Presented at the 29th IEEE/AIAA Digital Avionics Systems Conference (DASC), Salt Lake City, UT, October 2010.

Mixed-Criticality Networks For Adaptive Systems

Wilfried Steiner
Chip IP Design
TTTech Computertechnik AG
wilfried.steiner@tttech.com

Günther Bauer
Chip IP Design
TTTech Computertechnik AG
guenther.bauer@tttech.com

***Abstract*—**
A crucial aspect of mission-critical vehicles is their reaction to environmental or system-internal changes. Ideally, the new situation is autonomously assessed and the vehicle's behavior is adapted accordingly.

Modern mission-critical vehicles realize a network-centric design approach, for one reason because it follows the inherent requirements of interconnecting the sensors and actuators of a control system and secondly, to tolerate a partial failure of the system. Therefore, in order to construct a truly adaptive system, the underlying network infrastructure must harmonically interplay with the top-level adaptive decisions. *TTEthernet* is such a COTS network infrastructure supporting network-centric system design that provides built-in integrity and redundancy management mechanisms as well as sharing the same physical network for applications of mixed criticality.

This paper discusses adaptable QoS and adaptable protocol-control flow in general and in particular using *TTEthernet* as case study.

I. Introduction

Modern vehicles like airplanes, military ground vehicles, or premium cars realize a multitude of software/hardware applications. These applications not only are often inherently distributed and have real-time requirements, but also different applications may be of different criticality levels. Still, these applications should be able to share a common network. Indeed, today's system architecture is highly influenced by the capabilities of the network. Therefore, the more intelligent the network is, the more architectural freedom it provides and, ultimately, the more services the overall system may realize.

On the other hand, today's vehicles have already reached a degree of complexity that tackles limits of system design, testability, and certification. Yet these systems operate reliably and dependably. A key factor for that is the precise definition of environment and purpose of these vehicles, which directly specifies the requirements according to which a system is then developed. The more environments and purposes a vehicle has to manage, the more functionality it has to provide, up to a point where we face the problem of reuse of onboard resources. In order to keep the complexity of the overall system manageable, in current systems this reuse problem is addressed by "operation modes", for example dispatch mode, flight mode, and landing mode in an airplane.

When we continue to extend the possible set of environments and purposes we reach another point from which on the expression in form of classic modes becomes infeasible. This may be because there are simply too many modes of operation for explicit enumeration. In one extreme case, the system may even face an infinite number of environments, when some parameters are not known. For these cases the classic static mode concept has to be extended by a dynamic mode concept in which the mode is constructed online rather than configured offline.

Dynamic modes require some basic building blocks from which they can be constructed. A network for mixed-criticality applications, like *TTEthernet* for example, provides such basic building blocks in form of different levels of quality of service (QoS) and a synchronized global time service. We assume the presence of an "Adaption Authority" that defines at least some aspects, if not all aspects, of the properties of the dynamic mode, and call the responding construction process of the dynamic mode the "Adaptive Action".

The temporal characteristic of the Adaptive Action is a key aspect of adaptivity. In critical situations we may face the Adaptive Action to complete within milliseconds or below. Adaption of a high-end car potentially during a regular maintenance service may be less demanding in the order of minutes or hours. The electronic re-design of a generation of vehicles may also be interpreted as Adaptive Action and may take months or years. Also, the frequency with which the adaptive changes occur has to be considered, which is often directly linked to their execution time. In this paper we focus on systems in which the time of the Adaptive Action has to be short and/or the frequency of Adaptive Action is high.

We recapture and introduce some general concepts in Section II. In Section III we discuss network-specific Adaptive Actions and discuss the impact of changes to the QoS of a dataflow. We discuss the impact of Adaptive Actions of the synchronized global time, for example, the consequences of reconfiguring the set of timing masters that generate the synchronized global time in Section IV. Throughout this paper we use *TTEthernet* as a reference protocol for mixed-criticality networks, although, the presented concepts are not *TTEthernet*-specific. Finally, we conclude in Section V.

II. General Concepts

In this section we discuss general networking concepts and in particular three different traffic classes that are used for different levels of criticality: time-triggered, rate-constrained, and best-effort traffic. We then discuss the concept of self-stabilization and introduce a lightweight formalism for adaptive systems based on this concept.

A. Mixed-Criticality Networks

Formally, the physical topology of a network is defined by an undirected graph $G(V, E)$, where end systems and switches are vertices V and the physical communication links connecting vertices are edges E. Figure 1 depicts an example of a network topology with eight vertices (six end systems and two switches).

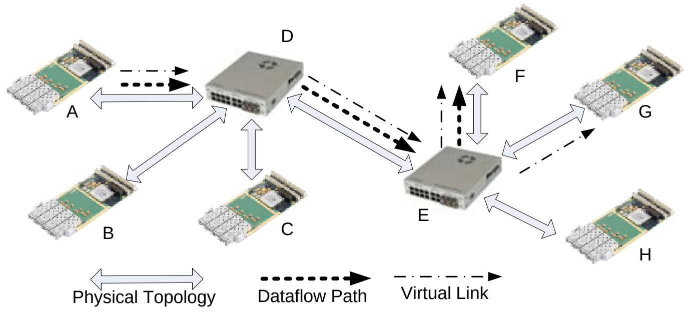

Fig. 1. Example of a *TTEthernet* network with six end systems and two switches connected in multi-hop

In this paper we focus on bi-directional physical communication links. Hence, each physical communication link connecting two vertices defines two directed "dataflow links". We denote the set of dataflow links by L, which is formally defined:

$$\forall v_1, v_2 \in V : (v_1, v_2) \in E \Rightarrow [v_1, v_2] \in L, [v_2, v_1] \in L \quad (1)$$

where "(x, y)" denotes an unordered tuple and "$[x, y]$' denotes an ordered tuple.

A sequence of dataflow links l_i is said to form a "dataflow path". The dataflow path in a network is represented by a directed path connecting one vertex (the sender) with exactly one other vertex (the receiver). An example of a dataflow path from A to F is depicted by the dotted line in Figure 1.

We denote the set of dataflow paths by DP and formally express a dataflow path p from a sender v_1 to a receiver v_r by the sequence of its dataflow links:

$$p = [[v_1, v_2], ..., [v_{(r-1)}, v_r]] \quad (2)$$

where the shortest dataflow path has length one (i.e. $p = [v_1, v_r]$).

A dataflow path defines, thus, a route from a sender to exactly one receiver. Information between the sender and receiver is communicated in the form of messages that we call "frames" according to the Ethernet convention. We denote the set of all frames by F with $f_i \in F$ and express the association of a frame f_i with a respective dataflow path p via its dataflow links: the instance of frame f_i on a dataflow link $[v_1, v_2]$ is denoted by $f_i^{[v_1,v_2]}$.

Frames may be delivered from a sender to multiple receivers. Therefore, we say the individual dataflow paths between the sender and each single receiver together form a "virtual link" (according the ARINC 664-p7 definition). We denote the set of virtual links by VL. Formally, a virtual link vl is expressed by the union of its dataflow paths: $vl = \bigcup p_i$.

Furthermore, we are interested in well-formed virtual links only: any two vertices $v_a, v_b \in V$ that are part of a dataflow path $p_i \in vl$ and are also part of another dataflow path $p_j \in vl$ are linked by the same sequence of dataflow links. Hence, a virtual link defines a directed tree structure having the sender as root and the receivers as leaves. For each frame $f_i \in F$, we formally denote the root by $first(f_i)$ (i.e., the first dataflow link in the virtual link of f_i) and, likewise, we denote the set of leaves by $last(f_i)$ (i.e., the set of last dataflow links in the virtual link of f_i).

Time-triggered frames are messages that are dispatched according to an offline compiled dispatch schedule, the tt-schedule. Furthermore, all devices that source time-triggered messages will proceed synchronously through their tt-schedules according to a global timebase. Hence, conflicts between time-triggered frames at the network can be avoided by having their transmissions scheduled at different points in time. A time-triggered frame f_i on a dataflow link $[v_k, v_l]$, $f_i^{[v_k,v_l]}$, is fully temporally specified by the following triple (as defined by the Time-Triggered Architecture [1]):

$$f_i^{[v_k,v_l]} = \{f_i.period, f_i^{[v_k,v_l]}.offset, f_i.length\} \quad (3)$$

The time-triggered frame period and frame length are given *a priori*; it is the task of a "tt-scheduler" tool to assign values to $F^L.offset$ for all frames F on all dataflow links L in the network. In case of a faulty component that violates this assigned triple, the switches in the network can be configured to execute a window enforcement scheme based on the synchronized global time. In this case they accept a time-triggered frame only if it is received within a predefined window around the expected receive point in time. The time-triggered traffic class is used in several protocols [2], [3], [4].

Rate-constrained frames are unsynchronized with bounded rate. A sender will respect a specified minimum duration in between any two frames with the same identifier f_i. This minimum duration specifies the maximum rate of frames with the same identifier. A rate-constrained frame f_i on a dataflow link $[v_k, v_l]$, $f_i^{[v_k,v_l]}$, is fully temporally specified by the following tuple:

$$f_i^{[v_k,v_l]} = \{f_i.rate, f_i.length\} \quad (4)$$

The rate-constrained frame rate and frame length are *a priori* given; it is the task of an "rc-checker" tool to calculate bounds on the maximum latency and jitter as well as on the worst-case memory use in the system. In case of a faulty component that violates this tuple, the switches in the system can be configured to execute a rate-enforcement algorithm, like a leaky-bucket or token-bucket algorithm. This algorithm drops rate-constrained frames in the switch when they are received too early. The rate-constrained traffic class is the communication paradigm used in ARINC 664-p7 [5].

Best-effort frames are unsynchronized without limits on their bound. The best-effort traffic is the standard Ethernet traffic and as there are no bounds given on the frame generation rate, no temporal enforcement action is possible. A mixed-criticality network is likely to assign best-effort traffic lowest priority, so that the impact of best-effort traffic on time-triggered and rate-constrained traffic is minimized.

B. Self-Stabilization Concepts for Adaptive Systems

Self-stabilization is a general concept that separates all system states into legitimate states (L) and illegitimate states (I), where L is defined by all states satisfying a given invariant (INV). Complementary, I is defined by all states not satisfying INV.

A system is self-stabilizing if two properties are satisfied: closure and convergence. Closure means that when a system resides in a legitimate state it will not enter an illegitimate state through the algorithm execution. Convergence means that the system, once placed in an illegitimate state will, through its algorithm's execution, eventually reach a legitimate state. Closure and convergence are depicted in Figure 2.

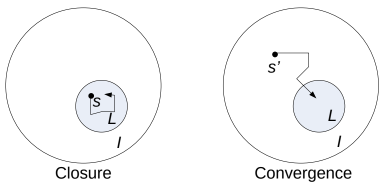

Fig. 2. The self-stabilization principle: closure and convergence

Convergence is often more complex to satisfy than closure as there are typically far more illegitimate states than legitimate ones.

A mixed-criticality system is a physical system and as such real-time has to be considered also to be part of its state. In particular, the pure progress in time may lead the system to leave the set of legitimate states and therefore violate the closure property. An example would be the non-synchronization of the local clocks in the distributed components, where in the set of legitimate states the clocks are assumed to be synchronized. Due to different oscillator drifts in the physical components, the closure property is potentially violated after a given interval of free-running clocks. In order to prevent the closure invalidation the clocks have to be continually re-synchronized.

For adaptive systems we can borrow the concepts and terminology of self-stabilizing systems: we interpret the properties of the dynamic mode as INV. A configuration of the basic building blocks that satisfies INV is then the implementation of the dynamic mode. The compilation of a new dynamic mode is a change of INV to INV' and the Adaptive Action (modification of the current configuration of the basic building blocks). The adaptive system determines the sets of possible INV, INV', and the transition from INV to INV' through its basic building blocks and Adaptive Actions.

A change from INV to INV' implies a change from L to L'. Figure 3 depicts different variants of change in INV. The variants are quite self-explaining. In short, "increasing" softens the invariant INV such that more states become legitimate. "Decreasing" is, of course, the opposite of increasing; here INV becomes strengthened thereby reducing L. The "overlapping" variant changes INV to INV' such that $L \cap L' \neq \{\}$ and "separating" causes $L \cap L' = \{\}$.

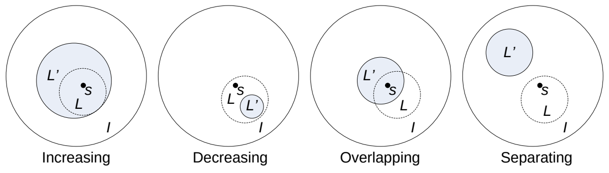

Fig. 3. Dynamic Legitimate States

As depicted in Figure 3 the change in INV may cause the current state of the system to become illegitimate. In this case we rely on a convergence property of the Adaptive Action to transition the system into a legitimate state.

We assume the presence of an Adaption Authority. The Adaption Authority changes some parameters in INV relevant to the new dynamic mode. Parameters not changed by the Adaption Authority have to be calculated by the Adaptive Action. Hence, INV' can be generated as a combined effort from the Adaption Authority and Adaptive Action. Furthermore it is then in the responsibility of the Adaptive Actions to actually re-configure the basic building blocks, such that INV' becomes satisfied (convergence property) and once INV' is reached it remains satisfied (closure property).

C. Adaptable Networks

In the case of a mixed-criticality network, as introduced above, the basic building blocks are threefold. Each message may be either time-triggered, rate-constrained, or best-effort traffic. The first set of basic building blocks of a mixed-criticality network is the frame classification and detailed temporal specification. We call Adaptive Actions on these sets of basic building blocks the "adaptive dataflow".

The mixed-criticality network assigns functions to physical components which execute the network's services. In *TTEthernet* the functions are Synchronization Master (SM), Synchronization Client (SC), and Compression Master (CM) used to establish and maintain the synchronized global timebase; SMs are assigned to end systems in the network, while CMs are assigned to the switches (SC may be both). The second set of basic building blocks of a mixed-criticality network is the assignment from function to physical component for the network's services. We call Adaptive Actions on this set of basic building blocks the "adaptive protocol-control flow".

The problem of finding a message path in the network is already well-studied, e.g., in the context of Steiner trees, and therefore outside the scope of this paper. Hence, we assume that all virtual links are a priori given and specified in terms of their individual dataflow paths. The Adaptive Actions on the message paths may re-use standard adaptive routing procedures.

III. Adaptive Dataflow

In this section we discuss adaptive dataflow. In particular we are interested in the Adaptive Action to change the traffic class of a frame or a set of frames and the resulting system-wide impact.

A. Dataflow Performance Parameters

In a network for mixed-criticality systems we are typically interested in the temporal quality of a frame's transmission as well as the maximum buffer sizes required in the network devices. The $latency(f_i)$ of a frame f_i is the time it takes from its transmission by a sender until its reception by a receiver. The $jitter(f_i)$ of a frame f_i is the difference between its maximum latency and the minimum latency.

Without giving particular formulas for $latency$ and $jitter$ for the different traffic classes, it is generally correct to state the following relations:

$$latency(TT) << latency(RC) \tag{5}$$

$$jitter(TT) << jitter(RC) \tag{6}$$

In a network as depicted in Figure 1, several dataflow links have to be shared between different end systems and switches (e.g., dataflow link from D to E). When unsynchronized frames, such as rate-constrained or best-effort traffic, have to traverse a shared dataflow link they may be received in a switch while the shared dataflow link is busy. Hence, unsynchronized frames may have to be queued in the switches and potentially also in the end systems. As a result, the buffer sizes for rate-constrained traffic typically have to be much larger than for time-triggered traffic:

$$buffer(TT) << buffer(RC) \tag{7}$$

In general, the temporal quality of time-triggered communication outperforms rate-constrained traffic. The same is true for time-triggered and best-effort traffic. We have to take these relations into account when the Adaptive Action changes the traffic class of a frame or a set of frames.

B. Dataflow Invariant

An invariant (INV) describing the dataflow of frames in the network has to define upper bounds on $latency$ and $jitter$ for each frame f_i in the network. These bounds may be individual for each frame f_i. Furthermore, INV has to define an upper bound on memory consumption in the end systems and switches. Formally, the dataflow invariant is defined as ($\forall f_i \in F, \forall v_i \in V$):

$$\begin{aligned} INV \quad = \quad & \bigwedge latency(f_i) \leq bound(latency(f_i)) \qquad (8) \\ & \bigwedge jitter(f_i) \leq bound(jitter(f_i)) \\ & \bigwedge buffer(v_i) \leq bound(buffer(v_i)) \\ & \bigwedge traffic\_class((f_i)) = \{BE \vee RC \vee TT\} \end{aligned}$$

A change in the invariant from INV to INV' is therefore reflected in a change of the bounds on latency, jitter, and buffer size. The Adaption Authority causes an Adaptive Action by a change to these bounds, e.g., an application requires a lower latency for a given frame. This means that the Adaptive Action has to find a configuration of the basic building blocks to satisfy these bounds. For adaptive dataflow the basic building blocks are the traffic classes to which frames are assigned, plus the presence or absence of a synchronized timebase.

The adaptive action on the dataflow as change in a frame's traffic class has three aspects. The first aspect requires a calculation process to determine the new maximum values on the dataflow performance parameters and compare them with the desired bounds in INV'. The second aspect is the actual reconfiguration of the traffic classes within a device. The third aspect is the propagation of the new adaptive information throughout the network in a reliable and potentially fault-tolerant way. We discuss these aspects in the following.

C. Adaptive Performance Calculation

The Adaptive Action in response to the a new INV' depends on the direction of traffic-class change. The change from time-triggered to rate-constrained traffic is a checking problem, while rate-constrained to time-triggered is a scheduling problem.

1) Time-Triggered to Rate-Constrained Adaption: We know from Equations 5, 6, and 7 that a change in traffic class from time-triggered to rate-constrained traffic is reflected by an increase in the bounds on $latency$, $jitter$, and $buffer$.

The network is then in charge of calculating the new actual maximum values for the dataflow performance parameters. If the maximum values exceed the bounds, the network may report the failure in the desired adaption to the Adaption Authority which in turn may decide to relax INV' even further (e.g., by reducing the overall number of frames).

The maximum values of the dataflow performance parameters may be established as listed below with increasing computational overhead.

a) Classic Modes: In the classic modes concept only a limited set of traffic class changes is possible, and for each of these modes INV and Adaptive Action are pairwise defined offline and statically stored. The Adaptive Action is therefore reduced to the adaptive reconfiguration and adaptive information distribution.

b) Simple Checking: When the maximum values for a particular configuration of the basic building blocks are not offline calculated they have to be established online. This calculation typically results in a conservative approximation of the actual maximum and often allows a trade-off on accuracy of the approximation and computational overhead. By simple checking we mean a calculation procedure that keeps the computational overhead small such that the procedure may even be realized in an embedded platform. Simple checking procedures may also impose restrictions on the frame characteristics such as on the relative frame rates of rate-constrained traffic. On the other hand they may even exclude traffic classes such as the best-effort traffic class.

An example of a simple checking procedure is the dynamic bandwidth reservation protocol of the Audio/Video Bridging working group (amendment 12 to IEEE 802.1Q, P802.1Qav).

Another simple checking procedure is based on a simplified calculation of the maximum sequence of frames per dataflow link. Let "maximum burst" $burst^{[v_x,v_y]}$ be this maximum chain of frames that may be sent sequentially on a given dataflow link $[v_x, v_y]$.

When the frame rates of all frames (time-triggered and rate-constrained) are about equal, all frames are of same priority and best-effort traffic is not present then $burst^{[v_x,v_y]}$ can be calculated as:

$$burst^{[v_x,v_k]} = \sum_i (f_i^{[v_x,v_k]}.length) \tag{9}$$

where $f_i^{[v_x,v_k]}$ denotes the frame instances transported on dataflow link $[v_x, v_k]$.

$burst^{[v_x,v_k]}$ is a very conservative upper bound on $buffer$ required for the dataflow link $[v_x, v_k]$ in v_k. Latency and jitter of a frame f_i can be calculated from the addition of all $burst^{[v_x,v_k]}$ in the dataflow path of f_i.

Note, that in this calculation f_i refers to all frames in the system (time-triggered and rate-constrained). The approximation of the maximum performance values can be significantly improved by taking the regularity of time-triggered traffic into account.

c) Sophisticated Checking: There are several benefits resulting from an increasing precision in the approximations of the maximum values. On the one hand, the applications using the frames may provide better service. On the other hand more frames may be possible in the network, while still providing sufficient dataflow performance for all applications.

More sophisticated checking procedures than the simple ones outlined above may also take into account the distributed application state and communication requirements in order to maximize the bandwidth efficiency. However, sophisticated checking procedures require computational resources that may exceed embedded platforms. Hence, dedicated components like a "Resource Management Authority" [6] may be necessary to execute more sophisticated checking procedures.

2) Rate-Constrained to Time-Triggered Adaption: A change of a frame (or set of frames) from rate-constrained to the time-triggered traffic class is reflected by decreasing bounds in the dataflow invariant. Similar to the change in traffic class as discussed above, we can distinguish different levels of realization of this change with increasing computational complexity.

a) Classic Modes: In the classic mode-based approach, the number of changes from rate-constrained to time-triggered traffic is restricted. Hence, it is feasible to generate the time-triggered schedules offline for all possible modes. The Adaptive Action is, again, reduced to the adaptive reconfiguration and the adaptive information distribution problem.

b) Simple Scheduling: There are several schemes for simple scheduling. In contrast to the classic mode, the configuration of the sender of a time-triggered frame to its transmission time in the time-triggered schedule is not pre-configured, but has to be assigned dynamically.

One way to realize a simple scheduling strategy is the offline configuration of a time-triggered schedule skeleton. This skeleton is then populated online with time-triggered frames. Among others, three factors determine the layout of such a skeleton and the slot assignment procedure: the lengths and periods of frames and the relative number of frames with respect to their lengths and periods.

The frame lengths drive the communication slot lengths. In some systems it will be sufficient to use a single slot length for all frames. In other systems there may be different slot sizes. For example, when video data as well as control data is communicated as time-triggered traffic, at least two lengths of slots (long for video, short for control data) increases bandwidth efficiency in contrast to an equally sized scheme.

The assignment procedure of time-triggered frames to slots in the skeleton can take the relative frame periods into account. Different segments on the timeline can be reserved for different frame periods. The sizes of these segments are determined by the knowledge on the relative number of frames with respect to their periods. The actual assignment process may be as simple as assigning the first empty slot within a segment to the new time-triggered frame.

c) Sophisticated Scheduling: Sophisticated scheduling strategies can reduce the level of offline configuration in the skeleton schedule, if a skeleton is present at all. On the other hand sophisticated scheduling strategies may include more information in the assignment process of time-triggered frames to communication slots. For example, an application may be synchronized to the network and the communication slot must be in temporal proximity to the point in time when the respective information, e.g., sensor data, to be carried in the time-triggered frame is generated.

As sophisticated scheduling strategies likely require significant computational resources, the network can use some dedicated component for sophisticated scheduling procedures.

The simple and sophisticated scheduling schemes as discussed above may be combined with the traditional scheme. In this case a subset of critical time-triggered frames is assigned offline to communication slots, which will not change through the Adaptive Action.

D. Adaptive Reconfiguration

Once the checking procedure for new rate-constrained traffic and the scheduling procedure for new time-triggered traffic have concluded, the actual change in the traffic class has to be executed. The pure classification of a particular frame f_i is easily changed by an update in the configuration of the respective device. However, the dispatch, relay, and receive logic is typically specific to a traffic class.

The straight-forward solution is the implementation of independent functionality for time-triggered and rate-constrained traffic communication. However, in order to reduce the overall size of the device these functionalities may be integrated. Indeed, first prototypes in VHDL show, that the integrated implementation only slightly increases the size compared to one individual communication function.

E. Adaptive Information Distribution

The change in the traffic class of a frame inherently effects several devices in the network, e.g., because a switch will execute different acceptance checks on frames of different traffic classes. Furthermore, the Adaption Authority as well as the adaptive performance calculation will likely be distributed throughout independent devices in the network. The consistent information distribution throughout the network is therefore of upmost importance.

In standard Ethernet networks the simple network management protocol (SNMP) is widely accepted. However, in case of a network for mixed-criticality systems we also have to consider faulty devices. *TTEthernet*, for example, realizes a fault-tolerant reconfiguration protocol. Here, a switch may only change its configuration when a configurable set of end systems send "unlock" frames to the switch within a small temporal duration. End systems can be re-configured through the host interface. If necessary, the host or a middleware layer can implement an interactive consistency protocol before the dataflow is adapted.

Alternatively, the change in traffic class can be masked from the network by assigning two identifiers to a frame, a time-triggered and a rate-constrained identifier. Depending on the current traffic class the application uses the one or the other and switches in case of a dataflow adaption. This simplicity is, again, a trade-off to the overall configuration memory required.

F. Use Cases

One use case of adaptive dataflow is an emergency mode for rare situations in which synchronization is lost. In this case critical time-triggered frames can be adapted to rate-constrained frames.

Another use case for the dataflow adaption from rate-constrained to time-triggered traffic allows the realization of "Synchronized Traffic On-Demand". Here, all frames are assigned to the rate-constrained and best-effort traffic classes. Only, when the network latencies become too high or too many frames are lost, some frames adopt their traffic class from rate-constrained to time-triggered traffic. This is of particular interest for power-save modes during which the communication links shall not be kept synchronized (e.g., Gbit Ethernet).

G. New Frame Addition and Existing Frame Removal

The concepts developed in this section describe the adaption of the traffic classes from existing frames. It is also likely that an Adaption Authority needs to define new frames or remove existing frames throughout the mission time. The concepts for traffic-class adaption can be re-used for this purpose using the classic, simple, or sophisticated approaches for new frame integration.

IV. Adaptive Protocol-Control Flow

A. Protocol-Control Flow Performance Parameters

The precision in a system is one quality parameter for synchronization, which defines the maximum difference of any two correct local clocks representing the local view of the global synchronized time. As defined in [7], we denote the internal representation of time in a device by $\mathcal{C}$ and the real-time by $\mathcal{R}$. The clock of device v_i is then represented by the function:

$$C(v_i) : \mathcal{R} \rightarrow \mathcal{C} \tag{10}$$

meaning that at each point in real-time t there exists a corresponding assignment of a device v_i's counters that represent the node's local view of time $C(v_i, t)$. The precision Π in the network of synchronized devices is the maximum difference between the local clocks of any two non-faulty devices:

$$|C(v_i, t) - C(v_j, t)| < \Pi \tag{11}$$

A second quality parameter is the number of faulty devices and their respective failure modes the system can tolerate while preserving the precision. When the failure hypothesis is exceeded it may happen that the precision in the system exceeds its defined maximum bounds. The maximum recovery time of the system in such scenarios is another quality parameter of synchronization.

B. Protocol-Control Flow Invariant

We define the invariant for adaptive protocol-control flow based on the definition of Π and the synchronization roles assigned to the devices:

$$\begin{aligned} \mathit{INV} \;=\; & (|C(v_i, t) - C(v_j, t)| < \Pi \vee \\ & \mathit{role}(v_i) = \mathit{none} \vee \mathit{role}(v_j) = \mathit{none}) \\ & \bigwedge \mathit{role}(v_{SM_1} \dots v_{SM_k}) = SM \\ & \bigwedge \mathit{role}(v_{CM_1} \dots v_{CM_l}) = CM \end{aligned} \tag{12}$$

where $\mathit{role}(v_i)$ is a function that returns the current synchronization role to which v_i is configured. As introduced in Section II-C, *TTEthernet* defines three roles in the synchronization algorithms: Synchronization Master (SM), Synchronization Client (CM), and Compression Master (CM). In case that a device v_i has no synchronization role assigned, $\mathit{role}(v_i)$ returns none. Note, that we do not include the set of SCs in INV.

A change in the invariant from INV to INV' is reflected by a change in the basic building blocks of the protocol. In *TTEthernet* these basic building blocks are the assignment of the roles of SM, CM, and SC to the physical devices. Here, we interpret the power-on of a device also as change of role from none to SM, CM, or SC and vice versa. The transition from INV to INV' may impose any one of the four modes depicted in Figure 3. INV' is an increasing change when the sets of SMs or CMs are reduced and decreased if the sets are enlarged.

On the other hand, INV' imposes an overlapping change when some new SMs or CMs are powered-up while some running ones are shut-down. Finally, INV' is a separating change if the complete set of running SMs and CMs is shut-down, while a new set of devices is powered-up.

TTEthernet also specifies synchronization priorities which we do not cover in the definition of INV. Different SMs and CMs may source *Protocol Control Frame*s with different priorities. A *TTEthernet* device can decide whether it automatically deflects to the *Protocol Control Frame*s with highest priority in the system or may deflect only upon host approval. The synchronization priorities can therefore also be part of the basic building blocks for adaptive protocol-control flow.

Similar to the adaptive dataflow, the changes in these assignments are triggered by an Adaption Authority. Furthermore, we can distinguish three aspects of the Adaptive Action for protocol-control flow: adaptive performance calculation, adaptive reconfiguration, and the adaptive information distribution.

C. Adaptive Performance Calculation

Adaptive actions on the *TTEthernet* synchronization strategy are changes in the assignment of algorithmic roles to the physical devices. The adaptive performance calculation for adaptive protocol-control flow is the calculation of an assignment or roles to the physical devices such that the precision defined in the invariant is satisfied. Again, we can distinguish three levels of complexity and associated computational overhead in the determination of such an assignment.

1) Classic Modes: In the classic modes approach we define independent assignments of the roles to the devices. The network is then capable of switching between these modes as triggered by the Adaption Authority.

In order to keep the network synchronized, a sufficient number of SMs and CMs have to be present. Furthermore, the connectivity of the SMs to the CMs directly relates to the number of faulty devices that can be tolerated. Classic modes switch between minimum synchronization groups that are well-connected.

2) Simple Synchronization Group Formation: Dynamic modes construct the synchronization group rather than selecting one group from a set of possible groups. A simple procedure of such a synchronization group formation is the selection of SMs and CMs from a list of available devices. For this simple procedure, the network has to maintain the lists of operational devices and their connectivity. When the Adaption Authority demands a change in INV, the simple formation procedure accesses these lists and aims to compensate for the imposed changes. The selection of new SMs and CMs may be as simple as picking the first available member from the respective list.

3) Sophisticated Synchronization Group Formation: More sophisticated formation procedures may try to optimize one or several of the synchronization performance parameters (precision, fault-tolerance, recovery time). For this purpose the Adaptive Action may even actively transfer roles between the physical devices. If the formation procedure aims to optimize the precision in the system, it may implement dedicated diagnostic procedures that measure the current precision as perceived in the devices in the network. Then the formation procedure may change the set of SMs and check the impact on the precision. If the precision improves, the INV' may become the new INV. If the precision becomes worse, the Adaptive Action may backtrack to the original INV.

D. Adaptive Reconfiguration

The change of synchronization role can either be done by changing the configuration of a physical device or by its shutdown. If one physical device shall be capable of providing more than one synchronization role, all roles may be present in the device or they may be loaded upon demand. Again, this is a trade-off between device size and download complexity.

E. Adaptive Information Distribution

The devices in the network can be explicitly informed of a change in the protocol-control flow. In this case similar measures as discussed for the adaptive dataflow can be realized. On the other hand, the protocol-control flow often implements periodic message exchange. In *TTEthernet*, for example, Protocol Control Frames (*Protocol Control Frame*s) are communicated. A change in the assignment of synchronization role to physical devices can be propagated implicitly by referring to the `pcf_membership_new` field inside the *Protocol Control Frame*. This field represents a membership vector with a one-to-one relation of bit to SM. A change in the set of SMs is reflected, therefore, by a change of the bits set in the membership vector.

F. Use Cases

One apparent use case for adaptive protocol-control flow are power-saving modes as for example in deep-space missions. Depending on the current purpose and environment of the vehicle, different nodes providing different functionality can be activated/deactivated. Hence, in order to provide its service, the network has to coordinate which physical devices execute which service role.

Another use case is optimal robustness in a network's service. When the network monitors the available devices and their connectivity it may still be able to provide full service in the failure case or different levels of degraded service.

V. Conclusions

Today's onboard networks of vehicles for high-criticality missions are mostly statically configured. In order to re-use the on-board resources in an efficient way, several configurations may be present in a vehicle. These configurations are called the modes and the network may switch between these modes with respect to the current mission situation.

With the growing complexity and multitude of on-board functionality, this traditional modes approach may become ineffective, calling for a finer granularity of change. In this paper we have presented such an approach, that we call the dynamic modes for the communication infrastructure.

Dynamic modes assume the presence of a higher layer Adaption Authority that defines at least some aspects of the intended dynamic mode. The Adaptive Actions are functions implemented within the network that complete the properties of the dynamic mode and modify the network in such a way that it satisfies the properties. In this paper we discussed Adaptive Action for the dataflow and the protocol-control flow in the network using *TTEthernet* as a case study.

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 G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112 – 126, Jan. 2003.

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

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

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

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

[6] B. Huber, C. El Salloum, and R. Obermaisser, "A resource management framework for mixed-criticality embedded systems," in *Industrial Electronics, 2008. IECON 2008. 34th Annual Conference of IEEE*, 10-13 2008, pp. 2425 –2431.

[7] J. Rushby, "Systematic formal verification for fault-tolerant time-triggered algorithms," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 651–660, sep 1999. [Online]. Available: http://www.csl.sri.com/papers/tse99/