Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

A Timed Verification of the IEEE 1394 Leader Election Protocol

J.M.T. Romijn

Software Engineering (SEN)

SEN-R9919 August 31, 1999

Report SEN-R9919
ISSN 1386-369X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

A Timed Verification of the IEEE 1394 Leader Election Protocol *

Judi Romijn
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
judi@cwi.nl

ABSTRACT

The IEEE 1394 architecture standard defines a high performance serial multimedia bus that allows several components in a network to communicate with each other at high speed. In the physical layer of the architecture, a leader election protocol is used to find a spanning tree with a unique root in the network topology. If there is a cycle in the network, the protocol treats this as an error situation. This paper presents a formal model of the leader election protocol in the language IOA as well as a correctness proof. The verification shows that under certain timing restrictions the protocol behaves correct. The timing constants proposed in the IEEE 1394 standard documentation obey the requirements found in this proof.

1991 Mathematics Subject Classification: 68M10, 68Q05, 68Q45, 68Q60, 94C15
1991 ACM Computing Classification System: F.1.1, F.4.1, G.2.2
Keywords and Phrases: protocol verification, I/O automata, safety, liveness, real time, refinement, simulation
Note: The results reported in this paper have been obtained as part of the research project "Specification, Testing and Verification of Software for Technical Applications", carried out by the Stichting Mathematisch Centrum for Philips Research Laboratories under Contract RWC-061-PS-950006-ps.
The author's current affiliation is: Computing Science Institute, University of Nijmegen, P.O. Box 9010, 6500 GL Nijmegen, The Netherlands.

1 Introduction

The IEEE 1394-1995 serial bus standard [10] defines an architecture that allows several components to communicate at very high speed. Originally, the architecture was designed by Apple (FireWire). Currently, more than 70 companies are involved in the standardisation effort. Although the IEEE 1394-1995 standard has been finalised, the architecture is still being refined and adapted. Part of this ongoing work is reflected in the IEEE P1394a standard proposal document [11], which is intended to be a supplement to IEEE 1394-1995. In this paper, 1394 will refer to IEEE 1394-1995 unless otherwise stated.

The IEEE 1394 standard allows several components to be connected either with cables and IEEE 1394 chips (cable environment), or with an IEEE 1394 backplane in one physical device (backplane environment). We restrict our attention to the cable environment situation, and refer to the whole of components, cables, etc. as *the network*.

Like in the OSI model, the IEEE 1394 architecture has several layers of which the physical layer is the lowest. This layer takes care of the actual communication on the bus, which happens

* A short version of this report appeared in S. Gnesi and D. Latella, editors, *Proceedings of the Fourth International Workshop on Formal Methods for Industrial Critical Systems (FMICS'99)*, pages 3–29, May 1999.

by sending signals on a wire by asserting voltages. The physical layer is responsible for the knowledge that a component has of the network topology and of components present, and for issues such as timing of asynchronous and synchronous communication and arbitration for use of the bus. These tasks are taken care of in several phases. The first phase in the physical layer is the bus reset phase, which is entered whenever a component is powered up, when the network topology changes or an error is discovered, or on request of higher layers in the architecture. After completion of the bus reset phase, the tree identify phase starts. In the three identify phase the network topology is determined by spanning a tree in the network. The root of the tree will act as the bus master. After the tree identify phase, the self identify phase follows in which all components inform the rest of the network of their capabilities and get a unique ID. Finally, in the normal operation phase, the arbitration for and actual use of the bus by higher layers and applications takes place.

In this paper we study the tree identify phase in the physical layer. The components employ a leader election protocol to span a tree in the network, with the root acting as the leader. A side effect of the protocol is that it detects whether there is a cycle in the network, and if so, does not terminate with a leader but halts in the initial phase of the protocol and issues error messages. Our intention is to prove that an abstraction of the protocol, which is as close to the description in the IEEE 1394 documents [10, 11] as possible, works correct. There already are some correctness proofs for other abstractions of this protocol [4, 26, 7, 28]. We reuse part of this work for proving the correctness of our model of the protocol. This is done by establishing an implementation relation between the most detailed model from [7] and our more detailed model of the protocol. In this way, our verification adds to a stepwise refinement of IEEE 1394 in which more detail is added to models in each step.

The verification is carried out by establishing timed trace inclusion between timed I/O automata through a timed refinement [16, 17, 19]. The I/O automata are presented in the IOA language [5]. We reuse an untimed I/O automaton from [7] to which we add a harmless time-passage action to turn it into a timed I/O automaton and use timed refinements as presented in [19]. As mentioned in [19], we could equally well establish an untimed refinement between the timed I/O automata, so timed trace inclusion follows if the time-passage action is visible in both models. Some related work that is interesting in the timed vs. untimed respect is the work presented in [23], which discusses safety and failure refinements between timed and untimed CSP models [3]. Some results are presented for failure refinements between communicating processes, which may be useful in the I/O automata setting.

The proofs show that under the assumptions made, the behaviour of the models is correct when we use the timing constants proposed in IEEE 1394-1195 and IEEE P1394a. It still remains to be seen whether further refinement of the models preserves the correctness.

This paper is organized as follows. Section 2 explains the IEEE 1394 tree identify, discusses related verifications and presents our abstraction. Section 3 introduces our I/O automata models of the tree identify protocol and shortly discusses the IOA language. Section 4 is an intermezzo about network topologies, in which general results are derived that we need in the verification. Section 5 presents the formal verification of the protocol. In Section 6 we sum up the conclusions that can be drawn from this exercise. We have added Appendix A to give the basic definition and principles used in this verification. Here we also present a new result for reusing invariants in a stepwise refinement proof, which is used in Section 5, as well as new sufficient conditions for feasibility, which are used in Section 5.3.

Note that to improve readibility, we often use Lamport's list notation [13] for conjunction or disjunction in formulas.

2 The protocol

In this section, the IEEE 1394 tree identify phase is described, other verifications of this protocol are discussed, and our our abstraction is introduced. The IOA models are presented in Section 3. The tree identify phase has already been described in several articles. The following text and pictures borrow from [4].

2.1 The IEEE 1394 tree identify phase

We refer to the components connected to 1394 bus as *devices*. Each device has a number of *ports* which are used for bidirectional *connections* to other devices. Each port has at most one connection. The device at the other side of the connection is called the *peer* device. The tree identify phase follows on completion of the bus reset phase, which is started as soon as a total reset of the network is demanded. This can occur on request of applications, or because the network configuration has changed or an error situation has been detected. The bus reset phase clears all topology information except local information on a device, namely which ports have connections. During the tree identify phase a spanning tree is constructed in the network. After the tree identify phase completes, the tree structure will be used in the normal bus operation. An example of a network topology at the start of the tree identify phase is presented in Figure 1.

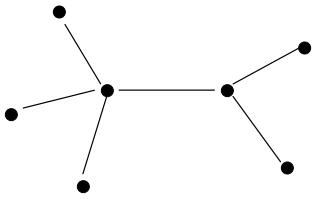

Figure 1: Initial network topology

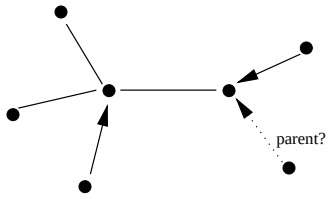

Figure 2: Intermediate configuration

Informally, the basic idea of the protocol is as follows: each device starts in the initial phase, in which it may receive a "parent request" on from a peer device on one of its ports. The receiving device then sends an acknowledgement message to the peer device and adds the port to its collection of children. A peer device which is connected to the child port, is then considered to be a child in the tree structure. See Figure 2. When a device is in the initial phase and it has no more than one port left on which no communication has taken place yet, it can send a parent request on that port and leave the initial phase. It is obvious that leaf devices (i.e. devices with one connected port) have exactly one such port at the start of the protocol, so they can send their parent request and leave the initial phase immediately. In this manner, a tree is constructed that grows from the leaves inward, until all ports of one device are children, and that device is the *root* of the tree. See Figure 4.

It is possible that two devices end up asking each other to be the parent. This situation is called "root contention". The devices both signal the reception of a parent request on a port on which they already sent a parent request, and turn to a symmetry breaking protocol in which random bits are used. See Figure 3. This root contention protocol has been formally specified and verified in [28].

When a cycle is present in the network, all the devices that are on such a cycle will not get a parent request from their peers on the cycle. So they will have more than one port on which no parent request was received, and can therefore not send a parent request themselves or leave

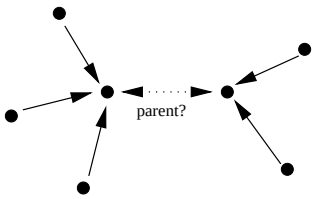

Figure 3: Two contending devices

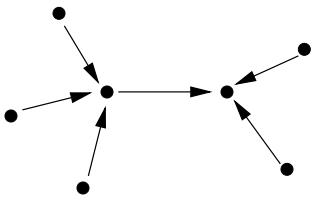

Figure 4: Final spanning tree

the initial phase. Also devices that are not on a cycle, but are wedged between two or more cycles will not get a parent request on at least two ports, and will not send a parent request themselves or leave the initial phase. Such a situation is solved by a timer which is started at the start of the tree identify phase, and which is supposed to expire only in the situation of a cycle in the network. When there is a cycle in the network, a root should not be elected, since the operation of the bus in the following phases relies on the topology being a tree structure.

A device may influence its own chances at becoming root by waiting for some time before sending the parent request, even if it is already possible to proceed. A device will only do so if it has the flag force_root set to true.

Devices may enter the tree identify phase at different times. This is due to the difference in the moments at which different devices signal that the bus reset phase (preceding the tree identify phase) should be entered.

2.2 Other verifications of the protocol

Parts of the IEEE 1394 architecture have been formally specified and/or verified in several articles [14, 12, 4, 7, 27, 26, 28]. Of these, [14, 12, 27] focus on the link layer. The articles [4, 26, 7, 28] study the tree identify phase of the physical layer, like this paper does. In Figure 5 we give an overview of the results of these articles, and their relation to the research presented in this paper. The results of the different articles are in the dashed boxes. The names of the formal models are listed, arrows between these indicate a (proved) implementation relation. The vertical position of a model name indicates the level of abstraction of that model with respect to the IEEE 1394 documentation. Very abstract models do not consider implementation details such as timing, signals etc. The most detailed models incorporate more detail from the IEEE 1394 documentation. In the picture, we have given some models the same vertical position to indicate that they have a comparable degree of detail. We now explain the results of each article in short.

Devillers, Griffioen, Romijn and Vaandrager [4] have shown that the election in the tree identify phase works correct, under the assumption that there are no cycles in the network, that the network topology is fixed throughout the protocol, that a root contention situation is solved in one atomic step, and that no device tries to become root by having the corresponding force_root flag set to true. The models are at a high level of abstraction: there is no timing and communication is modelled with finite queues. The models are I/O automata [16, 17] presented in a precondition/effect style. The proofs use invariants and simulation techniques from [18]. The proofs have been checked with the theorem prover PVS [22].

Shankland and van der Zwaag [26] have also shown that the election in the tree identify phase works correct, under the assumption that there are no cycles in the network, that the network topology is fixed throughout the protocol, and that no device tries to become root by having

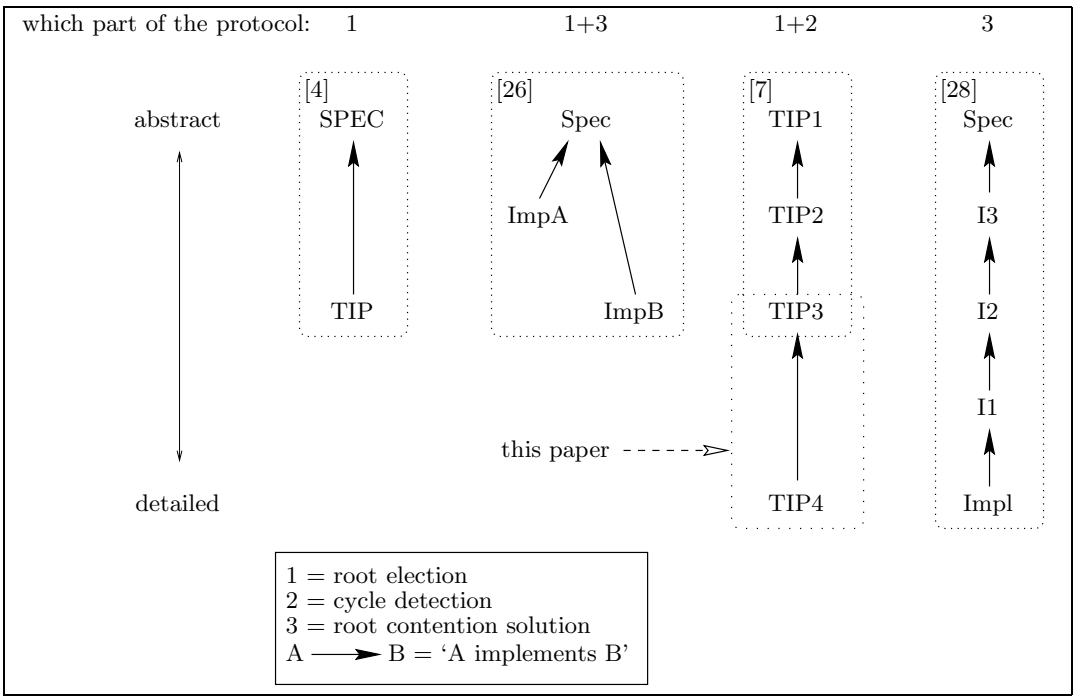

Figure 5: An overview of research on the IEEE 1394 tree identify phase

the corresponding force_root flag set to true. The models are at a high level of abstraction: there is no timing and communication is modelled with finite queues. The models are presented in μCRL [8], a process algebra language with data. The proofs use invariants and the cones and foci method from [9]. Note that the paper gives no proof that the root contention protocol actually terminates within bounded time, since for the verification it is enough to show that it *can* terminate.

Griffioen and Vaandrager [7] have shown that the election in the tree identify phase works correct, under the assumption that the network topology is fixed throughout the protocol, that a root contention situation is solved in one atomic step, and that no device tries to become root by having the corresponding force_root flag set to true. The models are at a high level of abstraction: there is no timing and communication is modelled with finite queues. The models are I/O automata [16, 17] presented in the IOA language [5]. The paper introduces a new simulation proof technique, called *normed simulations*. The proofs use invariants and the proposed simulation technique. The proofs have been checked with the theorem prover PVS [22]. Note that cycle detection is done with a predicate that takes the structure of the whole network into account, and does not use timing information, as in IEEE 1394. The predicate used implies that nodes that are part of a cycle will detect this with an error message. In IEEE 1394 (and in the models presented in this paper), the error situation is also detected by nodes that are not part of a cycle themselves, but wedged in between of two cycles.

Stoelinga and Vaandrager [28] have shown that the root contention solving protocol in the tree identify phase works correct under the assumption that the network topology is fixed throughout the protocol. The models are at an intermediate level of abstraction: on the one hand timers and probabilities are used, but on the other hand communication is modelled with finite queues. The models are probabilistic timed I/O automata [24, 25] presented in the IOA language [5].

The paper introduces two simulation proof techniques, which are special cases of the simulation techniques in [24, 25]. The proofs use invariants and the proposed simulation techniques.

The model of the protocol that is presented in [4] is essentially the same as one of the I/O automata examples in the book *Distributed Algorithms* by Lynch [15]. A correctness proof of this protocol is not given in [15]. The models that either include cycle detection or the root contention protocol are can be considered refinements of the protocol in [15].

2.3 This verification

As can be seen in Figure 5, this paper aims to give an implementation relation between the most detailed model from [7] and a more detailed model. In this way, our verification adds to a layered verification of IEEE 1394 in which models are refined, that is, more and more detail is added in each step. In order to keep our proof obligations manageable, we do not add too much detail, and hence our model has an intermediate degree of detail with respect to IEEE 1394.

The verification is carried out by establishing trace inclusion between timed I/O automata through a refinement [16, 17, 19]. The I/O automata are presented in the IOA language [5].

The most detailed model of [7] is an untimed model. This means that the cycle detection is done with a predicate that takes the structure of the whole network into account. In this verification, we want to establish that cycle detection based on the timing in IEEE 1394 works correct. In order to do this, we add timers to the model which expire when the leader election takes too much time. We also add timing information to the messages sent, in order to model the delay in communication in IEEE 1394. As argued above, this paper uses a different predicate for cycle detection than the one used in [7], in order to conform to the error behaviour of IEEE 1394. As in [7] we assume that the network topology is fixed throughout the protocol, that a root contention situation is solved in one atomic step that no device has the force_root flag set to true, and that communication can be modelled with finite queues.

Since our aim is to show that whenever timers in the model expire, there is indeed a cycle in the network, and that the timers will expire in case of a cycle in the network, we are trying to show that the timers do not expire too soon or too late. In our proofs we use invariants that express worst case scenarios in terms of delay. So we are actually performing a worst case analysis on the timing proposed in IEEE 1394. In this way, we establish a relation between the parameters of the protocol in terms of minimal and maximal values.

We expect that in a next refinement step it is possible to include the result from [28], to get closer to the IEEE 1394 behaviour without much effort. The next refinement step could then be to add a force_root flag to the model, thus expressing that devices behave a little different to increase their chances at leadership. In order to obtain a correctness statement about IEEE 1394 with all its detail, it still has to be shown that modelling the IEEE 1394 communication of voltages on wires by messages and finite queues is correct. We expect that in this situation, we will not just have a judgement on correctness, but we will also be able to say how the timing constants in IEEE 1394 could/should be adjusted.

Our assumptions As a specification of the desired behaviour, we have taken the most detailed model `TIP3` from [7]. In [7] it is shown that the behaviour of `TIP3` meets the requirements for the tree identify phase.

`TIP3` is a very abstract model of the tree identify phase, in the sense that it abstracts from a lot of details. We introduce a model `TIP4`, which is more detailed than `TIP3`, and prove that it is a refinement of `TIP3`. In this way, the correctness of the behaviour of `TIP4` can be derived

from the correctness of the behaviour of TIP3.

Our justification for still leaving out many implementation details that may affect the correctness of the protocol, is that we intend to reuse as much as possible of the proofs already established. This can only be done in a manageable way if we do not add too many details at once. As it is, the proofs for our verification are already quite lengthy and involved. See also Section 6 for a discussion of our results.

The abstractions have been chosen as follows.

- In TIP3, it is assumed that the devices signal a cycle by merely checking the network topology. In TIP4, the devices use a timer, which conforms to IEEE 1394.
- In both TIP3 and TIP4, communication between devices is modelled by sending messages on queues. In a IEEE 1394 network, the devices communicate by asserting signals (defined in terms of voltages) on wires for a certain time.
- In both TIP3 and TIP4, it is assumed that no device has the force_root flag set to true.
- In both TIP3 and TIP4, the network is assumed to be connected and to be fixed throughout the protocol. There may be cycles in the topology.
- In both TIP3 and TIP4, the root contention situation is solved in one atomic step, as opposed to the IEEE 1394 protocol which involves picking random bits, and which repeats until the symmetry is broken. Note that the root contention protocol has been formally specified and proved correct in [28].
- In both TIP3 and TIP4, all devices enter the tree identify phase at the same time.
- In TIP3, no timing is used whatsoever. In TIP4, timing is used for determining whether the network topology contains a cycle (see above), and for determining the actual delivery time of messages. The IEEE 1394 delay between the moment of sending and reception and processing of a signal is caused by difference in clocks of the devices, the length and propagation delay of the wires, and the difference in the tree identify phase enter moment of the sending and receiving device. In TIP4, the delay of message is determined at the moment that the message is being received. This delay may vary between the bounds caused by difference in clocks of the devices, and by the length and propagation delay of the wires. Although the second factor is constant, we have modelled the choice of delay to be completely free for each receive operation. Since we are after the bounds on the timing constants in relation to the network topology with respect to detecting cycles, we are establishing the property that the cycle detection timer will not expire too soon or too late. Therefore we are actually performing a worst case analysis. The worst case scenarios for IEEE 1394 and our model are the same, under the assumption that all devices enter the tree identify phase at the same moment.

3 IOA models

We present two models in the IOA language [5] of the tree identify protocol, namely TIP3 and TIP4. The IOA model for TIP3 comes (almost) literally from [7] and gives an abstract and untimed model of the protocol behaviour. It has been shown in [7] that this model has the desired behaviour of electing exactly one device for root if there is no cycle in the network. If

```
signature
  internal childrenknown(d:  Dev),
           addchild(d:Dev, p:Port),
           receivemes(d:Dev, p:Port, m:Mes),
           solverootcontent(d:Dev, p:Port)
  output   root(d:Dev),
           loopdetect(d:Dev)
  time     δ
```

Figure 6: Signature for TIP3 and TIP4.

there is a cycle in the network, all devices that are part of this cycle will detect this and give an error message.

The IOA language The IOA language facilitates precise and readable descriptions of I/O automata [16, 17]. Since our models are timed, we have added a **time** action, according to the definition in [19].

IOA contains the basic types `Bool`, `Nat`, `Int` and `Real` with their standard operators. In addition type constructors `Array`, `Seq` (finite sequences) and `Set` (finite sets) are part of the language. The notation `__[__]` is used for array subscripting, an array with a value `e` in all cells is denoted by `const(e)`. The operation `__⊢__` appends an element at the end of a sequence and the operations `head` and `tail` have the usual meaning. We assume the type `Time` which is the (predefined) type `Real` restricted to nonnegative values.

We assume the extra types `Mes` to represent the different message contents that may be exchanged between devices, as follows:

Type `Mes` enumeration of `parent`, `ack`

In Section 4 we give several definitions and operations that concern network topologies. Given a network $N = \langle D, P, \texttt{dev}, \texttt{peer}\rangle$, we assume the types `Dev`=D and `Port`=P and all operations as defined in Section 4.

The TIP models The signature part for both models is shown in Figure 6. The connected network $N = \langle D, P, \texttt{dev}, \texttt{peer}\rangle$ is a parameter for both models. In addition, the constants `MinDelay`, `MaxDelay`, `MinLpdtime`, and `MaxLpdtime` are parameters for `TIP4`. We assume `MinDelay` $\leq$ `MaxDelay` and `MinLpdtime` $\leq$ `MaxLpdtime`.

The IOA description of `TIP3` is shown in Figure 7. The action definitions are almost equal to those of `TIP4`, so we refer to the explanation below. The model `TIP3` comes (almost) literally from [7]. The first change is the addition of the time action, whose precondition is true, and whose effect is empty. The second change is the use of the `oncycle` predicate, which recognizes not just devices that are on an ordinary cycle, but also devices that are on a path between two cycles (see Section 4). Our verification shows that these devices also detect a cycle in the protocol and give an error message (see property I_{12} in Definition 5.1, Section 5.1).

The IOA description of `TIP4` is shown in Figure 8. The model `TIP4` is a proper timed IOA model: there is a state variable `time` which is used as a global clock, and per message queue

```
automaton TIP3
  states
    child:  Set[Port] := {}
    mq:  Array[Port,Seq[Mes]] := const({})
    init:  Array[Dev,Bool] := const(true)
    rc, root, lpd:  Array[Dev,Bool] := const(false)
  transitions
    internal childrenknown(d)
      pre init[d] ∧ size(ports(d)-child) ≤ 1
      eff  init[d] := false;
           for p in ports(d) do if p ∈ child
                                then mq[p] := mq[p] ⊢ ack
                                else mq[p] := mq[p] ⊢ parent fi od
    internal addchild(d,p) where d = dev(p)
      pre init[d] ∧ head(mq[peer(p)]) = parent
      eff  child := insert(p, child);
           mq[peer(p)] := tail(mq[peer(p)])
    internal receivemes(d,p,m) where d = dev(p)
      pre ¬ init[d] ∧ ports(d)-child = {p} ∧ head(mq[peer(p)]) = m
      eff  if m = parent then rc[d] := true fi;
           mq[peer(p)] := tail(mq[peer(p)])
    internal solverootcontent(d,p) where d = dev(p)
      pre rc[d] ∧ rc[dev(peer(p))]
      eff  child := insert(p,child);
           rc[d] := false;
           rc[dev(peer(p))] := false
    output root(d)
      pre ¬ init[d] ∧ ¬ root[d] ∧ ports(d) ⊆ child
      eff  root[d] := true
    output loopdetect(d)
      pre oncycle(d) ∧ ¬ lpd[d]
      eff  lpd[d] := true
    time δ where δ > 0
      pre true
      eff
```

Figure 7: Automaton TIP3.

```
automaton TIP4
  states
    child:  Set[Port] := {}
    mq:  Array[Port,Seq[Mes]] := const({})
    delay:  Array[Port,Time] := const(0)
    init:  Array[Dev,Bool] := const(true)
    rc, root, lpd:  Array[Dev,Bool] := const(false)
    time:  Time := 0
  transitions
    internal childrenknown(d)
      pre init[d] ∧ size(ports(d)-child) ≤ 1
      eff  init[d] := false;
           for p in ports(d) do delay[p] := 0;
                                if p ∈ child
                                then mq[p] := mq[p] ⊢ ack
                                else mq[p] := mq[p] ⊢ parent fi od
    internal addchild(d,p) where d = dev(p)
      pre init[d] ∧ head(mq[peer(p)])=parent ∧ delay[peer(p)] ≥ Mindelay
      eff  child := insert(p, child); mq[peer(p)] := tail(mq[peer(p)])
    internal receivemes(d,p,m) where d = dev(p)
      pre ∧ ¬ init[d] ∧ ports(d)-child = {p}
          ∧ head(mq[peer(p)])=m ∧ delay[peer(p)] ≥ Mindelay
      eff  if m = parent then rc[d] := true fi;
           mq[peer(p)] := tail(mq[peer(p)])
    internal solverootcontent(d,p) where d = dev(p)
      pre rc[d] ∧ rc[dev(peer(p))]
      eff  child := insert(p,child);
           rc[d] := false; rc[dev(peer(p))] := false
    output root(d)
      pre ¬ init[d] ∧ ¬ root[d] ∧ ports(d) ⊆ child
      eff  root[d] := true
    output loopdetect(d)
      pre init[d] ∧ ¬ lpd[d] ∧ time ≥ MinLpdtime
      eff  lpd[d] := true
    time δ where δ > 0
      pre ∀ d,p:
          ∧ ¬ pre(childrenknown(d)) ∧ ¬ pre(root(d))
          ∧ if init[d] ∧ ¬ lpd[d] then time+δ ≤ MaxLpdtime fi
          ∧ if mq[p]≠{} then delay(mq[p])+δ ≤ MaxDelay  fi
      eff  time := time+δ
           for p in Port do delay[p] := delay[p]+δ od
```

Figure 8: Automaton TIP4.

there is a variable delay that is reset for each message sent on the corresponding queue. A message is available at least after MinDelay time units have passed or ultimately after MaxDelay time units have passed. The condition for detecting a cycle in the network also depends on time, and not (as in TIP3) on the predicate oncycle which is based on the structure of the network. It is our goal to show that cycle detection will occur if and only if there really is a cycle present in the network.

We now give a short explanation of each action of TIP4. Whether a device is in the initial phase is reflected in the state variable init. When init is true, only actions addchild and childrenknown can be enabled. With addchild a parent request may be received (if the value of delay indicates that the parent request is available) and the corresponding port is added to the collection of children. The action childrenknown marks the end of the init phase. It can only be performed when there is at most one port left which is not a child port, and when it is performed, an acknowledgement is sent to all peer devices that are connected to a child port and a parent request is sent to the peer device connected to the port that is not a child, if any. If a device is on a cycle, then it does not ever reach the state in which childrenknown is enabled, because two of its ports are connected to peer devices which are also on a cycle. In this situation, the action loopdetect should be performed. In TIP3, the cycle is detected with the oncycle predicate. In TIP4, a timer signals that the device stays in the init phase too long, and therefore must be on a cycle. As soon as a device has left the init phase, it must wait for a message on the one remaining port that is not a child. If there is no such port, then the device is the root of the tree, and can perform the root action. If there is such a port, then the action receivemes can be performed as soon as the message is available. The expected message is an acknowledgement, after which the contribution of the device to the leader election is over. If an unexpected parent request is received, then the device is in root contention with the peer device that sent the parent request. The peer device has received or will receive the parent request that was sent earlier, and thus has signalled or will signal the root contention. As soon as both devices have signalled root contention, the action solverootcontent can be performed to break the symmetry and add one of the two ports involved to the child collection. The device whose port is added to child can then perform the root action. The time action signals the passing of time, by increasing the value of time. Time passage may not occur if there are other actions that cannot be delayed any further. Actions childrenknown and root are urgent, which means that they should happen at the first moment when they are enabled. Actions addchild and receivemes are also urgent, but they are enabled only when a message becomes available. Since the message is available only when the value of delay is in the interval [MinDelay, MaxDelay], we require that the value of delay does not pass beyond the right-hand border of this interval. The action loopdetect depends on the the value of time and can happen anywhere in the interval [MinLpdtime, MaxLpdtime], so we require that time does not pass beyond MaxLpdtime. The only action that is not mentioned in the precondition of the time action, is solverootcontent. The reason for this is that in the IEEE 1394 documentation, there is a small sub-protocol with timers that is used to break the symmetry, instead of the one action that represents this sub-protocol in TIP4. Since this sub-protocol is not guaranteed to end in finite time (due to randomly drawn bits), we cannot say at what time the action solverootcontent will take place. Hence we have put not requirement on the time action for solverootcontent. The root contention solving protocol is discussed and proved correct in [28].

4 Network preliminaries

This section gives some definitions and properties of network topologies which are needed in the verification.

4.1 Networks

Definition 4.1 A *network* is a quintuple $\langle D, P, \texttt{dev}, \texttt{peer}\rangle$, where

- D is a non-empty set of devices.
- P is a set of ports.
- $\texttt{dev} : P \to D$.
- $\texttt{peer} : P \to P$ with for all p: $\texttt{peer}(\texttt{peer}(p)) = p$ and $p \neq \texttt{peer}(p)$.

For $d \in D$, we define the abbreviation $\texttt{ports}(d) = \{p \in P | \texttt{dev}(p) = d\}$.
Given D' and $d \in D'$, the predicate $\texttt{leaf}(D', d)$ holds iff $\forall p_1, p_2 \in \texttt{ports}(d) : \texttt{dev}(\texttt{peer}(p_1)) \in D' \wedge \texttt{dev}(\texttt{peer}(p_2)) \in D' \to p_1 = p_2$.

The network consists of a collection of devices, each of which has a set of ports. Each port is connected to one other port with a cable, which is captured by the function `peer`. Each port has a connection and no port is connected to itself. The cable connection itself is referred to as a *cable hop*. Since for each $p \in P$, $\texttt{dev}(p)$ is defined, it follows that $P = \bigcup_{d \in D} \texttt{ports}(d)$.

Throughout this paper, we fix a network $N = \langle D, P, \texttt{dev}, \texttt{peer}\rangle$ and let variables $p, p', p'', p_0, \ldots$ range over ports in P, and $d, d', d_0, \ldots$ over devices in D.

4.2 Paths, cycles

The following definitions and lemmas are necessary to identify paths, cycles, etc. in the network.

Definition 4.2 A *path* π is a non-empty sequence of ports $\pi = p_0 p_1 \ldots p_n$, such that:

- n is odd
- $p_0 \neq p_n$
- for all $i > 0$, if i is odd then $p_{i-1} = \texttt{peer}(p_i)$ else $\texttt{dev}(p_{i-1}) = \texttt{dev}(p_i)$)

We denote the first and last port of π with $\texttt{first}(\pi) = p_0$ and $\texttt{last}(\pi) = p_n$. We denote the length of π with $\texttt{length}(\pi) = (n+1)/2$. We denote the path obtained by reversing π with $\texttt{reverse}(\pi) = p_n \ldots p_0$.
Path π is a path *from* d_1 *to* d_2 if $\texttt{dev}(\texttt{first}(\pi)) = d_1$ and $\texttt{dev}(\texttt{last}(\pi)) = d_2$. We say that a device d *is on* π iff there is a port p in π such that $d = \texttt{dev}(p)$. A *cycle* is a path $\pi = p_0 \ldots p_n$ such that $\texttt{dev}(p_0) = \texttt{dev}(p_n)$.
The predicate $\texttt{oncycle}(p)$ is true iff there is a cycle such that p is on it. The predicate $\texttt{oncycle}(d)$ is true iff there is a port $p \in \texttt{ports}(d)$, such that $\texttt{oncycle}(p)$ holds.

A path reflects a walk through the network by the concatenation of cable hops, in which a $p_1 p_2$ cable hop may not be followed immediately by the reverse hop $p_2 p_1$. The length of a path is the number of cable hops included in that path. A cycle may include a path π which is wedged in between smaller cycles ρ and τ, resulting in the shape of a pair of glasses: $\rho\pi\tau\texttt{reverse}(\pi)$.

Ports that are part of a cycle (of whatever shape) remain inactive during the protocol, as we will show later.

Lemma 4.3 If $\pi = p_0 p_1 p_2 \dots p_n$ is a cycle, then $p_2 \dots p_n p_0 p_1$ and $\texttt{reverse}(\pi)$ are also cycles.

Lemma 4.4 $\texttt{oncycle}(p) \rightarrow \texttt{oncycle}(\texttt{peer}(p))$

Lemma 4.5 $\texttt{oncycle}(p) \rightarrow \texttt{size}(\{p' | p' \in \texttt{ports}(\texttt{dev}(p)) \wedge \texttt{oncycle}(p')\}) \geq 2$

Proof Let $\pi = p_0 p_1 \dots p_n$ be a cycle such that $p = p_i$.
If $i = 0$, then by definition of a cycle, $\texttt{dev}(p_n) = \texttt{dev}(p)$, and by definition of a path, $p_n \neq p$.
If i is even and $i > 0$, then by definition of a path, $\texttt{dev}(p_{i-1}) - \texttt{dev}(p)$ and $p_{i-1} \neq p$.
If $i = n$, then by definition of a cycle, $\texttt{dev}(p_0) = \texttt{dev}(p)$, and by definition of a path, $p_0 \neq p$.
If i is odd and $i < n$, then by definition of a path, $\texttt{dev}(p_{i+1}) - \texttt{dev}(p)$ and $p_{i+1} \neq p$. ⊠

Lemma 4.6 Let $N = \langle D, P, \texttt{dev}, \texttt{peer} \rangle$ be a connected network, and $d_1, d_2 \in D$.
If $\texttt{oncycle}(d_1)$, $\texttt{oncycle}(d_2)$, and π is a path from d_1 to d_2, then for each $p \in \pi : \texttt{oncycle}(p)$

Proof Let ρ be a cycle such that d_1 is on it. Let τ be a cycle such that d_2 is on it. We will show for each port p in π that $\texttt{oncycle}(p)$, as follows.

Let $\pi = p_0 \dots p_n$. If p_i is in ρ or τ, then $\texttt{oncycle}(p_i)$. We take a fragment $\pi' = p_i \dots p_j$ from π such that $i > 0$ implies that p_{i-1} in ρ or in τ, and $j < n$ implies that p_{j+1} in ρ or in τ, and for each port p on π', p is not on ρ and not on τ. If we can construct a cycle such that the fragment π' is part of it, we are done.

Note that by definition, $\texttt{dev}(p_i)$ is on ρ or on τ, and $\texttt{dev}(p_j)$ is on ρ or on τ.

In the following case distinction we leave out all cases which are symmetric to a case proved earlier.

1. $p_i = p_j$.
   We assume w.l.o.g. that $\texttt{dev}(p_i)$ is on ρ. Let $\rho = \rho_1 \rho_2$ such that $\texttt{dev}(p_i) = \texttt{dev}(\texttt{last}(\rho_1)) = \texttt{dev}(\texttt{first}(\rho_2))$ and $\texttt{length}(\rho_1)$ is even. By assumption, $\texttt{last}(\rho_1) \neq p_i = p_j \neq \texttt{first}(\rho_2)$. We construct the path $\rho_2 \rho_1 \pi'$ and see that it is a cycle.

2. $p_i \neq p_j$.
   We assume w.l.o.g. that $\texttt{dev}(p_i)$ is on ρ and $\texttt{dev}(p_j)$ is on τ. Let $\rho = \rho_1 \rho_2$ such that $\texttt{dev}(p_i) = \texttt{dev}(\texttt{last}(\rho_1)) = \texttt{dev}(\texttt{first}(\rho_2))$ and $\texttt{length}(\rho_1)$ is even. Let $\tau = \tau_1 \tau_2$ such that $\texttt{dev}(p_j) = \texttt{dev}(\texttt{last}(\tau_1)) = \texttt{dev}(\texttt{first}(\tau_2))$ and $\texttt{length}(\tau_1)$ is even. By assumption, $\texttt{last}(\rho_1) \neq p_i \neq \texttt{first}(\rho_2)$ and $\texttt{last}(\tau_1) \neq p_j \neq \texttt{first}(\tau_2)$. We construct the path $\rho_2 \rho_1 \pi' \tau_2 \tau_1 \texttt{reverse}(\pi')$ and see that it is a cycle.

⊠

4.3 Connected networks

The following definitions and lemmas are necessary to identify the distance of devices in the network to the edge of the network, that is, how many times we have to take all the leaf devices away before a device becomes a leaf in the remaining set. The distance measure defined here will be used in the protocol to quantify the worst-case time that it takes for a device to complete its part in the protocol.

Definition 4.7 N is *connected* if for each two devices $d, d' \in D$ there is a path from d to d'.
If N is connected, we denote the maximum length of the shortest path in N between any two

devices by $\texttt{MaxHop} = \texttt{max}(\{n | d_1, d_2 \in D \wedge n = \texttt{min}(\{\texttt{length}(\pi) | \pi \text{ is path from } d_1 \text{ to } d_2\})\})$.
The function `Steps` is defined by the following equation:

$$\texttt{Steps}(D', d) = \begin{cases} 0 & \textit{if } \texttt{leaf}(D', d) \textit{ or } \texttt{oncycle}(d) \\ 1 + \texttt{Steps}(D'', d) & \textit{otherwise} \end{cases}$$
$$\textit{where } D'' = D' - \{d' \in D' | \texttt{leaf}(D', d')\}$$

We abbreviate $\texttt{Steps}(d) = \texttt{Steps}(D, d)$.
The function `Shrink` is defined by the following equation:

$$\texttt{Shrink}(D', n') = \begin{cases} D' & \textit{if } n' = 0 \\ \texttt{Shrink}(D'', n' - 1) & \textit{otherwise} \end{cases}$$
$$\textit{where } D'' = D' - \{d' \in D' | \texttt{leaf}(D', d')\}$$

We abbreviate $\texttt{Shrink}(n) = \texttt{Shrink}(D, n)$.

The value `MaxHop`, which is an upper bound to the mininum number of cable hops between any two devices, is used in the IEEE documentation as a restriction on the networks on which the protocol is to operate.

The function `Steps` gives the one but greatest distance between a device and a leaf in the network. This number is determined by the number of steps it takes for such a device to become a leaf, when in each step all leafs are removed. For a device that is part of a cycle, the value of `Steps` has no meaning and will not be used.

The function `Shrink` gives the set of devices that remains when in each step the leaf devices are removed and this is repeated for the indicated number of times, starting with the given set. The correspondence between `Steps` and `Shrink` is obvious: if $\texttt{Steps}(d) = n$ then $\texttt{leaf}(\texttt{Shrink}(n), d)$ holds and if $\texttt{Steps}(d) \geq n$ then $d \in \texttt{Shrink}(n)$.

In the remainder of this paper, we assume that N is connected.

Lemma 4.8 Let $d \in D$ such that $\neg\texttt{oncycle}(d)$.
If $\texttt{Steps}(d) = n$ then $\texttt{size}(\{p' \in \texttt{ports}(d) | \texttt{oncycle}(\texttt{dev}(\texttt{peer}(p')) \vee \texttt{Steps}(\texttt{dev}(\texttt{peer}(p'))) \geq n\}) \leq 1$.

Proof By contradiction. Assume $\neg\texttt{oncycle}(d)$ and $\texttt{Steps}(d) = n$. Let $p, p' \in \texttt{ports}(d)$ such that $p \neq p'$ and $\texttt{oncycle}(\texttt{dev}(\texttt{peer}(p)) \vee \texttt{Steps}(\texttt{dev}(\texttt{peer}(p))) \geq n$ and $\texttt{oncycle}(\texttt{dev}(\texttt{peer}(p')) \vee \texttt{Steps}(\texttt{dev}(\texttt{peer}(p'))) \geq n$. Since $\texttt{Steps}(d) = n$, either $\texttt{oncycle}(d)$ or $\texttt{leaf}(\texttt{Shrink}(n), d)$. Since we assumed $\neg\texttt{oncycle}(d)$, apparently $\texttt{leaf}(\texttt{Shrink}(n), d)$. By our assumption $\texttt{dev}(\texttt{peer}(p))$ and $\texttt{dev}(\texttt{peer}(p'))$ are both in $\texttt{Shrink}(n)$. But $p \neq p'$, which contradicts $\texttt{leaf}(\texttt{Shrink}(n), d)$. We conclude that $\texttt{size}(\{p' \in \texttt{ports}(d) | \texttt{oncycle}(\texttt{dev}(\texttt{peer}(p')) \vee \texttt{Steps}(\texttt{dev}(\texttt{peer}(p'))) \geq n\}) \leq 1$. ⊠

Lemma 4.9 For each $d \in D$

$$\texttt{Steps}(d) \leq \begin{cases} \lfloor \texttt{MaxHop}/2 \rfloor & \textit{if } \forall d' \in D : \neg\texttt{oncycle}(d') \\ \texttt{max}(0, \texttt{MaxHop} - 1) & \textit{otherwise} \end{cases}$$

Proof By contradiction.

1. Suppose $\forall d \in D : \neg\texttt{oncycle}(d)$.
   Suppose $\texttt{Steps}(d) = m > \lfloor n/2 \rfloor$. We show that we can construct a shortest path π with $\texttt{length}(\pi) > n$, by starting with d and extending the path in each step with one cable hop

in two directions. We use induction on $n' \in \{1, \ldots, m\}$ in the following hypothesis:
There is a path $p_0 \ldots p_{4n'-1}$ with $\texttt{Steps}(\texttt{dev}(p_0)) \geq m - n'$ and $\texttt{Steps}(\texttt{dev}(p_{4n'-1})) \geq m - n'$ and there is no other path from $\texttt{dev}(p_0)$ to $\texttt{dev}(p_{4n'-1})$.

- (Base step) $n' = 1$
  Since $m > 0$, certainly $\neg\texttt{leaf}(\texttt{Shrink}(m-1), d)$, and since $\texttt{leaf}(\texttt{Shrink}(m), d)$, there must be $p, q \in \texttt{ports}(d)$ such that $p \neq q$ and $\texttt{dev}(\texttt{peer}(p))$ and $\texttt{dev}(\texttt{peer}(q))$ in $\texttt{Shrink}(m-1)$. Fix p, q. Clearly, $\texttt{Steps}(\texttt{dev}(\texttt{peer}(p))) \geq m-1$ and $\texttt{Steps}(\texttt{dev}(\texttt{peer}(q))) \geq m-1$. Consider $\texttt{peer}(p)pq\texttt{peer}(q)$. This is a path if $\texttt{peer}(p) \neq \texttt{peer}(q)$. Since $\neg\texttt{oncycle}(d')$ for all $d' \in D$, we see that $\texttt{dev}(\texttt{peer}(p)) \neq \texttt{dev}(\texttt{peer}(q))$, so $\texttt{peer}(p) \neq \texttt{peer}(q)$. If was another path from $\texttt{dev}(\texttt{peer}(p))$ to $\texttt{dev}(\texttt{peer}(q))$ then this would contradict the assumption that $\neg\texttt{oncycle}(d')$ for all $d' \in D$. We conclude that $\pi = \texttt{peer}(p)pq\texttt{peer}(q)$ is a path that meets the requirements.
- (Induction step) $n' = n'' + 1 \leq m$ and the hypothesis holds for n''
  Let $\pi = p_0 \ldots p_{4n''-1}$ such that $\texttt{Steps}(\texttt{dev}(p_0)) \geq m - n''$ and $\texttt{Steps}(\texttt{dev}(p_{4n''-1})) \geq m - n''$ and there is no other path from $\texttt{dev}(p_0)$ to $\texttt{dev}(p_{4n''-1})$. We abbreviate $d_1 = \texttt{dev}(p_0)$ and $d_2 = \texttt{dev}(p_{4n''-1})$ for the first and last device of π. Since $n'' < m$, $\texttt{Steps}(d_1) > 0$ and $\texttt{Steps}(d_2) > 0$. So $\neg\texttt{leaf}(\texttt{Shrink}(m - n'' - 1), d_1)$ and $\neg\texttt{leaf}(\texttt{Shrink}(m - n'' - 1), d_2)$. So there must be $p, p' \in \texttt{ports}(d_1)$ and $q, q' \in \texttt{ports}(d_2)$ such that $p \neq p'$, $q \neq q'$, and $\texttt{dev}(\texttt{peer}(p))$, $\texttt{dev}(\texttt{peer}(p'))$ $\texttt{dev}(\texttt{peer}(q))$, and $\texttt{dev}(\texttt{peer}(q'))$ in $\texttt{Shrink}(m-n''-1)$. Fix p, p', q and q'. Now for $x \in \{p, p', q, q'\}$: $\texttt{Steps}(\texttt{dev}(\texttt{peer}(x))) \geq m - n'' - 1 = m - (n'' + 1) = m - n'$ We assume without loss of generality that $p \neq p_0$ and $q \neq p_{4n''-1}$. Consider $\texttt{peer}(p)p\pi q\texttt{peer}(q)$. This is a path if $\texttt{peer}(p) \neq \texttt{peer}(q)$. Since $\neg\texttt{oncycle}(d')$ for any $d' \in D$, we see that $\texttt{dev}(\texttt{peer}(p)) \neq \texttt{dev}(\texttt{peer}(q))$, so $\texttt{peer}(p) \neq \texttt{peer}(q)$. If there was another path from $\texttt{dev}(\texttt{peer}(p))$ to $\texttt{dev}(\texttt{peer}(q))$ then this would contradict the assumption that $\neg\texttt{oncycle}(d')$ for all $d' \in D$. We conclude that $\texttt{peer}(p)p\pi q\texttt{peer}(q)$ is a path that meets all the requirements for the induction step.

We conclude that there is a shortest path in the network of length $2m$. Since $m > \lfloor n/2 \rfloor$, certainly $2m > (2\lfloor n/2 \rfloor) + 1 \geq n$. So $2m > n$ and we have a contradiction.

2. Suppose $\exists d' \in D : \texttt{oncycle}(d')$.
   Suppose $\texttt{Steps}(d) = m > \texttt{max}(0, n-1)$. Then $m > 0$ and by definition of $\texttt{Steps}$, certainly $\neg\texttt{oncycle}(d)$. We show that we can construct a path π with $\texttt{length}(\pi) > n$, by starting with d and a neighbour of d on a cycle, and extending the path in each step with one cable hop to a neighbour which is not on a cycle. We use induction on $n' \in \{0, \ldots, m\}$ in the following hypothesis:
   There is a path $p_0 p_1 \ldots p_{2n'+1}$ with $\texttt{oncycle}(\texttt{dev}(p_0))$, $\texttt{Steps}(\texttt{dev}(p_{2n'+1})) \geq m - n'$ and for all $1 \leq i \leq 2n' + 1$: $\neg\texttt{oncycle}(\texttt{dev}(p_i))$, and there is no shorter path from $\texttt{dev}(p_0)$ to $\texttt{dev}(p_{2n'+1})$.

   - (Base step) $n' = 0$
     Suppose there is no $p \in \texttt{ports}(d)$ such that $\texttt{oncycle}(\texttt{dev}(\texttt{peer}(p)))$. Since N is connected, there must be π, d' such that π is a path $\pi = p_0 \ldots p_n$ from d' to d with $\texttt{oncycle}(d')$ and for each $i > 0$: $\neg\texttt{oncycle}(\texttt{dev}(p_i))$. Fix d', π. Since $\texttt{oncycle}(d')$ and $\neg\texttt{oncycle}(\texttt{dev}(p_1))$, we can use Lemma 4.8 to conclude that $\texttt{Steps}(\texttt{dev}(p_1)) > \texttt{max}(\{\texttt{Steps}(\texttt{dev}(p') | p' \in \texttt{ports}(\texttt{dev}(p_1)) \wedge p' \neq p_1\})$. Then it is not hard to show

(using induction and Lemma 4.8) that $\forall i \in \{1,3,\ldots,n-2\} : \texttt{Steps}(\texttt{dev}(p_i)) \geq \texttt{Steps}(\texttt{dev}(p_{i+2}))$. Then we easily have $\forall i \in \{1,3,\ldots,n\} : \texttt{Steps}(\texttt{dev}(p_i)) \geq m$. Since d is chosen arbitrarily with $\texttt{Steps}(d) > n-1$, any of the devices on π except $\texttt{dev}(p_0)$ would do. So we assume without loss of generality that there is a $p \in \texttt{ports}(d)$ such that $\texttt{oncycle}(\texttt{dev}(\texttt{peer}(p)))$. Fix p.

We now have $\texttt{Steps}(d) \geq m$, $\neg\texttt{oncycle}(d)$, and $\texttt{oncycle}(\texttt{dev}(\texttt{peer}(p))$. We see that $\texttt{peer}(p)p$ is a path that meets the requirements, since there cannot be a shorter path from $\texttt{dev}(\texttt{peer}(p))$ to d.

- (Induction step) $n' = n'' + 1 \leq m$ and the hypothesis holds for n''
  Let $\pi = p_0 \ldots p_{2n''+1}$ such that $\texttt{oncycle}(\texttt{dev}(p_0))$, $\texttt{Steps}(\texttt{dev}(p_{2n''+1})) \geq m - n''$ and for all $1 \leq i \leq 2n'' + 1$: $\neg\texttt{oncycle}(\texttt{dev}(p_i)$, and there is no shorter path from $\texttt{dev}(p_0)$ to $\texttt{dev}(p_{2n''+1})$. We abbreviate $d' = \texttt{dev}(p_{2n''+1})$ to indicate the last device in π. Since $n'' < m$, $\texttt{Steps}(d') > 0$. So $\neg\texttt{leaf}(\texttt{Shrink}(m - n'' - 1), d$. So there must be $p, p' \in \texttt{ports}(d')$ such that $p \neq p'$, and $\texttt{dev}(\texttt{peer}(p))$ and $\texttt{dev}(\texttt{peer}(p'))$ in $\texttt{Shrink}(m - n'' - 1)$. Fix p, p'. Now for $x \in \{p, p'\} : \texttt{Steps}(\texttt{dev}(\texttt{peer}(x))) \geq m - n'' - 1 = m - (n'' + 1) = m - n'$. We assume without loss of generality that $p \neq p_{2n''+1}$.
  Consider $\pi p\texttt{peer}(p)$. This is a path if $\texttt{peer}(p) \neq p_0$. Suppose that $\texttt{peer}(p) = p_0$. Then $p = p_1$ and $\texttt{dev}(p_1) = d_1$, hence $p_2 \ldots p_n$ is a cycle, which contradicts our assumption. We conclude that $\texttt{peer}(p) \neq p_0$ and $\pi p\texttt{peer}(p)$ is a path.
  Suppose that $\texttt{oncycle}(\texttt{dev}(\texttt{peer}(p)))$. Then by Lemma 4.6 we have that for all $1 \leq i \leq 2n''+1$, $\texttt{oncycle}(\texttt{dev}(p_i))$, which contradicts our assumption. So we conclude that $\neg\texttt{oncycle}(\texttt{dev}(\texttt{peer}(p)))$.
  Suppose a shorter path than $\pi p\texttt{peer}(p)$ exists from $\texttt{dev}(p_0)$ to $\texttt{dev}(\texttt{peer}(p))$. This enables us to conclude that $\texttt{oncycle}(\texttt{dev}(\texttt{peer}(p_i)))$ with $p_i \in \pi$, which contradicts our assumptions. So we conclude that no shorter path than $\pi p\texttt{peer}(p)$ exists from $\texttt{dev}(p_0)$ to $\texttt{dev}(\texttt{peer}(p))$.
  We see that the path $\pi p\texttt{peer}(p)$ meets all the requirements for the induction step.

So there is a shortest path in the network of length $m + 1$. Since $m > n - 1$, $m + 1 > n$ and we have a contradiction.

⊠

5 Verification

In this section we prove that the IEEE 1394 tree identify protocol is correct relative to our model. In Section 5.1 some properties are given which have been proved invariant for the model TIP3in [7]. Some additional properties are given, which are be proved invariant for the model TIP4, provided that the invariants for TIP3 are also invariant for TIP4. This provision is solved in the next section, Section 5.2, in which it is proved that under certain timing restrictions the behaviour of TIP4 is included in the behaviour of TIP3. The proofs in Section 5.2 allow us to conclude that the safety aspects of cycle detection and root election in TIP4 meet the IEEE 1394 requirements. In Section 5.3 we prove some liveness properties for TIP4. Finally, in Section 5.4 we discuss whether the IEEE 1394 timing constants obey the restrictions that we found in Section 5.2.

The proofs in Sections 5.1 and 5.2 use simulation techniques from [19] which are listed in Appendix A. These appendices also present a new result for using invariants in stepwise refinement, which is useful for this verification because it allows us to reuse invariants properties from [7] without extra effort. In Appendix A.2, some new sufficient conditions for feasibility can be found. These lessen the proof burden when proving that there are no time deadlocks in the model.

Throughout this section we fix a connected network $N = \langle D, P, \mathtt{dev}, \mathtt{peer}\rangle$ as the parameter for TIP4. We let s,t range over states of TIP4, δ over Time, and m over Mes.

5.1 Invariants for TIP3 and TIP4

We first define the properties, of which some are taken from the PVS code used to check the proofs for [7]. All of the following properties are necessary for the proofs in Sections 5.2 and 5.3.

Definition 5.1 $I_1(d) \triangleq \mathtt{init}[d] \rightarrow \neg\mathtt{rc}[d]$

$I_2(p) \triangleq \mathtt{init}[\mathtt{dev}(p)] \rightarrow \mathtt{mq}[p] = \{\}$

$I_3(p) \triangleq \mathtt{init}[\mathtt{dev}(p)] \rightarrow \mathtt{peer}(p) \notin \mathtt{child}$

$I_4(d) \triangleq \mathtt{init}[d] \vee \mathtt{size}(\mathtt{ports}(d) - \mathtt{child}) \leq 1$

$I_5(p) \triangleq \mathtt{length}(\mathtt{mq}[p]) \leq 1$

$I_6(p) \triangleq p \in \mathtt{child} \rightarrow \mathtt{mq}[\mathtt{peer}(p)] = \{\}$

$I_7(p) \triangleq \mathtt{rc}[\mathtt{dev}(p)] \rightarrow \mathtt{mq}[\mathtt{peer}(p)] = \{\}$

$I_8(p) \triangleq \mathtt{rc}[\mathtt{dev}(p)] \rightarrow \mathtt{peer}(p) \notin \mathtt{child}$

$$\begin{aligned} I_9(p) \triangleq {} & \vee\ \mathtt{init}[\mathtt{dev}(p)] \\ & \vee\ \mathtt{head}(\mathtt{mq}[p]) = \mathtt{parent} \\ & \vee\ \mathtt{peer}(p) \in \mathtt{child} \\ & \vee\ \mathtt{rc}[\mathtt{dev}(\mathtt{peer}(p))] \\ & \vee\ p \in \mathtt{child} \end{aligned}$$

$I_{10}(p) \triangleq \mathtt{mq}[p] \neq \{\} \rightarrow \mathtt{delay}[p] \leq \mathtt{MaxDelay}$

$$\begin{aligned} I_{11}(d) \triangleq {} & \wedge\ \neg\mathtt{oncycle}(d) \wedge \mathtt{init}[d] \rightarrow \mathtt{time} \leq \mathtt{Steps}(d) * \mathtt{MaxDelay} \\ & \wedge\ \neg\mathtt{oncycle}(d) \wedge \mathtt{time} > \mathtt{Steps}(d) * \mathtt{MaxDelay} \\ & \quad \rightarrow \forall p' \in \mathtt{ports}(d) : \\ & \qquad \mathtt{head}(\mathtt{mq}[p']) = \mathtt{parent} \\ & \qquad \rightarrow \mathtt{time} - \mathtt{delay}[p'] \leq \mathtt{Steps}(d) * \mathtt{MaxDelay} \end{aligned}$$

$$\begin{aligned} I_{12}(d) \triangleq {} & \wedge\ \mathtt{MinLpdtime} > \mathtt{max}(0, \mathtt{MaxHop} - 1) * \mathtt{MaxDelay} \\ & \wedge\ \mathtt{init}[d] \\ & \wedge\ \neg\mathtt{oncycle}(d) \\ & \rightarrow \mathtt{time} < \mathtt{MinLpdtime} \end{aligned}$$

$I_{13}(d) \triangleq \mathtt{oncycle}(d) \rightarrow \mathtt{init}[d]$

$I_{14}(d) \triangleq \texttt{oncycle}(p) \wedge \neg \texttt{lpd}[d] \rightarrow \texttt{time} \leq \texttt{MaxLpdtime}$

We let $I_1 \triangleq \bigwedge_d I_1(d)$, $I_2 \triangleq \bigwedge_p I_2(p)$, et cetera.

Some of the properties in Definition 5.1 have been taken from [7], from which we also repeat the following result.

Lemma 5.2 Properties I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8, and I_9 are invariant for TIP3.

Even though the predicate oncycle has a different meaning in [7], we can assume that the proofs [7] still hold here, since the oncycle predicate is not used in the proofs.

Now we prove that under the assumption that some of the properties from Definition 5.1 hold in each reachable state for TIP4, it follows that others hold in each reachable state for TIP4 as well. In Section 5.2, the assumptions will be fulfilled by the corresponding properties for TIP3.

Lemma 5.3

1. I_{10} is inductive relative to $(I_2 \wedge I_5)$ for TIP4.
2. I_{11} is inductive relative to $(I_1 \wedge I_3 \wedge I_5 \wedge I_9)$ for TIP4.
3. For each $s \in \mathit{reachable}(\texttt{TIP4})$, $s \models I_{11}$ implies $s \models I_{12}$.
4. I_{13} is inductive relative to I_3 for TIP4.
5. I_{14} is inductive relative to I_{13} for TIP4.

Proof

1. Trivial.
2. Suppose $\neg\texttt{oncycle}(d)$.
   Initially, $s.\texttt{time} = 0$ and $\forall p : s.\texttt{mq}[p] = \{\}$. Since $\texttt{Steps}(d) \geq 0$, $\texttt{Steps}(d) * \texttt{MaxDelay} \geq 0 = s.\texttt{time}$. Since $\forall p \in \texttt{ports}(d) : s.\texttt{mq}[p] = \{\}$, it follows that $s \models I_{11}$.

   Suppose from $s \xrightarrow{a} t$ and $s \models I_1 \wedge I_3 \wedge I_5 \wedge I_9 \wedge I_{11}$. We have to show that $t \models I_{11}$. Fix n such that $n * \texttt{MaxDelay} \leq s.\texttt{time} < (n+1) * \texttt{MaxDelay}$.

   We make the following case distinction.

   (a) $s.\texttt{time} > \texttt{Steps}(d) * \texttt{MaxDelay}$
   By $s \models I_{11}$ we see that $\neg s.\texttt{init}[d]$. By the effect of a, we conclude that $\neg t.\texttt{init}[d]$. Now we just need to show for each $p' \in \texttt{ports}(d)$ that if $\texttt{head}(t.\texttt{mq}[p']) = \texttt{parent}$, then $t.\texttt{time} - t.\texttt{delay}[p'] \leq \texttt{Steps}(d) * \texttt{MaxDelay}$. Assume $p' \in \texttt{ports}(d)$ and $\texttt{head}(t.\texttt{mq}[p']) = \texttt{parent}$. By $\neg s.\texttt{init}[d]$, $s \models I_5$ and the precondition and effect of a, $\texttt{head}(s.\texttt{mq}[p']) = \texttt{parent}$. Since $s \models I_{11}$, it follows that $s.\texttt{time} - s.\texttt{delay}[p'] \leq \texttt{Steps}(d) * \texttt{MaxDelay}$.

      i. $\forall \delta > 0 : a \neq \delta$
      Then by the effect of a, $t.\texttt{time} = s.\texttt{time}$, and by $\neg s.\texttt{init}[d]$ and the precondition and effect of a, $t.\texttt{delay}[p'] = s.\texttt{delay}[p']$. So $t.\texttt{time} - t.\texttt{delay}[p'] = s.\texttt{time} - s.\texttt{delay}[p'] \leq \texttt{Steps}(d) * \texttt{MaxDelay}$ and it follows that $t \models I_{11}(d)$.

      ii. $a = \delta$
      Then $t.\texttt{time} = s.\texttt{time} + \delta$, and $t.\texttt{delay}[p'] = s.\texttt{delay}[p'] + \delta$. So $t.\texttt{time} - t.\texttt{delay}[p'] = s.\texttt{time} + \delta - (s.\texttt{delay}[p'] + \delta) = s.\texttt{time} - s.\texttt{delay}[p'] \leq \texttt{Steps}(d) * \texttt{MaxDelay}$ and it follows that $t \models I_{11}(d)$.

   (b) $s.\texttt{time} \leq \texttt{Steps}(d) * \texttt{MaxDelay}$

i. $\neg s.\texttt{init}[d]$
By the effect of a, $\neg t.\texttt{init}[d]$. The remainder is equal to the proof for Step 2a.

ii. $s.\texttt{init}[d]$

A. $\forall \delta > 0 : a \neq \delta$
Then by the effect of a, $t.\texttt{time} = s.\texttt{time}$, so $t.\texttt{time} \leq \texttt{Steps}(d) * \texttt{MaxDelay}$, hence for each $p' \in \texttt{ports}(d)$, trivially $t.\texttt{time} - t.\texttt{delay}[p'] \leq \texttt{Steps}(d) * \texttt{MaxDelay}$ and it follows that $t \models I_{11}(d)$.

B. $a = \delta \wedge s.\texttt{time} + \delta \leq \texttt{Steps}(d) * \texttt{MaxDelay}$
Then for each $p' \in \texttt{ports}(d)$, trivially $t.\texttt{time} - t.\texttt{delay}[p'] \leq \texttt{Steps}(d) * \texttt{MaxDelay}$ and it follows that $t \models I_{11}(d)$.

C. $a = \delta \wedge s.\texttt{time} + \delta > \texttt{Steps}(d) * \texttt{MaxDelay}$
The effect of a leads to a violation of property I_{11}, so we have to show that our assumption on a leads to a contradiction.
First we prove by contradiction that for each d' with $\neg\texttt{oncycle}(d')$ and $\texttt{Steps}(d') < \texttt{Steps}(d)$, it follows that $\forall p' \in \texttt{ports}(d') : s.\texttt{mq}[p'] = \{\} \vee \texttt{head}(s.\texttt{mq}[p']) = \texttt{ack}$. Suppose $\neg\texttt{oncycle}(d')$, $\texttt{Steps}(d') < \texttt{Steps}(d)$ and $\texttt{head}(s.\texttt{mq}[p']) = \texttt{parent}$. By $s \models I_{11}$, we see that $s.\texttt{time} - s.\texttt{delay}[p'] \leq \texttt{Steps}(d') * \texttt{MaxDelay}$. Since $t.\texttt{time} - t.\texttt{delay}[p'] = s.\texttt{time} + \delta - (s.\texttt{delay}[p'] + \delta) = s.\texttt{time} - s.\texttt{delay}[p'] \leq \texttt{Steps}(d') * \texttt{MaxDelay}$, and since $s.\texttt{time} + \delta > \texttt{Steps}(d) * \texttt{MaxDelay} \geq (\texttt{Steps}(d') + 1) * \texttt{MaxDelay}$, we get $s.\texttt{delay}[p'] + \delta > \texttt{MaxDelay}$, which in contradiction with our assumption that s enables δ.
Now we prove by contradiction that for each d' with $\neg\texttt{oncycle}(d')$ and $\texttt{Steps}(d') \leq \texttt{Steps}(d)$, it follows that $\neg s.\texttt{init}[d']$. Fix d' such that $\neg\texttt{oncycle}(d')$, $s.\texttt{init}[d']$ and there is no d'' with $\texttt{Steps}(d'') < \texttt{Steps}(d')$ and $s.\texttt{init}[d'']$. Let $P' = \{p' \in \texttt{ports}(d') | \neg\texttt{oncycle}(\texttt{dev}(\texttt{peer}(p'))) \wedge \texttt{Steps}(\texttt{dev}(\texttt{peer}(p'))) \leq \texttt{Steps}(d') - 1\}$. By Lemma 4.8, $\texttt{size}(\texttt{ports}(d') - P') \leq 1$. Fix $p' \in P'$ and $d'' = \texttt{dev}(\texttt{peer}(p'))$. Note that $\texttt{Steps}(d'') < \texttt{Steps}(d') \leq \texttt{Steps}(d)$. By our assumption, $\neg s.\texttt{init}[d'']$. By $s.\texttt{init}[d']$ and $s \models I_3$, we see $\texttt{peer}(p') \notin s.\texttt{child}$. By $s.\texttt{init}[d']$ and $s \models I_1$, we see $\neg s.\texttt{rc}[d']$. Combining all of this with our observation that $s.\texttt{mq}[\texttt{peer}(p')] = \{\} \vee \texttt{head}(s.\texttt{mq}[\texttt{peer}(p')] = \texttt{ack}$ and $s \models I_9$, we get $p' \in s.\texttt{child}$. So $\texttt{size}(\texttt{ports}(d')) - s.\texttt{child} = 1$. Since $s.\texttt{init}[d']$, s enables $\texttt{childrenknown}(d')$ which is in contradiction with our assumption that s enables δ. We conclude that $\neg s.\texttt{init}[d']$.
From this observation, it trivially follows that $\neg s.\texttt{init}[d]$ which is in contradiction with our assumption. It follows that $a \neq \delta \vee s.\texttt{time} + \delta \leq \texttt{Steps}(d) * \texttt{MaxDelay}$.

3. Let $s \in \textit{reachable}(\texttt{TIP4})$ such that $s \models I_{11}$. Assume $\texttt{MinLpdtime} > \texttt{max}(0, \texttt{MaxHop} - 1) * \texttt{MaxDelay} \wedge s.\texttt{init}[d] \wedge \neg\texttt{oncycle}(d)$. By $s.\texttt{init}[d] \wedge \neg\texttt{oncycle}(d)$ and $s \models I_{11}$, $s.\texttt{time} \leq \texttt{Steps}(d) * \texttt{MaxDelay}$. Note that for each $n \geq 0$, $\lfloor n/2 \rfloor \leq \texttt{max}(0, n - 1)$. Combining this with Lemma 4.9, we get $\texttt{Steps}(d) \leq \texttt{max}(0, \texttt{MaxHop} - 1)$. So $s.\texttt{time} \leq \texttt{max}(0, \texttt{MaxHop} - 1) * \texttt{MaxDelay} < \texttt{MinLpdtime}$ and it follows that $s \models I_{12}$.

4. Suppose $\texttt{oncycle}(d)$.

Initially, $s.\texttt{init}[d]$, hence $s \models I_{13}$.

Suppose $s \xrightarrow{a} t$ and $s \models I_3 \wedge I_{13}$. By $\texttt{oncycle}(d)$ and $s \models I_{13}$, we see that $s.\texttt{init}[d]$. If $a \neq \texttt{childrenknown}(d)$ then $t.\texttt{init}[d] = s.\texttt{init}[d]$, so it suffices to show that $a \neq$

childrenknown(d). By Lemma 4.5 and oncycle(d), there must be $p_1, p_2 \in$ ports(d) such that $p_1 \neq p_2$ and oncycle(dev(peer(p_1))) and oncycle(dev(peer(p_2))). Since $s \models I_{13}$, we see that s.init[dev(peer(p_1))] and s.init[dev(peer(p_2))]. Since $s \models I_3$, we see that $p_1 \notin s$.child and $p_2 \notin s$.child. Since $p_1 \neq p_2$, we see that size(ports(d) $- s$.child ≥ 2, hence s does not enable childrenknown(d).

5. Trivial.

⊠

Note that by Items 2 and 3 it follows that I_{12} is inductive relative to $(I_1 \wedge I_3 \wedge I_5 \wedge I_9 \wedge I_{11})$ for TIP4.

5.2 TIP4 implements TIP3

We use the properties established in Section 5.1 to obtain that TIP4 implements TIP3. As an implementation relation we take inclusion of admissible timed traces. From Section 5.1, it follows that the behaviour of TIP4 meets these properties only when the parameters obey the following relation: MinLpdtime $>$ max$(0,$ MaxHop $- 1) *$ MaxDelay. Therefore, we assume throughout this section that this relation holds.

In order to obtain the implementation relation, we construct a function that is to be proved a weak timed refinement from TIP4 to TIP3. Given the complicated relations between the invariants for TIP3 and the properties for TIP4, we have been forced to either prove the properties for TIP4 that depend on invariants for TIP3 anew, or to prove the invariance of the properties for TIP4and the weak refinement in one proof, or to come up with a more elegant solution. The latter approach has given rise to some new sufficient conditions, which are presented in Appendix A.

To avoid confusion, all state variables from TIP3 are subscripted with $_3$, and all state variables from TIP4 are subscripted with $_4$. Since the action signatures are equal, we do not use these subscript on the action names.

Definition 5.4 The function ref from states of TIP4 to states of TIP3 is defined to be the identity function on state variables with the same name.

Lemma 5.5 Let $s \in$ *states*(TIP3). For all $I \in \{I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8, I_9\}$, ref$(s) \models I$ implies $s \models I$.

Proof Trivial. ⊠

Lemma 5.6 1. $s \in$ *Start*(TIP4) implies ref$(s) \in$ *Start*(TIP3).

2. $s \xrightarrow{a}_{\text{TIP4}} t$, $s \models I_{10} \wedge I_{11} \wedge I_{12} \wedge I_{13} \wedge I_{14}$ and ref$(s) \models I_1 \wedge I_2 \wedge I_3 \wedge I_4 \wedge I_5 \wedge I_6 \wedge I_7 \wedge I_8 \wedge I_9$ implies ref$(s) \xrightarrow{a}_{\text{TIP3}}$ ref(t).

Proof

1. Suppose $s \in$ *Start*(TIP4).

   Since the initial requirements are the same for every state variable in TIP3 as for the state variable with the same name in TIP4, and the state variables with the same name have the same value in s and in ref(s), ref$(s) \in$ *Start*(TIP3) follows from $s \in$ *Start*(TIP4).

2. Suppose $s \xrightarrow{a}_{\texttt{TIP4}} t$ $s \models I_{10} \wedge I_{11} \wedge I_{12} \wedge I_{13} \wedge I_{14}$ and $\texttt{ref}(s) \models I_1 \wedge I_2 \wedge I_3 \wedge I_4 \wedge I_5 \wedge I_6 \wedge I_7 \wedge I_8 \wedge I_9$. $s \in \mathit{reachable}(\texttt{TIP4})$ and $\texttt{ref}(s) \in \mathit{reachable}(\texttt{TIP3})$.

   Since for each a, the effect in TIP3 is equal to the effect in TIP4 on all state variables from TIP3, it follows that whenever $\texttt{ref}(s) \xrightarrow{a}_{\texttt{TIP3}} t'$, then $t' = \texttt{ref}(t)$.

   If $a \notin \bigcup_d \texttt{loopdetect}(d)$, then we see that the precondition of a in TIP4 trivially implies the precondition of a in TIP3, hence if $s \xrightarrow{a}_{\texttt{TIP4}}$, then $\texttt{ref}(s) \xrightarrow{a}_{\texttt{TIP3}}$. So we just need to show that if $a = \texttt{loopdetect}(a)$, then $\texttt{ref}(s) \xrightarrow{a}_{\texttt{TIP3}}$.

   Suppose $a = \texttt{loopdetect}(a)$. By precondition of a in TIP4, $\neg s.\texttt{lpd}_4[d]$ and $s.\texttt{time}_4 \geq \texttt{MinLpdtime}$. From $\neg s.\texttt{lpd}_4[d]$ we see $\neg \texttt{ref}(s).\texttt{lpd}_3[d]$. By $s.\texttt{time}_4 = \texttt{lpdtime}_4[d]$ and $s \models I_{12}$ we see that either $\neg s.\texttt{init}_4[d]$ or $\texttt{oncycle}(d)$. By precondition of a in TIP4 we see that $s.\texttt{init}_4[d]$, and we conclude that $\texttt{oncycle}(d)$. Hence $\texttt{ref}(s)$ enables a.

⊠

Corollary 5.7 I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8, I_9, I_{10}, I_{11}, I_{12}, I_{13} and I_{14} are invariant for TIP4.

Proof By Lemmas 5.2, 5.3, 5.5 and 5.6 we can use Lemma A.2. ⊠

Corollary 5.8 The function ref is a weak timed refinement from TIP4 to TIP3 with respect to $(I_{10} \wedge I_{11} \wedge I_{12} \wedge I_{13} \wedge I_{14})$ and $(I_1 \wedge I_2 \wedge I_3 \wedge I_4 \wedge I_5 \wedge I_6 \wedge I_7 \wedge I_8 \wedge I_9)$

Proof By Lemmas 5.2, 5.3, 5.5 and 5.6 we can use Lemma A.2. ⊠

The implementation relation now follows easily.

Theorem 5.9 *t-traces*(TIP4) $\subseteq$ *t-traces*(TIP3).

Proof Combine Corollary 5.8 with Theorem 6.14 from [19]. ⊠

5.3 Liveness results for TIP4

In this section we show some liveness results for model TIP4. As in Section 5.2, we assume that the parameters of TIP4 meet the following relation: $\texttt{MinLpdtime} > \texttt{max}(0, \texttt{MaxHop}-1)*\texttt{MaxDelay}$.

The liveness results are the following. We first show that TIP4 has no time deadlocks. For this, some new sufficient conditions are used, which are presented in Appendix A.2. Then we prove that when a cycle is present, it will be detected, and that otherwise a root will be elected. Notice that we cannot use results from TIP3, since notions as quiescence and fairness are not present at the timed level.

First we need to define a measure on reachable states, to indicate the number of discrete actions that must be performed before passing of time will be enabled again.

Definition 5.10 The function `Measure` gives a pair for each state s from `TIP4`, as follows:

$$\texttt{Measure}(s) = \langle I, C + R + M + L \rangle$$

where

$$\begin{aligned} I &= \texttt{size}(\{d | s.\texttt{init}[d]\}) \\ C &= \texttt{size}(P - s.\texttt{child}) \\ R &= \texttt{size}(\{d | \neg s.\texttt{root}[d]\}) \\ M &= \texttt{size}(\{p | s.\texttt{mq}[p] \neq \{\}\}) \\ L &= \texttt{size}(\{d | \neg s.\texttt{lpd}[d]\}) \end{aligned}$$

The ordering $\prec$ is the lexicographic ordering on pairs of naturals, based on the ordering $<$ on naturals.

Since $<$ is well-founded, $\prec$ is also well-founded.

Now we prove the properties that we need for deadlock freedom, namely that when no discrete action is enabled, then the passage of time is enabled, and that at every moment in time at most a finite amount of discrete activity can occur.

Lemma 5.11 For each $s \in \mathit{reachable}(\texttt{TIP4})$ the following holds:

1. s enables an action from $\mathit{acts}(\texttt{TIP4})$.
2. If $s \xrightarrow{a} t$ and $\forall \delta > 0 : a \neq \delta$, then $\texttt{Measure}(t) \prec \texttt{Measure}(s)$ otherwise $\texttt{Measure}(t) = \texttt{Measure}(s)$.

Proof

1. It suffices to show that if s does not enable a for all $a \in \mathit{acts}(\mathit{Tipvier}) - \bigcup_{\delta>0}\{\delta\}$, then s enables δ for some $\delta > 0$, which we prove by contradiction.

   Suppose that for all $a \in \mathit{acts}(\texttt{TIP4})$, s does not enable a. Apparently s does not enable any $\delta > 0$, so either $s.\texttt{time} = \texttt{MaxLpdtime}$ and $\exists d : s.\texttt{init}[d] \wedge \neg s.\texttt{lpd}[d]$, or $\exists p : s.\texttt{mq}[p] \neq \{\} \wedge s.\texttt{delay}([p]) \geq \texttt{MaxDelay}$.

   Suppose $s.\texttt{time} = \texttt{MaxLpdtime}$, $s.\texttt{init}[d]$ and $\neg s.\texttt{lpd}[d]$. Then s enables $\texttt{loopdetect}(d)$ and we have a contradiction.

   Suppose $s.\texttt{mq}[p] \neq \{\}$ and $s.\texttt{delay}[p] \geq \texttt{MaxDelay}$. Let $\texttt{head}(s.\texttt{mq}[p]) = m$. Since $\texttt{MaxDelay} \geq \texttt{MinDelay}$, $s.\texttt{delay}[p] \geq \texttt{MinDelay}$. Using Invariant I_2, we get $\neg s.\texttt{init}[\texttt{dev}(p)]$, and using Invariant I_6 we get $\texttt{peer}(p) \notin s.\texttt{child}$. Suppose $p \in s.\texttt{child}$. Using Invariant I_3 we get $\neg s.\texttt{init}[\texttt{dev}(\texttt{peer}(p))]$. Then s enables $\texttt{receivemes}(\texttt{dev}(\texttt{peer}(p)), \texttt{peer}(p), m)$ and we have a contradiction. So $p \notin s.\texttt{child}$. Using Invariant I_7, we see that $\neg s.\texttt{rc}[\texttt{dev}(\texttt{peer}(p))]$. Combining all of this with I_9 we get $m = \texttt{parent}$. Suppose $s.\texttt{init}[\texttt{dev}(\texttt{peer}(p))]$. Then s enables $\texttt{addchild}(\texttt{dev}(\texttt{peer}(p)), \texttt{peer}(p))$ and we have a contradiction. So $\neg s.\texttt{init}[\texttt{dev}(\texttt{peer}(p))]$. Then s enables $\texttt{receivemes}(\texttt{dev}(\texttt{peer}(p)), \texttt{peer}(p), \texttt{parent})$ and we have a contradiction.

2. Let $\texttt{Measure}(s) = \langle I_s, C_s + R_s + M_s + L_s \rangle$ and $\texttt{Measure}(t) = \langle I_t, C_t + R_t + M_t + L_t \rangle$.

   (a) $a = \texttt{childrenknown}(d)$
   By precondition of a, $\neg s.\texttt{init}[d]$ and by effect of a, $t.\texttt{init}[d]$. So $I_t < I_s$. We conclude that $\texttt{Measure}(t) \prec \texttt{Measure}(s)$.

(b) $a = \texttt{addchild}(d, p)$
By effect of a, $t.\texttt{init} = s.\texttt{init}$, $t.\texttt{root} = s.\texttt{root}$ and $t.\texttt{lpd} = s.\texttt{lpd}$, hence $I_t = I_s$, $R_t = R_s$ and $L_t = L_s$. By precondition of a, $\texttt{head}(s.\texttt{mq}[\texttt{peer}(p)] = \texttt{parent}$. Combining this with Invariant I_5, we get $s.\texttt{mq}[\texttt{peer}(p)] = \{\} \vdash \texttt{parent}$. By the effect of a, $t.\texttt{mq}[\texttt{peer}(p)] = \texttt{tail}(s.\texttt{mq}[\texttt{peer}(p)]) = \{\}$, so $M_t = M_s - \{\texttt{peer}(p)\}$, hence $M_t < M_s$. Combining $\texttt{head}(s.\texttt{mq}[\texttt{peer}(p)] = \texttt{parent}$ with Invariant I_6 we get $p \notin s.\texttt{child}$. By effect of a, $t.\texttt{child} = s.\texttt{child} \cup \{p\}$. So $C_t < C_s$. By effect of a We conclude that $\texttt{Measure}(t) \prec \texttt{Measure}(s)$.

(c) $a = \texttt{receivemes}(d, p, m)$
By effect of a, $t.\texttt{init} = s.\texttt{init}$, $t.\texttt{child} = s.\texttt{child}$, $t.\texttt{root} = s.\texttt{root}$ and $t.\texttt{lpd} = s.\texttt{lpd}$, hence $I_t = I_s$, $C_t = C_s$, $R_t = R_s$ and $L_t = L_s$. By precondition of a, $\texttt{head}(s.\texttt{mq}[\texttt{peer}(p)] = m$. Combining this with Invariant I_5, we get $s.\texttt{mq}[\texttt{peer}(p)] = \{\} \vdash m$. By the effect of a, $t.\texttt{mq}[\texttt{peer}(p)] = \texttt{tail}(s.\texttt{mq}[\texttt{peer}(p)]) = \{\}$, so $M_t = M_s - \{\texttt{peer}(p)\}$, hence $M_t < M_s$. We conclude that $\texttt{Measure}(t) \prec \texttt{Measure}(s)$.

(d) $a = \texttt{solverootcontent}(d, p)$
By effect of a, $t.\texttt{init} = s.\texttt{init}$, $t.\texttt{root} = s.\texttt{root}$, $t.\texttt{mq} = s.\texttt{mq}$ and $t.\texttt{lpd} = s.\texttt{lpd}$, hence $I_t = I_s$, $R_t = R_s$, $M_t = M_s$ and $L_t = L_s$. By precondition of a, $s.\texttt{rc}[\texttt{dev}(\texttt{peer}(p))]$. Combining this with Invariant I_8 we get $p \notin s.\texttt{child}$. By effect of a, $t.\texttt{child} = s.\texttt{child} \cup \{p\}$. So $C_t < C_s$. We conclude that $\texttt{Measure}(t) \prec \texttt{Measure}(s)$.

(e) $a = \texttt{root}(d)$
By effect of a, $s.\texttt{init} = t.\texttt{init}$, $s.\texttt{child} = t.\texttt{child}$, $t.\texttt{mq} = s.\texttt{mq}$ and $t.\texttt{lpd} = s.\texttt{lpd}$, hence $I_t = I_s$, $C_t = C_s$, $M_t = M_s$ and $L_t = L_s$. By precondition of a, $\neg s.\texttt{root}[d]$, and by effect of a, $t.\texttt{root}[d]$. So $R_t < R_s$. We conclude that $\texttt{Measure}(t) \prec \texttt{Measure}(s)$.

(f) $a = \texttt{loopdetect}(d)$
By effect of a, $s.\texttt{init} = t.\texttt{init}$, $s.\texttt{child} = t.\texttt{child}$, $s.\texttt{root} = t.\texttt{root}$ and $s.\texttt{mq} = t.\texttt{mq}$, hence $I_t = I_s$, $C_t = C_s$, $R_t = R_s$ and $M_t = M_s$. By precondition of a, $\neg s.\texttt{lpd}[d]$, and by effect of a, $t.\texttt{lpd}[d]$. So $L_t < L_s$. We conclude that $\texttt{Measure}(t) \prec \texttt{Measure}(s)$.

(g) $a = \delta$
By effect of a, $s.\texttt{init} = t.\texttt{init}$, $s.\texttt{child} = t.\texttt{child}$, $s.\texttt{root} = t.\texttt{root}$, $s.\texttt{mq} = t.\texttt{mq}$ and $t.\texttt{lpd} = s.\texttt{lpd}$. Hence $I_t = I_s$, $C_t = C_s$, $R_t = R_s$, $M_t = M_s$ and $L_t = L_s$, and we conclude that $\texttt{Measure}(t) = \texttt{Measure}(s)$.

⊠

Now we show that TIP4 cannot get into a time deadlock by its own discrete activity. A timed I/O automaton that has this property is called *feasible*.

Theorem 5.12 TIP4 is feasible.

Proof It can easily be seen that TIP4 fulfills the requirements for Lemma A.3. This lemma fulfills one of the requirements in Theorem A.4. The other requirement is fulfilled by the Measure function and the result in Proposition 5.11. It follows from Theorem A.4 that TIP4 is feasible. ⊠

We now show that whenever there is a cycle in the network, it is detected by the protocol.

Theorem 5.13 Let α be an admissible execution of TIP4.
If $\texttt{oncycle}(d)$ then α contains an occurrence of $\texttt{lpd}(d)$.

Proof Since time proceeds in α without bound, and since initially $s.\texttt{lpd}[d]$ is false and since $s.\texttt{lpd}[d]$ can only be made true by an occurrence of $\texttt{lpd}(d)$, it suffices to show that for each reachable state s in TIP4, if $s.\texttt{time} > \texttt{MaxLpdtime}$, then $s.\texttt{lpd}[d]$. This follows easily from Invariant I_{14}. ⊠

Unfortunately, it is not possible to prove that if there is no cycle in the network, then within finite time a root will be elected. This is due to the unknown duration of the root contention solving sub-protocol. The following theorem shows that if no root contention occurs, then indeed a root is elected in finite time.

Theorem 5.14 Let $\alpha = s_0 a_1 s_1 \dots$ be an admissible execution of TIP4.
If $\forall d : \neg\texttt{oncycle}(d)$ and $\forall i, d : \neg s_i.\texttt{rc}[d]$, then $\exists d$ such that α contains an occurrence of $\texttt{root}(d)$.

Proof Assume $\forall d : \neg\texttt{oncycle}(d)$. We observe the following:

1. Time proceeds in α without bound.

2. In each reachable state s in TIP4 the following holds. For all d: if s is an initial state then $\neg s.\texttt{root}[d]$, and if $s.\texttt{root}[d]$, then each execution leading to s must contain an occurrence of $\texttt{root}(d)$.

3. If there is a state s in α and a d such that $\texttt{ports}(d) - s.\texttt{child} = \{\}$, then α contains an occurrence of $\texttt{root}(d)$.

   This is easily seen by a few observations. Fix s and d such that $\texttt{ports}(d) - s.\texttt{child} = \{\}$. First, $s.\texttt{init}[d]$ or $s.\texttt{root}[d]$ or s enables $\texttt{root}(d)$. If $s.\texttt{root}[d]$, then by Item 2 we conclude that α contains an occurrence of $\texttt{root}(d)$. If $s.\texttt{init}[d]$ then s enables $\texttt{childrenknown}(d)$. If s enables $\texttt{childrenknown}(d)$ or $\texttt{root}(d)$, then s does not enable any $\delta > 0$. If s enables $\texttt{childrenknown}(d)$ and $s \xrightarrow{a} t$ then either $a = \texttt{childrenknown}(d)$ and t enables $\texttt{root}(d)$ or t enables $\texttt{childrenknown}(d)$. If s enables $\texttt{root}(d)$ and $s \xrightarrow{a} t$ then $a = \texttt{root}(d)$ or t enables $\texttt{root}(d)$.

4. In each reachable state s in TIP4 the following holds. If $\forall p \in P$ either $p \in s.\texttt{child}$ or $\texttt{peer}(p) \in s.\texttt{child}$, then there exists a d such that $\texttt{ports}(d) - s.\texttt{child} = \{\}$.

   Suppose $\forall p \in P$ either $p \in s.\texttt{child}$ or $\texttt{peer}(p) \in s.\texttt{child}$ and there is no d such that $\texttt{ports}(d) - s.\texttt{child} = \{\}$. Construct a longest path $\pi = p_0 \dots p_n$ such that for each $i \in \{0, 2, \dots, n-1\} : p_i \notin s.\texttt{child}$ and for all $i, j \in \{0, 1, 3, 5, \dots, n\} : i \neq j \rightarrow \texttt{dev}(p_i) \neq \texttt{dev}(p_j)$. Since $p_{n-1} \notin s.\texttt{child}$, and $p_n = \texttt{peer}(p_{n-1})$ certainly $p_n \in s.\texttt{child}$. Suppose that $p \in \texttt{dev}(p_n) : p \notin s.\texttt{child}$. If $\texttt{dev}(\texttt{peer}(p)) = \texttt{dev}(p_i)$ for some $i \in \{0, \dots, n\}$, then we can construct a cycle, and we have a contradiction. So $\texttt{dev}(\texttt{peer}(p)) \neq \texttt{dev}(p_i)$ for all $i \in \{0, \dots, n\}$. But then we can construct a longer path than π to meet the requirements, and we have a contradiction. So we conclude that there is no $p \in \texttt{dev}(p_n) : p \notin s.\texttt{child}$, hence $\texttt{ports}(\texttt{dev}(p_n)) - s.\texttt{child} = \{\}$.

We now show that there is a state s in α and a d such that $\texttt{ports}(d) - s.\texttt{child} = \{\}$. By definition of α, there exists an i such that $s_i.\texttt{time} > (\lfloor \texttt{MaxHop}/2 + 1 \rfloor + 1) * \texttt{MaxDelay}$ and $\forall j < i : s_j.\texttt{time} \leq (\lfloor \texttt{MaxHop}/2 \rfloor + 1) * \texttt{MaxDelay}$. Fix i.
By Lemma 4.9 and $\forall d : \neg\texttt{oncycle}(d)$, we have $\forall d : \texttt{Steps}(d) \leq \lfloor \texttt{MaxHop}/2 \rfloor$, hence $\forall d : (\texttt{Steps}(d) + 1) * \texttt{MaxDelay} < s_i.\texttt{time}$. Using invariant I_{11} we get $\forall d : \neg s_i.\texttt{init}[d]$. Using invariant I_4 we get $\forall d : \texttt{size}(\texttt{ports}(d) - s_i.\texttt{child}) \leq 1$.

constant	*value*	*reference*
min cable length	0	no restriction specified
max cable length	4.5 m	Section 1.1, Page 1, 1394-1995
max cable hops	16	Section 1.1, Page 1, 1394-1995
propagation delay	$\leq$5.05 ns/m	Section 4.2.1.4.3, Page 74, 1394-1995
min CONFIG_TIMEOUT	166.6 μs	Table 7-14, Page 89, 1394-1995
	166.6 μs	Table 8-14, Page 90, P1394a
max CONFIG_TIMEOUT	166.9 μs	Table 7-14, Page 89, 1394-1995
	166.9 μs	Table 8-14, Page 90, P1394a

Table 1: IEEE 1394 timing constants

Suppose $\exists d : \texttt{size}(\texttt{ports}(d) - s_i.\texttt{child}) = 0$. Fix d. It follows that $\texttt{ports}(d) - s_i.\texttt{child} = \{\}$. By Item 3 we may conclude that α contains an occurrence of $\texttt{root}(d)$.
Suppose $\forall d : \texttt{size}(\texttt{ports}(d) - s_i.\texttt{child}) = 1$. Suppose $\exists p : p \notin s_i.\texttt{child} \wedge \texttt{peer}(p) \notin s_i.\texttt{child}$. Fix p. By our assumption we have $\neg s.\texttt{rc}[\texttt{dev}(\texttt{peer}(p))]$. Combining this with $\neg s.\texttt{init}[\texttt{dev}(p)]$ and By invariant I_9, we get $\texttt{head}(s_i.\texttt{mq}[p]) = \texttt{parent}$. Combining this with $\neg\texttt{oncycle}(\texttt{dev}(p))$ and Invariant I_{11}, we get $s_i.\texttt{time} - s_i.\texttt{delay}[p]) \leq \texttt{Steps}(\texttt{dev}(p)) * \texttt{MaxDelay}$. Since $s_i.\texttt{time} > (\texttt{Steps}(\texttt{dev}(p)) + 1) * \texttt{MaxDelay}$, we have $s_i.\texttt{delay}[p] > \texttt{MaxDelay}$ which is in contradiction with Invariant I_{10}. We conclude that there is no p such that $p \notin s_i.\texttt{child} \wedge \texttt{peer}(p) \notin s_i.\texttt{child}$. Since $\forall p : p \in s_i.\texttt{child} \vee \texttt{peer}(p) \in s_i.\texttt{child}$ we can use Item 4 to conclude that there is a d such that $\texttt{ports}(d) - s_i.\texttt{child} = \{\}$. Fix d. By Item 3 we may conclude that α contains an occurrence of $\texttt{root}(d)$.

$\boxtimes$

5.4 Are the IEEE 1394 timing constants correct?

Table 1 gives the IEEE 1394 timing constants, and a reference to where they are to be found in the documentation. Here, 1394-1995 refers to [10] and P1394a refers to [11]. Note that the constants are the same for 1394-1995 and P1394a. From these numbers, we get the constants used for the formal verification as follows:

$$\begin{aligned}
\texttt{MinDelay} &= \text{min cable length} * \text{propagation delay} = 0\text{ns} \\
\texttt{MaxDelay} &= \text{max cable length} * \text{propagation delay} = 22.72\text{ns} \\
\texttt{MinLpdtime} &= \text{min CONFIG_TIMEOUT} = 166.6\mu\text{s} \\
\texttt{MaxLpdtime} &= \text{max CONFIG_TIMEOUT} = 166.9\mu\text{s} \\
\texttt{MaxHop} &\leq \text{max cable hops} = 16
\end{aligned}$$

The question is then, do these constants meet the requirement for correct implementation? We found in Theorem 5.9 that the model behaves correctly if the relation $\texttt{MinLpdtime} > \max(0, \texttt{MaxHop} - 1) * \texttt{MaxDelay}$ holds. Since $(16-1) * 22.72$ ns $= 340.80$ ns $< 166.9\mu$s, the answer is yes. If the devices in IEEE 1394 enter the tree identify phase at the same time, if there is no device with the force_root flag set to true, and if our model of the IEEE 1394 communication is correct, then we can say the following with certainty: If a loop is in the network, it is detected, and that if there is no loop in the network, no loop will be detected and a root will be chosen.

The difference between the actual MinLpdtime value and the minimal value as required by

our relation is rather large. One could wonder whether this implies that the limitations set by IEEE 1394 and P1394a can be loosened. This could be done by decreasing the `MinLpdtime` value, increasing the number of nodes allowed, increasing the delay between nodes (by allowing greater cable lengths), or a combination of these. However, the times at which the tree identify phase is entered can differ among nodes. The constant responsible for the duration of the bus reset signal being sent is based on a worst-case scenario for any node to notice that a bus reset period has started. This constant has a value of about 166 μs, and can be used as an indication of the difference in starting times for the tree identify phase. If the times can indeed be that far apart for peer nodes, the loop detection timer should be in the same order of magnitude to not run the risk of detecting a loop when it is not there. Moreover, the use of the force root flag increases the delay in participating in the tree identify phase even further. We conclude that it is not yet clear whether the IEEE 1394 and P1394a bounds are correct and may be loosened.

6 Conclusions

The verification shows that under the assumptions made, the IEEE 1394 definition of the tree identify phase meets the requirements. Exactly one root is chosen when there is no cycle present, and a cycle is detected if and only if there is a cycle present in the network. It is obvious from the proofs that the refinement step from an untimed model to a timed model in combination with the desired property of correct cycle detection is a complicated one. More proofs about network topologies are needed to make a quantitative analysis of the worst case scenarios. Also the invariant properties that are specific to the model `TIP4` become more complicated. The effort invested in the construction of these proofs adds up to about two months. We hope that in further refinement steps these proofs can be reused with little effort.

As to the remaining IEEE 1394 details that we have not considered, we believe that the addition of the delay in entering the tree identify phase will not affect the correctness of the protocol. Also the addition of the root contention solving protocol with its verification from [28] will probably not touch the critical behaviour parts of the root election or cycle detection. However, the correctness of a model obtained by adding the force_root flag and the assumption that the message queues model the IEEE 1394 signal communication is not that obvious. An extension of this work may show that either IEEE 1394 timing bounds can be tightened or should be loosened.

The advantage of the layered verification in this case is that we do not need to prove anything about the safety properties of root election, since our refinement proof give us safety immediately. Establishing the refinement was not as easy as expected, because of the complicated reuse of invariants at the abstract level. The extra lemma that was needed shows that the proof obligations can still be divided over small, clear proof steps.

Establishing the desired liveness properties that express that indeed a cycle is detected when there is a cycle in the network topology and indeed a root is elected otherwise, do not follow from the 'implements' relation, since we have only proved inclusion of admissible traces. In an untimed verification, liveness properties are proved by showing a *fair trace* inclusion, that is, each fair trace from the more detailed model is also a fair trace in the more abstract model. In most cases, the liveness property holds trivially for any fair trace of the abstract model, and therefore also for any fair trace of the detailed model. In a timed verification, liveness is often expressed in terms of timing bounds. Then again the (admissible) trace inclusion yields correctness. In our case, both of these methods do not work. We are comparing a timed model

to an essentialy untimed model and hence have no fairness that carries over from the more abstract level to the (timed) detailed level. On the other hand, we have no timing requirement for when the root should be elected. So we have had to prove the liveness completely on the level of model `TIP4`, without reusing proofs from the level of `TIP3`. In most cases and especially in timed verifications, proving safety involves many properties from which the greater part of the proof obligations for the liveness properties already follows. It can be argued from Section 5.3 that here the remaining proof obligation is not trivial. Nevertheless, we consider model `TIP4` to be highly complex, and expect that in other timed verifications it will be easier to show that once time has proceeded beyond a certain point, the safety properties combined with the feasibility proofs give the desired liveness properties.

We conclude that for proving safety and liveness properties in a situation with only untimed models or with only timed models, a layered verification is a very suitable proof method which allows one to 'divide and conquer' the proof obligations. In a situation where timed and untimed behaviour are compared, we think that other methods should be used in addition, or the degree of refinement should be very low in order for a layered verification to diminish the amount of work to be done in each layer. It would be very benificial if the proofs constructed for this paper were checked with a proof checker. Careful manual inspection can never replace the confidence obtained by such automated inspection. Some results have been obtained in checking invariant proofs for I/O automata, both timed and untimed, as can be seen in [2], and several papers which are under construction [1]. We expect that such an effort will be considerable, but manageable.

Acknowledgements The following people have contributed to this paper. David Griffioen was my partner in finding the essence of the TIP behaviour in the IEEE standard. Besides being the co-author of [4, 7], he also came up with an good version of model `TIP4`. Rudi Bloks explained some details of the IEEE standard which were vital for getting the models right. At the point where I got stuck in an early verification attempt of this protocol, Frits Vaandrager suggested and initiated the layered verification, and later he came up with the sufficient conditions for the weak timed refinement. Finally, the anonymous referees have given many helpful comments.

References

[1] M. Archer. Personal communication, March 1999.

[2] M. Archer and C. Heitmeyer. Mechanical verification of timed automata: A case study. In *Proceedings 1996 IEEE Real-Time Technology and Applications Symposium (RTAS'96)*. IEEE Computer Society Press, 1996. A full version is available as Report NRL/MR/5540–98-8180 from URL `http://www.itd.nrl.navy.mil/ITD/5540/publications/CHACS/1998/`.

[3] J.W. Davies and S.A. Schneider. A brief history of timed csp. *Theoretical Computer Science*, 138(2):243–271, 1995.

[4] M.C.A. Devillers, W.O.D. Griffioen, J.M.T Romijn, and F.W. Vaandrager. Verification of a leader election protocol — formal methods applied to IEEE 1394. Technical Report CSI-R9728, Computing Science Institute, University of Nijmegen, December 1997. Accepted, subject to revision, for Formal Methods in System Design.

[5] S.J. Garland, N.A. Lynch, and M. Vaziri. IOA: A language for specifying, programming, and validating distributed systems, December 1997. Available through URL `http://larch.lcs.mit.edu:8001/~garland/ioaLanguage.html`.

[6] R. Gawlick, R. Segala, J.F. Søgaard-Andersen, and N.A. Lynch. Liveness in timed and untimed systems. In S. Abiteboul and E. Shamir, editors, *Proceedings* 21^{th} *ICALP,* Jerusalem, volume 820 of

Lecture Notes in Computer Science. Springer-Verlag, 1994. A full version appears as MIT Technical Report number MIT/LCS/TR-587.

[7] W.O.D. Griffioen and F.W. Vaandrager. Normed simulations. In *Proceedings CAV'98*, volume 1427 of *Lecture Notes in Computer Science*, pages 332–344. Springer-Verlag, 1998.

[8] J.F. Groote and A. Ponse. The syntax and semantics of μCRL. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Algebra of Communicating Processes '94*, Workshops in Computing. Springer-Verlag, 1995.

[9] J.F. Groote and J.S. Springintveld. Focus points and convergent process operators — a proof strategy for protocol verification. Technical Report 142, Logic Group Preprint Series, Utrecht University, 1995. This report also appeared as Technical Report CS-R9566, CWI, 1995.

[10] IEEE Computer Society. IEEE Standard for a High Performance Serial Bus. Std 1394-1995, August 1996.

[11] IEEE Computer Society. Draft Standard for a High Performance Serial Bus (Supplement). P1394a Draft 3.0, June 1999.

[12] L. Kühne, J. Hooman, and W.P. de Roever. Towards mechanical verification of parts of the IEEE P1394 serial bus. In I. Lovrek, editor, *Proceedings 2nd International Workshop on Applied Formal Methods in System Design,* Zagreb, pages 73–85, 1997.

[13] Leslie Lamport. How to write a long formula. *Formal Aspects of Computing*, 6:580–584, 1994. Also appeared as SRC Research Report 119.

[14] S.P. Luttik. Description and formal specification of the Link layer of P1394. In I. Lovrek, editor, *Proceedings of the 2nd International Workshop on Applied Formal Methods in System Design,* Zagreb, pages 43–56, 1997. Also available as Report SEN-R9706, CWI, Amsterdam. See URL `http://www.cwi.nl/~luttik/`.

[15] N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., San Francisco, California, 1996.

[16] N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the* 6^{th} *Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, 1987. A full version is available as MIT Technical Report MIT/LCS/TR-387.

[17] N.A. Lynch and M.R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, September 1989.

[18] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations, I: Untimed systems. *Information and Computation*, 121(2):214–233, September 1995.

[19] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations, II: Timing-based systems. *Information and Computation*, 128(1):1–25, July 1996.

[20] Z. Manna and A. Pnueli. Verifying hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 4–35. Springer-Verlag, 1993.

[21] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, 1995.

[22] S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.

[23] S.A. Schneider. Timewise refinement for communicating processes. *Science of Computer Programming*, 28(1):43–90, 1997.

[24] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1995. Available as Technical Report MIT/LCS/TR-676.

[25] R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[26] C. Shankland and M.B. van der Zwaag. The tree identify protocol of IEEE 1394 in μCRL. *Formal Aspects of Computing*, 10:509–531, 1998.

[27] M. Sighireanu and R. Mateescu. Verification of the link layer protocol of the ieee-1394 serial bus (firewire): an experiment with e-lotos. *Springer International Journal on Software Tools for Technology Transfer (STTT)*, 2(1):68–88, December 1998.

[28] M.I.A. Stoelinga and F.W. Vaandrager. Root contention in IEEE 1394. In *Proceedings 5th Int. AMAST Workshop on Real-Time and Probabilistic Systems (ARTS'99)*. Springer-Verlag, 1999. To appear.

A Safe and Timed I/O Automata

In this appendix we review some basic definitions from [6, 21, 19], and we give sufficient conditions for including invariants in refinement proofs when the invariants at the refined level depend on invariants at the abstract level. These conditions are presented in Lemma A.2. Lemma A.6 is the timed version, which is used in the verification in this paper.

A.1 Safe I/O automata

A *safe I/O automaton* B consists of the following components:

- A set $states(B)$ of *states* (possibly infinite).
- A nonempty set $start(B) \subseteq states(B)$ of *start states*.
- A set $acts(B)$ of *actions*, partitioned into three sets $in(B)$, $int(B)$ and $out(B)$ of *input*, *internal* and *output* actions, respectively.
  Actions in $local(B) \triangleq out(B) \cup int(B)$ are called *locally controlled*.
- A set $steps(B) \subseteq states(B) \times acts(B) \times states(B)$ of *transitions*, with the property that for every state s and input action $a \in in(B)$ there is a transition $(s, a, s') \in steps(B)$.

We let s, s',.. range over states, and a,.. over actions. We write $s \xrightarrow{a}_B s'$, or just $s \xrightarrow{a} s'$ if B is clear from the context, as a shorthand for $(s, a, s') \in steps(B)$.

Enabling of actions An action a of a safe I/O automaton B is *enabled* in a state s iff $s \xrightarrow{a} s'$ for some s'. Since every input action is enabled in every state, safe I/O automata are said to be *input enabled*. The intuition behind the input-enabling condition is that input actions are under control of the environment and that the system that is modeled by a safe I/O automaton cannot prevent the environment from doing these actions.

Executions An *execution fragment* of a safe I/O automaton B is a finite or infinite alternating sequence $s_0 a_1 s_1 a_2 s_2 \cdots$ of states and actions of B, beginning with a state, and if it is finite also ending with a state, such that for all i, $s_i \xrightarrow{a_{i+1}} s_{i+1}$. An *execution* is an execution fragment that begins with a start state. We write $execs^*(B)$ for the set of finite executions of B, and $execs(B)$ for the set of all executions of B. A state s of B is *reachable* if it is the last state of some finite execution of B. We write $rstates(B)$ for the set of reachable states of B.

Traces Suppose $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$ is an execution fragment of B. Let $\gamma = a_1 a_2 \cdots$. Then the *trace* of α is the sequence $(\gamma \lceil in(B) \cup out(B))$, denoted by $\hat{\gamma}$. With $traces(B)$ we denote the set of traces of executions of B. For s, s' states of B and β a finite sequence of input and output actions of B, we define $s \stackrel{\beta}{\Rightarrow}_B s'$ iff B has a finite execution fragment with first state s, last state s' and trace β.

Invariants Let $P, Q \subseteq states(B)$. P is *invariant* for B if it is a superset of the reachable states of B, i.e. $rstates(B) \subseteq P$. P is *inductive relative to* Q if $start(B) \subseteq P$ and if for each $s' \in P \cap Q$: $s' \xrightarrow{a}_B s$ implies $s \in P$.

Refinements Let A and B be safe I/O automata. A *refinement* from A to B is a function $r : states(A) \to states(B)$ that satisfies:

1. If $s \in start(A)$ then $r(s) \in start(B)$.
2. If $s' \xrightarrow{a}_A s$ then $r(s') \stackrel{\beta}{\Rightarrow}_B r(s)$, where $\beta = \hat{a}$.

Let A and B be safe I/O automata with invariants P and Q, respectively. A *weak refinement* from A to B, with respect to P and Q, is a function $r : states(A) \to states(B)$ that satisfies:

1. If $s \in start(A)$ then $r(s) \in start(B)$.
2. If $s' \xrightarrow{a}_A s$, $s' \in P$, and $r(s') \in Q$, then $r(s') \stackrel{\beta}{\Rightarrow}_B r(s)$, where $\beta = \hat{a}$.

Theorem A.1 Let A and B be safe I/O automata. If there exists a (weak) refinement from A to B, then $traces(A) \subseteq traces(B)$.

Using abstract and refined invariants in a refinement Let A, B be safe I/O automata. The following lemma gives sufficient conditions for a weak refinement from A to B when one wants to use P_1, P_2, Q such that Q is invariant for B, P_1 is invariant for A depending on Q and the definition of the refinement function, and P_2 is invariant for A depending on P_1.

Lemma A.2 Let A, B be safe I/O automata. Let Q be invariant for B and P_2 be inductive relative to P_1 for A. Let $r : states(A) \to states(B)$ such that

1. $r(s) \in Q$ implies $s \in P_1$,
2. $s \in start(A)$ implies $r(s) \in start(B)$, and
3. $s' \in P_2$, $r(s') \in Q$ and $s' \xrightarrow{a}_A s$ implies $r(s') \stackrel{\beta}{\Rightarrow}_B r(s)$, where $\beta = \hat{a}$.

Then

1. P_1, P_2 are invariant for A.
2. r is a weak timed refinement from A to B with respect to P_2 and Q.

Proof

1. By induction.
   $\mathrm{IH}(n) \;=\; \forall s, \alpha : \; (s \in \mathit{rstates}(A) \wedge \alpha \in \mathit{execs}(A) \wedge \alpha = s_0\, a_1\, s_1 \ldots s_n \wedge s = s_n) \rightarrow (r(s) \in \mathit{rstates}(B) \wedge s \in (P_1 \cap P_2))$
   - Base step: $n = 0$.
     By definition of α, $s \in \mathit{start}(A)$. By definition of r, $r(s) \in \mathit{start}(B)$ so certainly $r(s) \in \mathit{rstates}(B)$. Since Q is invariant for B, $r(s) \in Q$. By definition of r, $s \in P_1$. Since $s \in \mathit{start}(A)$, and since P_2 is inductive relative to P_1 for A, $s \in P_2$.
   - Induction step: $\forall n \leq n' : \mathrm{IH}(n)$.
     Let $s \in \mathit{rstates}(A) \wedge \alpha \in \mathit{execs}(A) \wedge \alpha = s_0\, a_1\, s_1 \ldots s_{n'}\, a_{n'+1}\, s_{n'+1} \wedge s = s_{n'+1}$. Since $s_0 a_1 s_1 \ldots s_{n'} \in \mathit{execs}(A)$, certainly $s_{n'} \in \mathit{rstates}(A)$. Combining this with $n' \leq n'$, we get $\mathrm{IH}(n')$. Since $\mathrm{IH}(n')$, $r(s_{n'}) \in \mathit{rstates}(B) \wedge s_{n'} \in (P_1 \cap P_2)$. Since $r(s_{n'}) \in \mathit{rstates}(B)$ and Q is invariant for B, $r(s_{n'}) \in Q$. Since $s_{n'} \overset{a_{n'+1}}{\rightarrow}_A s_{n'+1}$ and by definition of r, $r(s_{n'}) \overset{\beta}{\Rightarrow}_B r(s_{n'+1})$ with $\beta = \widehat{a_{n'+1}}$, hence $r(s_{n'+1}) \in \mathit{rstates}(B)$, hence $r(s_{n'+1}) \in Q$. By definition of r, $s_{n'+1} \in P_1$. Since $s_{n'} \in (P_1 \cap P_2)$ and $s_{n'} \overset{a_{n'+1}}{\rightarrow}_A s_{n'+1}$ and since P_2 is inductive relative to P_1 for A, $s_{n'+1} \in P_2$.
2. By Item 1, the assumption that Q is invariant for B and by definition of r.

$\boxtimes$

A.2 Timed I/O Automata

A *timed I/O automaton* A is a safe I/O automaton whose set of actions includes R^+, the set of positive reals. Actions from R^+ are referred to as *time-passage actions*. Other actions are referred to as *discrete actions*. Performing one or more consecutive time-passage actions is called *idling*. We let $d, d', \ldots$ range over R^+ and, more generally, $t, t', \ldots$ over the set R of real numbers. The set of *visible* actions is defined by $\mathit{vis}(A) \triangleq (\mathit{in}(A) \cup \mathit{out}(A)) - \mathsf{R}^+$.
We assume that a timed I/O automaton satisfies the following axioms.

S1 If $s' \xrightarrow{d} s''$ and $s'' \xrightarrow{d'} s$, then $s' \xrightarrow{d+d'} s$.

For the second axiom, an auxiliary definition is needed. A *trajectory* for a step $s' \xrightarrow{d} s$ is a function $w : [0, d] \rightarrow \mathit{states}(A)$ such that $w(0) = s'$, $w(d) = s$, and

$$w(t) \xrightarrow{t'-t} w(t') \text{ for all } t, t' \in [0, d] \text{ with } t < t'.$$

Now we can state the second axiom.

S2 Each step $s \xrightarrow{d} s'$ has a trajectory.

Axiom **S1** gives a natural property of time, namely that if time can pass in two steps, then it can also pass in a single step. The *trajectory axiom* **S2** is a kind of converse to **S1**; it says that any time-passage step can be "filled in" with states for each intervening time, in a "consistent" way. Executions of timed I/O automata correspond to what are called *sampling computations* in [20].

Timed Traces The full externally visible behaviour of a timed I/O automaton can be inferred from its executions as follows: suppose $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$ is an execution fragment of a timed I/O automaton A. For each index j, let t_j be given by

$$\begin{aligned} t_0 &= 0, \\ t_{j+1} &= \text{if } a_{j+1} \in \mathsf{R}^+ \text{ then } t_j + a_{j+1} \text{ else } t_j. \end{aligned}$$

The *limit time* of α, notation $\alpha.ltime$, is the smallest element of $\mathsf{R}^{\geq 0} \cup \{\infty\}$ larger than or equal to all the t_j. We say α is *admissible* if $\alpha.ltime = \infty$, and *Zeno* if it is an infinite sequence but with a finite limit time. The *timed trace* $t\text{-}trace(\alpha)$ associated with α is defined by

$$t\text{-}trace(\alpha) \triangleq (((a_1, t_1)(a_2, t_2)\cdots)\lceil(vis(A) \times \mathsf{R}^{\geq 0}), \alpha.ltime).$$

Thus, $t\text{-}trace(\alpha)$ records the visible actions of α paired with their times of occurrence, as well as the limit time of the execution. A pair β is a *timed trace* of A if it is the timed trace of some finite or admissible execution of A. Thus, we explicitly exclude the timed traces that originate from Zeno executions. We write $t\text{-}traces(A)$ for the set of all timed traces of A, $t\text{-}traces^*(A)$ for the set of *finite* timed traces (the timed traces derived from the finite executions), and $t\text{-}traces^\infty(A)$ for the set of *admissible* traces (the timed traces derived from the admissible executions).

Moves We say $s' \overset{p}{\rightsquigarrow}_A s$ is a *t-move* of A if A has a finite timed execution fragment $\alpha = s_0 a_1 s_1 \ldots s_n$ such that $s' = s_0$, $s = s_n$ and $p = t\text{-}trace(\alpha)$.

Feasibility Let A be a timed I/O automaton. We say A is *feasible* if each element of $t\text{-}traces^*(A)$ is the prefix of some element of $t\text{-}traces^\infty(A)$.

Giving the proof for feasibility can be hard or tiresome. However, in some cases it follows rather straightforwardly from the definition of the timed I/O automaton. We give the following sufficient conditions, divided over two results, of which the first is rather simple, and the second is a bit more involved.

Lemma A.3 Let A be a timed I/O automaton with clock variables X and discrete variables Y. If

1. The precondition of time action d is of the following form, in which $\phi, \psi_1, \ldots, \psi_n$ are Boolean expressions over variables in Y, $x_1, \ldots, x_n \in X$ and $c_1, \ldots, c_n \in \mathsf{R}^+$:

   $$\neg\phi \wedge (\psi_1 \rightarrow x_1 + d \leq c_1) \wedge \cdots \wedge (\psi_n \rightarrow x_n + d \leq c_n)$$

2. The effect of time action d is of the following form:

   $$\forall x \in X : x := x + d$$

3. For each $s \in reachable(A)$:

   $$(s \models \phi) \rightarrow \exists a : s \xrightarrow{a} \wedge\, a \text{ is discrete}$$

4. For each $s \in reachable(A)$ and $0 \leq i \leq n$:

   $$(s \models \psi_i \wedge x_i \geq c_i) \rightarrow \exists a : s \xrightarrow{a} \wedge\, a \text{ is discrete}$$

then for each $s \in \mathit{reachable}(A)$ and $d > 0$, the following holds:

$$\begin{array}{ll} \vee & s \xrightarrow{d} \\ \vee & \exists a : s \xrightarrow{a} \wedge\, a \text{ is discrete} \\ \vee & \exists d', a, s' : d' < d \wedge s \xrightarrow{d'} s' \xrightarrow{a} \wedge a \text{ is discrete} \end{array}$$

Proof Let $s \in \mathit{reachable}(A)$, $d > 0$ and s does not enable d. Then $s \models \neg(\neg\phi \wedge (\psi_1 \rightarrow x_1 + d \leq c_1) \wedge \cdots \wedge (\psi_n \rightarrow x_n + d \leq c_n))$, which can easily be rewritten to $s \models \phi \vee (\psi_1 \wedge x_1 + d > c_1) \vee \cdots \vee (\psi_n \wedge x_n + d > c_n)$.

Suppose $s \models \phi$. By Assumption 3, there is a discrete action a such that $s \xrightarrow{a}$, and the result follows.

Suppose $s \models \neg\phi$. Then $s \models (\psi_1 \wedge x_1 + d > c_1) \vee \cdots \vee (\psi_n \wedge x_n + d > c_n)$. Take J to be the set of indices for which the disjunct is true, that is, $J = \{i | 1 \leq i \leq n \wedge s \models \psi_i \wedge x_i + d > c_i)\}$.

Suppose that for some $i \in J$, $s \models x_i \geq c_i$. Then by Assumption 4, there is a discrete action a such that $s \xrightarrow{a}$, and the result follows.

Suppose that for all $i \in J$, $s \models x_i < c_i$. Take d' to be the smallest value such that for some $i \in J$, $s \models x_i + d' = c_i$. Fix i. It is clear that for each $1 \leq j \leq n$ which is not in J, $s \models x_j + d' \leq c_j$. By assumption, $s \models \neg\phi$, so we now see that s enables d'. Let $s \xrightarrow{d'} s'$. By Assumption 2, and $s \models x_i + d' = c_i$, the effect of d' is such that $s' \models x_i = c_i$. Now by Assumption 4, there is a discrete action a such that $s \xrightarrow{a}$, and the result follows. ⊠

Theorem A.4 Let A be a timed I/O automaton. If

1. For each $s \in \mathit{reachable}(A)$ and $d > 0$, the following holds:

$$\begin{array}{ll} \vee & s \xrightarrow{d} \\ \vee & \exists a : s \xrightarrow{a} \wedge\, a \text{ is discrete} \\ \vee & \exists d', a, s' : d' < d \wedge s \xrightarrow{d'} s' \xrightarrow{a} \wedge a \text{ is discrete} \end{array}$$

2. Function $M : \mathit{states}(A) \rightarrow D$ is a measure function, $\prec$ is a well-founded ordering on D, and $C \in \mathsf{R}^+$ is a constant such that for each $s, s' \in \mathit{reachable}(A)$: $s \xrightarrow{a} s'$ implies that if a is discrete and s does not enable C, then $M(s') \prec M(s)$, otherwise $M(s') \preceq M(s)$.

then A is feasible.

Proof Suppose $\alpha \in \mathit{t\text{-}traces}^\infty(A)$. We define the function f that recursively builds an admissible execution from any state, as follows:

$$f(s) = \begin{cases} s\ C\ f(s') & \mathit{if}\ s \xrightarrow{C} s' \\ s\ a\ f(s') & \mathit{if}\ s \overset{C}{\not\rightarrow} \wedge\, s \xrightarrow{a} s' \\ s\ d\ s'\ a\ f(s'') & \mathit{if}\ s \overset{C}{\not\rightarrow} \wedge\, (\forall a' :\ a \text{ is discrete } \rightarrow\ s \overset{a'}{\not\rightarrow}) \wedge\ s \xrightarrow{d} s' \xrightarrow{a} s'' \end{cases}$$

Note that $f(s)$ may pick a and d in an arbitrary way when s does not enable C. For the proof this has no consequence.

Let $\alpha = \alpha' a s$ and let β be the execution resulting from $\alpha' a f(s)$. By Assumption 1, β can be constructed.

Suppose β is not admissible. Then there is an infinite suffix in β in which each occurrence of a time step implies that the time passing is smaller than C. Without loss of generality we

assume that the suffix starts after the prefix α', that is, in the part which is constructed by f. By definition of f, no state in this suffix enables C, so there are no two adjacent time steps in this suffix. We see that there are infinitely many occurrences of discrete actions in the suffix. Combining this with the fact that each state in the suffix does not enable C we have a contradiction with Assumption 2, our decreasing measure function. We conclude that β is admissible. ⊠

Implementation relation Let A and B be timed I/O automata. A *implements* B if *t-traces*$(A) \subseteq$ *t-traces*(B).

Refinements Let A and B be timed I/O automata. A *timed refinement* from A to B is a function $r : \mathit{states}(A) \to \mathit{states}(B)$ that satisfies:

1. If $s \in \mathit{start}(A)$ then $r(s) \in \mathit{start}(B)$.
2. If $s' \xrightarrow{a}_A s$ then $r(s') \overset{p}{\rightsquigarrow}_B r(s)$, where $p = \textit{t-trace}(s'as)$.

Let A and B be timed I/O automata with invariants P and Q, respectively. A *weak timed refinement* from A to B, with respect to P and Q, is a function $r : \mathit{states}(A) \to \mathit{states}(B)$ that satisfies:

1. If $s \in \mathit{start}(A)$ then $r(s) \in \mathit{start}(B)$.
2. If $s' \xrightarrow{a}_A s$, $s' \in P$, and $r(s') \in Q$, then $r(s') \overset{p}{\rightsquigarrow}_B r(s)$, where $p = \textit{t-trace}(s'as)$.

Theorem A.5 Let A and B be timed I/O automata. If there exists a (weak) timed refinement from A to B, then *t-traces*$(A) \subseteq$ *t-traces*(B).

Using abstract and refined invariants in a timed refinement We now present the timed version of Lemma A.2, since the timed version is used in the verification in this paper.

Lemma A.6 Let A, B be timed I/O automata. Let Q be invariant for B and P_2 be inductive relative to P_1 for A. Let $r : \mathit{states}(A) \to \mathit{states}(B)$ such that

1. $r(s) \in Q$ implies $s \in P_1$,
2. $s \in \mathit{start}(A)$ implies $r(s) \in \mathit{start}(B)$, and
3. $s' \in P_2$, $r(s') \in Q$ and $s' \xrightarrow{a}_A s$ implies $r(s') \overset{p}{\rightsquigarrow}_B r(s)$, where $p = \textit{t-trace}(s'as)$.

Then

1. P_1, P_2 are invariant for A.
2. r is a weak timed refinement from A to B with respect to P_2 and Q.

Proof Similar to the proof for Lemma A.2. ⊠