Four Formal Models of IEEE 1394 Link Layer

Hubert Garavel

INRIA – Univ. Grenoble Alpes

France

Bas Luttik

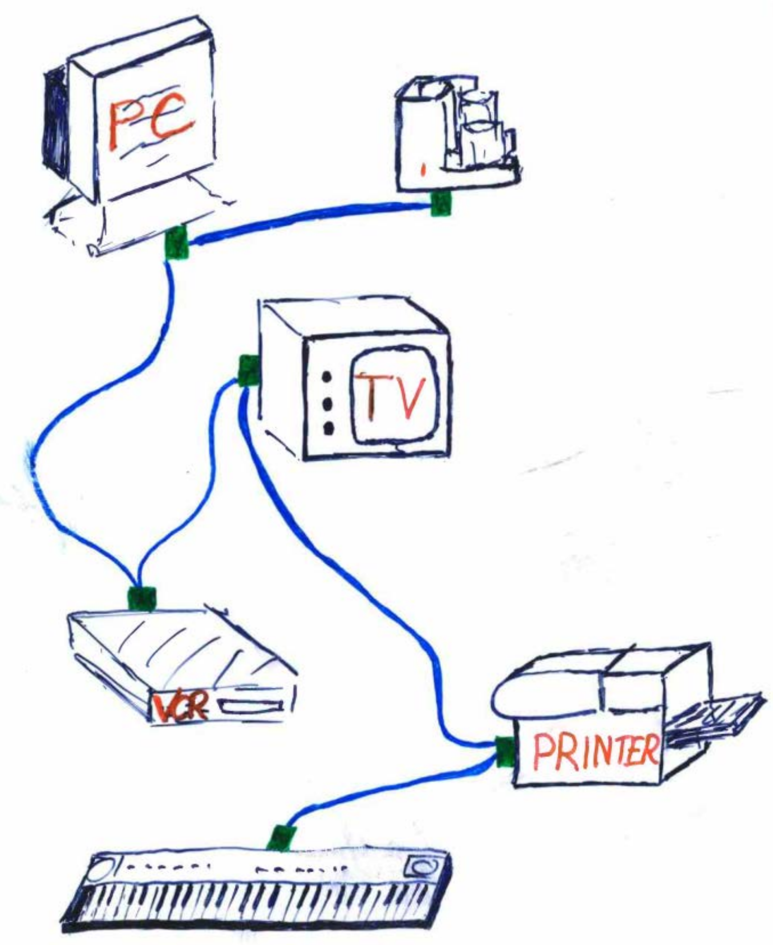
Eindhoven Univ. of Technology

The Netherlands

1. The FireWire bus

The FireWire idea

- High-speed serial bus
- Connect all computers and multimedia devices with the same thin cable
- Full-duplex transfers
- From 100 to 3200 Mbits/s
- Direct memory access
- Plug-and-play, hot swapping
- Power supply up to 30V-55W



FireWire: a 30-year history

- 1986: development initiated by Apple
- Many contributors: Hitachi, LG, Panasonic, Philips, Samsung, Sony, Texas Instruments, Toshiba, etc.
- 1995: IEEE 1394 standard (revised in 2008)
- 2000s: supported by BSD, macOS, Linux, Windows



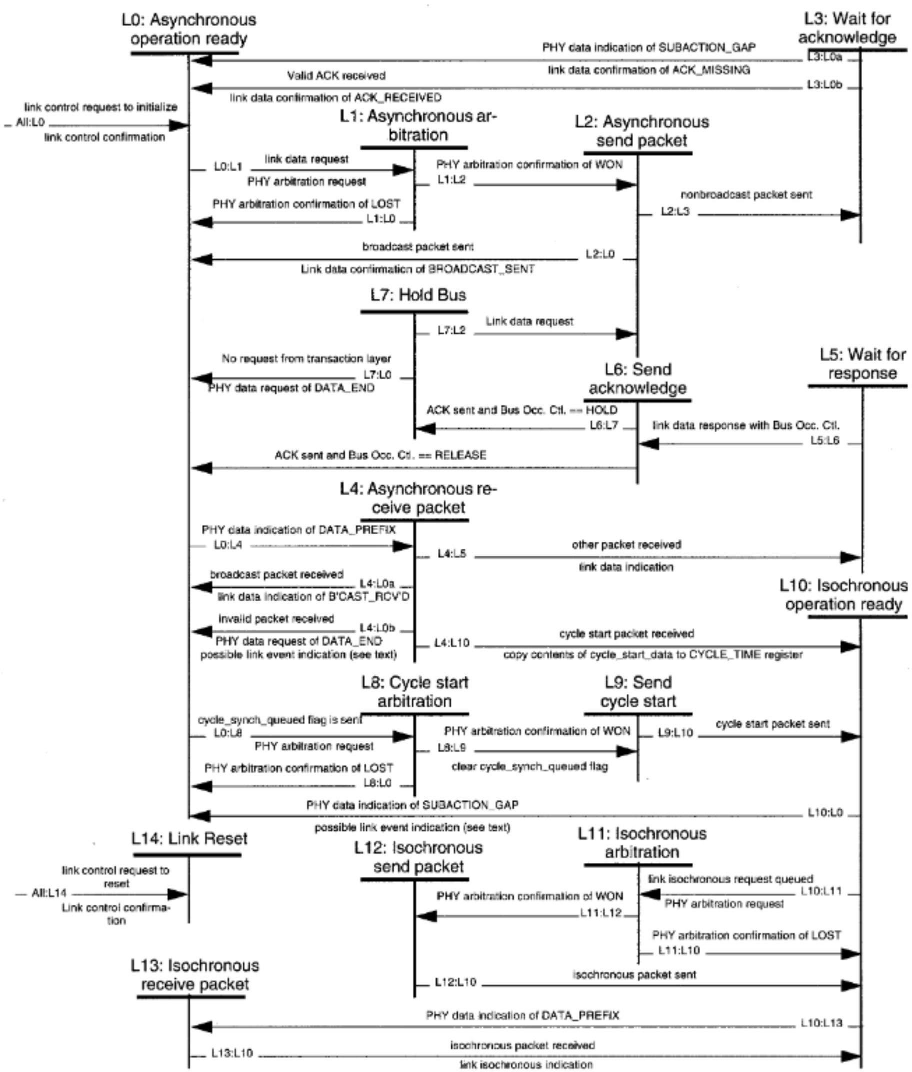

- But competition with USB-C and Thunderbolt
- 2016: last Apple product with FireWire

2. The IEEE 1394 protocol

IEEE 1394 standard

- A beautiful piece of engineering:
  - 1995 version: 384 pages
  - 2008 version: 906 pages
  - Many aspects: physical connectors, electric signals…
- Focus on the Link layer communication protocol
  - 40 pages of semi-formal descriptions
  - state machines / C++ code segments / English text with this order of priority
  - these descriptions are rather precise, but not totally

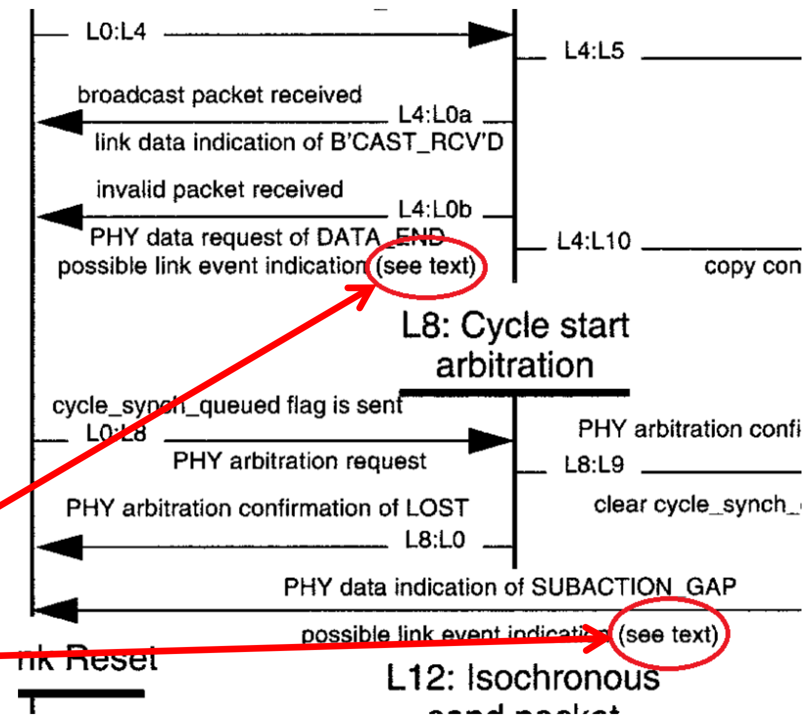
IEEE 1394 Link-layer state machine



- 14 "principal" states named L0, L1, ..., L13

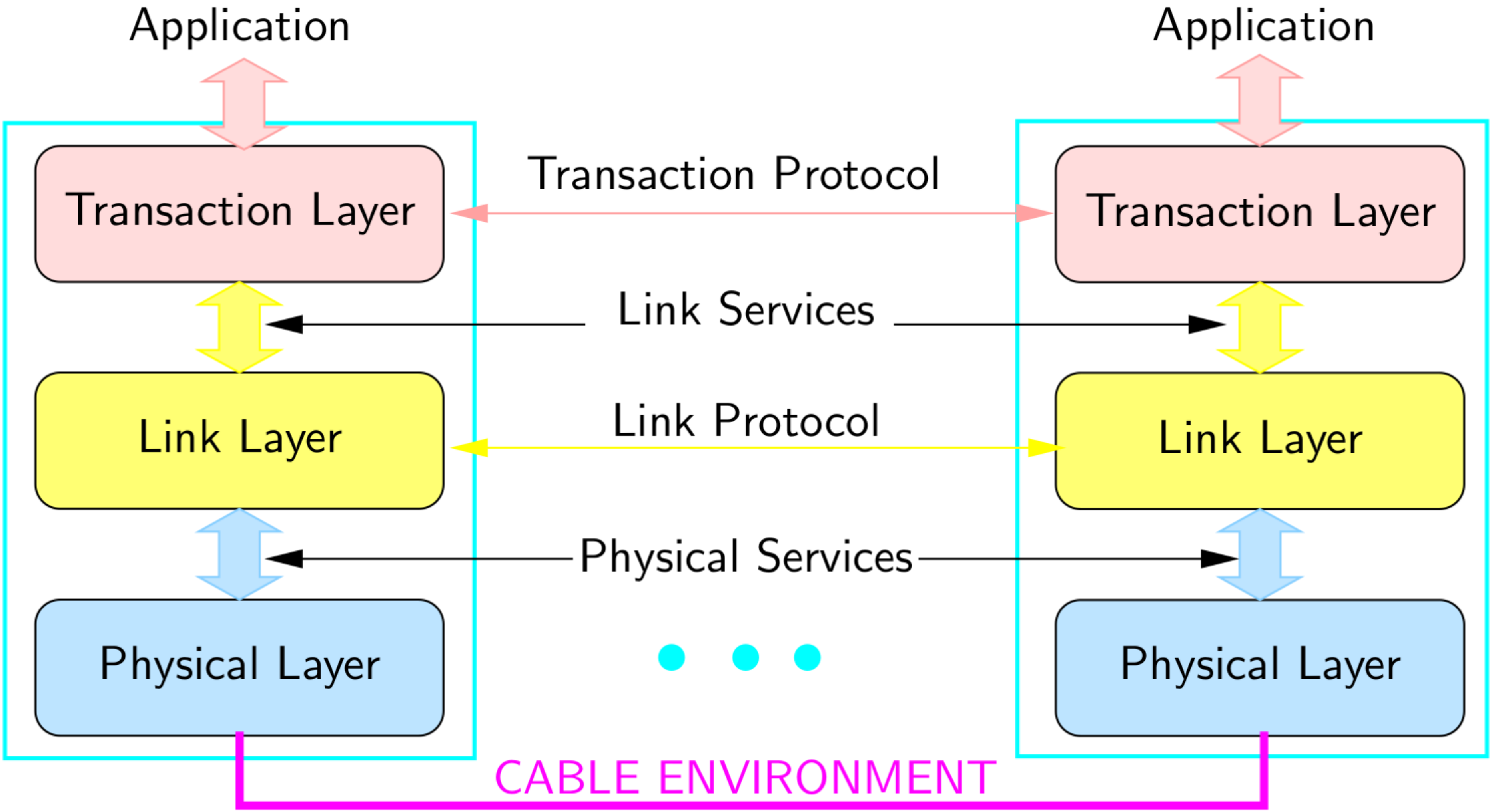
IEEE 1394 ambiguities

- The interconnection of state machines is not specified
- Actions are possible both on transitions and states
- State machines are incomplete and refer to informal English text



⇒ There is room left for formal methods

IEEE 1394 protocol stack



+ node controller (timeouts, reset) for all layers

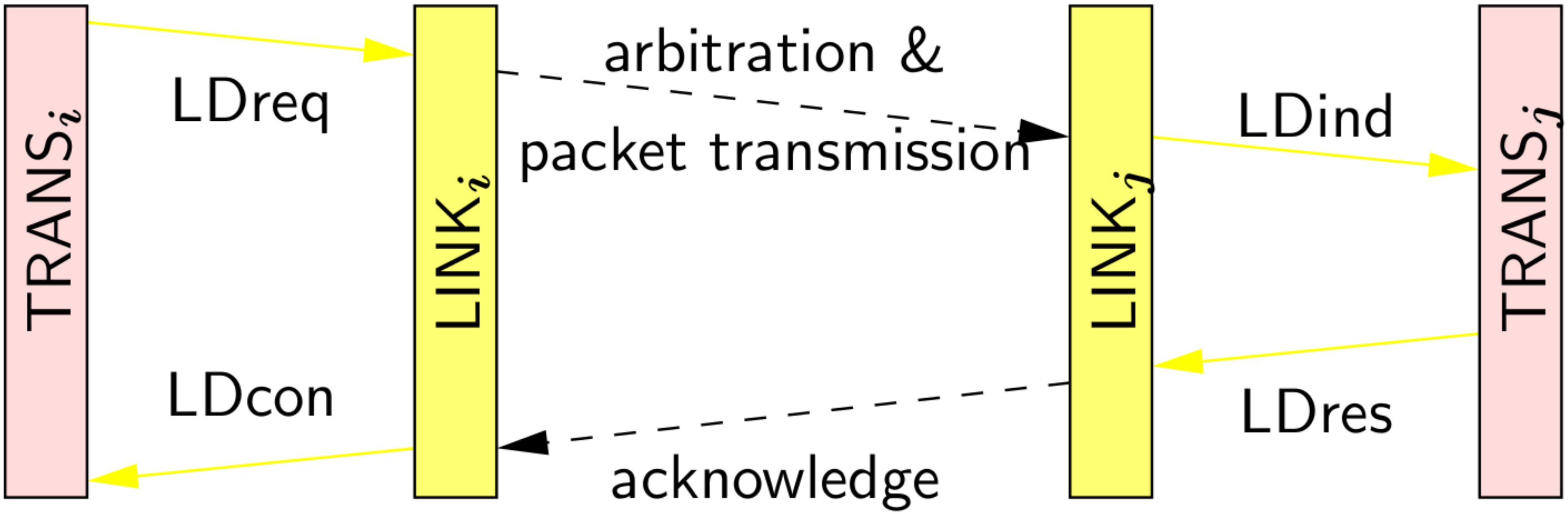
Transaction layer

- The TRANS layer provides the APPLI layer with three types of transactions:
  - READ: read data from another node
  - WRITE: write data to another node
  - LOCK: transfer to another node data to be processed, then transfer it back
- Transactions can be:
  - concatenated: response follows request immediately
  - split: response can be delayed

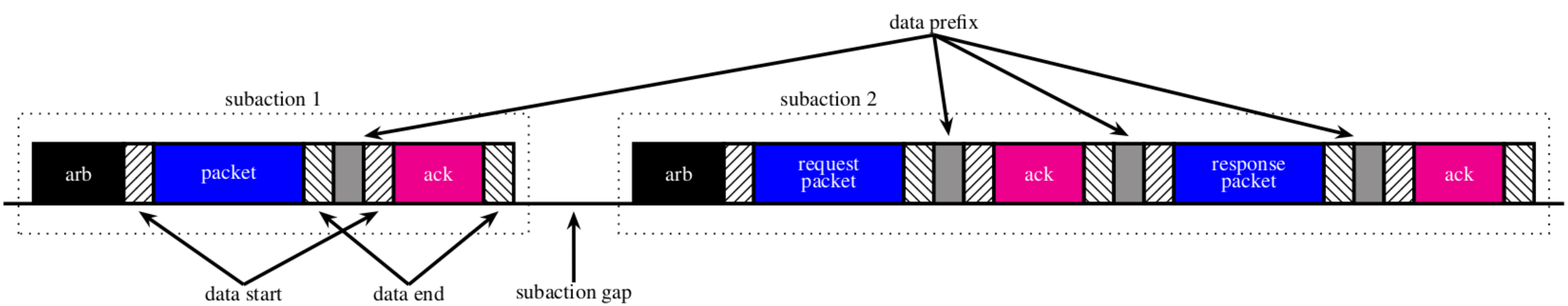
Link layer (1/2)

- Two types of data transfers:
  - isochronous mode (for multimedia):
    fast transfers of large amounts of data (audio/video)
    sent/received at constant rate (guaranteed bandwith)
    no acknowledgements
  - asynchronous mode (for computers):
    messages of arbitrary length
    sent at a lower priority
    acknowledgements from receiving nodes
- Either peer-to-peer or broadcast

Link layer (2/2)



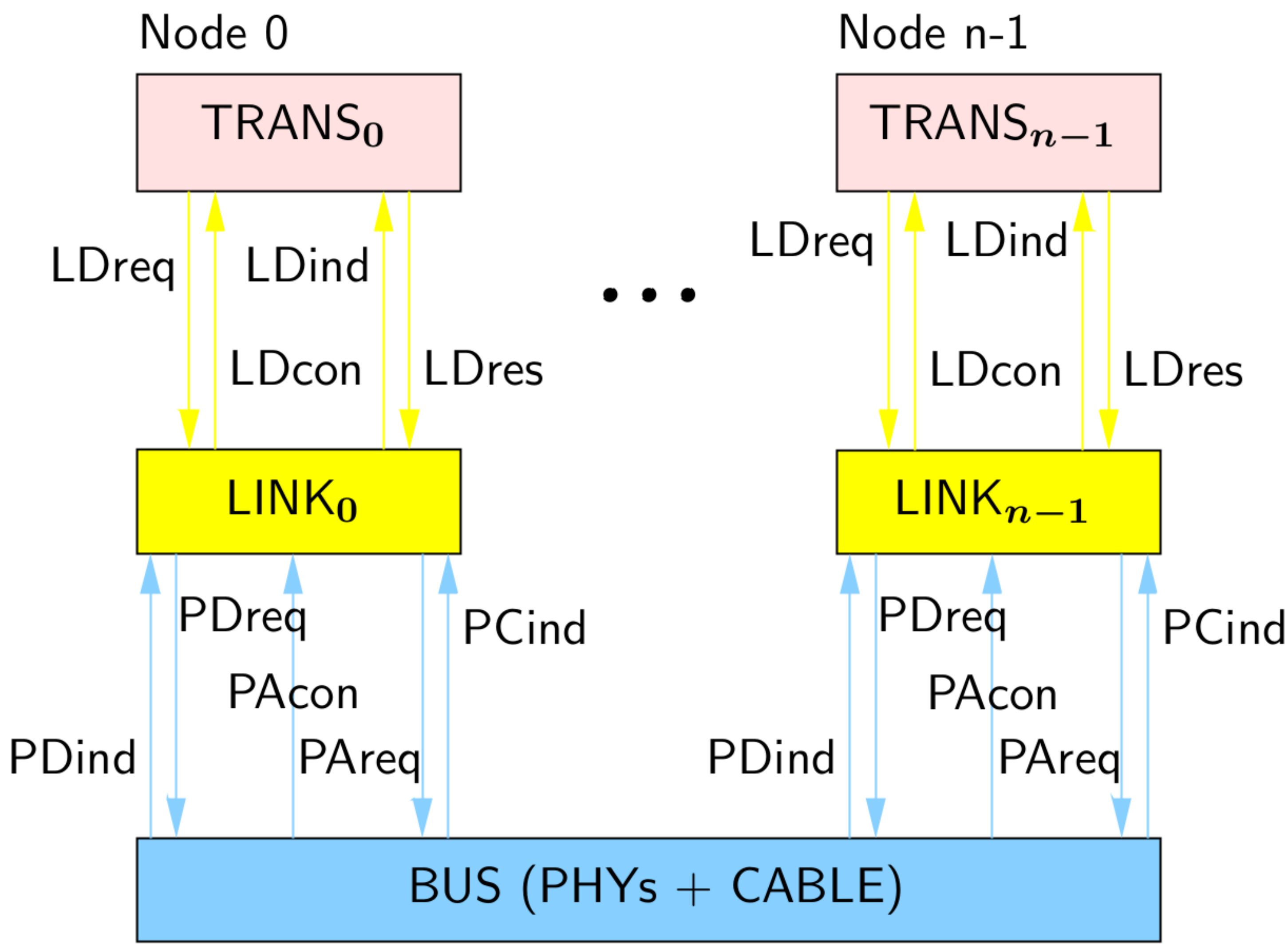
- Each subaction gathers one or two packets:



Physical layer

- The PHY layer converts link messages to signals
- It sends/receives signals on the cable
- It handles the loss or corruption of signals
- It also implements the arbitration protocol:
  - every second, 8000 arbitration slices (125 ms each)
  - isochronous transfers have priority
  - asynchronous transfers use the rest of the time slice
  - only one LINK can emit at a time
  - a LINK can emit at most once in each fairness interval

IEEE 1394 protocol events



3. The μCRL model

The μCRL model (1/2)

- Model written by Bas Luttik (1997)
  - feedback from H. Garavel, J. F. Groote, M. Sighireanu

- Features:
  - 809 non-blank lines (in the 1997 version of μCRL)
  - data types (term rewrite systems) are verbose
  - the MAIN process gathers *n* LINK entities and the BUS
  - the BUS represents *n* PHYSICAL entities and the cable

The μCRL model (2/2)

- Abstractions:
  - isochronous transfers are not modelled (too simple)
  - the model is untimed (no quantitative time)
  - the BUS is nondeterministic (signals lost or corrupted)
  - CRC checksums are not computed nor checked but error values to model lost / corrupted signals (i.e., Boolean abstractions)
- Verification:
  - Bas Luttik specified (in English) 5 involved safety and liveness properties of the Link layer

4. The LOTOS model

The LOTOS model (1/4)

- Model written by Mihaela Sighireanu (1997)
  - based on the μCRL model of Bas Luttik
  - same model written in two different languages:

    E-LOTOS (under standardization at the time)
    – model published in an STTT journal paper (1998)
    – one of the very few models written in E-LOTOS
    – no tool support

    LOTOS (standardized, supported by the CADP tools)
    – model used for verification by model checking
    – never published until MARS 2024

The LOTOS model (2/4)

- Features:
  - data types are much more concise than μCRL ones (predefined libraries for Bool and Nat, conditional rewrite rules, decreasing priority between rules)
  - the LINK and BUS processes of Bas Luttik are reused
- State-space explosion:
  - the state space of LINK and BUS is large, due to:
    - protocol complexity
    - fine granularity of signals
    - nondeterminism in the BUS

The LOTOS model (3/4)

- Data abstractions:
  - natural numbers in 0…*n* (where *n* = number of nodes)
  - DATA, HEADER, and ACK types reduced to one value
- Extra processes:
  - TRANS and APPLI processes to model upper layers
- 11 different scenarios:
  - Node 0 does one broadcast or point-to-point request
  - Each node does a broadcast or point-to-point request
  - Node 0 does *k* broadcast or point-to-point requests

All interesting cases are covered (split/concatenated…)

The LOTOS model (4/4)

- Further code simplifications by H. Garavel:
  - in 2005: the auxiliary C code was divided by 13 (from 2134 to 156 lines)
  - in 2023: the LOTOS code was reduced by 30% (from 2091 to 1385 lines) without loss of functionality and still preserving strong bisimilarity:
    - merged 2 TRANS processes into a parameterized one
    - merged 5 APPLI processes into a parameterized one
    - added a NODE process to factorize duplicated code

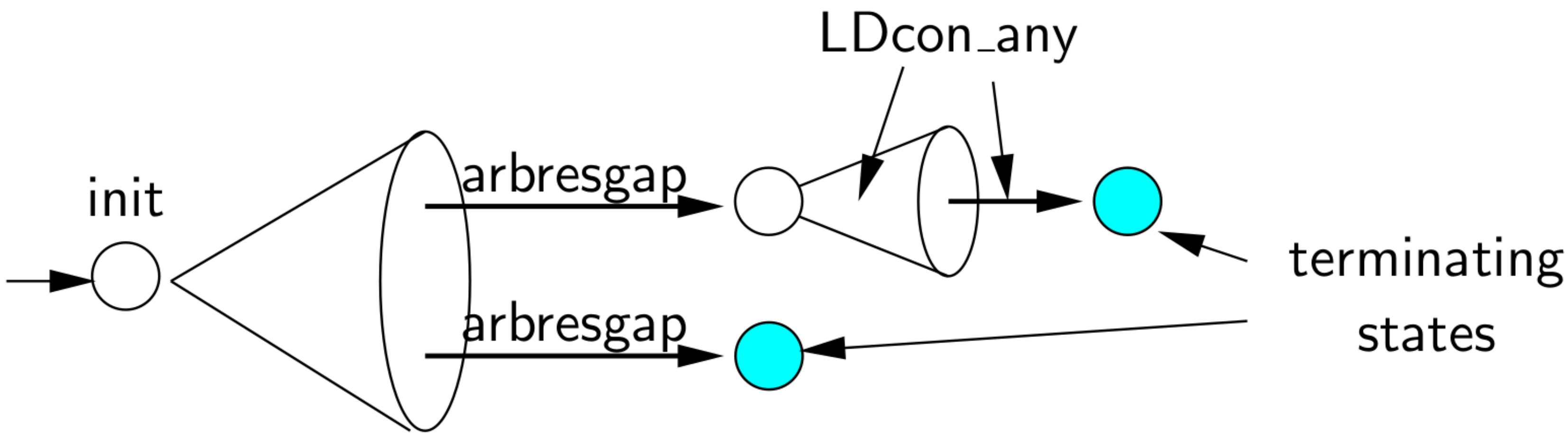
Verification of the LOTOS model

- The LOTOS models for the 11 scenarios were translated to LTSs (*Labelled Transition Systems*)
- Radu Mateescu formalized the 5 properties in the ACTL temporal logic [DeNicola & Vaandrager]
- These formulas were evaluated on all LTSs using the XTL tool of CADP
- Property 1 was violated in all scenarios

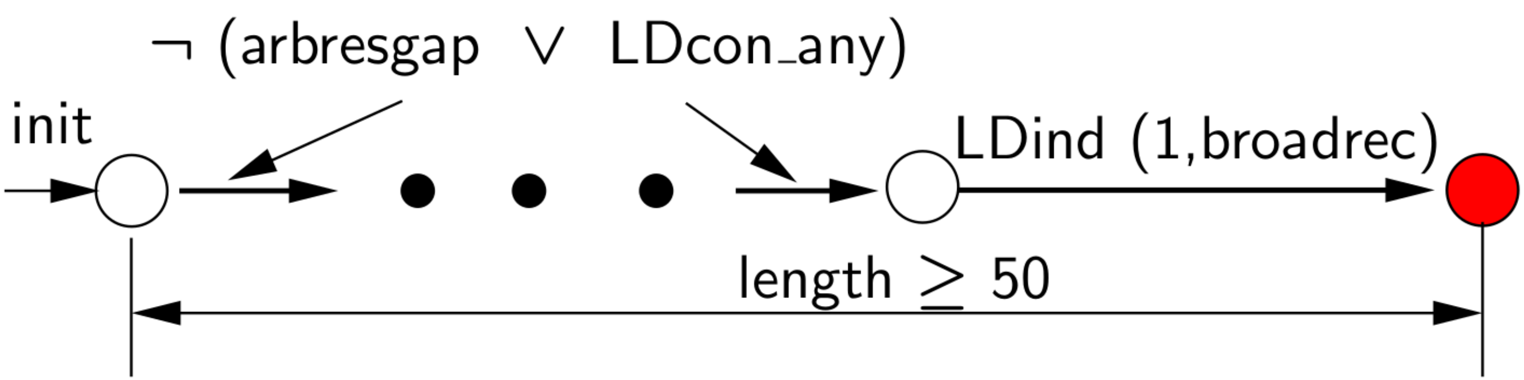
$$\mathbf{init} \Longrightarrow \neg\mathbf{EF}_{\mathbf{true}}\ \langle\neg(\mathtt{ARBRESGAP}\ \vee\ \mathtt{LDCON\_any})\rangle\ \mathbf{EF}_{\mathtt{LDCON\_any}}\ [\mathbf{true}]\ \mathbf{false}$$

Deadlock issue

- Expected "normal" termination

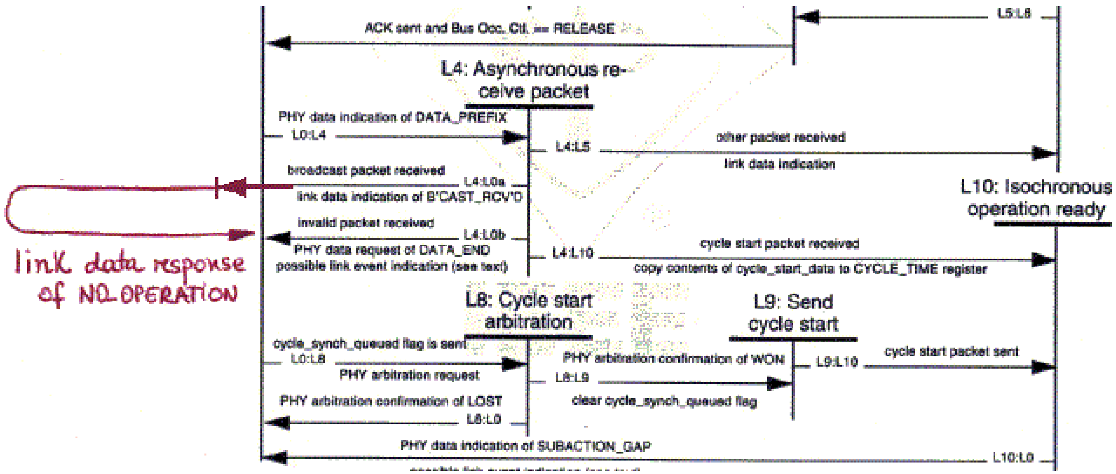


- Unexpected deadlock found after 50 events:



Two possible fixes

- The standard is wrong or, at least, ambiguous wrt the semantics of state-machine interconnection
- Solution A: handle unexpected event in LINK



- Solution B: modify TRANS to avoid this situation
  - 2 x 11 scenarios (with original and modified TRANS)

5. The mCRL2 model

The mCRL2 model

- Model translated from μCRL by J. F. Groote (2005)
- Features:
  - 60% smaller than the original μCRL model (327 non-blank lines of mCRL2, vs 809 lines of μCRL)
  - the size of data types was divided by 6.4 in mCRL2 (built-in types Bool and Nat, constructor types with automatic definition of equality, recognizer, and projection functions)
  - new syntax: A <| C |> B now noted C -> A <> B

6. The LNT model

The LNT model (1/2)

- Written in two successive steps (2022-2023):
  - systematic translation LOTOS → LNT (student project)
  - manual transformations to get readable LNT code:
    - – inline expansion of many auxiliary processes
    - – flattening nested **if-then-else** by adding **elsif** tests
    - – replacement of recursion by loops (**break**, **while**, **for**)
    - – factorization of similar code fragments, etc.
- Features:
  - LNT slightly more concise than LOTOS (~ 20%)
    774 non-blank lines of LNT vs 974 lines of LOTOS

The LNT model (2/2)

- Features:
  - 80% of LNT code is readable by non-experts
  - imperative style (write-many variables, assignments)
  - but also functional style (pattern-matching **case**)
  - partial functions, with explicit exceptions and **raise**
- Verification:
  - by model checking: the 5 ACTL formulas evaluate identically on LNT and LOTOS models
  - by equivalence checking: LTSs gerated from LNT and LOTOS are bisimilar (and have roughly the same sizes)

7. Conclusion

The FireWire case study

- A realistic problem:
  - at the interface between hardware (circuits and networking) and software (drivers and protocols)
  - a true success story of formal methods
  - model checking quickly found an unknown issue

- Semi-formal models are not enough:
  - (state machines + C code + text) may be ambiguous
  - even in an IEEE standard proofread by many experts

Four formal models of FireWire

- Rosetta stone of modelling languages:
  - evolution of formal methods over time:
    μCRL $\rightarrow$ mCRL2,   LOTOS $\rightarrow$ E-LOTOS $\rightarrow$ LNT
  - comparison of languages and specification styles
  - common example for benchmarking other languages

- Debate: different meanings of "minimality"
  - minimal languages (with small syntax/semantics)?
  - minimal models (faster to write, easier to read)
    using more complex / sophisticated languages

Acknowledgements

- Jan Friso Groote
- Charles Pecheur
- Marck-Edward Kemeh
- Judi Romijn
- Radu Mateescu
- Mihaela Sighireanu
- Laurent Mounier
- Bruno Vivien
- Oussama Oulkaid