TTEthernet: Time-Triggered Services for Ethernet Networks

Wilfried Steiner Günther Bauer
Chip IP Design
TTTech Computertechnik AG
forename.surname@tttech.com

***Abstract*—Following the networking trends in other high-volume industries such as automotive, industrial automation, railway and telecommunications, where high QoS is required, synchronous networking is gaining more importance for different mission-critical systems.**

***TTEthernet* is a communication infrastructure designed for systems with mixed-criticality requirements. It is for example possible to operate command and control functions and audio/video applications on the same physical network. In order to guarantee partitioning, *TTEthernet* realizes a fault-tolerant synchronization strategy. This synchronization strategy can be seen as the interplay of a collection of time-triggered services.**

In this paper we discuss this collection in a generic way and present the *TTEthernet* services in particular. We also discuss the interplay of the *TTEthernet* services in example use cases.

I. Introduction

In a real-time network, time-triggered services allow a set of individual components to work as a coordinated whole, which establishes two powerful properties: firstly, a strong system-wide determinism is established and, secondly, the given physical resources can be highly efficiently utilized. The time-triggered services bring the local clocks of the individual components into agreement and execute their continual alignment. These synchronized local clocks can then be used to trigger system-wide coordinated actions, such as the transmission of messages, which are said to be time-triggered. In addition to time-triggered communication only, the synchronized local clocks can also define intervals in which event-triggered communication is allowed, which enables mixed real-time/non-real-time communication on a single physical network.

Temporal Partitioning: The global time can be used as isolation mechanism when devices become faulty; we say that the global time operates as a "temporal firewall". In case of a failure it is not possible for a faulty application to have untimely access to the network. Depending on the location of the failure, either the communication controller itself or the switch will block faulty transmission attempts. Failures of the switch can be masked by end-to-end arguments such as CRCs or by high-integrity designs.

Efficient Resource Utilization: The global time contributes to efficient resource utilization in several ways. Time-triggered communication for example makes it possible to minimize the memory buffers in network devices such as switches, as the time-triggered communication schedule is free of conflicts. Hence, the switches do not have to be prepared for bursts of messages that have to be delivered over the same physical link. A minimal time-triggered switch design could even multiplex media access logic such as reception or transmission logic. A second way of effective resource utilization is buffer memory in the nodes, which can be minimized as the sensor values can be acquired according to the global time, immediately before sending the message. Finally, a third kind of effective resource utilization is power management, which can be saved analogously to memory.

Precise Diagnosis: A global time-stamping service simplifies the process of reconstruction of a chain of distributed events. At the same time, the synchronous capturing of sensor values makes it possible to build snapshots of the state of the overall systems.

Composability[1]: The global time allows the specification of devices not only in the value domain, but also in the temporal domain. This means that already during the design process of devices, the access pattern to the communication network can be defined. The devices can then be developed in parallel. Upon integration of the individual devices, it is guaranteed that prior services are stable and that the individual devices operate as a coordinated whole.

TTEthernet is an industrial development which advances the fundamental academic TT-Ethernet concept [2]. *TTEthernet* circumvents limitations of Ethernet technology for the application in safety-critical, hard real-time, and fault-tolerant systems, and offers completely new capabilities for design of innovative system architectures. It is designed to satisfy the requirements of the aerospace industry. Therefore it is interesting for cross-industry applications taking advantage of synchronous operation in critical embedded systems. *TTEthernet* is currently undergoing standardization at SAE with assigned draft standard number AS6802. Basis for the standard document is the *TTEthernet* specification [3].

In the next section we give an overview of the "Time-Triggered Service Classes" that describe the time-triggered services to be realized by a communication infrastructure for mixed-criticality systems in a generic way. Following the generic discussion we present the specific *TTEthernet* Services in Section III. In Section IV we discuss some example configurations of *TTEthernet*. We summarize and conclude in Section V.

II. Time-Triggered Service Classes

We observe throughout a broad range of application areas that the number of time-triggered communication protocols is increasing. While these time-triggered protocols differ significantly in the algorithms they implement to realize time-triggered communication, there is a common set of problems that has to be solved. We call this common set of problems the Time-Triggered Service Classes.

Scheduled Dispatch Service Class: This class specifies methods for time-triggered dispatch of messages according to an off-line specified schedule table. This includes the representation of the schedule in the components, e.g., how the schedule is stored in local memory.

Clock Synchronization Service Class: This service class represents services that ensure that the local clocks of the components in the communication infrastructure stay synchronized to each other once synchronization is established. Examples of clock synchronization algorithms can be found in [4], [5], [6].

Startup Service Class: The startup service class covers methods and services to initially synchronize the components in the communication infrastructure. This can be a coldstart procedure or an integration/reintegration procedure. Examples of startup algorithms can be found in [7], [8], [9].

Membership Service Class: Membership services are low-level diagnostic services that continually monitor the system's health state. In particular, such services could reflect which end systems are present in the systems and which are not - for example because of transient/permanent failures. An examples membership algorithm is the TTP membership algorithm [10].

Clique Detection and Resolution Service Class: This service class defines measures that detect clique scenarios. These are unintended scenarios where disjoint subsets of components are synchronized within the subset but not over subset boundaries. Clique Resolution services define methods that re-establish synchronization when cliques have been formed and detected. Example clique detection and resolution services are discussed in [11].

External Synchronization Service Class: This service class specifies methods that allow the communication infrastructure to synchronize to an external time source. Examples of external synchronization approaches are presented in [12].

Configuration and Maintenance Service Class: This service class defines services for how a communication infrastructure can be configured and maintained. Such services include for example configuration download procedures.

Dataflow-Integration Service Class: This service class defines measures for how message classes with different characteristics can be integrated such that all those message classes can use the same physical medium. In particular the integration of event-triggered and time-triggered messages classes is of interest in this service class.

Legacy Service Class: Existing protocols have inter-operability requirements. This service class aims to identify these requirements and provide glue functionality to allow inter-operability.

Integrity Service Class: This service class defines services that enhance the integrity of the communication infrastructure. In particular we are interested in two types of integrity measures: a guardian measure that can be central, local, or both, and end-to-end arguments, such as sequence numbers and time-stamps.

Availability Service Class: This service class defines services that enhance the availability of a communication infrastructure. Such services include redundancy management of communication channels and redundancy management in case of fault-tolerant computation entities such as Triple-Modular Redundant (TMR) configurations.

The complexity of the actual services that are realized for the service classes above heavily depends on the system requirements. A master-based system, for example, will allow the realization of very simple services, and a single function may be sufficient to address multiple service classes at the same point in time. A master-less system will require services to be realized in the form of distributed algorithms, which are inherently more complex. On the other hand, master-less systems provide higher system reliability, as the failure of a single device will typically not result in an overall system loss.

III. The *TTEthernet* Services

A. TTEthernet Scheduled Dispatch Service

The scheduling problem for time-triggered communication in bus-based networks is a well-understood problem and there are several commercial tools available. With the movement to tree-based networks, which is the standard Ethernet topology for wire-speeds of 100 Mbit/s and above, the scheduling problem becomes generalized to concurrent time-triggered dataflows: while the bus-based networks support only broadcast communication, the tree-based networks allow multicast or even unicast communication. The multicast/unicast dataflows may now be able to run in parallel.

Furthermore the store-and-forward switch design of the *TTEthernet* switches allows a decoupling of the receive schedule from the relay schedule in a *TTEthernet* switch. Hence, a time-triggered message that is received at time $t_{receive}$ will be intermediately stored and only relayed when the pre-configured relay point in time is reached t_{tt_relay}.

For a given set of messages to be scheduled and under the assumption that the store-and-forward delays imposed by the switches are acceptable, the scheduling complexity in a concurrent multicast/unicast network is equal to or less than in a broadcast-only network. This is because broadcast is just a special-case of multicast and each solution for broadcast would immediately be applicable also to multicast. On the other hand, multicast/unicast spans a larger solution space to the scheduling problem, where the size of the solution space grows directly with the concurrency of the time-triggered dataflows.

B. TTEthernet Clock Synchronization Service

The *TTEthernet* synchronization services use so-called *Protocol Control Frame*s as synchronization messages. For clock synchronization a special type of *Protocol Control Frame*, the "Integration Frame (IN)" is used.

TTEthernet specifies a two-step synchronization approach as depicted in Figure 1. In the first step Synchronization Masters send IN frames to the Compression Masters. The Compression Masters then calculate a configurable fault-tolerant average or median value from the relative arrival times of these IN frames and send a new IN frame out in a second step. This new IN frame is then also sent to Synchronization Clients.

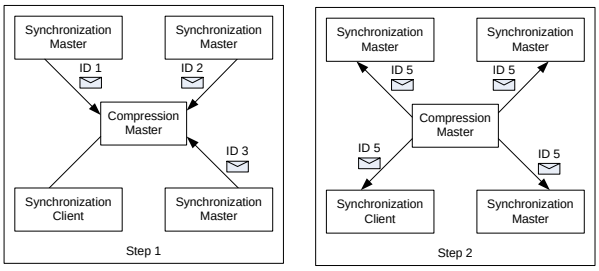

Fig. 1. TTEthernet two-step synchronization approach

The decision regarding which devices are configured as Synchronization Master, Synchronization Client, and Compression Master arises from the requirements of the system architecture. For simplicity, the examples in this *TTEthernet* specification configure end systems as Synchronization Master and switches as Compression Master. Note, that this is not a *TTEthernet* restriction; system configurations with end systems configured as Compression Masters and switches as Synchronization Masters are possible. Switches and end systems not configured either as Synchronization Master or Compression Master will be configured as Synchronization Client.

C. TTEthernet Startup/Restart Service

TTEthernet defines a four-step startup service that is also executed in case of a system-wide restart. The four steps realize two rounds of message exchange in the system, in which two additional types of *Protocol Control Frame*s are exchanged: the "Coldstart Frames (CS)" and the "Coldstart Acknowledge Frame (CA)".

In the first step, the *Synchronization Master*s will send out CS frames, if they do not manage to integrate to a running system for a configurable duration. In the second step the *Compression Master*s will distribute the CS frame as broadcast to all other *Synchronization Master*s in the system. The third step is started when a *Synchronization Master* receives a CS frame. Then it will wait for another configurable timeout after which it acknowledges the CS frame with a CA frame. The CA frame is again distributed by the *Compression Master*s as the fourth step. This finalizes the startup sequence. A *Synchronization Master* that receives a CA frame will wait for a configurable timeout and start synchronized operation hereafter. As either the CS or the CA frame are broadcasted consistently in the *TTEthernet* network, all *Synchronization Master*s that are already running will decide to enter synchronized operation at approximately the same point in time, and hence the initial synchronization is established.

In the first round of synchronized operation the *Synchronization Master* tests whether there is a sufficiently high number of other *Synchronization Master*s present; if so, it will stay in synchronized operation, if not it will repeat the startup service.

D. TTEthernet Membership Function

TTEthernet provides only a basic membership function as core service, which is used by the clock synchronization service, the startup/restart service, and the clique detection service.

The *Protocol Control Frame*s maintain a field called "membership_new". This field is a bit-vector with a one-to-one relation between bit and *Synchronization Master*. In step two of the clock synchronization service, the *Compression Master* generates a *Protocol Control Frame* in which it will set the respective bit in the membership_new field for all *Synchronization Master*s that provided an IN frame during step one of the clock synchronization service. Hence, a receiving *Synchronization Master* or *Synchronization Client* will know from the membership_new field, which *Synchronization Master*s contributed to the generation of this IN frame.

E. TTEthernet Clique Detection and Resolution Service

There are three clique detection services in *TTEthernet*: the synchronous clique detection, the asynchronous clique detection, and the relative clique detection.

The synchronous clique detection runs locally on all components in the system and checks if there are sufficient *Synchronization Master*s synchronized to the respective component. For this it evaluates the IN frame produced in step two of the clock synchronization service: if the number of bits set in the membership_new field falls below a configurable threshold then the synchronous clique detection returns successful.

The asynchronous clique detection also runs locally on all components in the system. It keeps track of how many *Synchronization Master*s there are operating but which are not synchronized to the respective component. This is done by locally storing the membership bits from those IN frames that it receives at unexpected points in time and, therefore considered to stem from *Synchronization Master*s that are unsynchronized to the respective component. At the beginning of each re-synchronization interval the component checks whether the number of bits set in the membership_new fields of unsynchronized IN frames is above a configurable threshold. If so, the asynchronous clique detection service returns successful.

The relative clique detection is executed in the *Synchronization Master*s. It checks at the beginning of each re-synchronization interval whether the number of synchronized *Synchronization Master*s is equal to or below the number of

unsynchronized *Synchronization Master*s. If this is the case, the relative clique detection returns successful.

When any of the clique detection services returns successful the respective component will go to an unsynchronized mode and try to re-integrate. In case of a *Synchronization Master* this is then already part of the the startup/restart service. That means, in case of global synchronization loss the *Synchronization Master*s will automatically restart the *TTEthernet* system.

F. TTEthernet External Synchronization Function

TTEthernet features external clock synchronization via the host interface. When enabled, the user can set an external clock synchronization value which is then applied additionally to the internal clock synchronization function.

G. TTEthernet System-of-Systems Synchronization Service

In *TTEthernet* the synchronization of the local clocks in the system is based on the exchange of *Protocol Control Frame*s. To support system-of-systems synchronization *TTEthernet* uses priorities which are carried in a field of the *Protocol Control Frame*s.

Each *Synchronization Master* and *Compression Master* has an *a priori* defined static priority which is set in all *Protocol Control Frame*s that they transmit. *Compression Master* will also only accept *Protocol Control Frame*s for synchronization and compression that have the *Compression Master*s priority set. *Synchronization Master*s and *Synchronization Client*s, however, have the capability to accept also *Protocol Control Frame*s of other priority than their own for synchronization. The decision whether to accept other priorities can be configured to require a host acknowledgment or can be done fully automatically always towards the highest priority in the system. This is also the rationale, why *Compression Master*s cannot change towards other priorities in the system (even towards higher priorities): as the decision whether to change or not is potentially application-driven the *Compression Master* which is typically realized in the *TTEthernet* switches would have to be informed of this decision via a fault-tolerant protocol. This was avoided for the sake of algorithmic simplicity.

H. TTEthernet Configuration and Maintenance Service

TTEthernet provides remote configuration via bus-download. In the end systems the configuration is done via the host interface and the decision whether to execute a re-configuration process can be rejected at this end, to prevent fault propagation. The switches are also remotely configured, but realize a dedicated configuration module, such that a host is not necessary. In the switches, an unlock protocol that involves interactions with a sufficiently high number of end systems is executed before a switch will change its configuration. Again this is required to maintain the fault-tolerance properties.

As *TTEthernet* realizes time-triggered as well as rate-constrained dataflows which do not demand a synchronized system, the system will even be in a state where re-configuration is possible when synchronization is not possible, for example when the failure hypothesis is violated. In this case, the re-configuration may also be used as failure isolation mechanism.

I. TTEthernet Dataflow-Integration Function

In *TTEthernet* a single physical network is used for concurrent synchronized/unsynchronized dataflow and control-flow. Hence, temporal interactions of their respective messages cannot be avoided and the *Protocol Control Frame*s are subject to dynamic temporal delays.

TTEthernet mitigates these dynamic delays with the so-called "permanence function". Each *Protocol Control Frame* will measure its delay through the network. This is done by maintaining a field in the *Protocol Control Frame*, called the "transparent clock" that is updated by each component in the *TTEthernet* network that sends, relays, or receives the *Protocol Control Frame* by the delay that it imposed. In the receiving component, the *Protocol Control Frame* will not immediately used for the synchronization process, but it will be delayed (maximum delay through the network minus the actual delay through the network as written in the transparent clock field). This means that the receiving component will always stretch the dynamic actual transmission time to the almost constant maximum transmission time. Hence, the permanence function provides a clean separation between the network jitter and the synchronization services.

J. TTEthernet Legacy Services

TTEthernet provides legacy services in a variety of forms. Firstly, and most obvious, *TTEthernet* is backward compatible to standard Ethernet. All *TTEthernet* dataflow and control-flow messages adhere to the standard Ethernet frame format. Dataflow messages are communicated according to a time-triggered, a rate-constrained, or a best-effort communication paradigm. All these so-called traffic classes do not require specific contents in the payload. This means that legacy protocols such IP, TCP, UDP, or any other higher layer protocol executed using the communication paradigm of choice. The identification of the traffic class is encoded in the Ethernet MAC destination field: time-triggered and rate-constrained messages use pre-configured multicast addresses. Best-effort messages are only restricted from using these multicast addresses, but free to select their Ethernet MAC destination address otherwise.

The compatibility to standard Ethernet means that a standard Ethernet receiver is able to operate all messages that are communicated in a *TTEthernet* network. So, for example, a maintenance engineer would be able to use a standard laptop for monitoring purposes. From a sender perspective, an end system is also able to transmit best-effort messages and to some extent also rate-constrained messages without the requirement for additional communication logic. For time-triggered communication the synchronization services have to be implemented. However, this implementation can be purely software-based using COTS Ethernet controllers.

The rate-constrained traffic class for dataflow messages is fully compatible to the communication paradigm defined in the ARINC 664 part 7 standard [13]. The *TTEthernet* switches, therefore, allow communication with legacy avionics.

The switch, as centralized component in a *TTEthernet* network, can be extended for bridging functionality. So it is easily possible to extend the modular Chip IP to become compatible to legacy protocols like TTP [14], FlexRay [15], CAN [16], or even Mil-Std 1553 and ARINC 429. Synchronous protocols (as TTP, FlexRay, or the Mil-Std 1553) the *TTEthernet* switch could even operate as Master component that "simulates" a set of legacy controllers if required.

K. TTEthernet Integrity Functions

There are two integrity concepts in *TTEthernet*: the central-guardian function and the high-integrity design. The central-guardian function is an integrity means that is realized in a remote component like the switches in a *TTEthernet* system. It can be used to protect the systems from arbitrarily faulty components as for example protection against a babbling CS or CA frame sender that tries to continually restart the system. The high-integrity design on the other hand is an integrity means that contains the error at the source. A self-checking pair design is an example of a high-integrity design.

For ultra-high dependable systems, the *TTEthernet* switches are always realized as high-integrity designs. *Synchronization Master*s though, can be realized as standard integrity if the central-guardian function in the switches is enabled. When the *Synchronization Master*s are realized as high-integrity components all dual-failures can be tolerated. When the *Synchronization Master*s are realized as standard-integrity components all single failures can be tolerated.

L. TTEthernet Availability Functions

Availability in *TTEthernet* is realized via active redundancy. In general, the intended topologies are replicated tree structures, where degree of replication (r) is a function of the failures to be tolerated (k): $r \geq k + 1$. For example the tolerance of two switch failures will require a system architecture of at least three redundant communication channels between any two end systems in the *TTEthernet* system.

IV. Time-Triggered Services – Example Use Cases

This section discusses some use cases of *TTEthernet* in which we highlight the interplay of the time-triggered services as discussed above. Figure 2 depicts two example TTEthernet networks: on the left a single-channel multi-hop network and on the right a dual-channel single-hop network.

A. Power-On Use Case

In this section we discuss example scenarios that show how *TTEthernet* will reach synchronized operation after initial power-on of the system. In particular we show synchronization scenarios in the fault-free case and in scenarios with a faulty *Synchronization Master* or faulty *Compression Master* present. Generally, the startup/restart service of *TTEthernet* does not demand a particular power-on sequence; the service is designed such that an initial synchronization is reached a bounded time after sufficient non-faulty components have been powered-up.

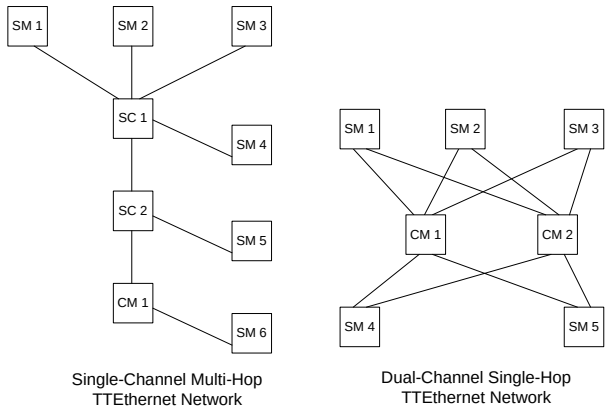

Fig. 2. TTEthernet Example Networks

1) Failure-Free Case: We can distinguish two types of fault-free startup scenarios: (a) the collision-free CS scenario and (b) the colliding CS scenario.

The collision-free startup scenario is essentially described in Section III-C, which describes the exchange of the CS and the CA frames. We will discuss the colliding CS scenario next:

CS frames from different *Synchronization Master*s may collide because the decision that the startup/restart service has to be executed is taken at a point in time that happens a pre-configured timeout after power-on of a *Synchronization Master*. Regardless, whether this pre-configured time is the same or different in any two *Synchronization Master*s, as the power-on time of a component is unknown and potentially arbitrary, it can always happen that two or more *Synchronization Master*s are powered-on at points in time so that they will reach their decision point to execute the startup/restart service at the same (or approximately the same) point in time. This then results in all these *Synchronization Master*s sending out a CS frame at approximately the same point in time.

Collisions of CS frames are resolved by the startup/restart service by always reacting to the last CS frame received: let us assume that n *Synchronization Master*s send CS frames at approximately the same point in time. All these CS frames are broadcasted by all *Compression Master*s in the network. A receiving *Synchronization Master* will now restart the third step of the startup/restart service with each CS frame it receives. So, basically that means that with each CS frame a *Synchronization Master* receives, its local timer that tells the *Synchronization Master* when to send its CA frame is reset. There are several notes on this approach:

- The *Synchronization Master* can be configured not to react to CS frames that it has sent itself in step 1 of the startup/restart service. The identification is done using the membership_new field (Section III-D) in the received CS frame, which will indicate its original sender.

- We rely on the *TTEthernet* dataflow integration function (Section III-I) to mask the network latency imposed by colliding CS frames.
- CS frames received in the *Synchronization Master*s from different *Compression Master*s will not be merged before used in the startup/coldstart service. This means that also in collision-free scenarios a *Synchronization Master* will execute a state transition with each single CS frame it receives: the first CS frame will cause the *Synchronization Master* entering the third step of the service, each following reception in the third step of a CS frame will cause a "re-entering" of the third step.

In the fault-free case, the *Synchronization Master*s are already synchronized after reception of the latest CS frame. After exchange of the CA frame the *Synchronization Master*s start the clock synchronization service (Section III-B). If the number of bits set in the membership_new value of the received IN frame in step two of the clock synchronization service is sufficiently high, then the *Synchronization Master*s will remain synchronized and continue clock synchronization. This first test on the number of bits set in the membership_new field can be interpreted as the first execution of the synchronous clique detection service (Section III-E).

2) Faulty Synchronization Master Case: In a high-integrity configuration, a faulty *Synchronization Master* is allowed to drop arbitrary sequences of input frames as well as to fail sending to an arbitrary number of channels. In a single failure case the failure to send either CS or CA will not have an impact on the startup/restart service and may be seen as failure free. However, there is an interesting failure case when the faulty *Synchronization Master* is started up with a low number of correct *Synchronization Master*s (without loss of generality, we assume only a single correct *Synchronization Master* to be powered-up): the faulty *Synchronization Master* can execute the startup/restart service and the first round of clock synchronization error-free, which causes the faulty *Synchronization Master* and the correct *Synchronization Master* to enter and stay in synchronized operation mode. Only now, the faulty *Synchronization Master* will start to exhibit its faulty behavior by not sending its following IN frames. Hence, the correct *Synchronization Master* will remain running on its own in synchronized operation. This is not critical *per se*. However, due to fault-tolerance reasons the threshold for integration may be set to at least two (this means a newly powered-on *Synchronization Master* will only integrate to an IN frame with a membership_new vector that has at least two bits set). Hence, *Synchronization Master*s that are powered-on late, will not be able to integrate. This situation is resolved by these *Synchronization Master*s to execute the startup/restart service themselves. In *TTEthernet* the startup/restart service takes precedence over synchronized operation, which prevents clique formation in such scenarios.

In a standard-integrity configuration a *Synchronization Master* is even allowed to send arbitrary sequences of frames. This failure mode is masked by the central guardian integrity function (Section III-K).

3) Faulty Compression Master Case: In a single failure scenario a faulty *Compression Master* has no impact as long as there exists at least one other communication channel with a correct *Compression Master*. However, as we do not require any particular power-on sequence it is also a valid scenario that only the faulty *Compression Master* and several *Synchronization Master*s are powered-up first. In such scenarios the *Compression Master* is able to generate clique scenarios by exhibiting its inconsistent relay behavior: a simple failure mode of a *Compression Master* is a static separation of communication flow, e.g. the faulty *Compression Master* will split its set of communication ports into two disjoint subsets and will relay messages received on a port of a given subset only to the ports that are in the same subset of ports. As there is no propagation from synchronization information between the respective subsets of *Synchronization Master*s, both subsets will execute the startup/restart service in isolation and will finally establish clique formations.

These clique formations will be resolved once a correct channel is present as then the *Synchronization Master*s will receive also the *Protocol Control Frame*s from the respective other clique and the asynchronous clique detection function will trigger the startup/restart service. As there exists now a non-faulty channel, CS and CA frames are propagated consistently and clique formations are prevented.

4) Concurrent Faulty Synchronization Master and Faulty Compression Master Case: The previously discussed power-on scenarios considered only a single faulty component at a time, which makes the argumentation of a successful execution of the startup/restart service simple. In this scenario we discuss startup with two faulty components present, a faulty *Synchronization Master* and a faulty *Compression Master*. Note, that this failure mode requires the *Synchronization Master*s to be high-integrity.

In this scenario the faulty *Synchronization Master* may only send its frames to the faulty *Compression Master*. The faulty *Compression Master* on the other hand will exhibit its inconsistent failure mode and relay frames only to a varying subset of *Synchronization Master*s. Hence, in this combined case of faulty *Synchronization Master* and faulty *Compression Master* an inconsistent transmission of *Protocol Control Frame*s is even possible when correct channels are already up and running.

The startup/restart service has been designed to solve this type of failure scenarios. In particular, the scenario in which the faulty *Synchronization Master* is first to send a CS frame, which is inconsistently relayed by the faulty *Compression Master* is mitigated by the CA frame, which is sent by each *Synchronization Master* that received the CA frame. As a correct *Synchronization Master* always sends on all channels, its *Protocol Control Frame*s will be distributed consistently in the *TTEthernet* network.

In those scenarios where the faulty *Synchronization Master* is not the first to send a CS frame, the potential inconsistent transmission of its CA frame will be masked by other *Synchronization Master*s, because the CS frame must have been sent

by a correct *Synchronization Master* and has been distributed consistently over a correct channel in the first place.

Note that a correct *Synchronization Master* and therefore also a faulty high-integrity *Synchronization Master* will not acknowledge its own CS frame. Thus, the faulty *Synchronization Master* and faulty *Compression Master* cannot maliciously cooperate to produce a CA frame at arbitrary points in time during operation.

B. Late Integration Use Case

A *Synchronization Master* or a *Compression Master* may have to integrate into an already synchronized *TTEthernet* network either because the component has been powered on late, or the component exhibited a transient failure. *TTEthernet* allows late integration or re-integration also in the presence of faulty components, which we discuss in the following.

1) Integration of Synchronization Master: An integrating *Synchronization Master* has to be prevented from synchronizing to a faulty *Synchronization Master* only as this potentially results in clique formation. In *TTEthernet* this problem is solved by synchronization only to IN frames with sufficient bits set in the membership_new field. For example, in a system that is configured to tolerate a single faulty *Synchronization Master* all *Synchronization Master*s would be configured to integrate only to IN frames with at least two bits set in the membership_new field, as this is a necessary and sufficient indication that the timeline that is associated with this IN frame is supported by at least one correct *Synchronization Master*.

2) Integration of Compression Master: An integrating *Compression Master* will execute the same integration procedure as an integration *Synchronization Master*, and has to realize some additional functionality to prevent system-wide reset in a scenario with two or more faulty *Synchronization Master*s.

The *TTEthernet* startup/restart service is based on the exchange of the CS and the CA frame. Hence, failure scenarios have to be prevented that consist of two faulty *Synchronization Master*s which cooperate to produce a faulty sequence of *Protocol Control Frame*s at a point in time when the overall system already reached synchronous operation. The *Compression Master*, therefore, will block CS frames after power-up for at least one re-synchronization interval. This time is sufficient for the *Compression Master* to integrate to a synchronized system. If the *Compression Master* is not able to integrate it will relay CS frames until it reaches synchronized operation with a sufficiently high number of *Synchronization Master*s present. Then it will again prevent CS frames from relay.

C. System-of-Systems Example

The previous use cases described *TTEthernet* network configurations with only a single priority. For System-of-Systems support, *TTEthernet* provides synchronization priorities as discussed in Section III-G. Figure 3 shows an example of a System-of-Systems network.

This example consists of a high-priority *TTEthernet* network and a low-priority *TTEthernet* network and a communication link that connects the two networks to each other.

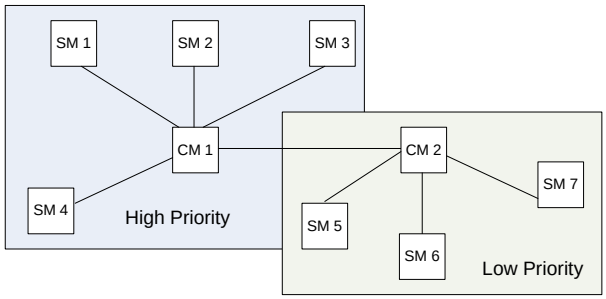

Fig. 3. Single-Channel System-of-Systems TTEthernet Network

1) Late Power-on of Low-Priority Network: One use case in system-of-systems architectures is that a high-priority network is running and lower-priority networks are powered-on as the current mission mode requires. For example, let us assume that the low-priority network in Figure 3 is powered-on late at a point in time when *Synchronization Master*s 1..4 and *Compression Master* 1 are already synchronized. This means that in the high-priority network *Protocol Control Frame*s are communicated that have their priority field set to high. As soon as *Compression Master* 2 is powered-up it will receive the high-priority *Protocol Control Frame*s and relay them to the *Synchronization Master*s 5..7. Note, that *Compression Master* 2 will not directly synchronize towards these high-priority frames, but just forward them to its attached *Synchronization Master*s. When the low-priority *Synchronization Master*s 5..7 receive the high-priority *Protocol Control Frame*s they can be off-line configured to automatically change their current priority for synchronization towards the highest priority available. The alternative configuration option is to require a host acknowledgment for this decision. In both cases the low-priority *Synchronization Master*s will not change the contents of their *Protocol Control Frame*s, hence, they will continue to send out *Protocol Control Frame*s with their low value set in the respective synchronization priority field. These low-priority *Protocol Control Frame*s will now be used by *Compression Master* 2 for synchronization. As already discussed under Section III-G, the indirect synchronization of *Compression Master* 2 towards the low-priority *Protocol Control Frame*s is done because the *Synchronization Master*s may be configured to require a host acknowledgment for a change in synchronization priority, which will not be directly accessible to the *Compression Master*.

2) Late Power-on of High-Priority Network: In this example we assume that the low-priority *TTEthernet* network is powered-up first and synchronized before the high-priority network. In general, such power-up scenarios may cause an overall system-of-systems restart as the high-priority *Synchronization Master*s may not accept the low-priority *Protocol Control Frame*s for synchronization and startup the high-priority *TTEthernet* network independently.

Although the strict priority mechanism for system-of-systems architectures may lead to a reset of the global time in

the low-priority network, it makes the selection of *Protocol Control Frame*s for synchronization simple and oscillation effects are easily excluded. Oscillation effects could otherwise be established when the components will not manage to converge to an agreed set of *Protocol Control Frame*s for synchronization, but continue to change this set differently.

The smooth integration of a high-priority *TTEthernet* network to a low-priority network can be provided via appropriate architectural measures. In this example, some additional high-priority *Synchronization Master*s (say SM H1 and SM H2) could be added to the low-priority *Compression Master* 2. SM H1 and SM H2 can be configured to accept lower priority *Protocol Control Frame*s, when they do not receive high-priority *Protocol Control Frame*s for a configurable number of re-synchronization intervals. As a second architectural choice the high-priority *Compression Master* 1 has to be powered-up before the high-priority *Synchronization Master*s 1..5. Now, these two architectural choices allow a smooth integration of the high-priority global time to the low-priority global time: SM H1 and SM H2 will source high-priority *Protocol Control Frame*s which are synchronized with the low-priority global time. Once CM 2 is powered-up it will forward these high-priority *Protocol Control Frame*s to its attached *Synchronization Master*s 1..5. These will then integrate to the high-priority global time sent by SM H1 and SM H2 and send themselves *Protocol Control Frame*s that are synchronized with the low-priority global time. The low-priority *Synchronization Master*s 5..7 can now change their current synchronization priority to high without reset.

V. Summary

In this paper we discussed the concept of time-triggered service classes which describes in a generic way services and functions necessary for time-triggered communication. Following the generic discussion we elaborated on the *TTEthernet* services in particular. Finally, we showed the interaction between the *TTEthernet* services in some example use cases that cover fault-free and faulty power-up scenarios as well as late integration and system-of-systems synchronization.

Acknowledgments

This paper solely represents the personal opinion of the listed authors. Under no circumstances can the material presented in this paper be credited as an official statement of their affiliations.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement $n°236701$ (*CoMMiCS*).

References

[1] H. Kopetz and R. Obermaisser, "Temporal composability," *IEE's Computing And Control Engineering Journal*, Jan. 2002.

[2] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, "The time-triggered ethernet (tte) design," *8th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC), Seattle, Washington*, May. 2005.

[3] W. Steiner, *TTEthernet Specification*, TTA Group, 2008, Available at http://www.ttagroup.org.

[4] H. Kopetz and W. Ochsenreiter, "Clock Synchronization in Distributed Real-Time Systems," *IEEE Transactions on Computers*, vol. C-36, no. 8, pp. 933–940, 1987.

[5] F. B. Schneider, "Understanding protocols for byzantine clock synchronization," Tech. Rep., 1987.

[6] L. Lamport and P. M. Melliar-Smith, "Synchronizing clocks in the presence of faults," vol. 32, no. 1, pp. 52–78, Jan. 1985.

[7] W. Steiner and M. Paulitsch, "The transition from asynchonous to synchronous system operation: An approach for distributed fault-tolerant systems," in *Proceedings of ICDCS*. Vienna, Austria: IEEE, Jul. 2002.

[8] W. Steiner and H. Kopetz, "The startup problem in fault-tolerant time-triggered communication," *International Conference on Dependable Systems and Networks (DSN 2006)*, Jun. 2006.

[9] M. Paulitsch and B. Hall, "Starting and resolving a partitioned brain," in *ISORC '08: Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 415–421.

[10] G. Bauer and M. Paulitsch, "An Investigation of Membership and Clique Avoidance in TTP/C," *19th IEEE Symposium on Reliable Distributed Systems, 16th - 18th October 2000, Nürnberg, Germany*, Oct. 2000.

[11] W. Steiner, M. Paulitsch, and H. Kopetz, "The tta's approach to resilience after transient upsets," *Real-Time Systems*, vol. 32, pp. 213–233, Feb. 2006.

[12] M. Paulitsch, "Fault-tolerant clock synchronization for embedded distributed multi-cluster systems," Doctoral thesis, Institut für Technische Informatik, Technische Universität Wien, Treitlstr. 1-3/3/182-1, Vienna, Austria, 2002, available at http://www.vmars.tuwien.ac.at.

[13] AEEC, *ARINC PROJECT PAPER 664, AIRCRAFT DATA NETWORKS, PART7, AFDX NETWORK (DRAFT)*, AERONAUTIC RADIO, INC., 2551 Riva Road, Annapolis, Maryland 21401-7465, Nov. 2003.

[14] H. Kopetz, *TTP/C Protocol – Version 1.0*. Vienna, Austria: TTTech Computertechnik AG, Jul. 2002, Available at http://www.ttpforum.org.

[15] *FlexRay Communications System - Protocol Specification - Version 2.1*. FlexRay Consortium, 2005, Available at http://www.flexray.com.

[16] H. Chen and J. Tian, "Research on the controller area network," *Networking and Digital Society, International Conference on*, vol. 2, pp. 251–254, 2009.