Proceedings of the of the 17th Conference on Computer Science and Intelligence Systems pp. 505–514
DOI: 10.15439/2022F176
ISSN 2300-5963 ACSIS, Vol. 30

Formal analysis of timeliness in the RaSTA protocol

Billy Naumann, Christine Jakobs, Matthias Werner
TU Chemnitz
Faculty of Computer Science
Chemnitz, Germany
Email: {billy.naumann,christine.jakobs,matthias.werner}@informatik.tu-chemnitz.de

***Abstract*—Formal reasoning about the correctness of safety-critical system properties is crucial since such systems may impact their environment when malfunctioning. The Rail Safe Transport Application (RaSTA) Protocol is a protocol for systems used in railway applications such as signaling. It claims to provide highly available and timely communication based on the application's demands. We investigate timeliness, i.e., the property that application data do not become obsolete.**

We analyze the protocol's specification and provide assumptions necessary to resolve imprecisions. Under the specified error model, we find that the deadlines proposed bound until messages are considered timely is too restrictive, disabling RaSTA's mechanisms to recover from lost messages in time. We formalize the specification of timeliness to provide a counterexample for the proposed bound and create an improved bound that does not lead to violated deadlines under the same assumptions and error model.

I. Introduction

THE *Rail Safe Transport Application (RaSTA)* [1] is a protocol used in railway signaling technology between diverse communication endpoints. It is independent of the overlaying application. Since it may be usable for *safety-critical* applications, its correctness is essential. The object of this paper is to formally verify the correctness of a part of the RaSTA protocol. We formally investigate RaSTA's timeliness property using networks of timed automata and the tool Uppaal for formal reasoning [2].

RaSTA's specification rests upon natural language. The interpretation of such a specification often relies on either explicit or implicit assumptions, allowing to focus on aspects considered necessary while abstracting from others. Those assumptions pose the danger of creating a model that cannot reflect the wanted properties. The following quote from Sir Tony Hoare shows that formal approaches are a necessary and helpful tool to discuss such assumptions:

> The job of formal methods is to elucidate the assumptions upon which formal correctness depends.

In this investigation, we discuss necessary assumptions about imprecisions in RaSTA's specification, prove that the bound for messages' timeliness given in RaSTA's specification is insufficient, and provide an improved bound.

The remainder of this paper is structured as follows: Section II provides an overview of the normative requirements applicable to RaSTA and an overview of the protocol itself, including assumptions made in the RaSTA specification. Section III introduces the formal semantics of networks of timed automata and Uppaal. In Section IV we present our model and the evaluation of RaSTA's timeliness property. Finally, section V gives a conclusion of our work.

II. RaSTA Protocol

The RaSTA [1] protocol is specified in a pre-standard by the DKE/UK 351.3, a national working committee of the Association for Electrical, Electronic and Information Technologies for railway signaling facilities. It is used in railway signaling technology to achieve safe and highly available communication.

A. Requirements

Strict normative requirements exist, as safety is a crucial concern in this field. RaSTA implements the requirements of [3] for safe communication in open communication systems of category 2, including networks consisting of safety-critical and non-safety-critical systems that can read, write, process, and transmit data. Safety-critical systems use safety-critical transmission functions, assuring Authenticity, Integrity, Timeliness, and Sequence of sent messages. Specifically, RaSTA defines timeliness as a state in which information is available in time according to the requirements. The number of users is generally unknown, as well as their application. Thus unknown amounts of data in arbitrary formats are sent in such networks. There might be routing and management facilities and the communication media may be prone to unforeseen external faults. Authorized access with malicious intentions is explicitly negligible in this category. Thus no cryptographic means are enforced.

The system's functionality must ensure the aforementioned properties of safety-critical transmission functions. The implementation of such a system implies an evaluation of possible safety threats. Appropriate means must be used to mitigate these threats. In [3], a list of specific safety means is provided, consisting of short descriptions and their requirements.

B. Architecture

RaSTA is implemented between a typical communication stack's application layer and the transport layer, as pictured in Figure 1. There are only a few requirements on the transport layer, making RaSTA suitable for different scenarios: It must be possible to send messages to specific receivers. The network, including the communication partners, must process the messages in a best-effort manner. It is unnecessary to

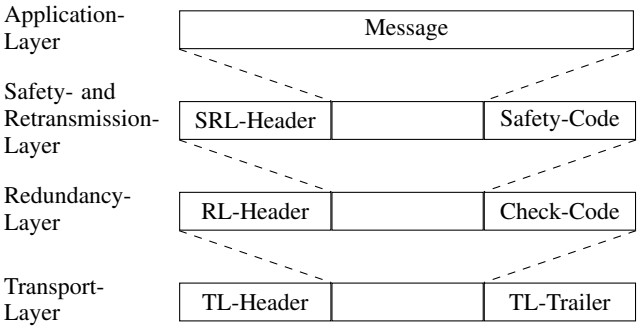


Fig. 1: Architecture of the RaSTA Protocol Stack

TABLE I: Protocol Data Unit of the SRL

Byte(s)	**Content**
0 - 1	length
2 - 3	type
4 - 7	receiver ID
8 - 11	sender ID
12 - 15	sequence number
16 - 19	confirmed sequence number
20 - 23	timestamp
24 - 27	confirmed timestamp
28 - 28+k-1	payload (k bytes)
28+k - (28, 35, 42)+k	safety code

enforce deadlines, but safety reasons imply fast transmission and adequate quality. Reference [1] states that *commercial of the shelf* transport services such as UDP or TCP are well suited.

RaSTA introduces two new layers between application and transport: the upper *Safety- and Retransmission-Layer (SRL)* and the lower *Redundancy-Layer (RL)*, also shown in Figure 1. The SRL provides a safe communication mechanism for networks according to [3], while the RL aims to provide a highly available communication service via so-called *Redundancy-Channels*. The RL uses multiple transport layer channels (possibly with different transport services) for redundant communication. In this way, messages that get lost or altered on a single transport channel do not affect the communication on SRL-level. Since most of the means to ensure the necessary properties reside in the SRL, this paper only briefly covers the RL.

Table I shows the design of the *Protocol Data Unit* (PDU) of the SRL. We omit a representation of the RL at this point.

The SRL makes use of *IDs* for sender and receiver to ensure authenticity as well as a *safety code* based on the message digest 4 (MD4) [4] algorithm to ensure integrity. Depending on the individual requirements, the latter can take either all, a few, or none of the result's bytes. The protocol uses the *sequence number (SN)* and the *confirmed sequence number (CS)* to maintain the correct sequence of all communicated messages: With each communicated message, SN gets incremented. At the same time, CS represents the last received *sequence number* of the communication partner. Both *timestamp (TS)* and *confirmed timestamp (CTS)* fields are used to ensure timeliness. Here, TS represents the time at which the sender created the message, and CTS represents the last received timestamp of the communication partner, analog to the confirmed sequence number. The insurance of both Sequence and Timeliness requires additional logic, discussed in the protocol specification.

The RL has a more lightweight PDU. There are two primary choices: it uses CRC for its *check code* with different possible configurations. Also, using an additional *sequence number* ensures noticing any race conditions between messages via the individual transport channels. Please note that additional logic is necessary to ensure a correct transmission on the receiver side.

C. Protocol specification

Since RaSTA is a relatively new protocol stack, there is not much work regarding formalizing and verifying its properties, even though its usage in safety-critical scenarios. However, a shortcoming is using MD4 as safety code as described in section II-B, shown by [5]. Here, possible changes to the protocol stack extend RaSTA's abilities to withstand attacks such as the injection of forged messages or replay attacks. Such malicious attacks are not in the scope of RaSTA's requirements for category 2 networks according to [3], but it raises the question if these assumptions are valid in the use case of railway signaling.

RaSTA defines message types for the SRL, used in different situations. Figure 2 shows the abstract state machine for the SRL. Defined are *Connection Request (ConnReq)*, *Connection Response (ConnResp)*, *Disconnect Request (DiscReq)*, *Heartbeat (HB)*, *Data*, *Retransmission Request (RetrReq)*, *Retransmission Response (RetrResp)*, and *Retransmitted Data (RetrData)* messages, from which *Data*, *RetrData* and *HB* messages are defined as relevant for time monitoring. Using the first two message types, connections are established and set up by performing a handshake between both communication partners. This is visualized in Figure 2 with arrows (a), (b), and (c). A *Disconnect Request* message is sent prior to closing an established connection or to indicate errors during the establishment or regular transmission. The corresponding transitions shows Figure 2 as dashed arrows. The communication partners monitor the connection quality via messages of type *HB*. Such Heartbeats are automatically sent after a defined time interval during which no other messages were sent. Application messages carrying a payload are transmitted as *Data* messages. The remaining message types handle error situations: A lost message leads to a corrupted sequence of messages is corrupted. The receiver can recognize this situation, in which he sends a *RetrReq* message. Figure 2 reflects this situation with transition (d). The original sender of the lost or corrupted message answers with a *RetrResp* message (corresponding to transition (f) in Figure 2), after which he repeats all messages with an unconfirmed sequence number as *RetrData* messages. To finalize, a *Data* or *HB* message is sent to indicate that the retransmission is completed, returning to normal operation by transition (e) in Figure

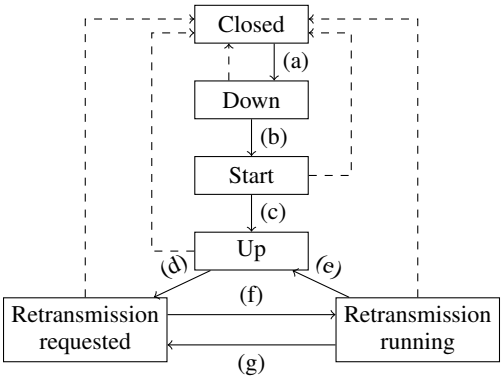


Fig. 2: Abstract State Machine of the SRL [1]

2. Finally, transition (g) corresponds to the situation where another message gets lost during a running retransmission, where another retransmission is initiated.

Besides the abstract states shown in Figure 2, each instance has to manage an internal state, consisting, among other things, of a set of sequence numbers, timestamps, and timers:

SN_T: the sequence number of the next transmitted message
SN_R: the expected sequence number of the next received message
CS_T: the confirmed sequence number of the next transmitted message
CS_R: the confirmed sequence number of the last received message
TS_R: the last received timestamp
CTS_R: the last received confirmed timestamp
T_{HB}: a timer representing the remaining time until the instance has to send a new message
T_{DL}: a timer representing the remaining time in which received messages are considered timely

The maximum values for the timer T_{DL} and T_{HB} are defined as configurable parameters, depending on the applications demands. We use $T_{DL,max}$ and $T_{HB,max}$ to represent them.

To transmit data, the sender has to create a new PDU according to Table I using $SN = SN_T$, $CS = CS_T$, $CTS = TS_R$, and $TS = t$, where t is the sender's current timestamp. After creating and sending the message, SN_T is incremented ($SN_T = SN_T + 1$) and T_{HB} is reset to $T_{HB,max}$. The sender has to store a copy of the message until a message with $CS \geq SN$ is received.

The message receiver has to check the sequence of SN, CS, and CTS. First, plausibility checks evaluate if $SN - SN_R$ is below a configured limit. Also, $CS_R \leq CS < SN_T$ has to hold, stating that the received and confirmed sequence number is plausible, too. If these conditions do not hold, the receiver discards the message. Otherwise, he performs more vigorous checks on SN and CTS. A boolean variable $SNinSeq$ is set to true, if $SN_{PDR} = SN_R$ holds. Also, another boolean variable $CTSinSeq$ is set to true, if for messages relevant for time monitoring $0 \leq CTS - CTS_R < T_{Dl,max}$ holds. Note that the receiver does not discard the messages if the checks result in negative results. The state transitions shown in Figure 2 depend on these variables. Summarizing the resulting behaviors, if $CTSinSeq$ does not hold, the receiver responds by sending a *DiscReq* and closing the connection. If $SNinSeq$ does not hold, the receiver initiates retransmission of all messages with unconfirmed sequence numbers (i.e. $CS_T \leq SN < SN_T$).

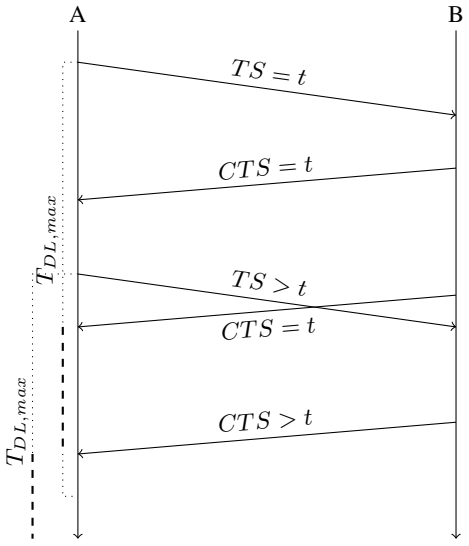


Fig. 3: Adaptive Channel Monitoring

D. Timeliness in RaSTA

RaSTA ensures timeliness by applying a concept called *Adaptive Channel Monitoring (ACM)*. The necessary information is included in each SRL-PDU but will be evaluated only for time monitoring relevant messages. According to [1], clocks may not be synchronized and can have different resolutions, consequentially disallowing the interpretation of timestamps of the communication partner.

ACM applies the *Double-Timestamp*-Principle, defined in [3] to check the message round trip times on both communication partners. A sent message carries the sender's local timestamp in TS. Upon receiving this message, the receiver stores this exact value as $TS_R = TS$ in his local state. The following answer of the receiver will carry the confirmed timestamp $CTS = TS_R$ back to the original sender, allowing the round trip time calculation as $RTT = t - CTS$ where t is the current timestamp. A message's timeliness is analyzable by constraining the round trip time since this value overestimates the timestamp of the send-event. Since multiple messages can carry the same CTS value, a round trip completes once a message with a new, greater CTS value arrives. This fact naturally implies that the original sender sent a message with $TS > t$ in the meantime. This message also starts the next round trip, overlapping the current one. This is visualized in Figure 3 for an error-free communication.

After a message with a specific new value of CTS arrives, the receiver updates CTS_R in the local state. All later messages with $CTS = CTS_R$ are considered timely, if

they arrive in an interval of length $T_{DL,max}$ since CTS_R. If another message with $CTS > CTS_R$ arrives, the stored value of CTS_R will be replaced by the new one, indicating that no further messages with the old CTS value should arrive anymore.

To implement these conditions, [1] uses the timer T_{DL}, which is reset with each update to CTS_R to $T_{DL} = T_{DL,max} - RTT$. If this timer reaches its limit, arriving messages may carry outdated information, which would be the case when $CTS = CTS_R$. Figure 3 visualizes this timer as dashed and dotted lines where the dashed portion represents the actual timer running while the dotted portion corresponds to the time since the reference point or until the deadline, respectively.

As $T_{DL,max}$ is a configurable parameter, it is possible to adjust it according to the application's needs. However, [1] states that $T_{DL,max}$ should include enough buffer to take possible retransmissions into account, and gives the following suggestion for its minimal value:

$$T_{DL,max} > 3 \cdot T_{HB,max} + 2 \cdot (T_{AB} + T_{BA}) + T_{RL,seq} \quad (1)$$

Here, according to [1], $T_{HB,max}$ refers to the communication partner's maximum time between two consecutive messages. T_{AB} and T_{BA} indicate the worst-case transmission time of the channel from sender to receiver, where A and B are the communication partners, and $T_{RL,seq}$ indicates the maximum time a message can get delayed in the RL because of race conditions.

As specified in [1], the reasoning behind this formula is as follows: the round trip time of a message can be overestimated by $T_{HB,max} + T_{AB} + T_{BA}$, a lost message will be noticed at worst after $2 \cdot T_{HB,max}$ and the following retransmission can be estimated as $T_{AB} + T_{BA}$, which sums up to the right hand side of Equation 1.

This bound results in an assumed error model where only one message per round trip can be lost. We examine this bound and see it as problematic since it introduces artificial dependencies between the communication partners and their communication channels: As soon as one channel loses one message, the other must deliver correct messages. This assumption is unrealistic since, in reality, the channels themselves cannot share such information.

E. Assumptions regarding RaSTA's specification

Before modeling RaSTA's communication to show timeliness, we need to state some assumptions regarding open or imprecisely defined aspects.

1) Violation of message sequence: Since the RL aggregates multiple transport connections between sender and receiver to a single redundancy channel, the correct order of the messages has to be assured since race conditions along the individual channels can occur. The configuration parameter $T_{RL,seq}$ states the delay of messages to be able to restore sequence before delivery to the SRL. We assume that this parameter is set to 0, effectively allowing messages to overtake each other unhandled. This assumption is reasonable since the SRL notices the incorrect message sequence in the same way a lost message would, triggering the retransmission.

2) Immediate Responses: The specification [1] is unclear about internal delays of messages that should be transmitted immediately. This delay is significant in the case of a retransmission of unconfirmed messages. We assume that no additional delay is introduced between two such messages, effectively sending all of them in the correct order simultaneously.

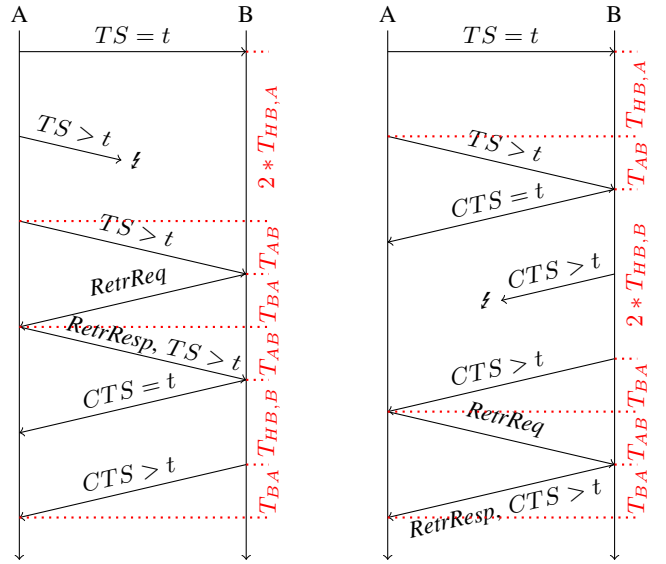
3) Handshakes: We focus on analyzing timeliness for the central part of the protocol: The data exchange, in essence, by the behavior of a message's round trip. By that decision, we exclude the handshake to establish the connection and any disconnection semantics. The initial handshake includes a final time-critical heartbeat message. However, this message must be sent immediately after receiving the connection response message. Thus, it will set the initial reference point for the upcoming data exchange. If the receiver discards the message, the handshake fails, and the connection is not established.

Additionally, we reduce the second part of the retransmission handshake, i.e., the *RetrResp, RetrData, HB* sequence to transmit all missing messages to a single *RetrResp* message. This reduction is feasible since we use the assumptions in Section II-E2, together with the transmission error model. The handshake is only carried out when a communication partner previously discarded a message. Hence, during this handshake, no further messages are discarded. All messages are sent without any delays. Hence, the retransmitting side immediately sends the (final) heartbeat message carrying the $CTS > t$ information. This assumption allows us to abstract from the specific messages to be confirmed.

F. Discussion of RaSTA's timelines property

A derivation of Equation 1 according to [1] is given in section II-D. However, we like to point out that the interpretation of $T_{HB,max}$ as the communication partners parameter is not feasible for all scenarios.

Under the assumption that only one message per round trip can get lost, there are, in essence, two different scenarios that lead to retransmission, shown in Figure 4. Both scenarios share the loss of the information used to finish the round trip, i.e., a timestamp $TS > t$ or a confirmed timestamp $CTS > t$. We describe these scenarios from the view of the final $CTS > t$ message, as the receiver will consider this message's timeliness. Hence, communication partner A is the receiver while B is the sender in the scenarios. In Figure 4a, the receiver's first message containing $TS > t$ is lost during transmission. Subsequently, the sender discards the following message also carrying this information. Since the message sequence is not correct at this point, the sender initiates retransmission. He continues to send heartbeat messages containing the $CTS = t$ information until the finalization of the retransmission. We omitted them for readability reasons. In the worst case, a heartbeat containing $CTS = t$ is sent just before the $TS > t$ information reaches the sender, leading



(a) receivers (A) fault (b) senders (B) fault

Fig. 4: Retransmission scenarios

to sending the $CTS > t$ information with the next heartbeat. Figure 4a also shows the individual worst case delays.

Accumulated, they lead to the following upper bound shown in Equation 2:

$$T_{DL,A} = 2 \cdot T_{HB,A} + T_{HB,B} + 2 \cdot (T_{AB} + T_{BA}) \quad (2)$$

Analogously, we show the situation where the sender's first message containing $CTS > t$ is lost in Figure 4b. The receiver's message containing $TS > t$ arrives at the sender just after a message containing $CTS = t$ was sent. Hence, the $CTS > t$ is sent as the next heartbeat and gets lost. The receiver discards the next heartbeat since the message sequence is incorrect. At this point, the receiver initiates the retransmission. After this retransmission, the $CTS > t$ information reached the receiver. Equation 3 presents the according upper bound.

$$T_{DL,B} = 2 \cdot T_{HB,B} + T_{HB,A} + 2 \cdot (T_{AB} + T_{BA}) \quad (3)$$

Since both scenarios can occur under our assumptions, we propose to use the maximum of both as the bound for the deadline, referred to by $T'_{DL,max}$, as shown in Equation 4.

$$T'_{DL,max} > \max(T_{DL,A}, T_{DL,B}) \quad (4)$$

Please note that we used $T_{HB,\cdot}$ and $T_{DL,\cdot}$ to indicate $T_{HB,max,\cdot}$ and $T_{DL,max,\cdot}$ for spacing reasons. In the following sections, we show a formal analysis of both limits presented in Equations 1 and 4 using timed automata supported by Uppaal.

III. Timed Automata and Uppaal

To check the correctness of both limits presented in Equations 1 and 4, we modeled the interesting aspects of RaSTA as timed automata and checked this model with the help of Uppaal.

A. Uppaal

Uppaal [6], [2] is described as "an integrated tool environment for modeling, validation, and verification of real-time systems modeled as networks of timed automata, extended with data types (e.g., bounded integers, arrays)." It is developed jointly by Basic Research in Computer Science at Aalborg University in Denmark and the Department of Information Technology at Uppsala University in Sweden.

Many applications of Uppaal in scientific case studies are shown on Uppaal's website [2], such as the verification of different versions of the well-known Fischer Protocol [7] for mutual exclusion in [8]. Nevertheless, also industrial protocols such as the Philips Audio Protocol for exchange of control information [9], or the Bang and Olufsen Audio/Video Protocol for transmission of messages between audio and video components over a single bus [10] have been model checked by Uppaal. Primarily the latter was known to be faulty. Uppaal's generation of (erroneous) traces allowed us to find the error. Also, Uppaal was used to find and verify a fix for this problem.

B. Modelling

Timed automata exist in multiple flavors, but they generally combine the known concept of finite state machines and clocks, unique variables used to represent time. Uppaal uses a dense time model, where clocks evaluate real numbers and advance synchronously. For evaluation, Uppaal can express clock valuations as symbolic constraints, thus reducing the state space by collapsing all clock valuations that share common properties. Further, Uppaal allows systems to be modeled as networks of timed automata by composition from individual automata. Every automaton may engage in (enabled) transitions, also used to synchronize multiple automata. [11]

In [11], the definition of the Timed Automata used in Uppaal is as follows:

A timed automaton $\mathcal{A}$ is a tuple (L, l_0, C, A, E, I), where L is a set of locations, l_0 is the initial location, C is the set of clocks, A is a set of actions, co-actions and the internal τ-action, $E \subseteq L \times A \times B(C) \times 2^C \times L$ is a set of edges between locations with an action, a guard and a set of clocks to be reset, and $I : L \to B(C)$ assigns invariants to locations.

Reference [11] also provides the definition and semantics of a network of timed automata, consisting of n timed automata $\mathcal{A}_i, 1 \leq i \leq n$. The location vector $\bar{l} = (l_1, ..., l_n)$ corresponds to the locations of each individual automation. Further, the invariants are merged to an invariant function over the location vectors $I(\bar{l}) = \wedge_i I_i(l_i)$. Finally, the notation $\bar{l}[l'_i/l_i]$ denotes the location vector where the ith element l_i is replaced by l'_i.

The semantics are then given by [11] as follows: Let $\mathcal{A}_i = (L_i, l_i^0, C, A, E_i, I_i)$ be a network of timed automata and $\overline{l_0} = (l_1^0, ..., l_n^0)$ the vector of initial locations. The semantics is defined as a labelled transition system $\langle S, s_0, \to \rangle$ with $S \subseteq (L_1 \times ... \times L_n) \times \mathcal{R}^C$ as the set of states, $s_0 = (\overline{l_0}, u_0)$ is the initial state and $\to \subseteq S \times S$ is the transition relation such that:

- $(\bar{l}, u) \xrightarrow{d} (\bar{l}, u+d)$ if $\forall d' : 0 \leq d' \leq d \implies u+d' \in I(\bar{l})$, and
- $(\bar{l}, u) \xrightarrow{a} (\bar{l}[l'_i/l_i], u'])$ if $\exists l_i \xrightarrow{\tau g r} l'_i : u \in g, u' = [r \mapsto 0]u$ and $u' \in I(\bar{l}[l'_i/l_i])$.
- Further, $(\bar{l}, u) \xrightarrow{a} (\bar{l}[l'_j/l_j, l'_i/l_i], u')$ if $\exists l_i \xrightarrow{c?g_i r_i} l'_i \wedge l_j \xrightarrow{c!g_j r_j} l'_j : u \in (g_i \wedge g_j), u' = [r_i \cup r_j \mapsto 0]u \wedge u' \in I(\bar{l}[l'_j/l_j, l'_i/l_i])$

Hence, possible transitions are categorized in *delay* and *action* transitions. The former are described by $(\bar{l}, u) \xrightarrow{d} (\bar{l}, u+d)$, letting the system evolve in time for d time units by mapping each clock $c \in C$ to the value $u(c) + d$, if the invariants of all locations aren't violated for any time point until the delay has occurred. The latter corresponds to a single edge or a pair of edges in the automaton. Here, either a single automaton or a pair of automata in the network change their locations according to action a in $(\bar{l}, u) \xrightarrow{a} (\bar{l}[l'_i/l_i], u'])$ and $(\bar{l}, u) \xrightarrow{a} (\bar{l}[l'_j/l_j, l'_i/l_i], u')$ respectively. Either are only possible if their guards are satisfied. Taking the transition resets all referenced clocks, and the resulting locations' invariants must be satisfied. When two automata perform such a pairwise transition, it is also necessary to label the transitions by corresponding co-actions, expressing active and passive synchronization at these points.

Uppaal extends such networks of timed automata with many features shown in the following list, as well as other constructs borrowed from C-like programming languages such as arrays, initializers, record- and custom-types as functions. [11]

- *Templates* for instantiating automata
- Invariants over *internal state variables*
- *Non-deterministic choice* of binary synchronization channels when multiple co-actions are possible
- *Urgent and committed locations* which disallow the passage of time

Within such templates, Uppaal uses additional labels for locations and edges, which allow to define the behavior of the automaton in an easy way, for example, to express a location's invariants. Actions can have *select* labels, which can be used to non-deterministically bind values from a given range to variables which can then be used in the remaining labels of the action. Also, *guards* enable actions upon fulfillment or disable them in case of violation. *Synchronization* labels allow the use of synchronization channels, where edges with complementary synchronization labels $c!$ and $c?$ over a shared channel c synchronize on taking the $c!$ labeled action, reassembling co-actions. Finally, *update* labels can alter the current internal state by changing variables' values or assigning values to clocks. [11], [12]

Figure 5 shows an example of a timely bounded synchronous communication channel. We use this cannel to model the communication between two RaSTA communication partners. Messages are accepted from the sender via the `send?` co-action, transiting from the `Idle` to the `Transmitting` location. The variable `content` refers to the channel's content, taken from whatever resides in `data_send`, the senders send buffer. To reduce the state space, the send buffer is

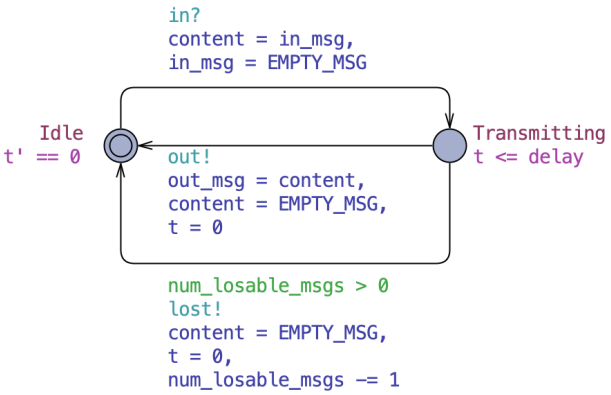


Fig. 5: Model of the RaSTA Communication Channel

reset to `empty` as soon as the channel has accepted the message. As long as `t <= delay`, the channel may reside in the `Transmitting` location, transitions back to `Idle` are possible immediately. There are two possible transitions corresponding to the correct and faulty transmission. In the latter case, the channel's content is not altered and copied into the receive buffer. Instead, the `lost!` co-action indicates an altered message to discard. The clock `t` is reset when transiting back to `Idle`, where it is stopped via the location's invariant. These measures are also made to reduce the state space.

To define a model, Uppaal uses a *system definition*, which allows instantiating templates to *processes*. Here, it is possible to bind the template's parameters to actual values or define partial instantiations to reuse similar processes. They are composed concurrently to a system by enumerating them after the `system` keyword.

C. Verification

Uppaal allows checking different properties for a system via model checking. Queries to the model express these properties using a simplified version of *Timed Computation Tree Logic (TCTL)* [13]. Queries consist of the path- and state formulae, where state formulae describe individual states and path formulae quantify over the model's traces. Uppaal does not allow the nesting of path formulae [11]. Such formulae are categorized depending on their semantic and matched runs. We explain the used state formulae and safety formulae in the following sections. Additionally, the standard version of Uppaal supports reachability formulae and liveness formulae. Note that many extensions extend classical Uppaal, for example, by examining statistical properties.

1) State Formulae: State formulae express the properties of individual states without considering the model's behavior. They are similar to guards in that they are described by side-effect-free expressions, for instance, `x == 42`. Besides statements over internal variables, it is also possible to test if a automaton `A` is in a certain location `l` by the expression `A.l`. Internal state variables of a single Automaton are accessible in the same way. Further, deadlocked states (where no outgoing action transitions from the state or delayed successors are possible) can be expressed via the keyword `deadlock`. [11]

2) Safety Formulae: Safety Formulae describe that *something bad will never happen*. A general technique is to express something bad in the model's terms, for instance, the violation of a deadline, and then invariantly assure that the model never fulfills this condition. Analogously, the model must fulfill *something good* invariantly. Uppaal uses TCTL formulae $A\Box\phi$, which are expressed in Uppaal as `A[]` ϕ for a state formula ϕ, expressing that ϕ must be true for all reachable states. [11]

IV. Verifying Timeliness

While it is possible to build a system in Uppaal reassembling the whole SRL and find a suitable formula to describe timeliness in this model, we decided to abstract from this approach for multiple reasons. First, we are only interested in the property of timelines. Therefore, it is feasible to abstract from unnecessary parts of the protocol. Our model abstracts the exact Protocol Data Unit shown in Table I while keeping each message's sequence and time information. Note that it is possible to omit CS since the receiver will discard messages if the plausibility checks fail as described in section II-C. Also, by maintaining an appropriate communication channel model, one can abstract from the RL, leaving only a concentrated portion of the protocol for verification. It is possible to reduce this model further since we are only interested in the timeliness of a single message, expressed as a round trip as shown in Figure 3. This reduction is possible since RaSTA claims to ensure timeliness for all time-critical messages, so it is sufficient to find a single situation where RaSTA fails to do so to show the violation of this property. This reduction aims to find a Uppaal model with only a few locations and internal state variables.

Most model checking tools try to explore the whole state space of the model by finding all possible execution paths and states on them. This approach becomes infeasible quickly since the number of possible states grows exponentially with the number of used internal state variables. Ultimately, this *state space explosion* leads to an infeasible time demand for evaluating the properties. There exist ways to approach this problem, such as symmetry reductions, but especially for software verification, this problem is still not solved [14]. This issue emphasizes the importance of a compact and abstract model.

A. Modelling assumptions

We discussed some possible abstractions and why they are feasible at the beginning of this section. Such abstractions' bases are usually on assumptions that restrict the system's modeled behavior in a certain way. While such abstractions allow the formulation of a simpler model, it is essential to ensure that the result is still a valid model of reality, including all necessary aspects of the system to reason about the properties of interest. Otherwise, the model might still be correct but becomes irrelevant since it does not lead to any desired statements.

We have shown assumptions necessary to formalize the specification of RaSTA in Section II-E. The following sections discuss the consequences of formalization and make assumptions regarding which aspects the model of the protocol stack needs and which can be abstracted.

1) Redundancy Channels: As described in section II-E1, we assume that the redundancy channels used in the RL lead to possible violations of message sequence. Since we are not interested in showing availability improvements, we decided to abstract from the RL and model only the SRL. The underlying communication channels are seen as per message, meaning that a new virtual communication channel is available for each message. Since Uppaal supports the dynamic instantiation of templates only for statistical queries, we have to limit ourselves to a constant number of available channels, thus limiting the number of messages sent simultaneously. Since, in real-world scenarios, communication always is limited by a specific throughput, we find this assumption feasible. We restrict our analysis to the case where $T_{AB} < T_{HB,max,A}$ and analogously for the values of B. This restriction allows the assumption of FIFO channels so messages cannot overtake each other. Alternating messages during transmission, leading to discarding the message by its receiver, is still possible. Such errors have the same impact on possible retransmissions initiated by the SRL.

2) Message semantics: Heartbeat and Data messages share the same information except for an empty application payload. The sender sends Heartbeat messages only when the application is ready to send (*Data*) messages in a defined time interval since the last message. Since RaSTA is independent of the overlying application, we can use this fact to abstract from both message types and reduce them to a single kind of message.

3) Timestamp relationship: Since we aim to show timeliness for a single message round trip, we can abstract from the specific values of the timestamps and use a relative representation during this round trip. Additionally, such a relative representation can abstract from the actual values since only the relationship between their corresponding send- and receive-events is necessary to capture the behavior, as shown in Figure 3. Hence, we directly represent the relationship between the ongoing time and the messages TS and CTS values using state variables.

4) Message Sequence: Another critical assumption is that the specific values of the sequence numbers and confirmed sequence numbers do not matter to show the timeliness of a single message. The sequence numbers are used to trigger retransmission if $SNinSeq$ is false, as described in Section II-C. The receiver sets this flag if two consecutive messages do not have consecutive sequence numbers. To be able to abstract from the specific values, we inform the receiver of a message about the alternation of the message. In reality, the receiver would check the message's integrity via the safety code and discard it if the check fails. Hence, we update $SNinSeq$ before the reception of the following message. The receiver can then react appropriately based on this information by

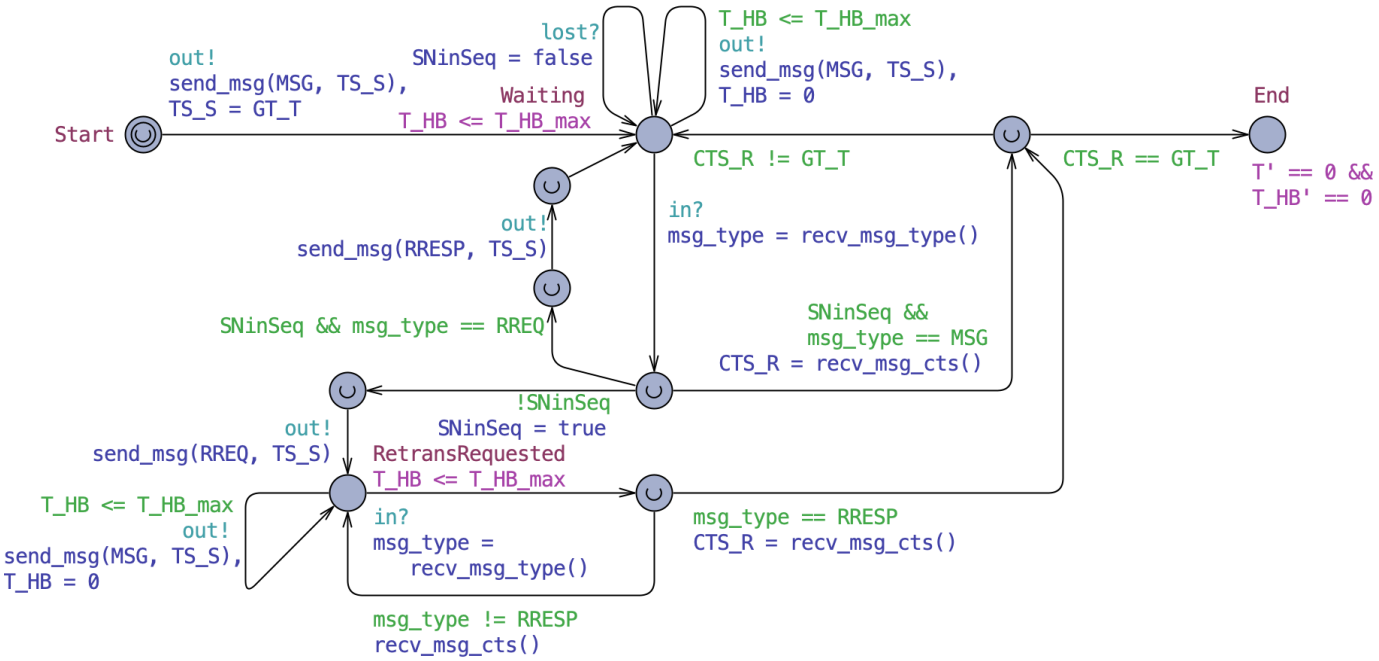


Fig. 6: Model of the RaSTA Receiver

engaging in a retransmission handshake.

5) Immediate Responses: We assume that the sender will not delay messages from being sent immediately. There are two significant points in the protocol where this is important: The initial *ConnReq, ConnResp, HB* handshake to establish the connection and the handshake to perform retransmission, including the transmission of the *RetrData* messages carrying the previously lost messages. As stated in II-E2, [1] covers this not explicitly: The visualizations show a delay of one time unit per message while the descriptions state that the sender has to send messages immediately.

6) Number of retransmitted messages: When a message is lost, the sender of this message must retransmit all unconfirmed messages, as described in Section II-C. Such messages may be sent before the modeled round trip. Hence, it is impossible in our model to find a representation of the messages to be retransmitted. However, the only information about these messages we care about is if one of them will complete the current round trip. Especially the final *HB* message will contain the $CTS > t$ information. Therefore, we reduced the set of retransmitted messages to a single one carrying the relevant information. This reduction is feasible since we already assumed that there is no additional delay between immediately sent messages in Section IV-A5.

B. Model and Verification

In this section, we show our model and the analysis of the SRL of RaSTA in Uppaal based on the assumptions shown in the previous sections. Since we want to model a single round trip, our model uses asymmetrical behavior, even if the RaSTA protocol is symmetrical after connection establishment. Hence, individual templates in Uppaal model sender and receiver. The channel is also an individual template model, which is instantiable multiple times to enable communication between sender and receiver.

Since we abstract from the specific values of the timestamps, it is necessary to model the relation between the reference point t^* when the receiver sends an initial message and the values of `TS_R` and `CTS_R`. We defined constant values `EQ_T` and `GT_T` to express if a received (confirmed) timestamp is equal to or greater than t^*. Transmitted messages carry this information instead of concrete time information.

Both sender and receiver use a clock `T_HB`. This clock represents the time since sending the last heartbeat message. With appropriate location invariants and action guards, we enforce that both communication partners never send two consecutive messages more than `T_HB_max` time units apart, where the concrete value depends on the chosen parameter for sender and receiver. Additionally, we use a clock `T` for the receiver model to indicate the current time during the round trip, by which the receiver determines the duration until the arrival in the `End` location, where this clock is stopped.

Further, sender and receiver use urgent states whenever a message is received and during retransmissions. This reflects that the time for the evaluation of the messages header and the decision of the upcoming actions is negligible and serves as an implementation of assumption in section II-E2.

The upcoming sections describe the individual templates.

1) Receiver: The Receivers model is shown in Figure 6. As the evaluation of a message's round trip time depends on a defined reference timestamp, the receiver starts sending a message carrying its current timestamp t^* in TS, which is implemented in the `send_msg(MSG, TS_S)` update. At this point `TS_S` has the value `EQ_T` and is updated to `GT_T`

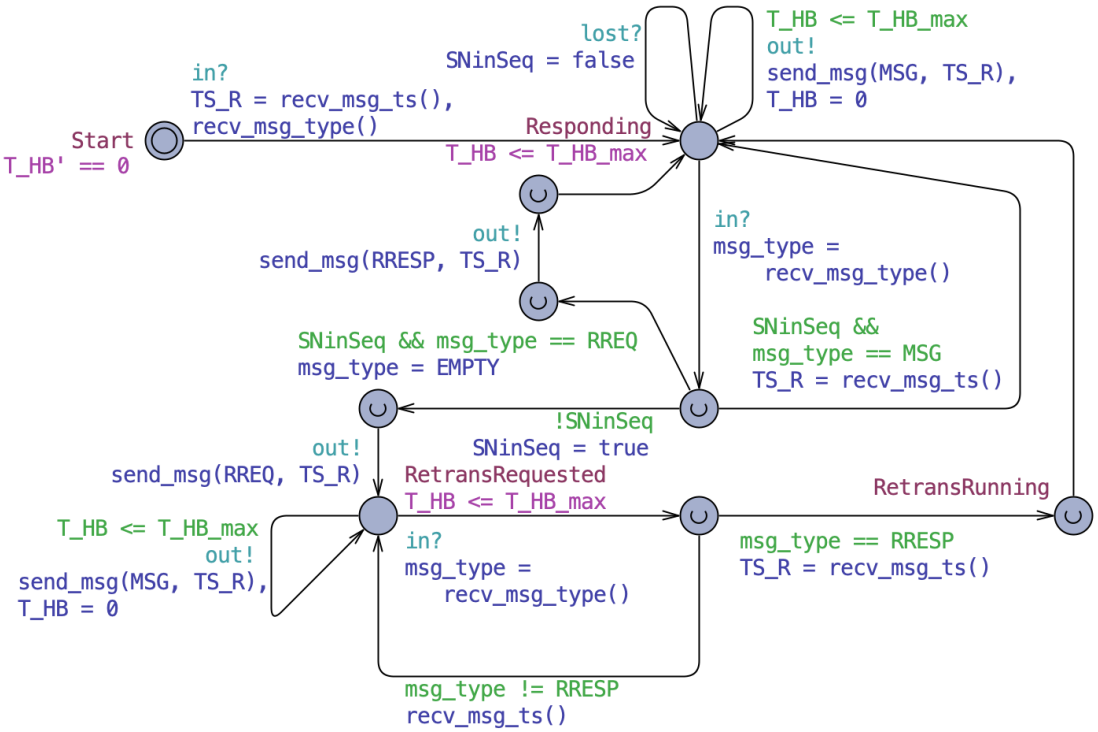


Fig. 7: Model of the RaSTA Sender

after the message is sent. The receiver will use this information for sending all further messages. He will reside in the location `Waiting` until either a new message has to be sent or a message is received or indicated as lost. By the assumption in section IV-A2, we abstract from distinct *HB* and *Data* messages at this point. We do not abstract from *RetrReq* messages. Such messages are identified via the `msg_type` variable and handled appropriately. The clock `T_HB` together with the location's invariant and the transition guards constrain the time between sending such messages. When a message is indicated as lost, the receiver sets `SNinSeq = false`. In reality, this would happen later when the following message is received, but since this is causally dependent on a lost message in our model, we decided to set this flag at this point. When a message is received, the receiver checks `SNinSeq` and either updates its `CTS_R` value based on the information in the message or triggers retransmission. If the message is a *RetrReq*, indicated by `msg_type == RREQ`, the according *RetrResp* message is sent immediately, carrying `GT_T` as its timestamp. By assumption in Section II-E3, this is the only message necessary to complete this retransmission in our model. Otherwise, and when no retransmission is necessary, the receiver either returns to the `Waiting` location when the received CTS value is less or equal to t^*, i.e., `CTS_R != GT_T`. In the same case, he moves to the `End` location, indicating that a new reference timestamp was confirmed by the sender via `CTS_R == GT_T`, thus ending the round trip. If retransmission is necessary, the receiver will initiate the corresponding handshake by sending the `RREQ` message and transitioning to the `RetransRequested` location. Here the receiver continues to send messages as in the `Waiting` location and waits until the retransmitted messages arrive. All other messages with `msg_type != RRESP` are ignored, their information will be part of the *RetrResp* message. Upon receiving his message, the receiver evaluates the contained `CTS_R` as for regular messages.

2) Sender: In our model, the sender takes the role of sending the messages whose timeliness is subject to verification. The sender starts by receiving the reference point t^* as its first message, represented by the value of `TS_R`, which is `EQ_T` at this point. At this point, the sender transitions into the `Responding` location. From here on, the model is similar to the receiver, differing only in interpreting the message's timing information. Where the receiver uses this information in the `CTS_R` variable, the sender uses it to update the state of the `TS_R` variable. Also, the `End` location is absent since the sender is not informed about finishing the round trip.

3) Channel: As described in Section IV-A1, we model the RL and underlying channels as per message. Uppaal supports the dynamic instantiation of templates only for statistical queries. Hence we are forced to limit our model by over-approximating the number of sent messages. At this point, a single channel will act synchronously, only accepting messages for transmission when it is in `Idle` and only allowing the delivery returning from the `Transmitting` location, as shown previously in Figure 5. As long as the channel is in location `Idle`, its clock `t` is stopped at 0 by the location's invariant. When a message is accepted via the `in?` co-action, the channel transitions to the `Transmitting` location, and

after at most the worst-case transmission delay `delay` the message is either delivered or lost. Both cases are modeled as a transition back to the `Idle` state, resetting both the internal state of the channel and its local clock `t`. Only for successful transmission, the channel copies the message to the receiver's buffer `out_msg`, the receiver is synchronized here via the `out!` co-action. The co-action `lost!` is used for a faulty transmission, where the channel ignores the message's contents. In the case of an erroneous transmission, the channel also decrements the variable `num_losable_msgs` to 0 to indicate that no further messages should be lost.

4) Verification and Results: Since, in our model, the absence of missed deadlines describes timeliness, the formulation of the property is possible as a safety formula, stating that there is no case of deadline violation. We checked both the original value shown in Equation 1 as well our adapted deadline shown in Equation 4 via the following formulae for verification in Uppaal, where both the values for $T_{DL,max}$ and $T'_{DL,max}$ are calculated as the bounds based on the parameters for the heartbeat and worst-case transmission times in Uppaal.

$$A\Box(Receiver.End \implies Receiver.T \leq T_{DL,max}) \quad (5)$$

$$A\Box(Receiver.End \implies Receiver.T \leq T'_{DL,max}) \quad (6)$$

Since Uppaal doesn't allow symbolic constants for model parameters, we used the values $T_{HB,max,A} = 5$, $T_{HB,max,B} = 3$, and $T_{AB} = T_{BA} = 1$, resulting in the bounds $T_{DL,max} = 13$ and $T'_{DL,max} = 17$. With these values, we could show that the proposed bound $T_{DL,max}$ by [1] is violated while our bound is still satisfied.

Even though we aim to instantiate channels for each message individually, the increased state space limits the feasibility of the evaluation. We decided to restrict the channel model to FIFO channels by instantiating only one pair of channels between sender and receiver. As shown in section IV-A1, this limits the validity of our results to the case where $T_{AB} < T_{HB,max,A}$, as in this case, heartbeats are not affected by unavailable channels.

We were able to show for a few selected values that this property holds, but a general statement for all possible assignments is not possible in this way. However, our model can be used to verify the timeliness of a concrete RaSTA communication instance within our assumptions.

V. Conclusion and Future Work

Safety-critical systems, such as railroad communication networks, demand a clear and comprehensible analysis of all aspects that potentially affect the correctness and safety of the user and the environment. Our analysis shows where the specification of RaSTA is unclear regarding timeliness. We were able to show that the recommended deadline for the RaSTA communication protocol is not guaranteed to hold for the corresponding error scenario. This inherent violation demonstrates that using formal methods for software verification is a viable approach not only to show formal correctness where necessary but also to elucidate underlying assumptions.

While we were able to show that the proposed bound is not sufficient, we could not provide a complete formal verification of the correctness of our bound. This open end is caused primarily by inherent problems of model checking, e.g., the state space explosion when stepping back from specific abstractions, such as using concrete timestamps instead of our approach. We aim to encounter the use of more general communication channels by lifting our FIFO assumption. Also, we will deal with more complex scenarios, for example, where the worst-case transmission delay is higher than the deadline for sending heartbeats.

References

[1] "Electric signalling systems for railways - part 200: Safe transmission protocol according to DIN EN 50159 (VDE 0831-159)," Jun. 2015.

[2] Home - Uppaal. Date accessed: 2022-21-04. [Online]. Available: https://uppaal.org

[3] "Railway applications - communication, signalling and processing systems - safety-related communication in transmission systems; german version EN 50159:2010," Apr. 2011.

[4] R. L. Rivest, "The MD4 Message-Digest Algorithm," Internet Requests for Comments, April 1992, http://www.rfc-editor.org/rfc/rfc1320.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1320.txt

[5] M. Heinrich, J. Vieten, T. Arul, and S. Katzenbeisser, "Security analysis of the rasta safety protocol," in *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2018. doi: 10.1109/ISI.2018.8587371 pp. 199–204.

[6] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Uppaal—a tool suite for automatic verification of Real–Time Systems," *BRICS Report Series*, vol. 3, no. 58, Jun. 1996. doi: 10.7146/brics.v3i58.18769. [Online]. Available: https://tidsskrift.dk/brics/article/view/18769

[7] L. Lamport, "A fast mutual exclusion algorithm," *ACM Trans. Comput. Syst.*, vol. 5, no. 1, p. 1–11, jan 1987. doi: 10.1145/7351.7352. [Online]. Available: https://doi.org/10.1145/7351.7352

[8] K. G. Larsen, P. Pettersson, and W. Yi, "Compositional and Symbolic Model-Checking of Real-Time Systems," in *Proc. of the 16th IEEE Real-Time Systems Symposium*. IEEE Computer Society Press, Dec. 1995. doi: 10.1109/REAL.1995.495198 pp. 76–87.

[9] ——, "Diagnostic model-checking for real-time systems," in *Hybrid Systems III*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996. doi: 10.1007/BFb0020977. ISBN 978-3-540-68334-6 pp. 575–586.

[10] K. Havelund, A. Skou, K. G. Larsen, and K. Lund, "Formal modeling and analysis of an audio/video protocol: An industrial case study using uppaal," in *Proceedings Real-Time Systems Symposium*. IEEE, 1997. doi: 10.1109/REAL.1997.641264 pp. 2–13.

[11] G. Behrmann, A. David, and K. G. Larsen, *A Tutorial on Uppaal*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, vol. 3185, p. 200–236. ISBN 978-3-540-23068-7. [Online]. Available: http://link.springer.com/10.1007/978-3-540-30080-9_7

[12] Uppaal documentation. Date accessed: 2022-21-04. [Online]. Available: https://docs.uppaal.org/

[13] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic model checking for real-time systems," *Information and Computation*, vol. 111, no. 2, p. 193–244, Jun 1994. doi: 10.1006/inco.1994.1045

[14] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, *Model Checking and the State Explosion Problem*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, vol. 7682, p. 1–30. ISBN 978-3-642-35745-9. [Online]. Available: http://link.springer.com/10.1007/978-3-642-35746-6_1