Analyzing Zeroconf with Timed Automata

Biniam Gebremichael, Frits Vaandrager, Miaomiao Zhang

Radboud Universiteit Nijmegen

ARTIST2 Meeting V&V, Eindhoven, 20-21 April 2006

Motivation

Our society increasingly depends on correct functioning of (implementations of) communication protocols.

Standards that define these protocol are written in informal language, with frequent ambiguities, omissions and inconsistencies.

We can blame the engineers (for not using formal methods), the companies (for playing political games), but also ourselves, i.e., the formal methods researchers.

Formal Methods

There is ample evidence that formal (mathematical) techniques may help to improve quality of protocol standards. Still, they are rarely used in (the authorative part of) protocol standards:

- Engineers have difficulties with formal notations (they do like FSMs and C).
- Relationship between formal models and protocols often unclear ("model hacking")

Uppaal

Recently, the timed automata model checker Uppaal has been extended with C-like functions, and the verification engine has become much more powerful (e.g. due to symmetry reduction).

Can we use Uppaal for building a model of (a fragment of) a protocol standard that
(a) is easy to understand by engineers,
(b) comes as close as possible to informal text,
(c) may serve as basis for automatic verification?

Case Study: Address Configuration in Zeroconf

Protocol for dynamic configuration of IPv4 link-local addresses.

Standardized by IETF in RFC 3927.

Philosophy: internet should be like electricity, i.e., work when you plug in a cable.

Several implementations available, notably Bonjour from Apple.

See `www.zeroconf.org`.

Our Results

1. Simple Uppaal model of critical part of Zeroconf, almost good enough for inclusion in standard.
2. Very close correspondence between model and standard; only probabilistic aspects cannot be handled.
3. Several mistakes/ambiguities found in standard.
4. Manual verification easy, model checking difficult.
5. Several suggestions for further improving TA technology.

Mutual Exclusion

Zeroconf can be viewed as a distributed mutual exclusion algorithm in which the resources are IP addresses.

Zeroconf is similar to Fisher's mutual exclusion algorithm and makes essential use of timing. But whereas Fischer uses a shared variable for communication, Zeroconf uses broadcast communication.

Within Zeroconf, hosts do not aim at acquiring access to a specific CS; but just to one out of 65024 available CSs.

Assumptions about Underlying Network

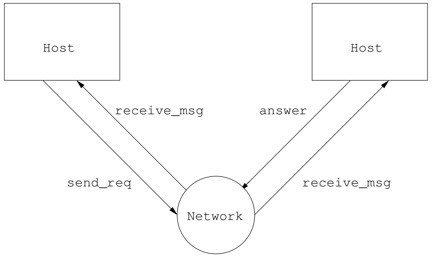
"This specification applies to all IEEE 802 Local Area Networks (LANs) [802], including Ethernet [802.3], Token-Ring [802.5] and IEEE 802.11 wireless LANs [802.11], as well as to other link-layer technologies that operate at data rates of at least 1 Mbps, have a round-trip latency of at most one second, and support ARP [RFC826]."

Address Resolution Protocol (ARP)

Widely used method for converting protocol addresses (e.g., IP) to local network (“hardware”) addresses (e.g., Ethernet).
Within Zeroconf all messages are ARP packets.

```
typedef struct{
HAType senderHA; // sender hardware address
IPType senderIP; // sender IP address
IPType targetIP; // target IP address
bool request; // is the packet a Request or a Reply
}ARP_packet;
```

Network Model



Several identical network automata in model. Each of these automata takes care of broadcasting single message (and reply) and has a clock to ensure roundtrip delay of 1sec.
We assume hosts handle incoming messages instantaneously.

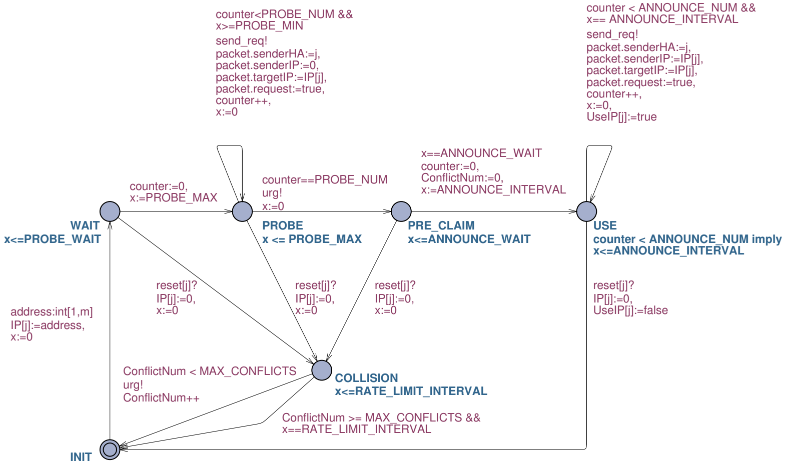
Model Architecture

For each host `j` there are three automata ("process based decomposition"):

1. An automaton that models address configuration.
2. An automaton that handles incoming messages.
3. An automaton that models the other processes running on the host (which may send regular ARP packets).

In addition we have a collection of `Network` automata.

Zeroconf Address Configuration



Close Correspondence Between Model and Standard

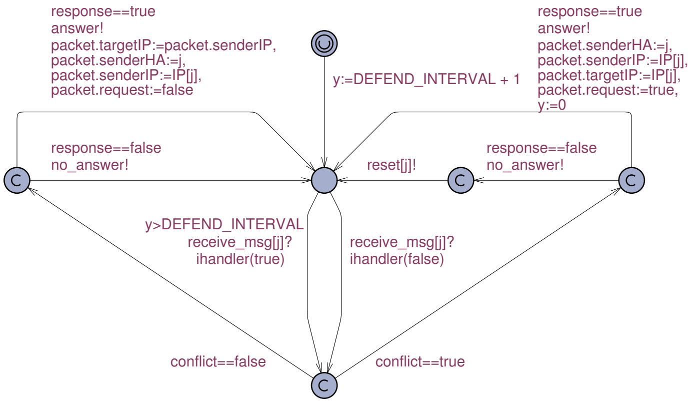
Example text from RFC [page 11, section 2.2.1]:

> *"When ready to begin probing, the host should then wait for a random time interval selected uniformly in the range zero to PROBE_WAIT seconds, and should then send PROBE_NUM probe packets, each of these probe packets spaced randomly, PROBE_MIN to PROBE_MAX seconds apart."*

“Mistakes” in Standard

- It does not specify upper and lower bounds on time that may elapse between sending last ARP Probe and sending first ARP Announcement.
- It does not specify whether a host may immediately start using a newly claimed address or whether it should first send out all ARP Announcements.
- It does not specify tolerance on timing of ARP Announcements.

Input Handling



Input Handling (cnt)

Input handling described at various places in standard. Conceptually it is natural to group everything together in one module.

Function `ihandler` distinguishes between 9 scenarios, each of which is described explicitly or implicitly in the standard.

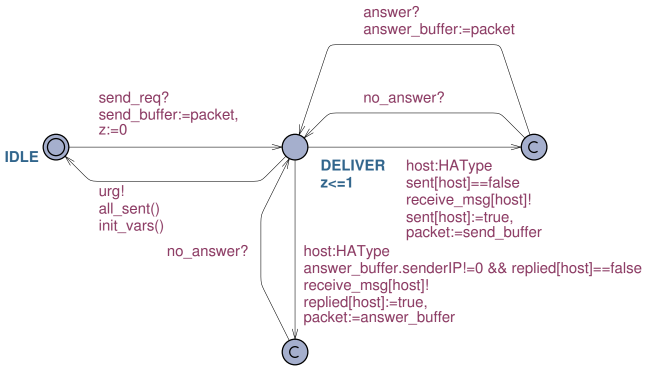
Example Scenario as Described in RFC

"In addition, if during this period [from the beginning of the probing process until ANNOUNCE_WAIT seconds after the last probe packet is sent] the host receives any ARP Probe where the packet's 'target IP address' is the address being probed for, and the packet's 'sender hardware address' is not the hardware address of the interface the host is attempting to configure, then the host MUST similarly treat this as an address conflict and select a new address as above. This can occur if two (or more) hosts attempt to configure the same IPv4 Link-Local address at the same time."

Further “Mistakes” in Standard

- Although standard states that Zeroconf requires an underlying network that supports ARP (RFC 826), we identified some cases where Zeroconf does not conform to RFC 826.
- It is not exactly clear in which situations a host may defend its address.

Network Automaton



Model Checking

Model checking Zeroconf is difficult!

Using latest version of Uppaal, we managed to prove mutual exclusion and absence of deadlock for model with 2 hosts, 1 IP address and 2 network automata.

Model checking failed for larger models.

Abstractions

1. Dead variable elimination
2. Over approximation: elimination of counter `ConflictNum`
3. Over approximation: elimination of clock `y`
4. Only consider single IP address (manual proof of correctness)

Using these abstractions (on top of those implemented in Uppaal) we can prove mutual exclusion for models with
3 hosts, 1 IP address and 3 network automata, or
2 hosts, x IP addresses and 2 network automata.

Limitations of Model Checking

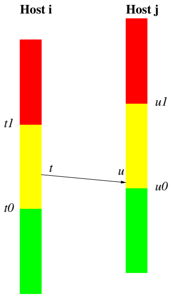
Thus far, we did not succeed in finding abstractions that permit us to handle larger instances.

Still, it is pretty obvious that Zeroconf satisfies mutual exclusion.

Large number of messages in the protocol "obscures" key correctness argument for model checker.

Manual Proof of Mutual Exclusion

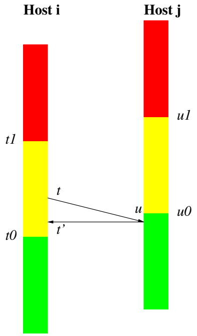
Suppose two hosts `i` and `j` enter CS. Assume wlog that `j` enters first. Case 1: last probe from `i` arrives at `j` before `j` enters CS.



Then `j` must be in trying region since
$u \geq t = t0 - 2 \geq u0 - 2 > u0 - 6 \geq u1$.
Hence probe will cause reset at `j`. Contradiction.

Mutual Exclusion (cnt)

Case 2: last probe from `i` arrives at `j` after `j` enters CS.
Then `j` will send reply message to `i`.



When reply arrives, `i` is still in trying region.
Hence reply will cause reset at `i`. Contradiction.

Future Work

- Zeroconf is a challenge for model checking tools
- Backward reachability analysis?
- Better support for computing abstractions (or proving their correctness) essential: only abstractions can bridge gap between realistic and tractable models
- Combine functionality Uppaal and PRISM
- Study host failure and joining of networks