Under consideration for publication in Formal Aspects of Computing

IEEE 1394 Tree Identify Protocol: Introduction to the Case Study

Savi Maharaj[1], Judi Romijn[2] and Carron Shankland[1]

[1]Department of Computing Science and Mathematics
University of Stirling, United Kingdom
[2]Department of Computing
University of Nijmegen, The Netherlands

Keywords: IEEE standard, FireWire, comparative case study, formal methods.

Abstract. We introduce a comparative case study on the application of formal methods and techniques to the Tree Identify Protocol of the IEEE standard 1394 serial multimedia bus. The Tree Identify Protocol makes an ideal subject for this purpose because it is small yet complex, and may be modelled in a variety of ways. We provide an informal explanation of the protocol, describe how the case study was conducted, and give an overview of the results.

1. Background

Networks of multimedia devices and equipment are increasingly common in homes and offices. Sophisticated systems have been developed for connecting the components of such networks, such as the Universal Serial Bus (USB) and the IEEE 1394 High Performance Serial Bus ("FireWire", "Lynx", "i.Link"). A key feature of these interconnection systems is that they allow dynamic reconfiguration of the network without requiring devices on the network to be switched off, a feature known as "Plug-and-Play", "hot-plugging", etc. To support such versatility, these systems have a complex architecture and behaviour, involving multiple layers of structure and multiple phases of activity, each of which may incorporate a variety of protocols and algorithms.

A full understanding of such a system is difficult to achieve. Many factors need to be taken into account, from the abstract, high-level structure, to low-level details such as timing characteristics. Complex behaviours must be analysed, both within each phase of the system's functioning and at the inter-phase boundaries, where subtle, unexpected behaviours may emerge. Yet there are high requirements for reliability, accuracy, and performance placed upon such systems, and widespread commercial interest in their correctness, so it is essential that they are well understood.

Formal methods can be of great value in helping to understand such a system. By appropriate choice of formal method, and of an appropriate formal *model* within the possibilities afforded by that method, it is possible to study the system at different levels of abstraction, or to zoom in on particular aspects of its behaviour. The papers in this special issue of Formal Aspects of Computing give an idea of the wide range of choices for modelling and analysis that are provided by formal methods at the present state of the art. The majority of the works presented deal with the formalization of the same problem, namely the protocol used in the Tree Identify Phase in the physical layer of the IEEE 1394 serial multimedia bus. This collection therefore constitutes a comparative case study in the use of formal methods, and adds to the growing body of work of this kind (e.g., [ABL96, BG00, BMS96, FH01, LL95]) which serve an important purpose in broadening awareness of the scope and possibilities of formal methods.

1.1. The workshop

The papers in this collection were submitted following a workshop held in Berlin in March 2001, as a satellite of the Formal Methods Europe conference. The IEEE 1394 bus was chosen as the subject of this study for several reasons. Firstly, it is an international standard, therefore there is a clear and widely available statement of its definition [IEE95]. Secondly, there have already been several efforts at applying formal methods to aspects of IEEE 1394 ([BSdRT00, BST00, D'A99, DGRV00, GV98, Rom01, SvdZ98, SV01, SS01, SV99], also see surveys in [MS00, Rom01, Sto02]), which provide a departure point for new modelling attempts. Finally, there is a wide and varied range of technical issues to be considered in formalizing IEEE 1394, so that it poses a challenge to the capabilities of formal methods.

The main focus of the workshop was the protocol prescribed for the Tree Identify Phase of IEEE 1394. For convenience, we shall refer to this protocol as the "Tree Identify Protocol", though this name is not used in the standard. Before the workshop, would-be contributers were provided with excerpts from IEEE 1394 comprising the description of the Tree Identify Protocol. After the workshop, participants were encouraged to evaluate their contribution by comparison with those of other participants, and by reflecting upon the answers to a number of questions, given in Section 3.

1.2. Structure of this paper

Section 2 gives an informal but detailed explanation of the salient aspects of IEEE 1394, and in particular, the Tree Identify Phase. This section is intended to complement the remaining papers in this collection by providing explanations of all technical terms relating to IEEE 1394. Section 3 reproduces the points of self-evaluation which the authors were asked to address. Finally, Section 4 gives an overview of the contributions that were accepted for this collection.

2. IEEE 1394

2.1. Overview

The 1394 bus is designed to connect together a number of distributed components (multimedia systems and devices). To provide maximum usability, the bus must be scalable and cope easily with changing network configurations. Each node has a number of ports which may be connected to other nodes.

To render the complexity of the system more accessible, the behaviour of the bus is described in layers in the style of OSI, with *physical*, *link*, and *transaction* layers, and a vertical *management* layer, as illustrated in Figure 1. In this collection of papers we are concerned with behaviour of the physical layer (referred to in the standard [IEE95][1] as PHY). The behaviour of each layer is in turn split into different phases, each associated with a different task. Figure 2 shows the PHY layer comprises the four phases Bus Reset, Tree Identify, Self Identify, and Normal Operation.

The Tree Identify Phase spans a tree in the network, and the node that is the root has the special task of being the cycle master (which we will not explain further). The idea behind the Tree Identify Phase is that since the network configuration is not fixed it makes no sense to expect a particular component to always

[1] In the following "the standard" should be taken to mean [IEE95]. The changes described in the Supplement [IEE00] do not affect the basic description of the protocol (although some important time parameters are changed).

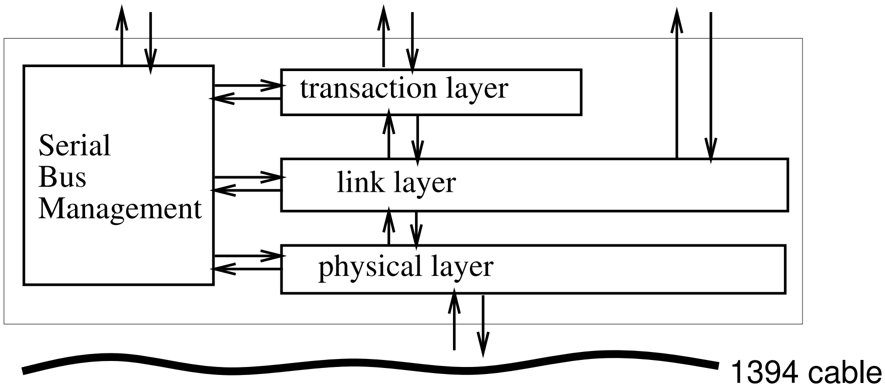

Fig. 1. Overall Architecture of IEEE 1394

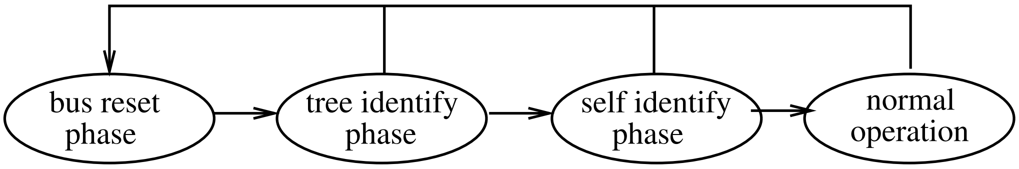

Fig. 2. Phases in PHY layer of IEEE 1394

be present and be the root. Therefore, any of the components may be the root[2], and a new root is elected every time the network configuration changes. The Tree Identify Phase elects the new root.

The description of the whole bus was published as an IEEE standard in 1995 [IEE95], and amended in the Supplement of 2000 [IEE00]. These documents use a combination of informal English text, state transition diagrams, and C code to describe the behaviour of the bus. We present the protocol informally, but in detail, below. The authors of the other papers in this collection worked directly from the standard IEEE 1394 documentation, including the C code which describes the activity in each state more formally and concretely. Figure 4-23 Tree-ID state machine and Table 4-45 Tree-ID actions and conditions reprinted in the Appendix with permission from IEEE Std 1394-1995 “IEEE Standard for a High Performance Serial Bus” Copyright©1996, by IEEE. The IEEE disclaims any responsibility or liability resulting from the placement and use in the described manner.

For analysis purposes, the liveness and safety properties required of the protocol are:

1. A root is eventually chosen.
2. Only one root is chosen.

Further, the protocol is designed for use on connected networks, will correctly elect a root if the network is acyclic, and will report an error if a loop is detected.

2.2. Tree Identify Phase

On entering the Tree Identify Phase all nodes in the network have equal status. They have only information about themselves; in particular, they know which of their ports are connected to other nodes. This is illustrated in Figure 3; there are connections, shown by solid lines, but no tree structure. Note that each link connects exactly two ports. Note also that nodes may actually enter the Tree Identify Phase at different times, caused mainly by the difference in times at which they signalled the preceding bus reset. We shall not go into

[2] Actually, not all nodes may have the capabilities to be cycle master. If a node turns out to be root and not capable of being cycle master, other 1394 functionality tries to find a more suitable candidate, sets the FORCE_ROOT flag at that node, clears the FORCE_ROOT flags at all other nodes and forces a new bus reset. The implications for the Tree Identify Protocol of setting this flag are discussed in Section 2.2.3.

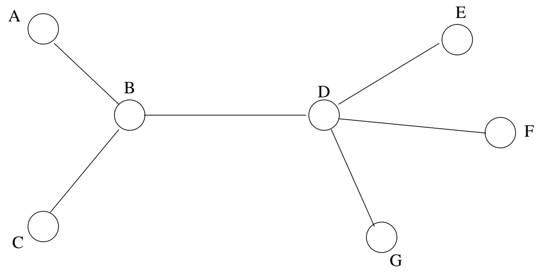

Fig. 3. Sample Network Topology

signal	*meaning*
IDLE	explicitly transmit "no message"
TX_PARENT_NOTIFY	transmit "be my parent"
RX_PARENT_NOTIFY	receive "be my parent"
TX_CHILD_NOTIFY	transmit "you are my child"
RX_ROOT_CONTENTION	receive "root contention"
RX_PARENT_HANDSHAKE	receive acknowledgement "I am your parent"
RX_CHILD_HANDSHAKE	receive acknowledgement

Fig. 4. Signals in the Tree Identify Phase and their interpretation

the details here; see the contribution of Romijn which deals specifically with this timing problem [Rom02]. All nodes behave in essentially the same way, modulo differences between manufacturers (which should still comply with the IEEE standard). This ensures that the election protocol is unbiased in the normal case. Special cases, including the use of a parameter (FORCE_ROOT) to bias the choice, are described in Section 2.2.3. The Tree Identify Protocol operates by constructing a spanning tree of the network, through nodes interacting with their neighbours.

Messages between nodes are defined in [IEE95, Tables 4-27 and 4-28] as combinations of voltages on wires. All signals of the Tree Identify Phase are shown in Figure 4, together with their intended meaning.

Signals are continuous, allowing new signals to be composed by combining existing signals, as illustrated in Figure 5. In fact, all signals starting RX are such combined signals. Note that, due to signal delay, what node A receives on a port is actually the result of what node A is currently transmitting and what the neighbour of A has been transmitting *a little while ago.*

As we shall see, the authors have used a variety of approaches to modelling communication, necessarily abstracting from the actual physical implementation. We discuss the implications of these choices in Section 4.

2.2.1. The Tree Identify Phase

The basic operation of the Tree Identify Phase is shown in Figure 6, which is based on [IEE95, Figure 4-23]. In the simplest situation, a node will move through the three states *T0: Tree-ID Start*, *T1: Child Handshake* and *T2: Parent Handshake*, and pass on to the Self Identify Phase. Broadly, the decision to move from one

A transmits	*B (neighbour of A) transmits*	*resulting signal on the wire connecting those nodes*
IDLE	IDLE	IDLE
IDLE	TX_PARENT_NOTIFY	RX_PARENT_NOTIFY
TX_PARENT_NOTIFY	TX_PARENT_NOTIFY	RX_ROOT_CONTENTION
TX_PARENT_NOTIFY	TX_CHILD_NOTIFY	RX_PARENT_HANDSHAKE
IDLE	TX_CHILD_NOTIFY	RX_CHILD_HANDSHAKE

Fig. 5. Combined Signals in the Tree Identify Phase

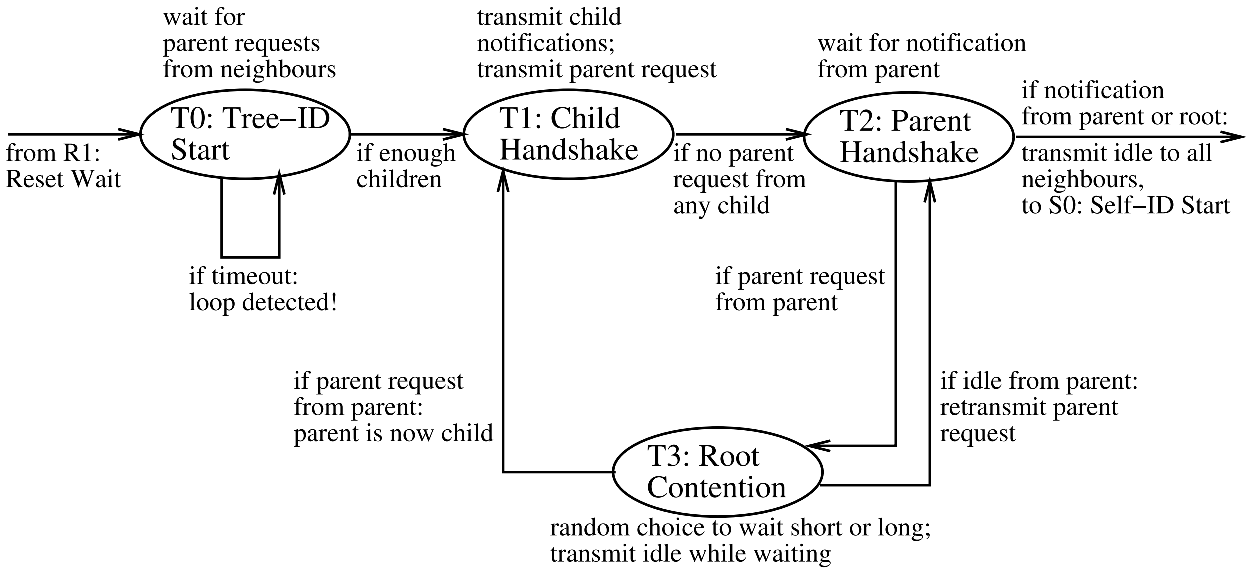

Fig. 6. States of the Tree Identify Phase

state to the next is based on the number of neighbour nodes requesting to be a child, c, the total number of neighbours, n, and interactions with those neighbours.

The node enters *T0: Tree-ID Start* from the Bus Reset Phase. If $n - c > 1$ in *T0* the node waits in *T0* for RX_PARENT_NOTIFY, "be my parent" requests, from its neighbours (potential children). As each request arrives, it is noted that that neighbour is now a child, and c is updated. When $n - c \leq 1$ the node transitions to state *T1: Child Handshake*.

Upon entering state *T1: Child Handshake*, the node starts transmitting TX_CHILD_NOTIFY, "you are my child" acknowledgements, to all children. In the case that $n - c = 1$ (in state *T1*) then the node has one neighbour which did not transmit a "be my parent" request. It asks to be a child of that neighbour by transmitting its own TX_PARENT_NOTIFY, "be my parent" request, on that port.

Then, in state *T1* the node waits for a handshake with each of its children. The handshake corresponds to receiving an RX_CHILD_HANDSHAKE, "I know you are my parent" acknowledgement. This signal is produced by the combination of TX_CHILD_NOTIFY from the parent and IDLE from the child. Upon reception, the node transitions to state *T2: Parent Handshake.* The IDLE signal from the child means it has transitioned to state *S0: Self-ID Start*, which is not described here. So the children of a node must finish the whole Tree Identify Phase before the node itself can move beyond state *T1*.

In state *T2: Parent Handshake* the node waits only for a handshake with its parent. The handshake corresponds to receiving an RX_PARENT_HANDSHAKE, "I am your parent" acknowledgement. This signal is produced by the combination of TX_PARENT_NOTIFY from the child and TX_CHILD_NOTIFY from the parent. Upon reception, the node transitions to state *S0: Self ID start*, and starts transmitting the IDLE signal to all its neighbours. It takes no further part in the election; it knows its children and its parent. One node will make this transition instantly, because it has $n - c = 0$, has no parent with whom to perform a handshake and is therefore the root node.

In the case where $n - c = 1$ in *T0* (i.e. leaf nodes), $n - c \leq 1$ is true immediately and the nodes transition from *T0* without waiting for parent requests. The spanning tree over the network is therefore built from the leaves inwards.

2.2.2. Example

To illustrate the tree identify process, given the network in Figure 3 and assuming D is transmitting IDLE initially, the following sequence of signals might be observed on the link between node D and E.

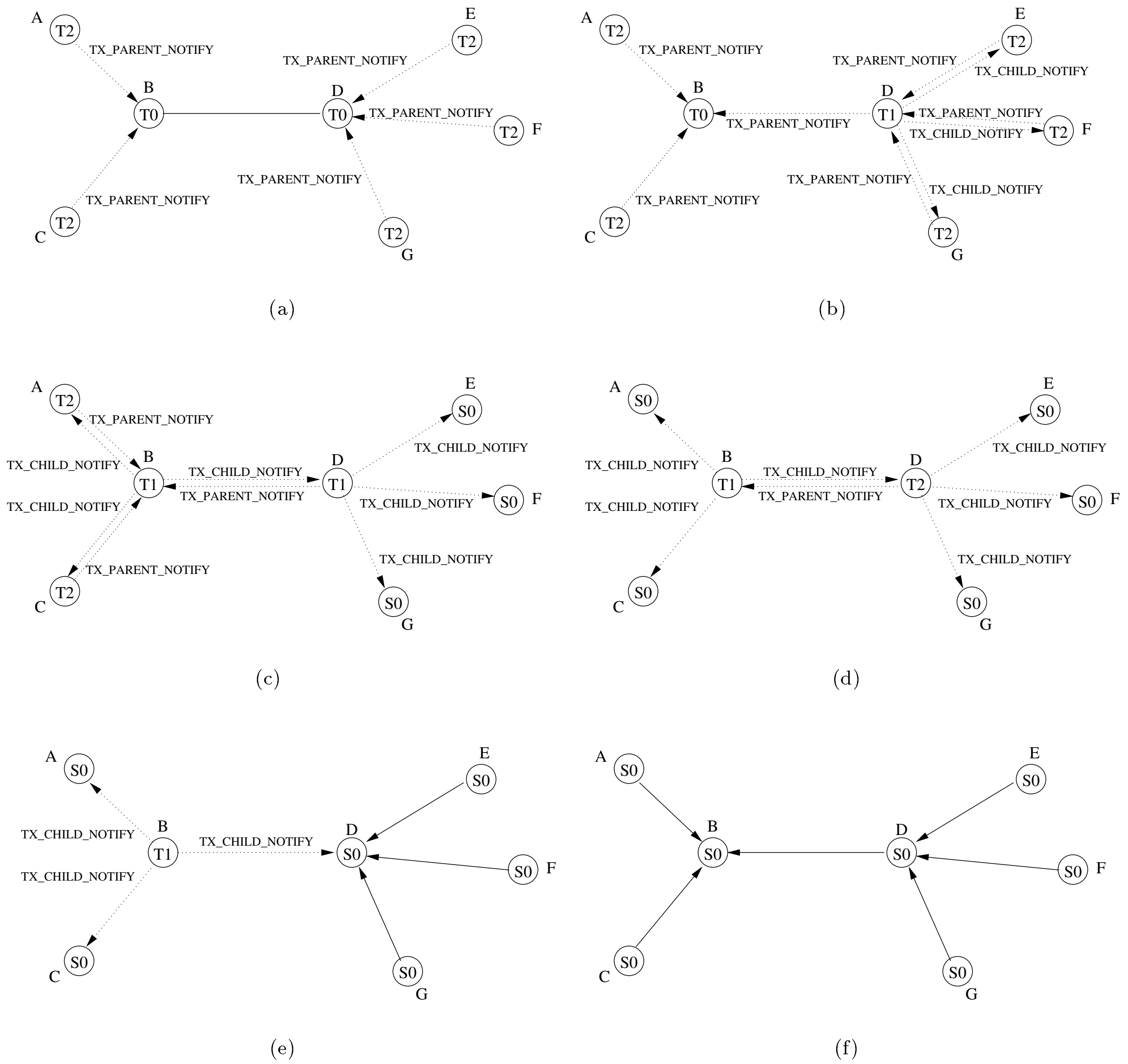

Fig. 7. Sample Tree Identify Process

from	*to*	*signal*	*meaning*
E	D	TX_PARENT_NOTIFY	D detects RX_PARENT_NOTIFY
D	E	TX_CHILD_NOTIFY	E detects RX_PARENT_HANDSHAKE
E	D	IDLE	D detects RX_CHILD_HANDSHAKE
D	E	IDLE	D and E have established parent-child relation

The signals and growing tree structure of the entire network are depicted in Figure 7. Here, labels inside nodes indicate the state of that node in the tree identify process. Dotted arrows are labelled with the signal that is sent in the direction of the arrow. The absence of an arrow indicates the IDLE signal. Solid arrows indicate that a parent-child relationship has been established, with the arrow pointing in the direction of the parent.

Problems occur if the expected confirmation is not received in state *T2*; this leads to *root contention*. Root contention is rather complex, and is dealt with in the next section.

2.2.3. Special Cases

Root contention Since signals experience a delay and may cross, requests are not atomic and *root contention* may arise (two nodes simultaneously transmit TX_PARENT_NOTIFY, "be my parent" requests, to each other yielding RX_ROOT_CONTENTION). This can only occur on one connection in the network, between the last two nodes that transition from state *T0* to *T1*, and the node that wins the contention will be the root of the tree.

Since only one node can be the root, the contention must be resolved; this is achieved through randomisation and timing. The standard specifies that each node chooses a random Boolean and waits for a long or short time, depending on the value of that Boolean. If, after the wait period is over, there is an RX_PARENT_NOTIFY signal from the other node ("be my parent") then this node becomes the root. If there is no such message then this node retransmits its own TX_PARENT_NOTIFY, "be my parent" message, and root contention may result again.

With reference to the state transition diagram of Figure 6, this works as follows. The root contention situation is recognised by each of the two nodes (although possibly at different times) in state *T2* upon reception of RX_ROOT_CONTENTION. At this point a node transitions to state *T3: Root Contention* and starts transmitting IDLE to the neighbour on the link in contention.

In state *T3: Root Contention* a random Boolean is drawn, and a timer is set to expire after a short time (ROOT_CONTEND_FAST, if the Boolean is 0), or a long time (ROOT_CONTEND_SLOW, if the Boolean is 1). The node then waits for the timeout. When this occurs, if the node receives IDLE on the link in contention it transitions to state *T2*, starts transmitting TX_PARENT_NOTIFY to the neighbour once again, and waits for a parent handshake, or root contention. If the node receives RX_PARENT_NOTIFY, it transitions to state *T1*, where it transmits the TX_CHILD_NOTIFY, "you are my child" acknowledgement, and waits for a child handshake. When a node ends up in state *T1* from *T3*, it is the winner of the contention and the root of the tree.

For example, given the network status in Figure 3 and assuming both B and D are transmitting IDLE initially, the following sequence of signals might be observed on the link between B and D.

B signals	D signals	meaning
TX_PARENT_NOTIFY	TX_PARENT_NOTIFY	B and D detect RX_ROOT_CONTENTION
IDLE	IDLE	back off and wait
		Assume D chooses short and is receiving IDLE from B.
IDLE	TX_PARENT_NOTIFY	D asks B to be its parent

At this point D will be in state *T2*. B is either still waiting in state *T3*, or has sent a parent request and is also in *T2*, but network delay means that D did not see the request before sending its own request again. In the first case the spanning tree shown in Figure 7(f) results, while in the second case contention results again.

The protocol here is similar to one presented in Distributed Algorithms [Lyn96, page 501] where a different method of contention resolution is used (the node with the highest identifier becomes the root). This approach will not work for IEEE 1394 because the nodes do not have usable identifiers during the Tree Identification Phase[3]. Such identifiers are not assigned until the Self Identify Phase. The protocols were developed independently.

Loop detection The protocol of the Tree Identify Phase automatically detects loop configurations. If there is a loop in the network, as in Figure 8, then for each node on the loop the transition from *T0* to *T1* does not get enabled because they will be waiting for 2 neighbours to communicate, so $n - c > 1$. Instead, a timeout generates an error message. The states resulting when the Tree Identify process gets stuck can be seen in Figure 8.

With reference to the state transition diagram of Figure 6, this works as follows. Upon entering state *T0*, a timer is started. When this timer reaches the value CONFIG_TIMEOUT, a loop is detected, and an error message for other 1394 layers is generated.

[3] The nodes do have hard-wired, unique identifiers but these are 64 bits long. Only the simple signals described above are transmitted during the Tree Identify Phase, and these can not carry such information.

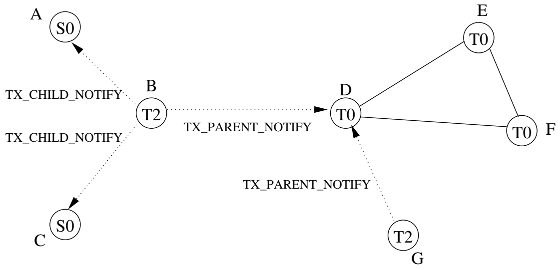

Fig. 8. Network topology with a loop

name	definition	1394 values	1394a values
CONFIG_TIMEOUT	detect a loop in T0	166.6μs – 166.9μs	166.6μs – 166.9μs
ROOT_CONTEND_FAST	the short time	0.24μs – 0.26μs	0.76μs – 0.85μs
ROOT_CONTEND_SLOW	the long time	0.57μs – 0.60μs	1.59μs – 1.67μs
FORCE_ROOT_TIMEOUT	wait in T0	83.3μs – CONFIG_TIMEOUT	83.3μs – 162.0μs

Fig. 9. Time constants in 1394 and 1394a

The FORCE_ROOT flag Lastly, the Tree Identify Protocol can be biased by use of the FORCE_ROOT parameter. This is a local Boolean parameter to the physical layer of each node, which can be set and cleared locally by the node itself, or remotely by any other node in the network. If the FORCE_ROOT parameter is set this builds in extra delay to state *T0*, potentially stopping the node from taking the *T0:T1* transition when $n - c \leq 1$ is true, and waiting for a timeout, at which point $n - c = 0$ should hold.

The intention is that if exactly one node in the network has FORCE_ROOT set, that node becomes the root of the tree in the Tree Identify Phase. The 1394 management layer ensures that this is the case. However, because of the dynamic plug-and-play functionality, a network may (temporarily) have more than one node with FORCE_ROOT set, in which case any node on a path between two biased nodes may become root in the Tree Identify Phase.

With reference to the state transition diagram of Figure 6, this works as follows. Upon entering state *T0*, a timer is started (the same timer as used for loop detection). The transition to state *T1* is enabled if either $n - c = 0$, or if $n - c = 1$ and FORCE_ROOT is not set, or if $n - c = 1$ and FORCE_ROOT is set and the timer has reached the value FORCE_ROOT_TIMEOUT.

2.3. Timing constraints of the Tree Identify Phase

The standard [IEE95, Table 4-32] and the amendment [IEE00, Table 8-14] define certain timing constants used in root contention. These are presented in Figure 9.

The standard also lays down certain physical limitations, which in turn affect timing values. Cable length is defined to have a maximum value of 4.5m. Signal velocity is defined to be less than 5.05ns/meter. These two determine the maximum delay in transmitting a message from one node to its neighbour of 22.725ns.

2.4. 1394-1995 versus 1394a-2000

The amendment [IEE00] to the IEEE 1394 standard [IEE95] contains some adjustments to the 1394 functionality. Any device being sold as IEEE 1394 compliant may now be compliant with either [IEE95] or as amended by [IEE00], hence with all the adjustments of that amendment. For the Tree Identify Phase the adjustments in the amendment concern only the timing constraints. This means that a 1394 network may consist of nodes, some of which have the 1394 timing constraints, with the others having 1394a [IEE00] timing constraints.

In all contributions to this special issue, networks are assumed to have uniform timing, i.e. either 1394 or

1394a time constants, but not both. We defer a more detailed explanation of the use of the timing constraints in these contributions to Section 4.

3. Points to Address

Taking our lead from the helpful questions posted by Abrial et al in the Steam Boiler case study [ABL96], we asked authors to evaluate their contribution by considering the following questions:

- Which parts of the system are specified, at which level, and in which way (informally, rigorously, formally)? Do you consider your chosen approach to be particularly expressive or particularly awkward for this sort of problem?
- Which parts of the system are analysed, and in what way (manual, informal, partially or fully automated)?
- Have assumptions about the architecture or the behaviour of the system been made explicit and are they documented?
- Is the solution easy to change? Following a change, can analyses be reused, or are all proofs invalidated?
- How long did it take:
  - How much time has been spent on producing the solution? Give the number of person months, if possible, for the various parts of the solution.
  - How much preparation is needed to become sufficiently expert in the used specification framework in order to be able to produce a solution to such a problem in that framework? Indicate the number of weeks of training which is believed necessary for an average engineer to learn the methods.
- What are the premises for a good understanding of the proposed solution?
  - Is a detailed knowledge of the used formalism needed?
  - Can an average engineer understand the solution (with no special training)?
  - How much time do you believe is necessary for the average engineer without knowledge of the used specification methods to learn what is needed to be able to understand the solution? Could you discuss this solution with a non formal methods literate person, e.g. a customer?
  - Do you believe your solution provides a suitable alternative formalisation to the IEEE standard?
- Which of the other solutions are comparable with yours, and what are the major differences with respect to them? Which other solutions complement your solution and in what respects?

The answers to these questions are given by the authors in their respective papers. Below, we attempt to answer the the last question in more depth, i.e. how do the solutions compare to each other?

4. Results

The papers submitted after the workshop were refereed by at least one other workshop participant, and at least one external reviewer supplied by the Formal Aspects of Computing editorial team. This modified co-refereeing approach allowed the reviewing process to benefit both from the domain expertise of the workshop participants, and the independence of the external reviewers.

Eight contributions were accepted for publication. Seven of these are formalizations of the Tree Identify Protocol. Section 4.1 contains a brief overview of these. The eighth paper, by Stoelinga [Sto02], is a survey of formalizations of the strategy for resolving root contention that is used in the Tree Identify Protocol, and is regarded as an interesting companion to the other papers in this collection.

4.1. The Formalizations of the Tree Identify Protocol

For each contribution, we give the following information: the formalism that is used; the techniques and tools used to support that formalism; the aspects of the Tree Identify Protocol that are formalized; the properties that were verified. The discussion in this selection is very brief; the reader is encouraged to consult the papers directly for full explanations.

[**ACM**] Abrial, Cansell, and Mery [ACM02] use the **B Method** formalism, which is applied using an *event driven approach*. The technique used is proof-based development, which is supported by the tool **Atelier B**. The Tree Identify Protocol is modelled as a series of refinements, starting from an abstract description of the underlying graph. The models do not cover real-time behaviour, performance, or probabilistic issues, and the treatment of root contention is left at an abstract level. The correctness of the protocol is demonstrated by proving basic graph properties as well as proof obligations arising from refinement.

[**VPM**] Verdejo, Pita, and Martí-Oliet [VPMO02] use the formalism of **rewriting logic**. The techniques used are formal specification, execution of the specification, formal model-checking analysis, and formal proof. The proof was carried out manually, while the other techniques were supported by the tool **Maude**. The model consists of three descriptions, at different levels of abstraction, and covers both synchronous and asynchronous communication, and timing aspects of the protocol. The properties that were verified include safety, liveness, and total correctness.

[**CM**] Calder and Miller [CM02] use the modelling language **Promela** and the model-checking tool **SPIN**. A Perl script is used to automatically generate all network configurations within certain bounds, and model-checking is used to verify a number of safety and liveness properties, expressed in Linear Temporal Logic (LTL), for each of these configurations. There is a discussion of how these results can be extrapolated to any size of network with a particular shape.

[**SB**] Schuppan and Biere [SB02] also apply a model-checking approach, this time using various implementations of the tool **SMV**. The protocol is modelled using **state machines** and the properties to be verified are stated using LTL and **computation tree logic (CTL)**. Various models are checked, including some representing a specific network configuration, and some which leave the network topology unspecified. Timing constraints are modelled using counters, and the FORCE_ROOT flag is also represented. Bounded model-checking is used to verify a number of safety and liveness properties for the given configuration.

[**FS**] Fidge and Shankland [FS02] use the **probabilistic Guarded Command Language** to model the root contention aspect of the Tree Identify Protocol in a highly abstract fashion. Formal proof (conducted manually) is used to derive probabilistic and timing results about the root contention protocol. Note that time is not part of the model, but that timing results are derived from those on probability.

[**KNS**] Kwiatkowska, Norman, and Sproston [KNS02] also deal with probability, using the formalism of **probabilistic timed automata** and the technique of **probabilistic model checking**. Like the previous paper, they focus on the behaviour of the root contention protocol. The aim of their analysis is to calculate the probability of successful contention resolution within a particular deadline. Their analyses are carried out with the help of the tools PRISM and HYTECH, and their paper includes a comparison of the performance of these tools.

[**Rom**] Romijn [Rom02] uses the formalism of **timed automata** and the technique of model-checking, as supported by the tool **UPPAAL**. Her model concerns both the Bus Reset and the Tree Identify Phase of IEEE 1394 and is focussed on a specific, problematic network configuration. She demonstrates the presence of a bug in the Tree Identify Protocol: in certain situations, the protocol will falsely conclude that there is a loop present in the network, and will generate an error message accordingly.

4.2. Comparison of the Contributions

To aid the reader in comparing the above approaches, we summarise in the following table how (and if) particular issues are dealt with by each contributor. An "•" entry means that a feature is explicitly addressed or used in the paper. A blank means not relevant or not addressed. Some entries have an "◦"; these are explained below.

	[ACM]	[VPM]	[CM]	[SB]	[FS]	[KNS]	[Rom]
Which part of the protocol?							
Tree spanning	•	•	•	•			•
Root contention	◦	•	◦	•	•	•	
Probability					•	•	
Timing		•		•	◦	•	•
Force root		•		•			•
Loop detection		•	◦	•			•
Method of Analysis							
Manual Proof		•	•		•	•	
Refinement	•						
Model Checking		•	•	•		•	•
Fixed Topology		•	•	•			•
- generated automatically			•				
Computer-Assisted Proof	•						

Which part of the protocol? Most of the contributions to this special issue choose to address the *tree spanning* algorithm as described in the Tree Identify Phase. Some ([KNS] and [FS]) focus more tightly on analysis of the sub-protocol of *root contention* resolution, using *probability* and/or *time* in the modelling. Other authors ([VPM] and [SB]) model root contention explicitly but do not particularly analyse this aspect, while others ([ACM] and [CM]) simply model contention resolution by nondeterminism or a single coin toss (this is represented in the table by a "◦" since contention resolution is modelled rather abstractly).

The model can be made more realistic by incorporating details such as probabilistic choice, or timing; the point then is to carry out an analysis using that extra detail, as is done for probabilities in [FS] and [KNS]. The contributions dealing with timing do so in different ways.

In the table, [FS] has a "◦" for time, since time is not included in their model, but the final analysis makes some comments about time. In [SB], and [VPM], counters over the domain of the natural numbers are used. In [SB], one counter per node is incremented from zero to the timeout value in one-tick steps, where one tick is exactly the delay of a message. The model is synchronous, in that each node in the network must perform exactly one transition (possibly without effect) per time tick. The timeout values used in [SB] are not related to the values of the standard. In [VPM], there is one counter for each message being transmitted, and per node one counter for the FORCE_ROOT flag (if set) and one for loop detection. All counters are decremented from the timeout value to zero in delay steps which are as large as possible, but minimally 1ns. The delays steps are performed only when no other step is possible. The timeout values for the FORCE_ROOT flag and for loop detection are within the 1394 value ranges. The timeout values for communication delay are constrained only in the analysis, and not by the 1394 physical limits. In [KNS] and [Rom], clocks over the domain of the real numbers are used, corresponding directly to the timers in the 1394 state machines. Both contributions assume one rate for all clocks. In [KNS], the timeout values are from 1394a. The communication delay varies: in some experiments this is the value as constrained by the 1394 physical limits (rounded up somewhat), and in some it is the larger, maximal value for which root contention was earlier shown to be correct (see also the survey contribution [Sto02]). In [Rom], the timeout values are from 1394a. The communication delays are not modelled, but it is argued that these are not significant for the error that is shown.

The overall behaviour of the protocol is affected by the FORCE_ROOT parameter. This is ignored by many authors. Although FORCE_ROOT affects timing in the protocol overall, and is intended to influence the choice of root, it does not, for example, affect the time to resolve root contention, and it should not affect the fact that a root is chosen. However, see [Rom] for a particular situation in which it has an effect on *loop detection*.

Loop detection as implemented in the standard is explicitly modelled by some ([VPM], [SB] and [Rom]) and plays an important part in the analysis. [CM] uses a more abstract means of modelling loop detection: in the analysis a stronger condition is checked by detecting infinite paths (in which the node does not make the desired transition, *T0* to *T1* in this case). This approach is represented in the table by a "◦". Others, such as [ACM] and [FS], describe the system at a more abstract level, and assume the network is acyclic.

Method of Analysis Although modelling the system can on its own be very useful, the advantage of using a formal method is that the modelling language is supported by analysis and reasoning techniques, and often tools to automate reasoning. A variety of analysis techniques were used. While [FS] made a *manual proof*, all other authors used tools to support their reasoning. In some cases, manual proof supported the abstractions made in the model checking (as in [KNS]), or extended the results given by the model checking (as in [CM]). [ACM] are unique in using a refinement approach to the problem. This is supported through formal *computer-assisted proof*.

Most approaches use some form of *model checking*. An important question for those using model checking relates to the range of models checked; is a single, *fixed topology* checked, or is the analysis somehow more abstract? The analysis done by [Rom] is focussed on a specific, problematic network configuration, and the question of generalising the results is not tackled. By contrast, although the basic method of model checking used by [VPM], [SB] and [CM] uses a fixed topology, they all also generalise their results in some way. [CM] for example uses Perl scripts to *automatically generate* configurations up to a fixed size and run the model checker automatically. In addition, manual proof is used to generalise the result to all networks of a particular form (of any size). [VPM] uses model checking as a prototype stage, building confidence in the description before a full (manual) formal analysis is carried out. Similarly, [SB] go on to further model checking using a *symbolic* model checker to check networks of unspecified topology. They also use a C preprocessor to help them set up particular topologies for checking. [KNS] also use a symbolic model checker, since using probabilities adds to the complexity of the model checking problem.

Other Modelling Issues Finally, we note that several authors make modelling decisions which diverge to some degree from the IEEE description of the protocol. For example, [VPM] model messages as discrete, and assume that the network is secure - no messages are lost or corrupted. This is a useful abstraction used by many authors, and makes the protocol simpler; however, in some models, the effect of assuming no loss is that there is no "child acknowledgement" phase, and the parent node spends less (or even zero) time in *T1*. In contrast, [SB] and [Rom] model line *states*, rather than discrete messages.

Another simplification is the decision of [CM] to model contention resolution in one step. In a model checker it is important to reduce state space as much as possible. In this case, adding more attempts to resolve contention would not have affected the validity of the properties checked, but would have seriously affected the performance of the model checker. Moreover, if an artificial limit is to be imposed on a value, then it makes sense to choose as small a value as possible.

A different modification of the root contention procedure can be seen in [VPM] where contention resolution is modelled in an eager way. That is, rather than waiting the specified time and then checking for a message, the node constantly monitors for a message while waiting. [VPM] allows the wait to be aborted early, meaning that a contention round may take less than the time specified in the standard. This does not affect the final result, i.e. the same node will become root; however, the time taken to resolve contention is likely to be lower than in the standard.

In fact, the crux of successful formal modelling is choosing a sensible abstraction. The aim is to remove detail which unnecessarily complicates the model or reasoning about that model, but without throwing away crucial information. Some of the models presented are relatively high level: they view the network as a whole, and simply model the changes from one network state to another. [FS], for example, does not make the topology of the network explicit, therefore modelling all state changes over all possible network topologies. The approach taken by [ACM] is similarly general, though in their final refinement the global view of the network is augmented with local views from the point of view of each node.

One of the benefits of a comparative case study like this one is that it helps to draw out the differences and similarities in approach of various formal methods (and their proponents).

4.3. Conclusions

In an ideal world we would like to see more widespread use of formal methods in developing standards such as the IEEE 1394; however, the people who develop such standards often have no training in formal methods and are perhaps not yet appreciative of the benefits of using them. As part of a drive to promote the use of formal methods, comparative studies such as this have an important role to play. For formal methods practitioners this study is offered as a benchmark, allowing them to apply their particular language, method and tools to a well understood but tricky problem. For industrialists, we hope that this issue of Formal

Aspects of Computing will be educational, allowing them to use their understanding of the basic problem to gain understanding of the formal approaches being applied.

The contributions contained herein span a range of different techniques and styles of abstraction. Many techniques used previously in the literature have been event based models; while there are some more of those here, there are also state based models and property based models. More than anything, the papers which follow demonstrate that there is no ideal formalisation which meets all needs (even for this relatively small example). Each approach described here has something new to offer, some aspect in which it is particularly useful. For example, the benefits of refinement are shown in the contribution of [ACM], allowing the model to be developed highly abstractly at first. The model of [FS] is also rather abstract, but this allows concentration on the probabilistic aspects of the protocol while ignoring other concerns. In contrast, the analyses of [KNS] and [Rom] are much more detailed, bringing out particular timing and probabilistic details. The model checking approaches of [CM], [SB] and [VPM] allow the model to be executed in some sense, which may be a more appealing technique for industrialists more used to programming.

Formal approaches are most effective when applied in tandem with the system development process, so that bugs can be detected early. In this case, the IEEE 1394 and 1394a standards were developed without the benefits of formal methods; therefore all formal analysis is post-hoc, with the disadvantage that there is no possibility to change the standard when errors are discovered. Since the standards have been fixed for some time, and explored by others fairly extensively, we did not expect any bugs to turn up. We were therefore surprised that a significant yet rare error situation was detected by [Rom]. Other authors have identified areas where the standard is ambiguous, or where there is a seeming inconsistency between different areas of the standard. We are grateful to one of the anonymous referees for alerting us to an ambiguity of this kind. We hope that the experience of work on formalising and analysing the Tree Identify Protocol will be useful in future efforts to incorporate the use of formal methods in the development of such standards.

References

[ABL96] J-R. Abrial, E. Börger, and H. Langmaack. *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*. Number 1165 in Lecture Notes in Computer Science. Springer-Verlag, 1996.

[ACM02] J-R. Abrial, D. Cansell, and D. Mery. A mechanically proved and incremental development of IEEE 1394 Tree Identify Protocol. *Formal Aspects of Computing*, 14(3):xx–yy, 2002.

[BG00] E. Boerger and R. Gotzhein. Requirements Engineering Case Study: Light Control. *Journal of Universal Computer Science*, 6(7), 2000.

[BMS96] M. Broy, S. Merz, and K. Spies. *Formal Systems Specification: The RPC-Memory Specification Case Study*. Number 1169 in Lecture Notes in Computer Science. Springer-Verlag, 1996.

[BSdRT00] G. Bandini, R.L. Lutje Spelberg, R.C.H. de Rooij, and W.J. Toetenel. Application of parametric model checking - the root contention protocol using LPMC. In *Proceedings of the 7th ASCI Conference, Beekbergen, The Netherlands*, pages 73–85, February 2000.

[BST00] G. Bandini, R.L. Lutje Spelberg, and W.J. Toetenel. Parametric Verification of the IEEE 1394a root contention protocol using LPMC. In *Proceedings of the 7th International Conference, on Real-Time Computing Systems and Applications (RTCSA 2000)*, pages 207–214. IEEE Computer Society Press, December 2000.

[CM02] M. Calder and A. Miller. Using SPIN to Analyse the Tree Identification phase of the IEEE 1394 High Performance Serial Bus (FireWire) Protocol. *Formal Aspects of Computing*, 14(3):xx–yy, 2002.

[D'A99] P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. PhD thesis, University of Twente, November 1999. Available via `wwwhome.cs.utwente.nl/~dargenio`.

[DGRV00] 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. *Formal Methods in System Design*, 16(3):307–320, 2000.

[FH01] M. Frappier and H. Habrias. *Software Specification Methods: An Overview Using a Case Study*. FACIT. Springer-Verlag, 2001.

[FS02] C. Fidge and C. Shankland. But What If I Don't Want To Wait Forever? *Formal Aspects of Computing*, 14(3):xx–yy, 2002.

[GV98] W.O.D. Griffioen and F.W. Vaandrager. Normed simulations. In *Proceedings of CAV'98*, number 1427 in Lecture Notes in Computer Science, pages 332–344, 1998.

[IEE95] Institute of Electrical and Electronics Engineers. *IEEE Standard for a High Performance Serial Bus. Std 1394-1995*, August 1995.

[IEE00] Institute of Electrical and Electronics Engineers. *IEEE Standard for a High Performance Serial Bus — Amendment 1. Std 1394a-2000*, June 2000.

[KNS02] M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic Model Checking of Deadline Properties in the IEEE 1394 FireWire Root Contention Protocol. *Formal Aspects of Computing*, 14(3):xx–yy, 2002.

[LL95] C. Lewerentz and T. Lindner. *Formal Development of Reactive Systems: Case Study Production Cell*. Number 891 in Lecture Notes in Computer Science. Springer-Verlag, 1995.

[Lyn96] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc, 1996.

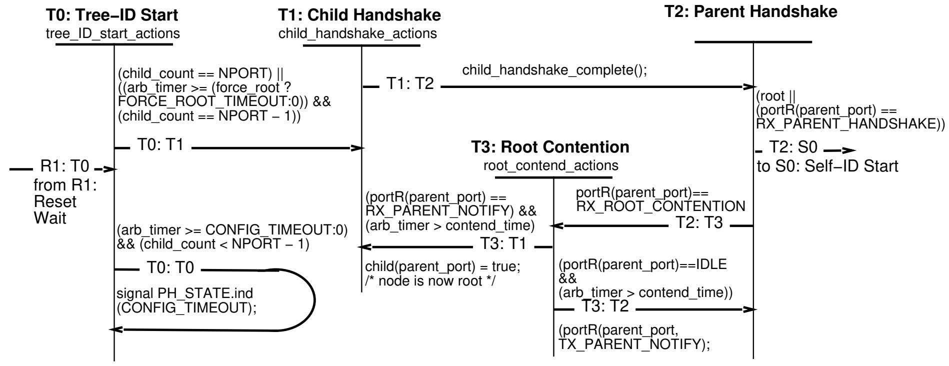

Fig. 10. The state machine of a node in IEEE 1394

[MS00] S. Maharaj and C. Shankland. A Survey of Formal Methods applied to IEEE 1394. *Journal of Universal Computer Science*, 6(11):1145–1163, 2000.

[Rom01] J.M.T. Romijn. A Timed Verification of the IEEE 1394 Leader Election Protocol. *Formal Methods in System Design*, 19(2):165–194, September 2001. Special Issue on the Fourth International Workshop on Formal Methods for Industrial Critical Systems.

[Rom02] J.M.T. Romijn. False loop detection in the IEEE 1394 Tree Identify Phase. *Formal Aspects of Computing*, 14(3):xx–yy, 2002.

[SB02] V. Schuppan and A. Biere. Verifying the IEEE 1394 FireWire Tree Identify Protocol with SMV. *Formal Aspects of Computing*, 14(3):xx–yy, 2002.

[SS01] D.P.L. Simons and M.I.A. Stoelinga. Mechanical Verification of the IEEE 1394a Root Contention Protocol using Uppaal2k. *Springer International Journal of Software Tools for Technology Transfer*, pages 469–485, 2001.

[Sto02] M.I.A. Stoelinga. Fun with FireWire: a comparative study of formal verification methods applied to the IEEE 1394 Root Contention Protocol. *Formal Aspects of Computing*, 14(3):xx–yy, 2002.

[SV99] M.I.A. Stoelinga and F.W. Vaandrager. Root Contention in IEEE 1394. In *5th AMAST Workshop on Real-Time and Probabilistic Systems*, LNCS 1601, pages 53–74. Springer-Verlag, 1999.

[SV01] C. Shankland and A. Verdejo. A case study in abstraction using E-LOTOS and the FireWire. *Computer Networks*, 37:481–502, 2001.

[SvdZ98] C. Shankland and M. van der Zwaag. The Tree Identify Protocol of IEEE 1394 in μCRL. *Formal Aspects of Computing*, 10:509–531, 1998.

[VPMO02] A. Verdejo, I. Pita, and N. Marti-Oliet. Specification and Verification of the Tree Identify Protocol of IEEE 1394 in Rewriting Logic. *Formal Aspects of Computing*, 14(3):xx–yy, 2002.

A. Excerpts from the IEEE standard

The main description of the tree identify protocol is in Section 4.4.2.2 of the IEEE standard [IEE95]. The major transitions of the protocol are described using state machine diagrams, and the actions of the states themselves are described by C code and informal text. Other information about the protocol (such as the physical realisation of messages as voltages, and timing information can be found elsewhere in Section 4 (Cable PHY Specification). There is an informative example of operation of the protocol in Annex E of the standard [IEE95].

For reference purposes we reproduce here the Tree-ID state machine diagram of Figure 4-23 of the standard [IEE95] and the accompanying C code of Table 4-45 of the standard [4].

```
void tree_ID_start_actions () {
int i, temp_count;
arb_timer = 0;
while (true) {
  temp_count = 0;
  for (i = 0; i < NPORT; i++)
    if (~connected[i] || portR(i) == RX_PARENT_NOTIFY) {
      child[i] = true;
      temp_count++;
    child_count = temp_count;
```

```
  }
}

void child_handshake_actions () {
int i;
root = true;
for (i = 0; i < NPORT; i++) {
  if (connected[i] && child[i])
    portT(i, TX_CHILD_NOTIFY);
  else if (connected[i]) {
    portT(i, TX_PARENT_NOTIFY);
    parent_port = i;
    root = false;
  }
}

boolean child_handshake_complete () {
int i;
for (i = 0; i < NPORT; i++)
  if (child[i] && connected[i] && (portR(i) != RX_CHILD_HANDSHAKE)
    return false;
return true;
}

void root_contend_action () {
int i;
contend_time = (random_bool() ? CONTEND_SLOW : CONTEND_FAST);
for (i = 0; i < NPORT; i++) {
  if (child[i])
    portT(i, TX_CHILD_NOTIFY);
  else
    portT(i, IDLE);
  }
arb_timer = 0;
}
```

Note that there appears to be a right brace missing from `tree_ID_start_actions` and `child_handshake_actions`. These omissions are copied from the standard, but their position can be inferred from the indentation of the code.