Chapter 7
Real-Time Communication

Overview The focus of this chapter is on the architectural view of real-time communication. The chapter commences by summarizing the requirements of a real-time communication system: *low protocol latency* with *minimal jitter*, the establishment of a *global time base*, fast *error detection at the receiver,* and the need for *temporal error containment* by the communication system, such that a *babbling node* cannot hinder the communication among the correct nodes. The next section presents a waistline model of a real-time communication system. At the center of the waist is the *basic message transport service (BMTS)* that transports a message from a sender to a set of receivers within a given latency and with a given reliability. In real-time systems, the tradeoff between reliability and timeliness has to remain in the hands of the application and should not be hardwired in the BMTS. The protocols above the BMTS, called *higher-level protocols*, implement services that require the bidirectional exchange of messages such as a simple *request-reply service*. The protocols below the BMTS, called *lower-level protocols*, implement the basic message transport service. The important topic of flow control, the different types of flow control and the *phenomenon of thrashing* are discussed in the following section. From the temporal point of view, three different communication services can be distinguished: *event-triggered communication*, *rate-constrained communication*, and *time-triggered communication*. The section on event-triggered communication contains the Ethernet protocol, the CAN protocol, and the UDP protocol from the Internet suite of protocols. Since there are no temporal constraints on the sender of event-triggered messages, it is not possible to provide temporal bounds for latency and jitter, given the limited bandwidth of any communication system. The rate-constrained protocols provide bounds for latency and jitter. The protocols covered in this section are the ARINC 629, and ARINC 684 (AFDX). The final section presents the time-triggered protocol TTP, TTEthernet, and FlexRay. These protocols require the establishment of a global time base among all communicating nodes. A cycle is assigned to every time-triggered message. The start of transmission of the message is triggered exactly when the global time reaches the start of cycle. Time-triggered communication is deterministic and well suited for the implementation of fault tolerance by the active replication of components.

H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Real-Time Systems Series, DOI 10.1007/978-1-4419-8237-7_7,

7.1 Requirements

The architectural requirements for the communication infrastructure of a distributed real-time system follow from the discussion about the properties of *real-time data* elaborated in the previous chapters. These requirements are substantially different from the requirements of non-real-time communication services.

7.1.1 Timeliness

The most important difference between a real-time communication system and a non-real-time communication system is the requirement for short message-transport latency and minimal jitter.

Short Message-Transport Latency. The real-time duration of a *distributed real-time transaction* (see Sect. 1.7.3), starting with the reading of sensors and terminating with the output of the results to an actuator depends on the time needed for the computations within the components and the time needed for the message transport among the involved components. This duration should be *as small as possible*, such that the *dead time* in control loops is minimized. It follows that the *worst-case message transport latency* of a real-time protocol should be small.

Minimal Jitter. The jitter is the difference between the worst-case message-transport latency and the best-case message-transport latency. A large jitter has a negative effect on the duration of the *action delay* (see Sect. 5.5.2) and the *precision* of the clock-synchronization (see Sect. 3.4).

Clock Synchronization. A real-time image must be *temporally accurate* at the *instant of use* (see Sect. 5.4). In a distributed system, the temporal accuracy can only be checked if the duration between the instant of observation of an RT-entity, observed by the sensor node, and the instant of use, determined by the actuator node, can be measured. This requires the availability of a global time base of proper precision among all involved nodes. It is up to the communication system to establish such a global time and to synchronize the nodes, e.g., by following the IEEE 1588 standard for clock synchronization. If fault tolerance is required, two independent self-checking channels must be provided to link an end system to the fault-tolerant communication infrastructure. The clock synchronization messages must be provided on both channels in order to tolerate the loss of any one of them.

7.1.2 Dependability

Communication Reliability. In real-time communication, the use of robust channel encoding, the use of error-correcting codes for forward error correction, or the deployment of diffusion based algorithms, where replicated copies of a message are

sent on diverse channels (e.g., *frequency hopping* in wireless systems), possibly at different times, are the *techniques of choice* for improving the communication reliability. In many *non-real-time* communication systems, reliability is achieved by time redundancy, i.e., a lost message is retransmitted. This tradeoff between time and reliability *increases the jitter significantly*. This tradeoff should not be part of the basic message transport service (BMTS), since it is up to the application to decide if this tradeoff is desired.

Example: In the *positive acknowledgment-or-retransmission (PAR) protocol*, widely used in event-triggered non-real-time communication, a sender waits for a given time until it has received a positive acknowledgement message from the receiver indicating that the previous message has arrived correctly. In case the timeout elapses before the acknowledgement message arrives at the sender, the original message is retransmitted. This procedure is repeated *n-times* (protocol specific) before a permanent failure of the communication is reported to the sender. The jitter of the PAR protocol is substantial, since in most cases the first try will be successful, while in a few cases the message will arrive after n times the timeout value plus the worst-case message transport latency. Since the timeout value must be longer than two worst-case message transport latencies (one for the original message and one for the acknowledgment message), the jitter of PAR is longer than $(2n)$ worst-case message-transport latencies.

Example: Consider a scenario, where a sensor component sends periodically, e.g., every millisecond, a message containing an observation of an RT entity to a control component. In case the message is corrupted or lost, it makes more sense to wait for the next message that contains a more recent observation than to implement a PAR protocol that will resend the lost message with the older observation.

Temporal Fault Containment of Components. It is ***impossible*** *to maintain the communication among the correct components using a shared communication channel if the temporal errors caused by a faulty component are not contained.* A shared communication channel must erect *temporal firewalls* that contain the temporal faults of a component (a *babbling idiot*), so that the communication among the components that are not directly affected by the faulty component is not compromised. This requires that the communication system holds information about the intended (permitted) temporal behavior of a component and can disconnect a component that violates its temporal specification. If this requirement is not met, a faulty component can block the communication among the correct components.

Example: A faulty component that sends continuously high-priority messages on a CAN bus will block the communication among all other correct components and thus cause a total loss of communication among the correct components.

Error Detection. A message is an *atomic unit* that either arrives correctly or not at all. To detect if a message has been corrupted during transport, every message is required to contain a CRC field of redundant information so the receiver can validate the correctness of the data field. In a real-time system, the detection of a corrupted message or of message loss by the *receiver* is of particular concern.

Example: *Error detection on output*. Consider a node at a control valve that receives output commands from a controller node. In case the communication is interrupted because the wires are cut, the control valve, the *receiver*, should enter a safe state, e.g., close

the valve autonomously. The receiver, i.e., the control valve, must detect the loss of communication autonomously in order to be able to enter the safe state despite the fact that the wire has been cut.

Example: *Error detection on input*. Consider a sensor node that periodically sends an observation to a control node. In case the communication is interrupted because the wires are cut, the control node, the *receiver*, must immediately detect the loss of communication.

The failure of a component of a distributed system should be detected by the communication protocol and should be reported consistently to all remaining correct components of the ensemble. In real-time systems, the prompt and consistent detection of component failures is the function of a *membership service*.

End-to-End Acknowledgment. End-to-end acknowledgement about the success or failure of a distributed action is needed in any scenario where multiple nodes cooperate to achieve a desired result [Sal84]. In a real-time system, the *definitive* end-to-end acknowledgment about the *ultimate* success or failure of a communication action can come from a component that is different from the receiver of an outgoing message. An outgoing message to an actuator in the environment must cause some *intended physical effect* in the environment. A sensor component that is different from the actuator component monitors this *intended physical effect*. The result observed by this sensor component is the definite end-to-end acknowledgement of the outgoing message and the intended physical action.

Example: Figure 7.1 shows an example of an end-to-end acknowledgment of the output message to a control valve by a flow sensor that is connected to a different node.

Example: A wrong end-to-end protocol can have serious consequences, as seen in the following quote [Sev81, p. 414] regarding the Three Mile Island Nuclear Reactor #2 accident on March 28, 1979: *Perhaps the single most important and damaging failure in the relatively long chain of failures during this accident was that of the Pressure Operated Relief Valve (PORV) on the pressurizer. The PORV did not close; yet its monitoring light was signaling green (meaning closed).* In this system, the fundamental design principle *never trust an actuator,* was violated. The designers assumed that the acknowledged arrival of a control output signal that commanded the valve to close, implied that the valve *was* closed. Since there was an electromechanical fault in the valve, this implication was not true. A proper end-to-end protocol that mechanically sensed the closed position of the valve would have avoided this catastrophic false information.

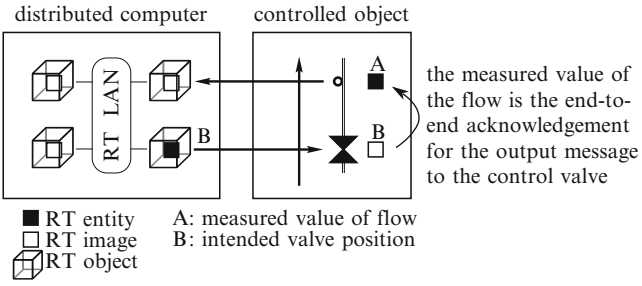

Fig. 7.1 End-to-end acknowledgment in a real-time system

Determinism. The behavior of the basic message transport service (BMTS) should be *deterministic* such that the order of messages is the same on all channels and the instants of message arrival of replicated messages that travel on redundant independent channels are close together. This desired property, which has been discussed at length in Sect. 5.6, is required for the implementation of fault tolerance by active redundancy.

Example: If in a fault-tolerant configuration the message order on two independent communication channels is not the same, then the fault-masking capability may be lost due to the missing replica determinism.

7.1.3 Flexibility

Many real-time communication systems must support different system configurations that change over time. A real-time protocol should be flexible to accommodate these changes without requiring a software modification and retesting of the operational nodes that are not affected by the change. Since the bandwidth of any communication channel is limited, there exists an upper bound on the increase in communication traffic that can be handled within the given time constraints.

Topology. The standard communication topology in distributed real-time systems is multicast, not point-to-point. The same image of an RT entity is needed at a number of different components, e.g., at the man-machine interface, at a process-model component, and at an alarm-monitoring component. A message should be delivered to all receivers of the receiver group within a short and known time interval.

Dynamic Addition of a Partner. It should be possible to add a new communication partner dynamically. If this new partner is *passive*, i.e., it is only receiving messages but not sending messages, the multicast topology can support this requirement by adding the new partner to the receiver group. If the new partner is *active*, i.e., it is sending messages, then the communication infrastructure should provide the necessary bandwidth without violating the temporal guarantees given to the already existing partners.

Example: A communication system within a car must support different configurations of nodes, depending on customer demand. One customer might demand a car with a sunroof and automatic seats with memory, while another customer might opt for a special air-conditioning system and a sophisticated anti-theft system. All possible combinations of nodes must be supported by the communication system without a need for retesting existing nodes.

7.1.4 Physical Structure

The physical structure of a real-time communication system is determined by technical needs and economic considerations.

Example: In the harsh environment of a plant, the physical transmission system must be more robust than in a benign office environment.

Physical Fault Isolation. The communication system should provide for the physical isolation of nodes that are placed at different locations in space, such that common mode node failures, e.g., those caused by a lighting stroke, will not occur. The transducer circuits that link the wires to the nodes must withstand the specified high-voltage disturbances. Fiber-optic transmission channels provide for the best physical isolation.

Example: Consider an airplane with a fly-by-wire system. The nodes that form a fault-tolerant unit for this critical function should be at different locations within the plane and connected by well-isolated channels, such that a high-voltage disturbance or a physical damage of a section of the plane during an incident (e.g., lightning stroke) will not result in the correlated loss of the safety-critical system function of all replicated nodes.

Low Cost Wiring. In many embedded systems, e.g., in a car or an airplane, the weight and cost of the wiring harness is substantial. The selection of the communication protocols and in particular the physical transmission layer is influenced by the desire to minimize the wiring weight and cost.

7.2 Design Issues

In Chap. 2 we emphasized the *need to design a generic model for expressing the behavior of an embedded system that avoids the characteristics of difficult tasks* (see Table 2.2 on *difficult tasks*). The communication among the computational components of a distributed system forms an integral part of the behavior. At the architecture level we thus need a *simple model* for describing the communication that captures the system aspect of the real-time message transport without getting detracted by the detailed mechanisms and complexities of the implementation of the transmission channels or the logic of high-level protocols.

7.2.1 A Waistline Communication Model

The *waistline model* depicted in Fig. 7.2 seems to be fit for this purpose. The center of the model, the *waistline*, depicts the *basic message transport service (BMTS)* that is provided at the architecture level. Considering the discussion in the previous chapters, the BMTS should transport a *message* from a sending component to one or more receiving components with a high reliability, a small delay, and minimal jitter (see Sect. 4.3). Since the sender of the message must not be directly impacted by a failure of the receiver, the message flow of the BMTS must be *unidirectional*.

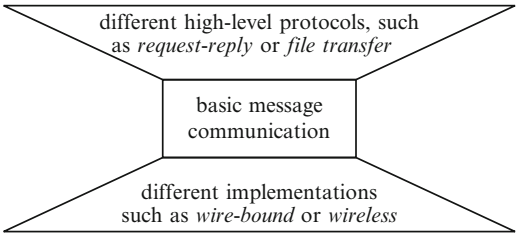

Fig. 7.2 Waistline model of message transport

In the literature, the meaning behind the notion of a *datagram service* comes close to the semantics of the BMTS, but there are no temporal requirements (e.g., short transport latency, minimal jitter) for a datagram.

Depending on the temporal properties of this BMTS, we distinguish three types of messages:

1. *Event-triggered messages.* The messages are produced sporadically whenever a significant event occurs at the sender. There is no *minimum time* between messages established. No temporal guarantees about the delay between sending a message and receiving a message can be given. In case the sender produces more messages than the BMTS can handle, either back-pressure is exercised on the sender or messages are lost.
2. *Rate-constrained messages.* The messages are produced sporadically, however the sender guarantees not to exceed a maximum message rate. Within the given fault-hypothesis, the BMTS guarantees not to exceed a maximum worst-case transport latency. The jitter depends on the load of the network and is bounded by the worst-case transport latency minus the minimum transport latency.
3. *Time-triggered messages.* Sender and receiver agree *a priori* on the exact instants when messages are sent and received. Within the given fault-hypothesis, the BMTS guarantees that the messages will be delivered at the specified instants with a jitter that is determined by the precision of the global time.

There are many different means how to implement the BMTS on wire-bound or wireless channels by *low-level protocols.* A given BMTS is characterized by the transport latency, the jitter, and the reliability of transporting a single message unidirectionally form a sender to the set of destined receivers.

The BMTS provides the basis for building *high-level protocols*, such as a *request-reply protocol*, a *file-transport protocol,* or any other rule-based exchange of message sequences that is captured in a higher-level concept.

Example: A simple high-level protocol is the mentioned *request-reply protocol* that consists of two BMTS messages, one from the sender to the receiver and a second related (but from the point of view of the BMTS *independent*) BMTS message from the receiver back to the sender.

7.2.2 Physical Performance Limitation

Any physical communication channel is characterized by its *bandwidth* and its *propagation delay*. The *bandwidth* denotes the number of bits that can traverse a channel in unit time. The length of the channel and the transmission speed of the wave (electromagnetic, optical) within the channel determine the *propagation delay* which is the duration it takes for a single bit to travel from one end of the channel to the other end. Because the transmission speed of a wave in a cable is approximately 2/3 of the transmission speed of light in vacuum (about 300,000 km/s), it takes a signal about 5 μs to travel across a cable of 1 km length. The term *bit length of a channel* is used to denote the number of bits that can traverse the channel within one propagation delay.

Example: If the channel bandwidth is 100 Mbit/s and the channel is 200 m long, the bit-length of the channel is 100 bits, since the propagation delay of this channel is 1 μs.

In a bus system, the data efficiency of any media access protocol on a shared channel is limited by the need to maintain a minimum time interval of one propagation delay between two successive messages. Assume the bit length of a channel to be *bl* bits and the message length to be *m* bits. Then an upper bound for the data efficiency of any media access protocol in a bus system is given by:

$$data\ efficiency < m/(m + bl)$$

Example: Consider a 1 km bus with a bandwidth equal to 100 Mbits/s. The message length that is transmitted over this channel is 100 bits. It follows that the bit length of the channel is 500 bits, and the limit to the best achievable data efficiency is 100/(100 + 500) = 16.6%.

If the message length is less than the bit length of the channel, then the *best channel utilization* that can be realized with any media access protocol is less than 50%. This physical limit has implications for the design of protocols. If a channel is long and supports a high bandwidth, it is wasteful to send short messages. Table 7.1 contains the bit length of a channel as a function of its bandwidth and its length.

Example: From Table 7.1, we can deduce that if we want to achieve a channel utilization better than 50% for a 100 m long channel with 1 Gbit/s bandwidth, then the *minimum message length* must be larger than 500 bits. In such a scenario, it is *waste of bandwidth* to send BMTS messages that are only a few bits long.

7.2.3 Flow Control

Flow control is concerned with the control of the speed of information flow between a sender and a receiver (or the communication system, in this case the term *congestion* control is sometimes used) in such a manner that the communication system and the receiver can keep up with the sender. In any communication scenario, it is the

Table 7.1 Bit-length of a channel as a function of channel length and bandwidth

Channel length and propagation delay in seconds	Bandwidth of the channel in bits per second and bit length in seconds						
	10 kbit 100 μs	100 kbits 10 μ	1 Mbit 1 μ	10 Mbit 100 ns	100 Mbit 10 ns	1 Gbit 1 ns	10 Gbit 100 ps
1 cm – 50 ps	<1	<1	<1	<1	<1	<1	<1
10 cm – 500 ps	<1	<1	<1	<1	<1	<1	5
1 m – 5 ns	<1	<1	<1	<1	<1	5	50
10 m – 50 ns	<1	<1	<1	<1	5	50	500
100 m – 500 ns	<1	<1	<1	5	50	500	5 k
1 km – 5 μs	<1	<1	5	50	500	5 k	50 k
10 km – 50 μs	<1	5	50	500	5 k	50 k	500 k

receiver or the limited capacity of the communication system rather than the sender that determines the *maximum* speed of communication. We distinguish between the following types of flow control:

Back-pressure flow control. In case the communication system or the sender cannot accept any further messages the sender is forced to delay the sending of further messages until the overload condition has disappeared. We say that *back-pressure* is exercised on the sender.

> **Example:** In a CAN system a node cannot access the bus if a transmission by another sender is in progress, i.e., the access protocol at the sender exerts *backpressure* on the node that intends to send a message.

Explicit flow control. In *explicit flow control* (a form of *back-pressure* flow control) the receiver sends an explicit acknowledgment message back to the sender, informing the sender that the sender's previous message arrived correctly, and that the receiver is now ready to accept the next message. The most important protocol with explicit flow control is the well-known *Positive-Acknowledgment-or-Retransmission (PAR)* protocol, described in the example of Sect. 7.1.2.

Backpressure flow control and explicit flow control are based on the sometimes-overlooked assumption that the sender is within the sphere of control (SOC) of the receiver, i.e., that the receiver can control the transmission speed of the sender (see Fig. 7.3). This is not the case in many real-time scenarios, since the progress of the physical process cannot be influenced by the communication protocol.

> **Example:** On August 8, 1993 a prototype of a fly-by-wire fighter plane crashed, because the plane responded too slowly to the pilot's commands [Neu95, p.37].

Best-effort flow control. In best effort flow control, the communication system provides intermediate buffer-storage for messages in case the link for the further transport of messages to the final receiver is not available. If a buffer overflows, messages are dropped.

> **Example:** In switched Ethernet, the Ethernet switch contains a buffer storage before the link to a final receiver. In case this buffer overflows, backpressure is exercised or further messages are dropped. The duration during which a message has to reside in this buffer is difficult to predict.

Fig. 7.3 Explicit flow control in a real-time system

Best-effort flow control cannot be used in hard real-time systems, since the uncontrollable delay or the *throw away* of messages is not acceptable.

Rate-constrained flow control. In *rate-constrained flow control*, the sender, the communication system and the receiver agree *a priori,* i.e., before the communication starts, on the maximum message rate. The communication system and the receiver will accept all messages sent by the sender, as long as the sender rate remains below the agreed maximum.

Implicit flow-control. In *implicit flow control*, the sender and receiver agree *a priori,* i.e., before the communication starts, on the *instants* when messages are sent and received. This requires the availability of a global time-base. The sender commits itself to send a message only at the agreed instants, and the receiver commits itself to accept all messages sent by the sender, as long as the sender fulfills its obligation. No acknowledgment messages are exchanged during run time. Error detection is the responsibility of the receiver that can determine (by looking at its global clock) when an expected message fails to arrive. Error detection latency is short, since it is determined by the precision of the clock synchronization. Implicit flow-control is the most appropriate flow-control strategy for the exchange of real-time data.

7.2.4 Thrashing

The often-observed phenomenon of the throughput of a system decreasing abruptly with increasing load is called *thrashing*. Thrashing can be observed in many systems and is not limited to computer systems.

> **Example:** Consider the example of a traffic system in a large city. The throughput of the road system increases with increasing traffic up to a certain critical point. When this critical point is reached, further increase in traffic can lead to a reduction in throughput, or in other words, a *traffic jam*.

Many systems can be characterized by a throughput-load dependency as shown in Fig. 7.4. An ideal system exhibits the load throughput curve labeled *ideal* in Fig. 7.4. The throughput increases with increasing load until the saturation point has been reached. From thereon, the throughput remains constant. A system has a *controlled* load-throughput characteristic if the throughput increases monotonically

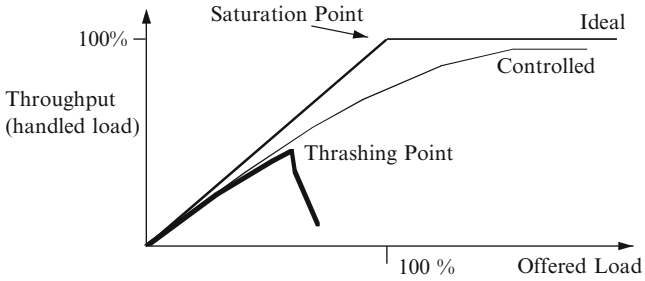

Fig. 7.4 Throughput-load characteristic

with the load and reaches the maximum throughput asymptotically. If the throughput increases up to a certain point, the *thrashing point,* and thereafter decreases abruptly, we say the system is *thrashing*.

Real-time systems must be free of the thrashing phenomena. If a real-time system contains a mechanism that can cause thrashing, then, it is likely that the system fails in the important *rare-event* scenarios discussed in Sect. 1.5.

Mechanisms that can cause thrashing. Mechanisms that require a more than proportional increase in resources as the load increases are prone to cause thrashing. Two examples of such mechanisms are:

1. A time-monitored retry mechanism (a type of PAR protocol) in a request-reply protocol: If a communication system slows down because it can barely handle the offered load, a request-reply protocol reaches its time-outs, and generates *additional* load.
2. Operating system services: In a dynamic scheduling environment, the time needed to find a feasible schedule increases more than linearly as the requested load reaches the capacity limit. This increase in the amount of scheduling overhead further decreases the computational resources that are available for the application tasks.

Similar arguments hold for the overhead required for queue management.

A successful technique to avoid thrashing in explicit flow-control schemes is to monitor the resource requirements of the system continuously and to exercise a stringent backpressure flow control at the system boundaries as soon as a decrease in the throughput is observed.

> **Example:** If too many users try to establish a telephone connection and thereby overload a switch, the switch exercises backpressure by presenting a *busy signal* to the users.

Remember that in a real-time system, such a backpressure flow control mechanisms is not always possible.

> **Example:** Consider a monitoring and control system for an electric power grid. There may be more than 100,000 different RT entities and alarms that must be monitored continually. In the case of a rare event, such as a severe thunderstorm when a number of lightning strikes hit the power lines within a short interval of time, many correlated alarms will occur. The computer system cannot exercise explicit flow control over these alarms in case the system enters the thrashing zone. It follows the design must be capable to handle the simultaneous occurrence of 100,000 different alarms.

7.3 Event-Triggered Communication

Figure 7.5 depicts event-triggered communication at the architectural level. A sender sends a message whenever a significant event (e.g., termination of a task, an interrupt signal, etc.) occurs at the sender. This message is placed in a queue at the sender's site until the basic message transport service (BMTS) is ready to transport the message to the receiver. The communication channel can be even-triggered, rate-constrained, or time-triggered. After arrival of the message at the receiver, the message is placed in a receiver queue until the receiver consumes the message. Using the CRC field contained in every message, the BMTS checks at the receiver's site whether the contents of a message have been corrupted during transport and simply discards corrupted messages. From the architectural point of view, a BMTS is characterized by a maximum bandwidth, a transport latency, a jitter, and a reliability for the transport of correct BMTS messages. These transport parameters can be characterized by probability distributions.

Whenever queues are involved in a scenario, the possibility of queue overflow must be considered. Queue overflow will occur if the transmission rate of the sender is larger than the capacity of the network (overflow of the sender's queue) or the delivery rate of the network is larger than the reception rate at the receiver (overflow of the receiver's queue). Different event-trigged protocols take different approaches to the handling of queue overflow.

*It is **impossible** to provide temporal guarantees in an open event-triggered communication scenario*. If every sending component in an open communication scenario is autonomous and is allowed to start sending a message at any instant, then it can happen that all sending components send a message to the same receiver at the same instant (the *critical instant*), thus overloading the channel to the receiver. In fielded communication systems, we find three strategies to handle such a scenario: (1) the communication system *stores* messages intermediately at a buffer before the receiver, (2) the communication system exerts *backpressure* on the sender, or (3) or the communication system *discards* some messages. None of these strategies is acceptable for real-time data.

A *lower-level protocol*, e.g., a link-level protocol that increases the reliability of a link at the cost of additional jitter, is not directly visible at the BMTS level – although its effects, the increased reliability and the increased jitter, are reflected in the characterization of the BMTS service. When a BMTS message has been sent

Fig. 7.5 Event-triggered communication

across the Internet, we don't know what types and how many different low-level protocols have been activated.

In Sect. 4.3.3 an *exactly-once semantics* has been demanded for the transmission of event information. The implementation of an *exactly-once semantics* requires a bi-directional information flow between sender and receiver that is not provided at the BMTS level of our model. In our model, the *exactly-once semantics* must be implemented by a *higher-level protocol* that uses two or more (from the point of view of the communication service *independent*) BMTS messages.

7.3.1 Ethernet

Ethernet is the most widely used protocol in the non-real-time world. The original *bus-based Ethernet*, controlled by the CSMA/CD (carrier sense multiple access/collision detection) with exponential back-off access control strategy [Met76] has, over the years, morphed into a switched Ethernet configuration with star topology, standardized in IEEE standard 802.3. An Ethernet switch deploys a *best-effort flow-control* strategy with a buffer before the link to the final receiver. If this buffer overflows, further messages to this receiver are discarded. If an *exactly-once semantics* must be implemented in an Ethernet system, a higher level protocol that uses two or more Ethernet messages must be provided. An extension of standard Ethernet, time-triggered (TT) Ethernet that supports a *deterministic message transport* is described in Sect. 7.5.2.

7.3.2 Controller Area Network

The CAN (Controller Area Network) Protocol developed by Bosch [CAN90] is a bus-based CSMA/CA (carrier sense multiple access/collision avoidance) protocol that exercises *backpressure flow control* on the sender. The CAN message consists of six fields as depicted in Fig. 7.6. The first field is a 32-bit *arbitration field* that contains the message identifier of 29 bits length. (The original CAN had only a arbitration field of 11 bits, supporting at most 2,024 different message identifiers.) Then there is a 6-bit control field followed by a data field of between 0–64 bits in length. The data in the first three fields are protected by a 16-bit CRC field that ensures a Hamming distance of 6. The fields after the CRC are used for an immediate acknowledgment message.

Field	Arbitration	Control	Data Field	CRC	A	EOF
Bits	32	6	0-64	16	2	7

Fig. 7.6 Data format of a CAN message

In CAN, access to the CAN-bus is controlled by an arbitration logic that assumes the existence of a recessive and a dominant state on the bus such that the dominant state can overwrite the recessive state. This requires that the propagation delay of the channel is smaller than the length of a bit-cell of the CAN message (see Table 7.1). Assume that a 0 is coded into the dominant state and a 1 is coded into the recessive state. Whenever a node intends to send a message, it puts the first bit of the arbitration field (i.e., the message identifier) on the channel. In case of a conflict, the node with a 0 in its first identifier bit wins, and the one with a 1 must back off. This arbitration continues for all bits of the arbitration field. A node with all 0 s always wins – this is the bit pattern of the highest priority message. In CAN, the message identifier thus determines the message priority.

7.3.3 User Datagram Protocol

The user datagram protocol (UDP) is the stateless *datagram* protocol of the Internet protocol suite. It is an efficient unreliable uni-directional message protocol that requires no set-up of transmission channels and supports multi-casting on a local area network using a best-effort flow-control strategy. Many real-time applications use UDP because the tradeoff between latency and reliability is not hard-wired in UDP (in contrast to the *Transmission Control Protocol TCP*) but can be performed at the application level, taking account of the application semantics. UDP is also used for multimedia streaming applications.

7.4 Rate-Constrained Communication

In rate-constrained communication, a *minimum guaranteed bandwidth* is established for each channel. For this minimum bandwidth, the *maximum transport latency* and the *maximum jitter* are guaranteed to be smaller than an upper bound. If a sender (an end-system) sends more messages than the minimum guaranteed bandwidth, the communication system will try to transport the messages according to a best-effort strategy. If it cannot handle the traffic, it will exercise backpressure flow control on the sender in order to protect the communication system from overload generated by a misbehaving sender (e.g., a *babbling* end system).

In order to be able to provide the guarantees, the communication system must contain information about the guaranteed bandwidth for each sender. This information can be contained in static protocol parameters that are pre-configured into the communication controller *a priori* or can be loaded into the communication controller dynamically during run-time. Rate constrained communication protocols provide temporal error detection and protection of the communication system from *babbling idiots*.

Table 7.2 Transport latency of AFDX on an A 380 configuration [Mil04]

Latency (ms)	0–0.5	0.5–1	1–2	2–3	3–4	4–8	8–13
Percent of traffic	2%	7%	12%	16%	18%	38%	7%

Rate-constrained protocols provide a *guaranteed maximum transport latency*. The actual transport latency will normally be significantly better (see Table 7.2), since under normal conditions the global traffic pattern is much smaller than the assumed peak. Since we cannot predict the instant of message delivery, rate constrained communication systems are *not deterministic* according to the definition of determinism given in Sect. 5.6.1.

7.4.1 Token Protocol

One of the early rate-constrained protocols is the *token protocol* controlling the access to a multi-access local area networks (LAN). In a *token system,* the right to transmit is contained in a special control message, the *token*. Whoever has the token is allowed to transmit. Two time parameters determine the response time of a token system, the *token-hold time* THT, denoting the longest time a node may hold the token, and the *token-rotation time* TRT, denoting the longest time for a full rotation of the token. The maximum TRT is the product of the number of nodes and the THT of each node, which determines the *guaranteed bandwidth* allocated to a node. A serious failure in any token system is the loss of the token, e.g., if the station that possesses the token fails. In such a situation, the network traffic is disrupted until some other node detects the *silence* by monitoring a time-out, and generates a new token. A token system can be structured as a bus or a ring. Token rings were standardized by IEEE standard 802.5. They have been in wide use some years ago.

7.4.2 Mini-slotting Protocol ARINC 629

In the mini-slotting protocol ARINC 629 [ARI91], the access to a shared bus is controlled by two time-out parameters, the *synchronization gap SG* controlling the entrance to a distributed waiting room, and *the terminal gap TG* controlling the access from the waiting room to the bus. The *synchronization gap SG* is identical for all nodes, whereas the *terminal gap TG,* the *personality timer,* is different for each node and must be a multiple of the propagation delay (called a *mini-slot)*. The following relation holds between these time-outs: $SG > Max\{TG_i\}$ for all nodes i. ARINC 629 is thus a *waiting-room protocol* similar to the *bakery algorithm* of Lamport [Lam74]. In the first phase, the set of nodes that wants to transmit a message is admitted to the *distributed waiting room* if there is silence on the bus for a longer duration than the *synchronization gap SG*. A node that has entered the

waiting room and senses silence on the bus for a duration that surpasses its personal terminal gap TG starts transmitting its message. This protocol logic guarantees that a node cannot monopolize the bus, since even the node with the shortest terminal gap TG (the highest priority node) is allowed to send a second message only after all other nodes that are in the waiting room have finished their message transmission. Typical values for the time-out parameters on a 2 Mbit/s channel are: terminal gap (determined by the propagation delay): 4–128 μs, synchronization gap SG longer than the longest terminal gap. The ARINC 629 protocol is used on the Boeing 777 airplane.

7.4.3 Avionics Full Duplex Switched Ethernet

Avionics Full Duplex Switched Ethernet (AFDX) is a rate-constrained protocol based on switched Ethernet. Whereas the message format and the physical layer of AFDX is in agreement with the Ethernet standard IEEE 802.3, the protocol allocates a statically defined bandwidth to each sender on a virtual link basis. A virtual link connects a sender with a specified number of receivers. An AFDX switch guarantees that

1. The delivery order of messages traveling on a virtual link is the same as the send order.
2. A minimal bandwidth and a maximum transmission latency and jitter is guaranteed on a virtual link basis.
3. There is no data loss due to buffer over-subscription in the switch.

The configuration table of the switch contains state information for every virtual link and enables the switch to protect the network from nodes that try to overload the network. The system integrator establishes the virtual links and sets the connection parameters. AFDX has been standardized under ARINC 664 and used in the Airbus A 380 and in the Boeing Dreamliner B787.

Table 7.2 shows typical values of the virtual link latency distribution, i.e., the jitter, on an AFDX configuration for the A 380 [Mil04]. This jitter will cause a significant action delay (see Sect. 5.5.2 and Problem 5.16).

7.4.4 Audio Video Bus

Physical connections in multimedia systems are predominantly unidirectional and point-to-point, resulting in substantial wiring harnesses. In order to simplify the wiring harnesses, special multi-media protocols have been developed. Some of these special protocols are incompatible with standard IT protocols such as Ethernet. On the other side, standard switched Ethernet does not provide the temporal quality of service needed in multi-media applications.

A communication system for audio-video streaming must support the following temporal *quality of service* requirements:

1. It must be possible to precisely synchronize multiple audio video streams that are generated at different physical locations, e.g., for lip synchronization or mixing content. The required precision of synchronization is in the microsecond range.
2. The worst-case transport delay of a multimedia data stream, including buffering delays at source and destination, must be bounded. The duration for the dynamic switching from one video stream to another video stream must be in the millisecond range.
3. The communication resources that are dynamically allocated to a multi-media stream must remain available for the duration of a session. This requires a dynamic resource reservation schema.

The IEEE 802.1 audio/video bridging (AVB) task force develops a set of protocols based on the Ethernet standard that meet the aforementioned requirements.

7.5 Time-Triggered Communication

In a time-triggered communication system, the sender and receiver(s) agree *a priori* on a cyclic time-controlled *conflict-free* communication schedule for the sending of time-triggered messages. This cyclic communication schedule can be expressed in the *cyclic model of time* (see Fig. 3.9 in Sect. 3.3.4), where the send and receive instants of a message, the *message cycles,* are represented by a *period* and *phase*. In every period, a message is sent at exactly the same phase. Since the communication system can be aware of the schedule, it can allocate the resources such that a time-triggered message is transported without any intermediate delay or buffering.

In some sense, time-triggered communication resembles *time-controlled circuit switching (TCCS)*, where a time-controlled dedicated channel between a sender and receiver is established for the *short duration of a single message transport*.

> **Example:** A coordinated set of traffic lights along a road that establishes periodically a *green wave* is a good metaphor for time-controlled circuit switching

We can distinguish the following three types of time-controlled circuit switching:

1. *Collision Avoidance Time-Controlled Circuit Switching (CA-TCCS).* In CA-TCCS, it is assumed that there are two message classes, the scheduled time-triggered messages and the sporadic event-triggered messages. The switch, knowing beforehand the conflict-free schedule of the time-triggered messages can shift the event-triggered messages such that conflicts between the event-triggered and time-triggered messages are avoided.
2. *Preemptive Time-Controlled Circuit Switching (P-TCCS).* In P-TCCS, it is assumed that there are two message classes, the scheduled time-triggered

messages and the sporadic event-triggered messages. In case of a conflict between an event-triggered message and a time-triggered message, the switch preempts the event-triggered message and sends the time-triggered message with a small delay and minimal jitter.

3. *Collision Tolerant Time-Controlled Circuit Switching (CT-TCCS).* In CT-TCCS, it is assumed that there are two message classes, the scheduled time-triggered messages and other uncontrollable messages or interfering signals, a situation characteristic for wire-less scenarios. The time-triggered communication controller will send multiple replicas of the same time-triggered message on diverse frequency channels at different preplanned instants, hoping that one of the replicas will reach the receiver unharmed.

Within the given fault hypothesis, time-triggered communication is deterministic according to the definition of determinism in Sect. 5.6.1. The *sparse model of time* ensures that messages that are sent at the same *active interval* of the sparse time base over independent (replicated) channels will arrive at the receivers at the same future *active interval* of the *sparse time base*. The jitter is bounded by the precision of the clock synchronization, which is usually in the sub-microsecond range.

Time-triggered control requires that the temporal control signals within a domain be derived from a *single* time source. This time source can be the synchronized global time (this is preferable) or the period of a single *leading process,* i.e., a process that establishes a basic period autonomously. In case a leading process establishes the time, the temporal control signals of all other cycles must be derived from the period of this leading process. The generation of the schedules is *simple*, if all periods are in a harmonic relationship.

> **Example:** If, within single synchronization domain, two processes are activated by unrelated control signals from two different unsynchronized time sources, then the phase relationship between these two processes will vary such that eventually the process executions will overlap.

The precise phase control of time-triggered communication makes it possible to align the processing actions and communication actions tightly within a distributed transaction and thus minimize the duration (the *dead-time*) of a control loop (see Fig. 3.9). This tight phase control is also performed if a cascade of switches must be traversed by a time-triggered message.

> **Example:** In the smart power grid timely end-to-end transport guarantees must be provided over an entire grid [Ter11]. Time-triggered communication minimizes the transport delay, supports fault-tolerance, and thus enables the realization of tight direct digital control loops over a wide area.

7.5.1 Time-Triggered Protocol

The Time-Triggered Protocol TTP, following the CA-TCCS schema, integrates time-triggered communication, temporal error detection, a fault-tolerant clock-

synchronization service, and a membership service in a *single protocol* with minimal protocol overhead [Kop93]. The system integrator must set up the parameters for the transmission slots of the end systems *a priori*. Event-triggered communication can be implemented by an overlay protocol on top of TTP.

Fault-tolerant clock synchronization is achieved by taking the difference between the *measured* and *specified* arrival time of every message as a measure for the difference of the clocks of the sender and receiver and by applying the *fault-tolerant average algorithm* (see Sect. 3.4.3) on these differences to calculate a correction factor for each local clock.

The *membership service of TTP* informs all connected nodes about the *health-state* of every cluster node and of the *violation of the fault-hypothesis* (should it occur) such that a never-give up (NGU) strategy can be activated quickly. The membership is encoded in a *membership vector* that contains as many bits as there are nodes in a cluster. A specified bit position of the membership vector is assigned to each node. When this bit is set to *TRUE,* a node is operating, if this bit is set to *FALSE*, this node is not operating correctly. The membership instant of a node is the periodic send instant of a message by this node. The *state* of a TTP controller (*C-state*) consists of the current time and the node membership vector. To enforce agreement on the C-state of all nodes of an ensemble, TTP calculates the CRC at the sender over the message contents concatenated with the C-state of the sender. The CRC at the receiver is calculated from the received message contents concatenated with the C-state of the receiver. If the result of the CRC check at the receiver is negative then either the message was corrupted during transmission or there is a disagreement between the C-states of the sender and receiver. In both cases, the message is discarded and the receiving node assumes that the sender has been faulty. We call this assumption the *self-confidence principle*. The self-confidence principle ensures that a single faulty node cannot kill a correct node in a system that is assumed to contain at most one faulty node.

If, in the above scenario, the sender has been correct – all other working nodes have received the message correctly – then the receiving node must have been faulty. An algorithm must tolerate that a faulty node makes further faulty decisions. It will send a message with a wrong membership vector and will be taken out of the membership by all other working nodes. If the receiving node had been right, a correct decision would have been taken and the sender of the original message would have been taken out of the membership. TTP operates on two physical channels and has an independent bus guardian at every node that protects the bus from babbling idiots – even a faulty node can send a message during its assigned time slot only. It fails silently outside its time slot.

Considering the services TTP provides it is a very data-efficient protocol well suited for applications that require a frequent update of short real-time data elements. Examples of such applications are industrial controls or the control of robot movements.

TTP has been formally certified for use in airborne system [Rus99]. It is deployed in the A 380 and the Boeing 787 aircraft and other aerospace and industrial control applications. TTP has been standardized in 2011 by the Society of Automotive Engineers (SAE) and is in the standardization process by ARINC.

7.5.2 Time-Triggered Ethernet

Time-Triggered Ethernet (TTEthernet) is an extension of the switched Ethernet standard IEEE 802.3 that supports *standard Ethernet traffic* on one side and provides a *deterministic message transport* on the other side [Kop08]. While the protocol controllers of the end systems can be standard Ethernet controllers, a TTEthernet switch distinguishes two message classes, the standard (event-triggered) Ethernet messages (ET-messages) and the deterministic time-triggered messages (TT-messages). Both ET-messages and TT-message formats are fully compliant with the Ethernet standard. The distinction between ET-messages and TT-messages can be made on the basis of the contents of the *standard Ethernet type field* or on other information in the standard Ethernet header (e.g., the address fields). The TTEthernet switch transports TT-messages with a constant small delay without intermediate storage in buffers, while ET-messages are transported during the time-intervals when no TT traffic is occurring. In case of conflict between an ET and TT messages, different conflict resolution strategies are applied.

The *entry-level TTEthernet system*, a 100 Mbit/s system, recognizes TT-messages based on the contents of the *standard Ethernet type field* and preempts an ET-message that is in its way employing the *P-TCCS strategy*. After the transport of the TT-message has been finished, the preempted ET-message is retransmitted autonomously by the *entry-level TTEthernet switch*. The *entry level TTEthernet switch* is stateless and does not require any parameterization on start up. It is up to the end systems to arrive at a conflict-free schedule for all TT messages. The *entry-level TTEthernet* system does not protect the communication system from babbling end systems, because it is stateless and does not know what is the intended correct temporal behavior of an end system.

Normal TTEthernet switches are parameterized with state-information that informs the switch about the periods and phases, the *cycles*, and the length of all time-triggered messages. Normal TTEthernet protects the communication system from babbling end systems. Since the switch has knowledge about the cycles of all TT-messages it can deploy the CA-TCCS strategy and shift the transmission of an ET message to its final destination such that a conflict with any TT-message is avoided. In some TTEthernet switches, the message schedules can be changed dynamically during the operation of the system.

Fault-tolerant TTEthernet switches provide redundant channels for the implementation of fault tolerant systems. A fault-tolerant clock synchronization establishes the required global time of sub-microsecond precision. The determinism makes the *fault-tolerant TTEthernet* the communication system of choice for the implementation of fault-tolerant systems.

TT Ethernet switches support transmission speeds of 100 Mbit/s and 1 Gbit/s. Some TT-Ethernet switches are certified to the highest-criticality class of Table 11.1. TTEthernet is in the process of standardization by ARINC. TTEthernet has been selected as the backbone communication system for the NASA Orion Program [McC09].

7.5.3 FlexRay

FlexRay is a communication protocol for automotive applications that has been designed by the FlexRay consortium [Ber01]. FlexRay is a combination of two protocols, a *time-triggered protocol* with fault-tolerant clock synchronization that is similar to TTP but without the membership service, and an *event-triggered protocol* that is similar to the ARINC 629 mini-slotting protocol but without the waiting room facility. The system integrator must partition the time into two successive intervals, one for the time-triggered communication and the other one for the event-triggered communication and set the parameters for the system operation accordingly. At present FlexRay is deployed in some car models of Audi and BMW.

Points to Remember

- The architectural requirements for the communication infrastructure of a distributed real-time system follow from the properties of *real-time data* that must be *temporally accurate* at the *instant of use*.
- A jitter of the message delay has a negative effect on the duration of the *action delay* and the *precision* of the clock synchronization.
- A shared communication system must detect and contain the temporal faults of any component (a *babbling idiot*), so that the communication among the components that are not directly affected by the faulty component is not compromised.
- The result of the intended physical action in the environment that is observed by a sensor component is the final end-to-end acknowledgement of an outgoing message.
- The behavior of a real-time communication system should be *deterministic* such that the order of messages is the same on all channels and the instants of message arrival of replicated messages that travel on redundant independent channels are close together.
- The communication topology in distributed real-time systems is multicast, not point-to-point. The same image of an RT entity is needed at a number of different components, e.g., at the man-machine interface, at a process-model component, and at an alarm-monitoring component.
- The *basic message transport service (BMTS)* that is provided at the architecture level is at the waist of a *waistline* model. The BMTS is implemented by (hidden) *low-level* protocols and is used to construct application-specific *high-level* protocols.
- Whenever a BMTS message has been sent across the Internet, we don't know what types and how many different *low-level protocols* have been activated.
- If an *exactly-once semantics* must be implemented, a *high-level protocol* that uses two or more BMTS messages must be provided.

- If the message length is less than the bit length of the channel, then the *best channel utilization* that can be realized with any media access protocol on a shared channel, such as a bus, is less than 50%.
- Implicit flow-control is the most appropriate flow-control strategy for the exchange of real-time data.
- Whenever queues are involved in a scenario, the possibility of queue overflow must be considered.
- In real-time systems, the tradeoff between reliability and jitter should be performed at the application level.
- Time-triggered communication resembles *time-controlled circuit switching (TCCS)*, where a time-controlled dedicated channel between the sender and receiver is established for the *short duration of a single message transport*.
- TTP integrates time-triggered communication, temporal error detection, a fault-tolerant clock-synchronization service, and a membership service in a *single protocol* with minimal protocol overhead.
- The self-confidence principle ensures that a single faulty node cannot kill a correct node in a system that is assumed to contain at most one faulty node.
- TTEthernet is an extension of the switched Ethernet standard IEEE 802.3 where standard (event-triggered) Ethernet messages (ET-messages) and time-triggered Ethernet messages (TT-messages) are supported.

Bibliographic Notes

The requirements for distributed safety-critical real-time systems onboard vehicles are analyzed in the SAE report J20056/1 "Class C Application Requirements" [SAE95]. An interesting report about a *Comparison of Bus Architectures for Safety Critical Embedded Systems* has been published by NASA [Rus03]. A rational for the design of time-triggered Ethernet is published in [Kop08].

Review Questions and Problems

7.1 Compare the requirements of *real-time* communication systems with those of *non real-time* communication systems. What are the most significant differences?

7.2 Why are application-specific end-to-end protocols needed at the interface between the computer system and the controlled object?

7.3 Describe the different flow-control strategies. Which subsystem controls the speed of communication if an *explicit* flow control schema is deployed?

7.4 Calculate the latency jitter of a high level PAR protocol that allows three retries, assuming that the lower level protocol used for this implementation has a dmin of 2 ms and a dmax of 20 ms. Calculate the error detection latency at the sender.

7.5 Compare the efficiency of event-triggered and time-triggered communication protocols at low load and at peak load.
7.6 What mechanisms can lead to thrashing? How should a system react if thrashing is observed?
7.7 Given a bandwidth of 500 Mbits/s, a channel length of 100 m and a message length of 80 bits, what is the limit of the protocol efficiency that can be achieved at the media access level of a bus system?
7.8 How do the nodes in a CAN system decide which node is allowed to access the bus?
7.9 Explain the role of the time-outs in the ARINC 629 protocol. Is it possible for a collision to occur on an ARINC 629 bus?
7.10 Describe the different types of *time-controlled circuit switching*!
7.11 What services are provided by the *TTP protocol*?
7.12 What is the *self-confidence principle?*
7.13 Explain the principle of operation of time-triggered Ethernet!