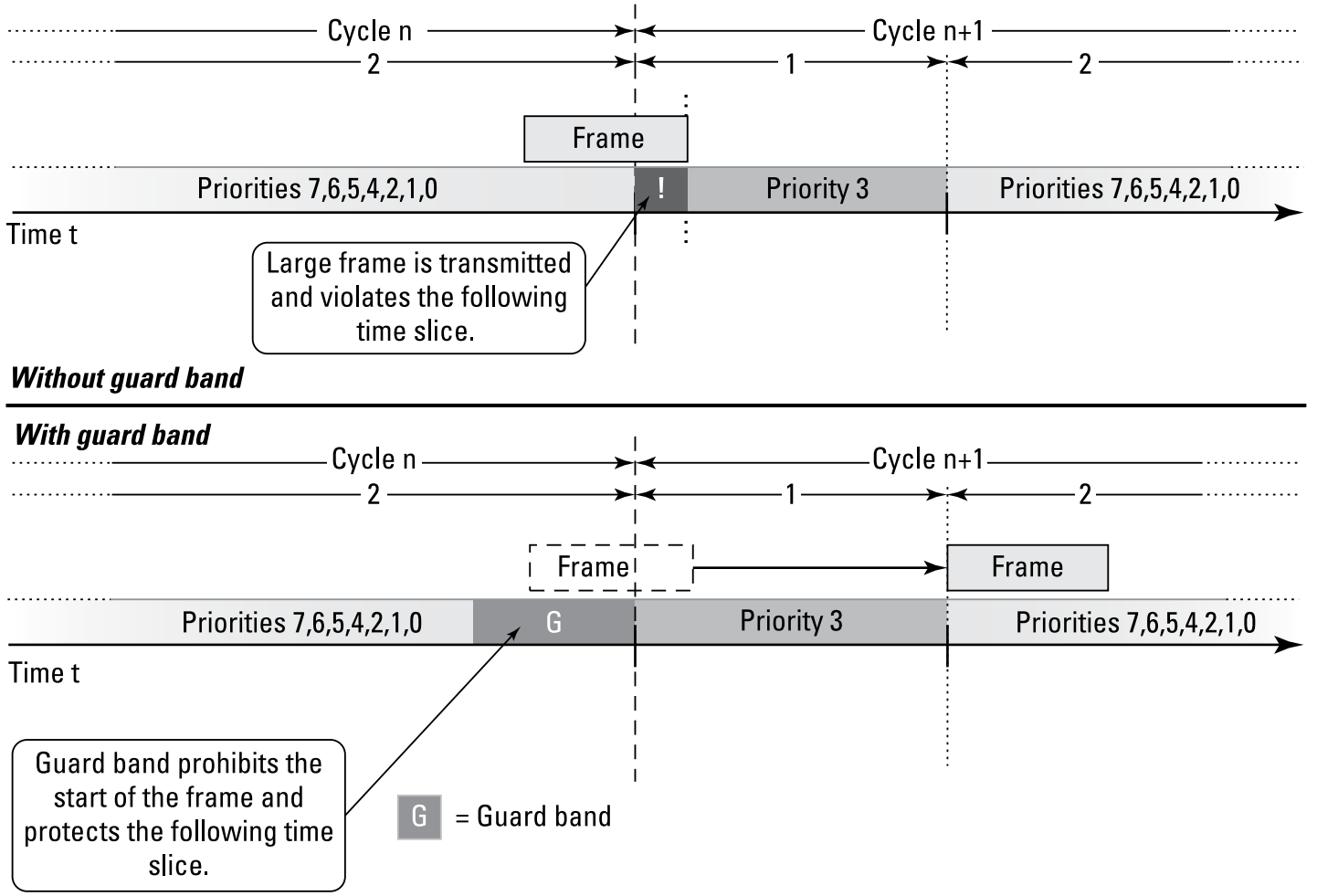
Dependability

Guard bands, Frame preemption

Redundancy: Frame Replication and Elimination

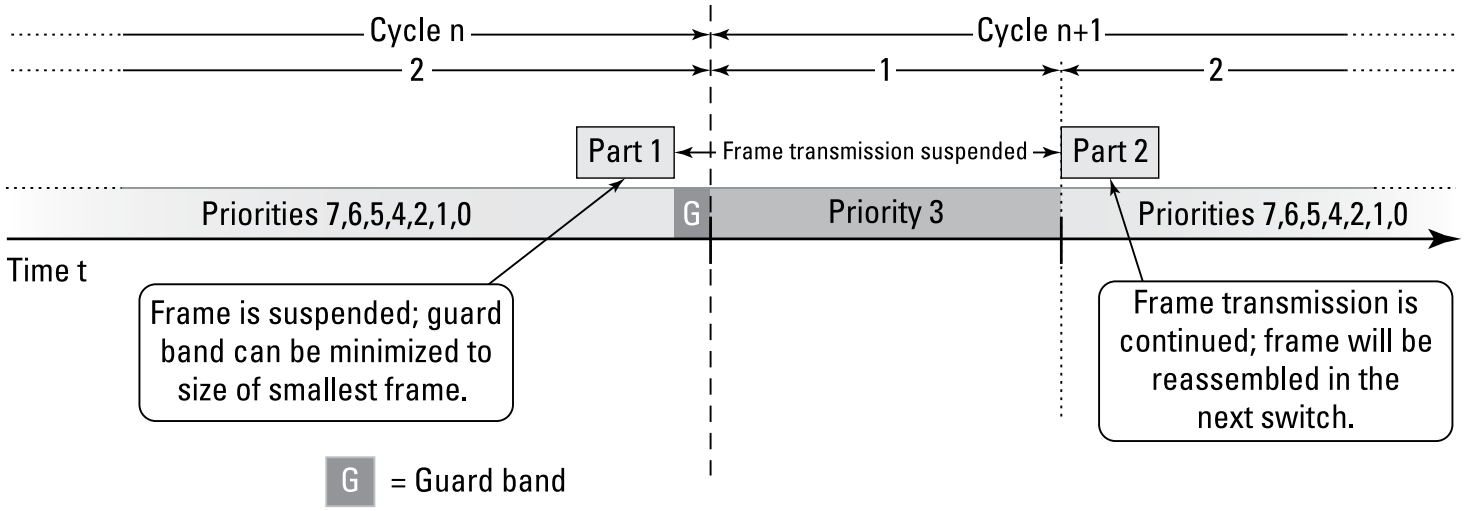
Guard bands

- Problem: a low-priority frame spills over into the time slot reserved for a higher-priority item.
- To prevent that from happening, the Time-Aware Scheduler employs a **guard band** that blocks the large frame from entering the time slot, so the frame gets bumped to the beginning of the next cycle.



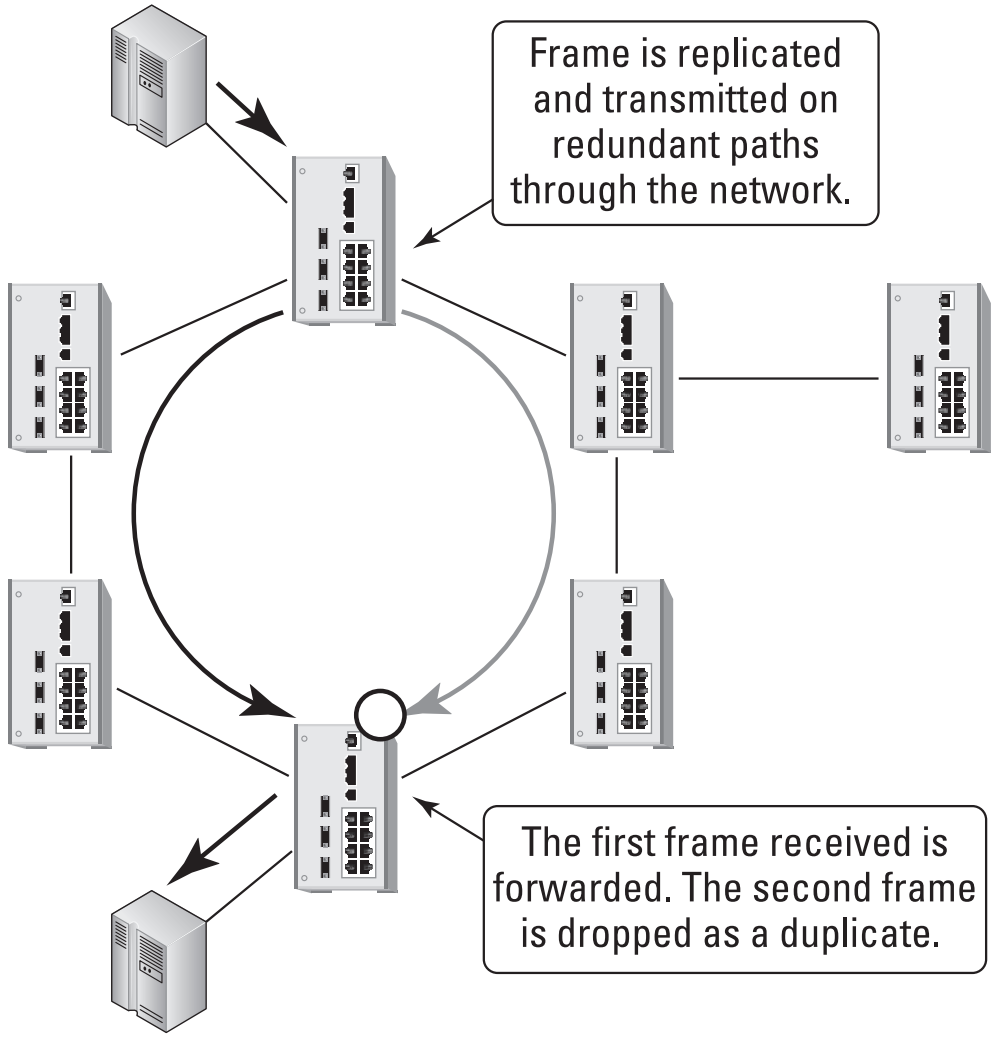
Frame preemption

- Guard band drawback: it blocks transmission of frames in the lower-priority queues, resulting in wasted bandwidth
- Frame preemption: blocking only the portion of it that won't fit in the time slot
- Needs to split a frame into smaller packets



IEEE 802.1CB: Seamless redundancy in TSN

- **Seamless redundancy:** All network paths are used in parallel, so no disruption occurs if one path fails.
- **Non-seamless (failover) redundancy:** The protocol recovers the fault by switching from the primary path to the secondary path; it may result in a very brief disruption.
- 802.1CB implements Frame Replication and Elimination for Reliability



802.1CB

- Frame Replication and Elimination for Reliability (FRER)
- Specified protocols for bridges and end systems:
  - Replication of packets
  - Identification of duplicate packets
  - Redundant transmission
  - Merge points and elimination of redundant packets
  - Optional: Proxy mode of operation
  - Optional: Auto-configuration to establish redundant paths

802.1CB history

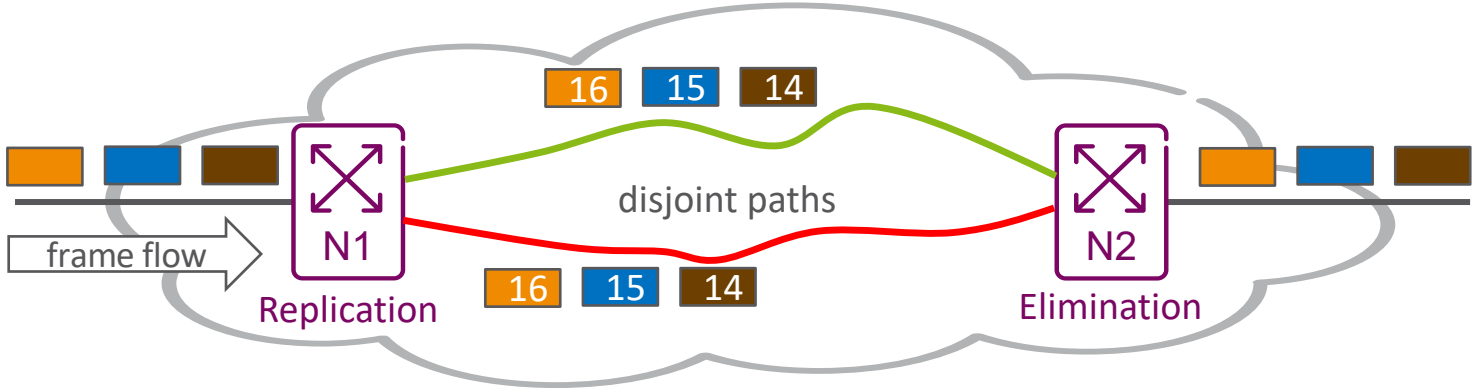
- Industrial automation systems already implemented redundancy on top (proprietary) Ethernet
  - PRP: Parallel Redundancy Protocol
  - HSR: High-availability Seamless Redundancy
- Need to standardize in 802.1
  - Industrial automation (converging towards IEEE 802 standardized Ethernet networks)
  - Professional audio/video needs redundancy for availability reasons
  - Automotive, and other safety critical application domains, have fail-operational requirements

802.1CB goals

- Increase probability that a given packet will be delivered on time
- Consider a range of failures in the communication path that could cause packet errors or packet drops:
  - Connector
  - Wire
  - Electrical components on PCB
  - PHY and MAC
  - Switch internal errors
  - Power
  - Software

Frame replication and elimination

- Add sequence numbers to frames
- Send on two maximally disjoint paths
- Then combine and delete extras



Redundancy without 802.1CB?

- The 802.1 Rapid Spanning Tree Protocol (RSTP) is a distributed agreement protocol used to disable loops in a given physical network topology
- In case of link or switch failures, RSTP will enable previously disabled links to re-establish connectivity
- But this takes time and there is no worst-case latency guarantee
  - Not acceptable for applications with stringent availability requirements (e.g., autonomous driving or “Superbowl” commercials)
  - Some applications need “instantaneous” response in failure modes

802.1CB

- Identification of streams
  - Identify and mark packets
- Replication
  - Create copies and forward on redundant paths
- Elimination
  - Eliminate duplicate packets
  - Recipient has an “acceptance window” for frame duplicates arriving out of order

NEW

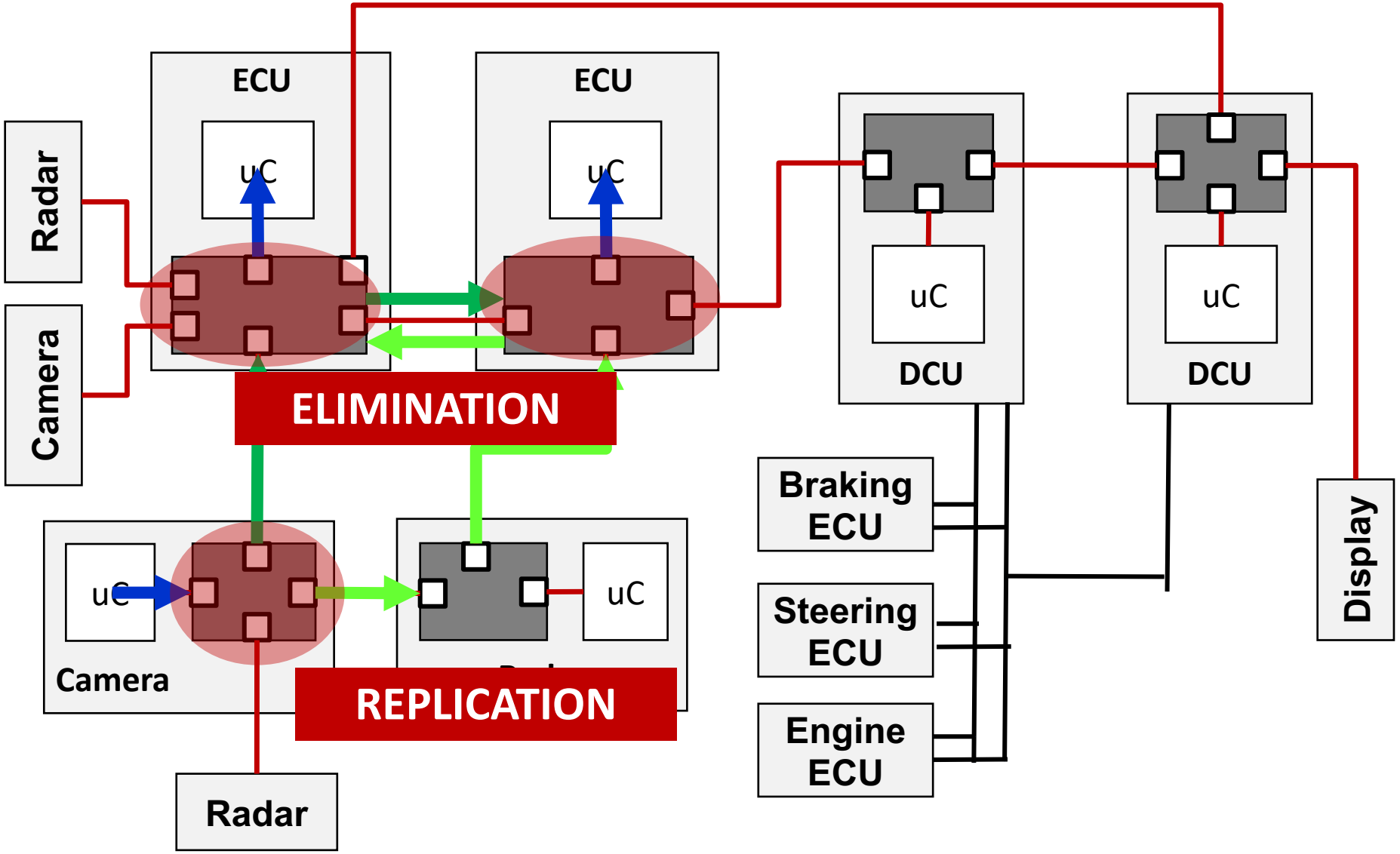
Field	Offset	Length
Destination MAC address	0	6
Source MAC address	6	6
C-tag EtherType	12	2
Priority, DE, VLAN ID	14	2
FRER Ethertype	16	2
sequence number	18	2
Payload Length/EtherType	20	2
data	22	*n*
Frame Check Sequence	22+*n*	4

Example Ethernet frame format with embedded R-Tag

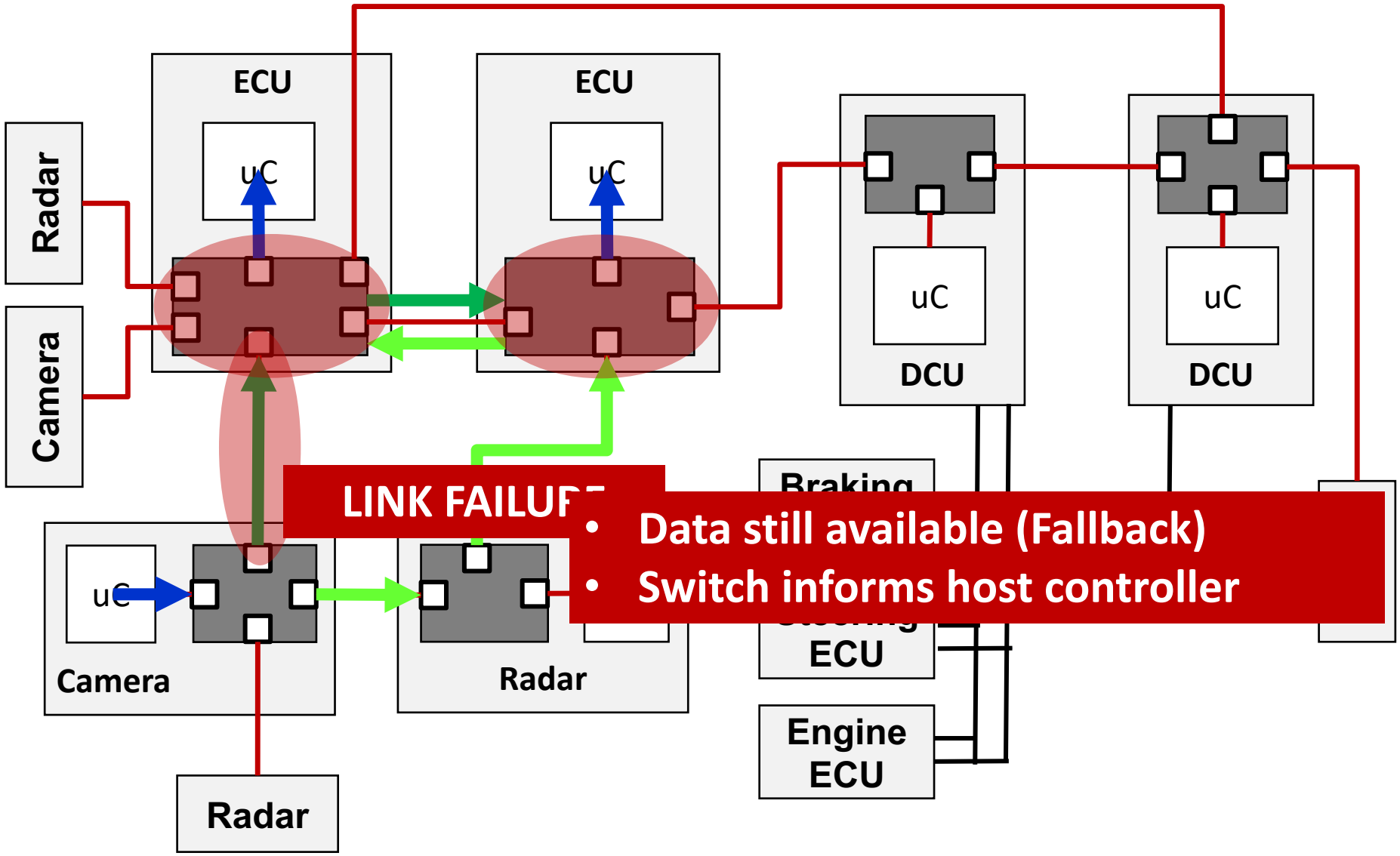
802.1CB operation

- Replicating packets at source or switch
- Send on separate paths
- Elimination of duplicates at sink or switch
- Proxy mode: all is handled by switches

Bridges with proxy mode



Link failure



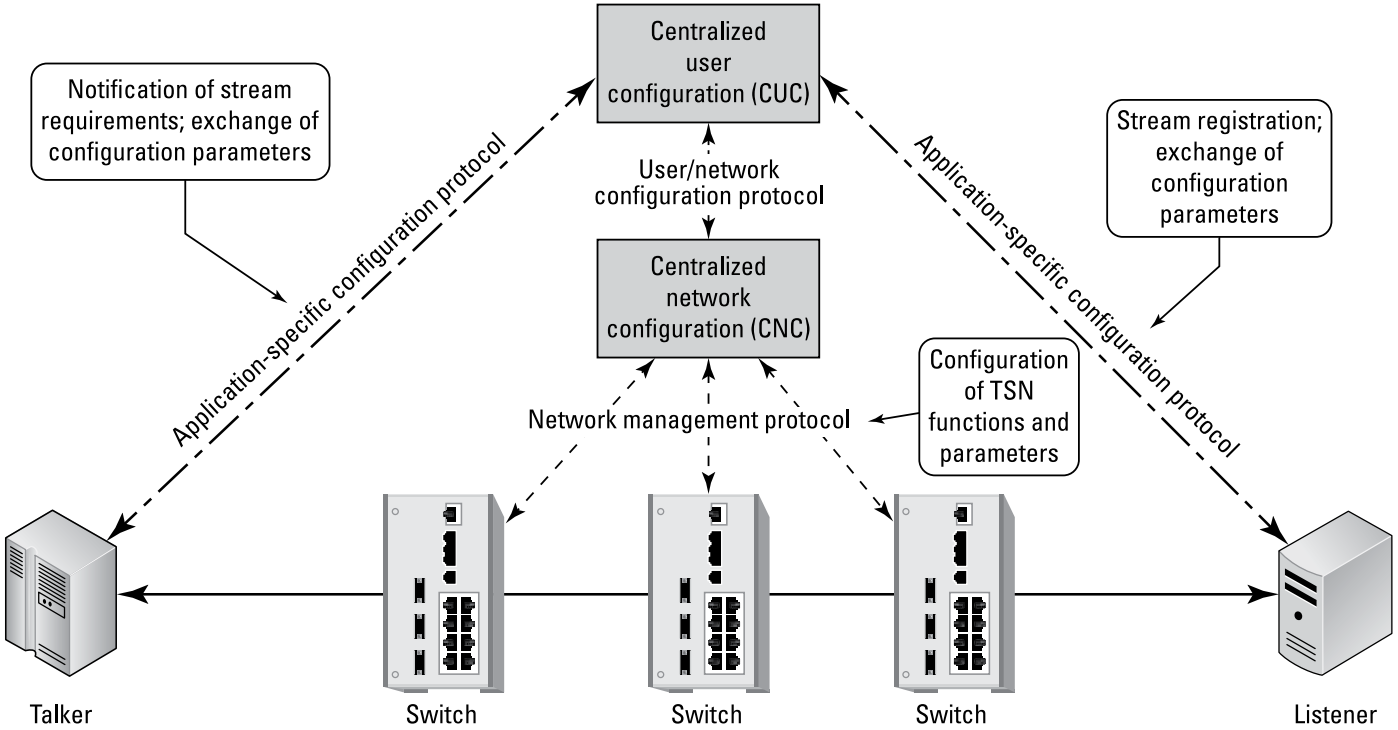
The trade-offs

- Need rings in the network (additional links and switches)
- More bandwidth usage due to duplication
  - Need to pay attention to specific ports
  - Need to pay attention to latency increase caused to other flows in the network

TSN network configuration

Centralized vs. distributed

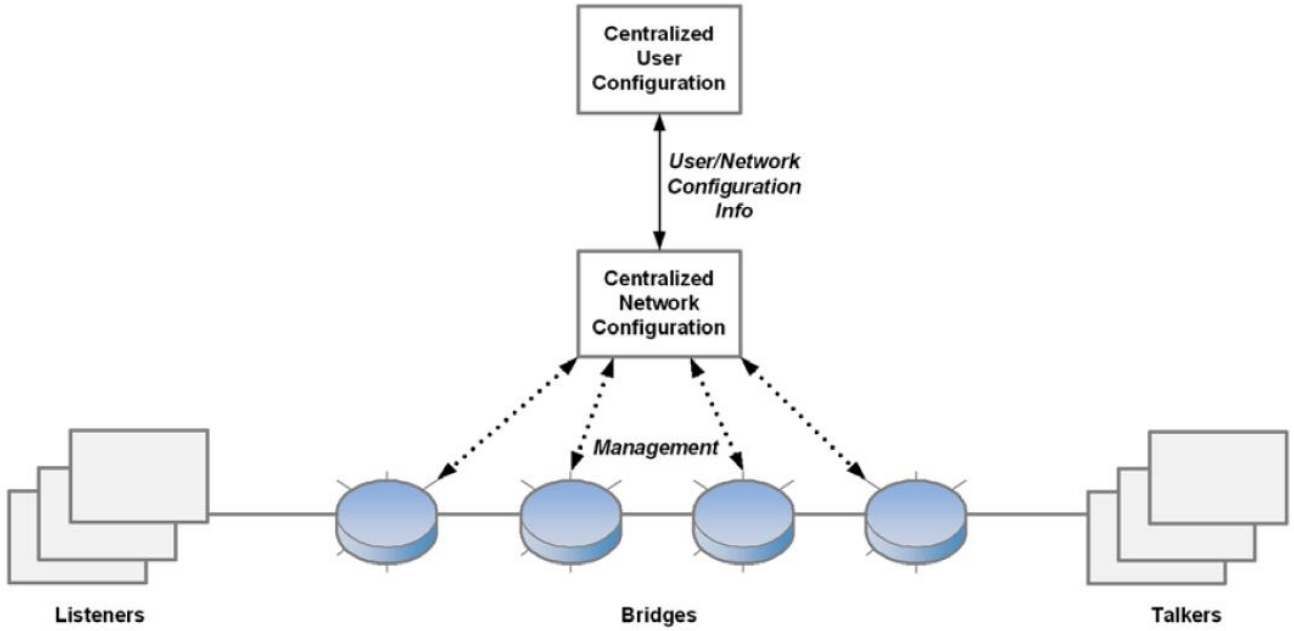
Configuring a TSN network



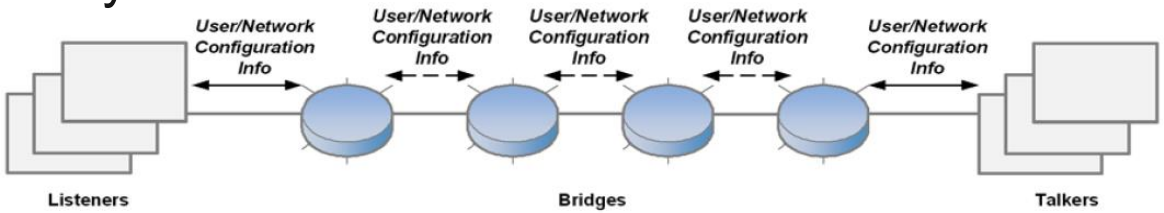
- End system announce their requirements to a central user configuration (CUC)
- The CUC computes a network configuration (this is where your optimization software is running)
- The CUC pushes the computed configuration to centralized network configuration (CNC)
- The CNC distributes the configuration modeled as YANG objects using the protocol NETCONF

TSN Configuration
[802.1Qcc]

Fully centralized



Fully distributed



Centralized network & distributed user