Loose Synchronization of Event-Triggered Networks for Distribution of Synchronous Programs

Jan Romberg
romberg@in.tum.de

Andreas Bauer
baueran@in.tum.de

Technische Universität München, Institut für Informatik
Boltzmannstrasse 3, D-85748 Garching, Germany

ABSTRACT

Dataflow synchronous languages have attracted considerable interest in domains such as real-time control and hardware design. The potential benefits are promising: Discrete-time semantics and deterministic concurrency reduce the state-space of parallel designs, and the engineer’s intuition of uniformly progressing physical time is clearly reflected. However, for deriving implementations, use of synchronous programs is currently limited to hardware synthesis, generation of non-distributed software, or deployment on time-triggered architectures.

In this paper, it is shown how synchronous dataflow designs can be used for synthesizing distributed applications based on target architectures that do not provide a global time base by default. We propose a distribution method called “synchronization cascade” where the nodes’ local clocks depend on each other in a tree-like manner. For evaluation of the method, we characterize some requirements for firm real-time applications, and evaluate our approach with respect to the postulated requirements.

Categories and Subject Descriptors
D.2.2 [Software Engineering]: Design Tools and Techniques

General Terms
Design, Languages

Keywords
Embedded Software, Synchronous Languages, AutoFocus, Code Distribution, Scheduling

1. INTRODUCTION

Dataflow synchronous programs and specifications, exemplified by LUSTRE and SIGNAL[2], or AUTOFOCUS[3], provide a discrete-time abstraction from real-time, concurrent implementations. This abstraction is familiar to control engineers and hardware designers: all parts of the program run in a uniform time-frame (time-synchronous). Concurrency in synchronous designs is based on lock-step composition, which usually results in more understandable designs, and avoids a blow-up of the state space of parallel programs.

While synchronous programs are well suited for synthesis of monolithic hardware and software, the problem of distributed implementation of a synchronous program is still a challenging one. Clearly, formal abstractions of distributed systems, along with an automated procedure to synthesize distributed implementations, are desirable for several reasons, including improved behavioral validation, for instance by simulation or formal verification.

However, there are numerous challenges to an (preferably automated) approach for distribution of synchronous programs, e.g. (1) integration of existing systems with the synthesized executives, (2) required compatibility with the numerous platforms in the domain, (3) compliance with tight nonfunctional requirements such as timeliness, memory consumption, and hardware cost.

Looking at real-time control application in the automotive sector, time-triggered protocols [8] are an attractive target platform for highly critical applications such as X-By-Wire. However, for applications where cost concerns and legacy integration issues are more dominant compared to criticality requirements, existing event-triggered bus architectures such as Controller Area Network (CAN) may play an important role for some time to come. Applications characterized by this requirement will be called medium-criticality applications in the sequel.

Synchronous approach vs. firm real time

In the context of medium-criticality applications, let us discuss the synchronous distribution issue in more depth: Certainly, semantically correct implementation of the distributed program is vital. But when looking at the state of the art in distributed real-time control applications, many of these applications meet their timing and criticality requirements even though they are based on communication media that provide no absolute guarantees about response times. As a possible explanation, some control applications are known to tolerate the loss of a bounded number of messages, e.g. state values. In real-time systems, this corresponds to the notion of firm real-time: transactions are discarded when they miss their deadlines, as there is no value to completing them afterwards. In contrast to hard real-time systems, a bounded number of deadline misses is not considered fatal. How can the notion of firm real-time be married with the distribution of synchronous programs?

It can also be questioned whether synchronous implementations require the existence of a precise global timebase. Existing works on asynchronous distribution of synchronous programs [9] have shown that this is not necessarily the case. On the other hand, it is quite clear that asynchronous distribution does not satisfy some requirements of real-time control applications when communica-
tion media are involved that allow message losses or unbounded latencies. Can we use a loose base, which may be cheaper to implement, and still obtain implementations suited for real-time control applications?

Based on this discussion, our approach is twofold: (1) Provide a distribution method for synchronous programs based on a synchronization/communication layer with a loose base. The method should ensure semantically correct execution of the synchronous program under normal operation conditions. (2) Make sure the synchronization/communication layer provides a reduced service, including synchronization, in case of certain faults. By adjusting some well-defined parameters, it is then up to the developer to ensure that the system remains in normal operation for the most part of its lifecycle, and meets the correctness and timeliness requirements imposed by the application.

In the following, we will describe the procedure to deploy synchronous programs onto event-triggered networks based on loose synchronization, and evaluate our method with respect to the requirements of medium-criticality applications. Section 2 introduces a simple denotational style [3] used for specifying synchronous programs and abstractions of communication channels. Section 3 introduces a synchronization/communication layer called synchronization cascade, which is used for distributed deployment of synchronous programs. Section 4 defines some requirements for distribution of synchronous programs for medium-criticality applications, and shows that our distribution method satisfies some essential properties related to these requirements. Section 5 finally, relates our approach to works of other authors, and gives an outlook discussing future directions.

2. DATAFLOW SYNCHRONOUS SPECIFICATIONS

We model distributed software as a network of components communicating over timed streams.

Streams

A stream is a finite or infinite sequence of messages from a set $M$. For such a set of messages, we use $M^c$ to denote the set of all finite and infinite streams over $M$. For a stream $\sigma \in M^c$, the $i$-th message is written as $\sigma_i$. We define a special message $\perp$, the absent message: for a given set $M$, we write $M_\perp = M \cup \{ \perp \}$ for the set obtained by adding the absent message.

The length operator $\#$ yields the length of the stream to which it is applied. Concatenation of streams, written $\sigma_1 \cdot \sigma_2$, yields a stream that starts with the messages of $\sigma_1$ followed by the messages of $\sigma_2$. The filtering operator $\circ$ is used to filter away messages. $M^c \circ i$ is the stream of $\sigma$ obtained by removing all messages in $\sigma$ that are not in the set $M' \subset M$.

In the following, we will use a time-synchronous interpretation of streams: for all streams, the position of a message in a stream is associated with a unique instant in a uniform discrete timeframe.

Components

A component $c$ has a set of input signals $I = \{ i_1, i_2, \ldots, i_m \}$ with types $M_{i_1}$ and a set of output signals $O = \{ o_1, o_2, \ldots, o_n \}$ with types $M_{o_1}$, $I = M_{i_1} \times M_{i_2} \times \cdots \times M_{i_m}$ and $O = M_{o_1} \times M_{o_2} \times \cdots \times M_{o_n}$ are the input and output domains of the component, respectively. We consider only deterministic components that are complete on their inputs: for these, inputs and outputs can be related by a total function $f : I \rightarrow O$. We call this function $f$ the I/O function of the component. For time-synchronous streams, we require I/O functions to be causal; that is, present outputs do not depend on future inputs, and length-preserving, i.e. there is a fixed correspondence between the lengths of inputs and outputs.

Networks of Components

For composition of two components $c_1$ (I/O function $f_1$, inputs $I_1$, outputs $O_1$) and $c_2$ (I/O function $f_2$, inputs $I_2$, outputs $O_2$) yields a component with inputs $(I_1 \setminus O_2) \cup (I_2 \setminus O_1)$, and outputs $O_1 \cup O_2$. Semantically, composition of $c_1$ and $c_2$ is then defined by intersection of $c_1$’s and $c_2$’s behavior on the shared inputs/outputs (synchronous composition); we restrict the scope of our semantics to the class of systems where the intersection is again a (total) I/O function. Such networks of components can be specified in a graphical style, using rectangles for components, and directed arrows for signals/channels [3].

Abstractions for Communication Channels

In the following, we will use components for abstractly defining properties of a given communication channel. This definition will be used in Sections 5 and 6 to formalize assumptions about the communication medium and guarantees provided by the synchronization/communication layer.

**Definition 1** (Channel). A component $c$ is a channel iff $|I| = |O| = 1$ and $I = O$.

**Definition 2** ($m$-LENGTH-PRESERVING CHANNEL). Given some $m \geq 0$, a channel is an $m$-length-preserving channel iff $\forall \sigma \in I : \#(f(\sigma)) = \#(\sigma) + m$.

**Definition 3** (UNIT DELAY CHANNEL). A channel is a unit delay channel with initial message $m$ iff $\forall \sigma \in I : f(\sigma) = m \circ \sigma_1$.

As a corollary, unit delay channels are 1-length-preserving.

**Definition 4** ($n$-BOUNDED LOSSY CHANNEL). Given some $n > 0$ and some $m$-length-preserving channel $ch$ with domain $I = O = M_{i_1}$, $ch$ is an $n$-bounded lossy channel iff, for all input/output streams $(\sigma_1, \sigma_2) \in f$, for all of $\sigma_1$’s substreams $\sigma_1^i$ of length $n$ at position $i$, for all of $\sigma_2$’s substreams $\sigma_2^{i+m}$ of length $m$ at position $i + n$, the following condition holds:

$$\#(M \circ \sigma_1^i) = n \Rightarrow \#(M \circ \sigma_2^{i+m}) \geq 1$$

Intuitively, if fed with messages from the set $M$, an $n$-bounded lossy channel will lose at most $n - 1$ subsequent messages. As a direct consequence of the above definition, any $n$-bounded lossy channel is also $n + 1$-bounded lossy.

3. SYNCHRONIZATION CASCADES

3.1 Terminology

A synchronization cascade provides a layer for synchronization and communication, and implements a logical network topology on top of some suitable physical topology where each link in the logical topology can be mapped to a physical counterpart. We call the underlying protocol(s) the base protocol(s) of the cascade.

A synchronization cascade is a rooted tree with nodes $N = \{ N_0, N_1, \ldots \}$, and edges $S \subseteq N \times N$. Each node corresponds to a processor or control unit in the distributed implementation. Edges

$^3$Our unit delay definition corresponds with a combined use of the pre and $\rightarrow$ operators in LUSTRE.
$s \in S$ are called (direct) synchronizing links: Each such link communicates a periodic message that is used by its child node to synchronize itself with the parent node.

The root of the tree is called master node $N_0$. For a non-master node $N$, we denote as $L_i(N)$ the set of those synchronizing links that form a path from $N_0$ to node $N$. If $(N, N_0) \notin S$, the links in $L_i(N)$ form an indirect link from $N$ to $N_0$. $Par(N)$ is the set of parent nodes along the path such that $N_0 \in Par(N)$ and $N \notin Par(N)$.

The rooted tree is extended to a (directed) multigraph by adding edges $U = \{u_1, u_2, \ldots\}$ (depicted as dashed edges). The edges $u \in U$ are called nonsynchronizing links: while their value is usually important to the receiver, the timing of their reception does not influence the receiver’s activation times.

An example for a synchronization cascade is shown in Fig. 1. Node $N_0$ is the master node. Links $s_1, s_2, s_3$ are synchronizing links, while $u_1$ and $u_2$ are nonsynchronizing. The master node emits a periodic synchronizing message with a predefined base period $T$.

**Processing phases**

Fig. 2 shows, schematically, the timing of computations performed by a single node. During each cycle, the node performs two subsequent computations: a send/receive phase, and a computation phase. For given $N_i \in N, j \in N_0$, instant $t_{i,j}$ denotes the activation instant of node $N_i$ at step $j$.

**Send/receive phase**

The send/receive phase is triggered by a periodically elapsing timer for the master, and by the respective synchronizing message for a non-master node. During this phase, the nonsynchronizing messages received since the last send/receive phase and the incoming synchronizing message are read, and all outgoing messages computed in the last computation phase are emitted. Because the send/receive phase requires nonzero time for execution, and the receiver node could potentially lose synchronizing messages if their interarrival time is too short, we define a quiet interval that overlaps the send/receive phase, and during which the node is not required to process incoming synchronizing messages. The remaining part of the cycle is called the receptive interval. For nonsynchronizing messages arriving in the quiet interval, the node may either read the message immediately, or leave it in the message buffer so it can be processed by the next send/receive phase. The quiet interval $[0, Q]$ starts at the beginning of each period. The analysis below will ensure that a node does not receive synchronizing messages in $[0, Q]$ under given operating conditions. $Q$ is typically a worst-case estimate of the send/receive period’s combined task response and execution times. In the following, we formally require $0 \leq Q < T \cdot (1 - 2 \varepsilon)$, where $\varepsilon$ is a clock drift constant introduced in Section 4.6. Because communication overhead is usually small compared to computation time, we expect typical assignments for $Q$ to be less than $T/4$.
periodically activated by the synchronizing message. In state MESSAGE_ABSENT, the node has detected a (possibly transient) absence of the synchronizing message. The send/receive phase is activated by the node’s own periodic timer in this state. We will show in Section 3.2.3 that, while in state MESSAGE_ABSENT, the node is able to re-synchronize itself with its parent node. In state SELF_TRIGGERED, the node is periodically activated by its own timer, and there are no guarantees about the node’s ability to re-synchronize itself with its parent node, if existent. The parameter \( T_{\text{max}} \) is called the message absence detection margin. It denotes the time interval after which, if no synchronizing message has been detected, a node in state EXTERNALLY_TRIGGERED changes to MESSAGE_ABSENT. Parameter \( n_{\text{pf}} \) is the parent fault detection count. It denotes the maximum number of periods the node will remain in state MESSAGE_ABSENT if no synchronizing message is detected. If this number exceeds \( n_{\text{pf}} \), the node will change to state SELF_TRIGGERED. Sender fault detection therefore initiates a fallback behavior in case either the parent node or the communication medium fails for a longer period of time. Note that re-synchronization with the parent after the node has entered state SELF_TRIGGERED is not in the scope of this paper.

3.2 Environment assumptions

We will now state some assumptions about the physical environment of the cascade. The assumptions will be necessary in order to show that the cascade meets its operational requirements. Some of the assumptions will be required independent of the network state, while others are prerequisites for normal operation of the network, and may be violated under fault conditions.

Physical clocks

Each node \( N_i \) has its own physical clock \( t_i \). A physical clock is typically subject to drifts and jitter w.r.t. the ideal physical time \( t \). Operation of a synchronization cascade requires that deviations of all the nodes’ clocks from ideal time are bounded by a constant.

DEFINITION 5 (\( \varepsilon \)-Bounded Clock Drift).

For a given cascade, let each node \( N_i \) be associated with a physical clock \( t_i \). The cascade is said to have an \( \varepsilon \)-bounded clock drift iff, for all intervals where \( t_i \) is not reset,

\[
\forall N_i \in \mathcal{N}, \left| \frac{dt_i}{dt} - 1 \right| \leq \varepsilon
\]

In combination with our definition of timer_task, the bounded clock drift assumption guarantees that the physical base period of each node is bounded by \( [T/(1+\varepsilon), T/(1-\varepsilon)] \), and the message absence detection period is bounded by \( [T_{\text{max}}/(1+\varepsilon), T_{\text{max}}/(1-\varepsilon)] \).

Message jitter

We define for each link \( s_j, u_j \) in the cascade a map \( \Delta_i \), mapping direct links, i.e. links between adjacent nodes, to their corresponding worst case message jitters, assuming that some adequate method for analysis is available. The minimum and maximum message latencies for direct or indirect links \( (N_i, N_j) \) will be denoted as \( d_{\text{min}}(i, i'), d_{\text{max}}(i, i') \), respectively, such that

\[
d_{\text{max}}(i, i') - d_{\text{min}}(i, i') = \sum_{s_j \in L_i(N)} \Delta_i(s_j)
\]

holds for all \( (N_i, N_j) \).

The message jitter summarizes the end-to-end jitter from the instant the send/receive phase at the parent is activated until the child node’s activation time. The worst-case jitter will typically include (1) execution time jitter of the sender’s send/receive code, (2) queuing jitter at the sender, (3) communication jitter of the medium, (4) response time jitter of the receiver’s task.

Because the message jitter includes the queuing jitter at the sender, the bound may be invalid if the communication medium is not accepting messages (e.g. due to unforeseen overload conditions or external disturbances). We therefore assume the existence of a simple communication layer that enforces the predetermined queuing interval by retracting the message when the precomputed worst-case queuing time is overrun. Note that this typically requires the layer to have some access to lower-layer operations of the controller.

For correct operation of the cascade, the end-to-end jitter from the master to any node must be bounded:

DEFINITION 6 (Bounded Sync. Message Jitter).

Let \( L_i(N) \) denote the set of all synchronizing links that form a path from the master to the node \( N \). The network is said to have a bounded synchronizing message jitter iff

\[
\forall N \in \mathcal{N}, \sum_{s_j \in L_i(N)} \Delta_i(s_j) < \min \left( \frac{T - Q}{2}, \frac{T(1 - 5\varepsilon)}{2} \right)
\]

This bound should be satisfactory for a large number of practical applications. For instance, in an automotive case study analyzed in [3], the bound is found to hold for most test cases of the controller ICs (Intel 82527 and Philips 82C200) allowing to retract messages when the precomputed worst-case queuing buffer has been used up.
for the case of a 1MBit/s CAN bus, most high-priority messages have a jitter of around $10^{-5}$ s, so for $\varepsilon = 10^{-6}$, $T = 10^{-5}$ s, and $Q = T/20$, cascades up to depth 4 (four synchronizing links between master and the “farthest” node) are possible.

### 3.4 Analysis of Operational Modes

This section will provide an analysis of the different operational modes of the cascade: operation under normal conditions, operation under transient fault conditions, and operation under permanent fault conditions. For normal operation, we will show that all non-masters remain in state EXTERNALLY_TRIGGERED, while under transient fault conditions, it will be shown that non-masters never enter state SELF_TRIGGERED.

The following two definitions deal with the property of synchronization – the offset of the node’s activation instant w.r.t. the master’s activation instant is bounded and receptiveness – an assertion about the principal ability of a node to receive the synchronizing message for the current step during its receptive interval. Note that for forwarding a synchronizing message over the entire length of an indirect link, all nodes along the link have to be receptive.

**Definition 7.** (I/O Function of a (Direct) Link). For a given execution of a cascade, the I/O function of a link $l \in S \cup U$, written $f_l$, is defined as the function mapping the sequence of messages written by the sender’s program to the sequence of messages arriving at the receiver node, where the special output symbol $\perp$ indicates a lost message.

**Definition 8.** (I/O Function of an Indirect Link). For a given execution of the cascade, the I/O function of an indirect link $l = l_1 \rightarrow l_2 \rightarrow \cdots \rightarrow l_m$, where $l_i \in S \cup U$ and $l_1$ and $l_m$ are the first and last links in the direction of message flow, respectively, is defined as the composition of the individual links’ I/O functions:

$$f_l = f_{l_m} \circ \cdots \circ f_{l_2} \circ f_{l_1}$$

Using these definitions, a lossy link models both message loss due to the sender’s communication layer aborting the send, and due to the medium losing messages. In order to define bounded message losses, we will use the definition of n-bounded lossy channels from Section 3 to capture the condition that a cascade may suffer from a bounded number of message losses on each of its direct and indirect synchronizing links.

**Definition 9.** (n-Bounded Lossy Cascade). A cascade with master $N_0$ is an n-bounded lossy cascade iff, for all executions of the cascade, all direct and indirect synchronizing links from $N_0$ to nodes $N \in N$, the link’s I/O function is an n-bounded lossy channel.

### 3.3 Choice of Parameters

This section gives some predefined values for the parameters $T_{ma}$ and $n_{pf}$ used by the send/receive phase activation algorithm. $T_{ma}$ is chosen such that loss of the synchronizing messages cause the receiver’s activations to be delayed by $T/2$ w.r.t. the master’s activation instants, while $n_{pf}$ results from an analysis of the maximum number of lost messages that can be tolerated by the synchronization algorithm:

$$T_{ma} = \frac{3}{2} T,$$

$$n_{pf} = \max_{N \in N} \left( \left\lfloor \frac{1}{4T\varepsilon} \left( T - \left( 2 \sum_{s_j \in L(N)} \Delta_{i,s_j} \right) + \text{max}(2Q, T\varepsilon) \right) \right\rfloor \right)$$

where the computed value for $n_{pf}$ is required to be positive. For $\varepsilon \ll 1$, $Q \ll T$, the choice for $T_{ma}$ can be shown to be very close to an optimally robust assignment, so that a maximal number of lost synchronizing messages can be tolerated in the presence of clock drifts. We will demonstrate in the next section that, for our assignment for $T_{ma}$, the cascade indeed satisfies this robustness requirement for an $n_{pf}$-bounded number of lost messages.
Lemma 2. Let \( N \) be a non-master node in a cascade under normal operating conditions. Then \( j \)-receptiveness of \( N \) and \( \text{Par}(N) \) imply \( j \)-synchronization of \( N \).

Proof. Straightforward adaptation of Lemma 2: Normal operating conditions ensure that the direct or indirect synchronizing link to \( N \) is \( 1 \)-bounded lossy, so synchronizing messages from \( N_0 \) are never lost. Because \( N_0 \) will send a message in every step \( j \), and both \( N \) and \( N_0 \)'s parent nodes are \( j \)-receptive, a synchronizing message will be received by \( N \) in its receptive interval for all \( j \).

Lemma 3. Let \( N \) be some non-master node in a cascade under normal operating conditions. Then \( (j-1) \)-synchronization of \( N \) implies \( j \)-receptiveness of \( N \).

A detailed proof is given in the appendix. Intuitively, it suffices to show that, given \( j \)-receptiveness of \( \text{Par}(N) \) and \( j-1 \)-synchronization of \( N \), messages will always arrive after the quiet interval has elapsed, and before timer of \( N \) reaches \( T \). Similarly to Lemma 2, the two corner cases are: (1) \( N_0 \)'s clock “fast”, \( N \)'s clock “slow”, synchronizing message at \( j-n \) with maximum latency, synchronizing message at \( j \) with minimum latency, and (2) \( N_0 \)'s clock “slow”, \( N \)'s clock “fast”, synchronizing message at \( j-1 \) with minimum latency, synchronizing message at \( j \) with maximum latency.

Lemma 6. Let \( N \) be some non-master node in a cascade under transient fault conditions. Then for a given step \( j > n_{pf} \), let \( J = \{ j-n_{pf}, \ldots, j-1 \} \) be a set of successive step indices. If \( N \) and \( \text{Par}(N) \) are \( j \)-receptive, then there is at least one \( j'' \in J \) such that \( N \) is \( j'' \)-synchronized.

Proof. Transient fault conditions imply that the synchronizing link from \( N_0 \) parent to \( N \) is \( n_{pf} \)-bounded lossy. According to our definition for the node’s behavior, \( N_0 \) will send a synchronization message in every step. Then from Definition 5, it is clear that \( N \) receives a synchronization message for at least one \( j'' \in J \), and so Lemma 5 is a direct adaptation of Lemma 3.

Lemma 7. Under transient fault conditions, all nodes in \( N \) are \( j \)-receptive for all \( j \).

Proof. The proof is again by double induction over \( j \) in the index set, and over the nodes on the paths from \( N_0 \) to nodes \( N \in J \).

1. (base case) \( \forall j, j \)-receptive(\( N_0 \)):
   
   By Definition 2.

2. (induction step) \( \forall j, j \)-receptive(\( \text{Par}(N) \)) \( \Rightarrow \) \( \forall j, j \)-receptive(\( N \)):
   
   Split into cases (2a) and (2b) for the inner induction.

   (2a – base case)
   
   \( \forall j \leq n_{pf}, j \)-receptive(\( N \)):
   
   By Definition 2. \( N \) is \( 0 \)-synchronized. So there exists an \( n \leq n_{pf} \) such that \( N \) is \( (j-n) \)-synchronized and, by Lemma 5, \( N \) is \( j \)-receptive.

2b – induction step)

   \( \forall j > n_{pf}, \{ j-n_{pf}, \ldots, j-1 \} \)-receptive(\( \text{Par}(N) \)) \( \land \) \( \{ j-n_{pf}, \ldots, j-1 \} \)-receptive(\( N \)) \( \Rightarrow \) \( j \)-receptive(\( N \)):
   
   Because \( N \) and \( \text{Par}(N) \) are \( j' \)-receptive for all \( j' \in \{ j-n_{pf}, \ldots, j-1 \} \), we know by Lemma 5 that there exists an \( n \leq n_{pf} \) such that \( N \) is \( (j-n) \)-synchronized. But this is just the precondition for Lemma 5 so \( N \) is \( j \)-receptive.

Theorem 1. Under normal operating conditions, a non-master node \( N \) will always remain in state \( \text{EXTERNALLY_triggered} \).

Proof. We observe that \( N \) is initialized to state \( \text{EXTERNALLY_triggered} \). Because of Lemma 3, \( N \) is \( j \)-receptive for all \( j \), we conclude from Definition 2 that the precondition for leaving state \( \text{EXTERNALLY_triggered} \), \( \tau_{\text{ref}} = T_{\text{max}} \), will never hold. Therefore, the only reachable state for \( N \) is \( \text{EXTERNALLY_triggered} \).

In case of temporary message losses, the network is operating under transient fault conditions: nodes affected by loss of their synchronizing message may transition temporarily to state \( \text{MESSAGE_ABSENT} \). We can show, however, that a given node will always re-synchronize itself with the master, and count never reaches \( n_{pf} \). As a consequence, the node will never enter state \( \text{SELF_triggered} \).

Definition 13. (Transient fault conditions)

A cascade is said to operate under transient fault conditions iff it is \( n_{pf} \)-bounded lossy, and the \( \varepsilon \)-bounded clock drift assumption holds.

Lemma 5. Let \( N \) be some non-master node in a cascade under transient fault conditions. Then if there exists an \( n \), \( 1 \leq n \leq n_{pf} \), such that \( N \) is \( (j-n) \)-synchronized, then \( N \) is \( j \)-receptive.

Again, the details of the proof can be found in the appendix. It demonstrates that, for any \( n' \) such that \( 1 \leq n' \leq n_{pf} \), if a node \( N \) has performed \( n' \) – 1 unsynchronized cycles, synchronizing messages will arrive after the quiet interval has elapsed, and before timer of \( N \) reaches \( T \). Similarly to Lemma 2, the two corner cases are: (1) \( N_0 \)'s clock “fast”, \( N \)'s clock “slow”, synchronizing message at \( j-n \) with maximum latency, synchronizing message at \( j \) with minimum latency, and (2) \( N_0 \)'s clock “slow”, \( N \)'s clock “fast”, synchronizing message at \( j-1 \) with minimum latency, synchronizing message at \( j \) with maximum latency.

Theorem 2. Under transient fault conditions, a non-master node will never enter state \( \text{SELF_triggered} \).

Proof. By Lemma 3, the non-master node will be always receptive when receiving a synchronizing message. Therefore, each arriving synchronizing message is received. Transient fault conditions guarantee that the medium loses at most \( n_{pf} - 1 \) subsequent messages. Consequently, count never reaches \( n_{pf} \), so the precondition count \( \geq n_{pf} \) for transitioning to state \( \text{SELF_triggered} \) will never hold.

We denote as permanent fault conditions all other operating conditions, such as non-\( n_{pf} \)-bounded lossy cascades, violation of the \( \varepsilon \)-bounded clock drift assumption, or complete failure of nodes. Behavior of the cascade under such conditions will not be discussed in the scope of this paper.
4. PROPERTIES

This section defines some essential requirements that a synchronization cascade has to satisfy for distribution of synchronous programs in medium-criticality applications, and demonstrates the corresponding formal properties.

4.1 Requirements

P-reactivity

Distributed real-time control applications typically contain periodic, reactive parts which continually compute output values from a given input. Because inputs may originate from other nodes, it is highly desirable to provide an architecture that allows reactive programs to safely synchronize their local processing with communication on the medium, eliminating the needs for special “watchdogs” or similar mechanisms. We will define a property called P-reactivity which captures the fact that a node performs communication actions with a certain minimal frequency. Local processing can therefore be triggered by the communicator handler.

**Definition 14 (P-reactivity).** A node \( N_i \) is called P-reactive for some \( P \in \mathbb{R}_+ \) if, for all possible executions of \( N_i \) and for all instants \( t \), there is at least one activation instant for a send/receive phase in the time interval \([t, t + P]\).

Unit delay and length preserving channels

Semantically correct deployment of a synchronous specification warrants that the communication channels provided by the communication layer are valid implementations of the corresponding abstract channels in the specification. As will be indicated in Section 4.2, we will use unit delay channels as our model for an abstract channel. Breaking down the original requirements for the cascade from Section 4.1 to the implementation of channels, the cascade should (1) provide a valid implementation of unit delay channels under normal operating conditions, and (2) provide some limited service, including synchronization, under transient fault conditions. The synchronization service can be abstracted as a lossy channel with the 1-length-preserving property. Length-preserving then captures the fact that sender and receiver never get “out of sync”.

In the following, we will show that the cascade indeed satisfies the stated requirements.

4.2 Properties of Synchronization Cascades

**Proposition 1.** Under normal operating conditions or transient fault conditions, all nodes are \( (T_{max}/(1 - \varepsilon)) \)-reactive.

By definition of timer task in Fig. 3, \( N_i \) is activated at least every \( T_{max} \) time units (measured by its physical clock) in all of the three possible states EXTERNALLY_TRIGGERED, MESSAGE_ABSENT, SELF_TRIGGERED. Because the bounded clock drift assumption holds under normal and transient fault conditions, the worst case of a “slow” physical clock is \( \beta \text{timer}/\beta t = 1 - \varepsilon \). In physical time, the greatest interval in between activations is therefore \( T_{max}/(1 - \varepsilon) \).

**Proposition 2.** Under normal operating conditions, the input/output behavior of a synchronizing link is a unit delay channel.

For an intuitive treatment, there are four parts constituting the unit delay channel property: (1) every message sent by the sender must be accepted by the communication medium, (2) once accepted, the message must reach the receiver, (3) the cascade receiver must be receptive, (4) a message computed at step \( j \) at the sender is processed at step \( j + 1 \) at the receiver. (1) and (2) are guaranteed by the 1-bounded message loss assumption. (3) and (4) are direct consequences of Lemma 1.

**Proposition 3.** Under normal operating conditions or transient fault conditions, the input/output behavior of a synchronizing link is a 1-length-preserving channel.

In the case of normal operating conditions, the 1-length-preserving property results from Proposition 1. For transient fault conditions, the fact that the synchronizing link itself is 1-length-preserving follows from Lemma 3: let \( N, N' \) be the sender and receiver of the synchronizing message, respectively. For a given step \( j \), there are two possibilities: (1) if the synchronizing message is not lost in step \( j \), then \( N \) and \( N' \) will both be \( j \)-synchronized. They will therefore agree on the step number, and \( N' \) will process in step \( j \) the result of \( N \)'s computation at step \( j - 1 \), so the channel’s behavior may be characterized as a unit delay for step \( j \). (2) if the synchronizing message is lost in step \( j \), then \( N' \) will detect a \( \bot \) symbol each time \( N \) emits a synchronizing message. In both cases, the input/output behavior of the link constitutes a 1-length-preserving channel.

4.3 Nonsynchronizing messages

In the discrete-time abstraction of the synchronous programs, synchronizing messages correspond to messages with deterministic timing: if the sender component has computed the synchronizing message at step \( j \), the synchronizing message will always be processed by the receiver component at step \( j + 1 \) in normal operation. This is why abstracting the link as a (deterministic) unit delay channel in the synchronous program, as shown in the Section 4.2, is justified for synchronizing links.

For nonsynchronizing links, the deterministic delay channel abstraction may not always be valid. A nonsynchronizing message computed in step \( j \) by a sender node may reach a receiver node at steps \( j, j + 1, \ldots \), depending on the timing of activations of the two nodes, and the timing of messages on the synchronizing link. Fortunately, best/worst-case analysis, such as in [2], can be used in theory to ensure that some nonsynchronizing messages have deterministic timing. For other messages, it may be necessary to either add some flow control mechanism to the communication layer, or to account for the nondeterminism in the discrete-time abstraction.

4.4 Mapping Synchronous Programs to Cascades

Consider the simple network in Fig. 3(a). The network includes components \( \{A, B, C, D\} \), and unit delay channels \( pre \) in between components. A sends messages through signal \( b \) to component \( B \), and through signal \( c \) to component \( C \). \( B \) sends messages to \( D \) (signal \( d_j \)), \( C \) sends messages through signal \( d_1 \) to component \( D \), and to \( A \) through signal \( a \). The dataflow network is mapped to the cascade of Fig. 4 with the mapping \( \{A, n_0\}, \{B, n_1\}, \{C, n_2\}, \{D, n_3\}\). Fig. 4(b) shows a depiction of the resulting cascade. The resulting distribution is correct for normal operation if the two nonsynchronizing channels \( u_1, u_2 \) have deterministic unit-delay behavior. Note that the synchronization messages carry values that the distributed program needs to communicate. If, for a given system step, no such value needs to be communicated, an empty message, with no relevant data, must be used as a synchronizing message.


5. CONCLUSION

Our method of distribution relies on the existence of delays at the partitioning boundaries in the specification. Clearly, introduction of such delays is somewhat implementation-driven: an “ideal” platform with infinite resources would not require delays in the model, except for breaking causal loops. Bottom-up introduction of delays will in most cases necessitate a re-validation of the entire design, and is in conflict with the idea of having implementation-independent synchronous specifications. We propose in [3] a methodology which enforces introduction of delays at the boundaries of (abstract) software components at early design stages, thus ensuring both implementation-independence and partitionability of the specification.

In comparison to the LTIA approach of Benveniste et al. [3], synchronization cascades are designed for protocols with restricted availability, while in LTIA, the bus is assumed to be ideally available. LTIA links may be abstracted as deterministic channels, while in cascades, even under normal conditions, some unsynchronized links may exhibit nondeterministic behavior. In cascades, the activation timing of non-master nodes always depends on the master node, and is roughly periodic. This is also true if synchronous programs with (delayed) feedback loops are deployed onto a cascade. For LTIA networks, convergence of the activation frequencies for periodic, length-preserving programs with feedback is not obvious from [3].

For the special case of the CAN protocol, Time-Triggered CAN (TTCAN) [6] is a CAN-based synchronization layer which, in its Level 1 stage, does not rely on additional hardware for synchronization, similar to cascades. Arbitration in TTCAN is primarily based on static assignment of message slots. An interesting question is whether existing unsynchronized nodes can be integrated with the synchronized network: In TTCAN, unsynchronized nodes may only have read access to the bus, while in a CAN-based cascade, full read/write interoperation is possible if the messages sent by unsynchronized nodes are included in the message jitter analysis in Section 4.

High-precision clock synchronization algorithms such as [9] are well-studied. This kind of algorithm provides a high-precision synchronization, where clocks are synchronized within an interval in the range of $\Delta(1 - 1/|N|)$, with $|N|$ the number of nodes, while cascades are merely synchronized in the range of $T/2$, with $T$ as the base period. However, we claim that our design of a synchronization cascade is more specifically suited to the requirements outlined in Section 4. For instance, Welch and Lynch’s algorithm requires $|N|^2$ synchronization messages for each round vs. $|N| - 1$ messages for a cascade. Welch and Lynch’s algorithm uses explicit synchronization rounds, where the synchronization messages could potentially block other real-time traffic on the medium. Cascades, on the other hand, provide synchronization using the regular real-time traffic of the distributed program. We also claim that the $T/2$ precision may be sufficient in cases where synchronization is important for correct, timely implementation of the distributed program’s semantics, but a precise absolute global time base is not necessary.

Our next goals will be an experimental evaluation of the method along with some tool support [2], and the definition of a fault-modular variant of the cascade, where subtrees within the cascade can safely retain their relative synchronization in the case of master faults.

Acknowledgements

Thanks to Bernhard Schätz and the anonymous reviewers for helpful comments on an earlier version of this paper.

6. REFERENCES


APPENDIX

Upper and Lower Bounds

We will use the following properties for working with upper and lower bounds: Let $S$ be a set, let $F, G : S \rightarrow \mathbb{R}$ be functions from $S$ to the reals such that upper and lower bounds exist for $F, G,$ and let $\min_{x \in S}(F(s))$ and $\max_{x \in S}(F(s))$ be the lower and upper bounds of $F$ on $S,$ respectively. Furthermore, let $C$ be some constant. Then the following properties hold:

\[
\begin{align*}
\max_{x \in S}(C) &= C \\
\min_{x \in S}(C) &= C \\
\max_{x \in S}(F(s) + G(s)) &\leq \max_{x \in S}(F(s)) + \max_{x \in S}(G(s)) \\
\min_{x \in S}(F(s) + G(s)) &\geq \min_{x \in S}(F(s)) + \min_{x \in S}(G(s)) \\
\max_{x \in S}(-F(s)) &= -\min_{x \in S}(F(s)) \\
\min_{x \in S}(-F(s)) &= -\max_{x \in S}(F(s)) \\
\max_{x \in S}(F(s)) < \min_{x \in S}(G(s)) &\implies \forall s \in S.F(s) < G(s) \\
\max_{x \in S}(F(s)) \leq \min_{x \in S}(G(s)) &\implies \forall s \in S.F(s) \leq G(s)
\end{align*}
\]

Proof of Lemma 3

For $j$-receptiveness of $N,$ Equations 8 (case (1)) and 9 (case (2)) must hold.

(1) We can rewrite Equation 8 as

\[Q + (t_{i,j-1} - t_{0,j-1}) < d_{\min}(0, i) + (t_{0,j} - t_{0,j-1}).\]

This condition is quantified over all possible executions of the cascade under normal operating conditions: we indicate the set of such executions with $\text{NOC}$. Using Equations 8 and 9, we eliminate the quantification and use lower/upper bounds instead:

\[Q + \max_{(\text{NOC})}(t_{i,j-1} - t_{0,j-1}) < d_{\min}(0, i) + \min_{(\text{NOC})}(t_{0,j} - t_{0,j-1}).\]

From Section 3.2, it follows that $T'(1 + \varepsilon)$ is a lower bound for $t_{0,j} - t_{0,j-1}$. Because $N$ is $(j - 1)$-synchronized by assumption, the bounded message jitter property from Section 3.2 yields $\max_{j}(\text{NOC})_{i}(t_{i,j-1} - t_{0,j-1}) = d_{\max}(0, i)$. Substituting and using Equation 9, the following must hold:

\[Q + \sum_{\sigma_j \in \mathcal{L}(N_1)} \Delta\nu(s_j) < T \frac{1}{1 + \varepsilon}.
\]

Using the $(T - Q)/2$ bound for the message jitter from Definition 3, we have to show

\[Q + \frac{T - Q}{2} < T \frac{1}{1 + \varepsilon}.
\]

Solving for $Q$ yields the condition $0 \leq \varepsilon < 1$

\[Q < T \frac{1}{1 + \frac{\varepsilon}{1 + \varepsilon}} ,
\]

which holds for $Q < T \cdot (1 - 2\varepsilon), 0 \leq \varepsilon < 1$. This proves case 1.

(2) By assumption, it is true that $(j - 1)$-synchronized($N$).

Rewriting Equation 8 and using upper/lower bounds yields:

\[
\frac{T_{\text{ma}}}{1 + \varepsilon} + \min_{(\text{NOC})}(t_{i,j-1} - t_{0,j-1}) > \max_{(\text{NOC})}(t_{0,j} - t_{0,j-1}) + d_{\max}(0, i)
\]

With $T'(1 - \varepsilon)$ as an upper bound for $t_{0,j} - t_{0,j-1}, d_{\max}(0, i)$ as a lower bound for $(t_{i,j-1} - t_{0,j-1}),$ and Equation 9, we obtain:

\[
\frac{T_{\text{ma}}}{1 + \varepsilon} > \frac{T}{1 - \varepsilon} + \sum_{\sigma_j \in \mathcal{L}(N_1)} \Delta\nu(s_j).
\]

Substituting our choice for $T_{\text{ma}}$ from equation 4 and solving for $\sum_{\sigma_j \in \mathcal{L}(N_1)} \Delta\nu(s_j)$ with $0 \leq \varepsilon < 1$ yields:

\[
\sum_{\sigma_j \in \mathcal{L}(N_1)} \Delta\nu(s_j) < \frac{T(1 - 5\varepsilon)}{2(1 - \varepsilon^2)}.
\]

For $0 \leq \varepsilon < 1$, this inequation follows from Definition 5. This proves case 2.

Proof of Lemma 5

By assumption, there exists an $n \in \{1, \ldots, n_{pf}\}$ such that $N$ is $(j - n)$-synchronized. Let $n'$ be the smallest such $n$. Then for $j$-receptiveness of $N$, Equations 5 (case (1)) and 8 (case (2)) must hold.

(1) We distinguish cases (1a) ($N$ is $(j - 1)$-synchronized) and (1b) ($N$ is $(j - n')$-synchronized, and $2 \leq n' \leq n_{pf}$).

(1a) See case (1) of the proof for Lemma 5.

(1b) We can rewrite Equation 5 as

\[Q + (t_{i,j-1} - t_{0,j-1}) < d_{\min}(0, i) + (t_{0,j} - t_{0,j-1}).\]

The equation is implicitly quantified over the set of executions under transient fault conditions, $TFC$. Quantification is removed by taking upper/lower bounds:

\[Q + \max_{(\text{TFC})}(t_{i,j-1} - t_{0,j-1}) < d_{\min}(0, i) + \min_{(\text{TFC})}(t_{0,j} - t_{0,j-1}).\]

(16)

For obtaining an upper bound for $t_{i,j-1} - t_{0,j-1}$, we split the term $t_{i,j-1} - t_{0,j-1}$ using the identity

\[t_{i,j-1} - t_{0,j-1} = (t_{i,j-n'} - t_{0,j-n'}) + (t_{i,j-1} - t_{i,j-n'}) - (t_{0,j-1} - t_{0,j-n'}).\]

Taking the maximum over executions $TFC$, and using Equations 5 and 6, we obtain

\[\max_{(\text{TFC})}(t_{i,j-1} - t_{0,j-1}) \leq \max_{(\text{TFC})}(t_{i,j-n'} - t_{0,j-n'}) + \max_{(\text{TFC})}(t_{i,j-1} - t_{i,j-n'}) - \min_{(\text{TFC})}(t_{0,j-1} - t_{0,j-n'}) ,
\]

which can be resolved as follows:

- $\max_{(\text{TFC})}(t_{i,j-n'} - t_{0,j-n'})$:
  We observe that, by assumption of Lemma 5 and using the right-hand side condition of Equation 5, $t_{i,j-n'} - t_{0,j-n'} \leq d_{\max}(0, i)$. Therefore, $d_{\max}(0, i)$ is a valid upper bound.

- $\max_{(\text{TFC})}(t_{i,j-1} - t_{i,j-n'})$:
  According to the operational definition of $N$, $N$ will first detect a message absence (yielding $T_{\text{ma}}/(1 - \varepsilon)$ as an upper bound for the duration of cycle $j - n'$) and then perform $n' - 2$ unsynchronized steps (yielding an upper bound of $(n' - 2) \cdot T'(1 - \varepsilon)$ for the remaining cycles). The total upper bound is $T_{\text{ma}}/(1 - \varepsilon) + (n' - 2) \cdot T'(1 - \varepsilon)$.

- $\min_{(\text{TFC})}(t_{0,j-1} - t_{0,j-n'})$:
  The lower bound for the duration of $n' - 1$ cycles of the master is $(n' - 1) \cdot T'(1 + \varepsilon)$.

We substitute the upper bound for $t_{i,j-1} - t_{0,j-1}$ into Equation 5, and use $T'(1 + \varepsilon)$ as a lower bound for $t_{0,j} - t_{0,j-1}$ and Equation 9 for $d_{\max} - d_{\min}$. Then the following property remains to be shown:

\[\frac{T_{\text{ma}}}{1 + \varepsilon} + (n - 2) \frac{T}{1 - \varepsilon} + Q + \sum_{\sigma_j \in \mathcal{L}(N_1)} \Delta\nu(s_j) < n' \frac{T}{1 + \varepsilon}.
\]
Solving for \( n' \) results in the condition \((0 \leq \varepsilon < 1, T > 0): \)

\[
n' < \frac{1}{4T^2} \left( T(1 + \varepsilon) - \left( 2Q + 2 \sum_{s_j \in L_i(N_i)} \Delta_h(s_j) \right) (1 - \varepsilon^2) \right).
\]

For \( 0 \leq \varepsilon < 1, \) this holds if

\[
n' < \frac{1}{4T^2} \left( T - \left( 2Q + 2 \sum_{s_j \in L_i(N_i)} \Delta_h(s_j) \right) (1 - \varepsilon^2) \right).
\]

This follows from \( n' \leq n_{pf} \) and Equation 3, so we’re done for case 1.

(2) We distinguish cases (2a) \((N \text{ is } (j - 1)\text{-synchronized})\) and (2b) \((N \text{ is } (j - n')\text{-synchronized and } 2 \leq n' \leq n_{pf})\).

(2a) See case (2) of the proof for Lemma 3.

(2b) For this case, it is true that \( \varepsilon < (j - 1)\text{-synchronized}(N)\).

Rewriting Equation 8 and using upper/lower bounds yields:

\[
\frac{T}{1 + \varepsilon} + \min\limits_{(TPC)} (t_{i,j-1} - t_{0,j-1}) > \max\limits_{(TPC)} (t_{0,j} - t_{0,j-1}) + t_{max}(0, i).
\]

A lower bound for \( t_{i,j-1} - t_{0,j-1} \) is again found by splitting up the term and using Equations 8–13.

- \( \min\limits_{(TPC)} (t_{i,j-1} - t_{0,j-1}) \):
  
  By assumption of Lemma 3 and using the left-hand side condition of Equation 8, \( t_{i,j-1} - t_{0,j-1} \geq d_{min}(0, i) \), so \( d_{min}(0, i) \) is a valid lower bound.

- \( \min\limits_{(TPC)} (t_{i,j-1} - t_{0,j-1}) \):
  
  \( N \) will first detect a message absence (lower bound \( t_{ma}/(1 + \varepsilon) \)) and then perform \( n' - 2 \) unsynchronized steps (lower bound \( (n' - 2) \cdot T/(1 + \varepsilon) \)) for the remaining cycles. The total lower bound is \( t_{ma}/(1 + \varepsilon) + (n - 2) \cdot T/(1 + \varepsilon) \).

- \( \max\limits_{(TPC)} (t_{0,j} - t_{0,j-1}) \):
  
  The upper bound for the duration of \( n' - 1 \) cycles of the master is \( (n' - 1) \cdot T/(1 + \varepsilon) \).

With \( T/(1 + \varepsilon) \) as an upper bound for \( t_{0,j} - t_{0,j-1} \) and Equation 11, substituting the above bounds into Equation 11 yields:

\[
\frac{T_{ma}}{1 + \varepsilon} + (n' - 1) \frac{T}{1 + \varepsilon} > n' \frac{T}{1 - \varepsilon} + \sum_{s_j \in L_i(N_i)} \Delta_h(s_j).
\]

Solving for \( n' \) yields \((0 \leq \varepsilon < 1, T > 0): \)

\[
n' < \frac{1}{4T^2} \left( T(1 - \varepsilon) - 2 \sum_{s_j \in L_i(N_i)} \Delta_h(s_j) (1 - \varepsilon^2) \right).
\]

For \( 0 \leq \varepsilon < 1, \) this constraint is satisfied if

\[
n' < \frac{1}{4T^2} \left( T(1 - \varepsilon) - 2 \sum_{s_j \in L_i(N_i)} \Delta_h(s_j) \right).
\]

Again, for \( n' \leq n_{pf}, \) this follows from Equation 3. This concludes the proof for case 2. \( \square \)