Automated Analysis of AODV using UPPAAL

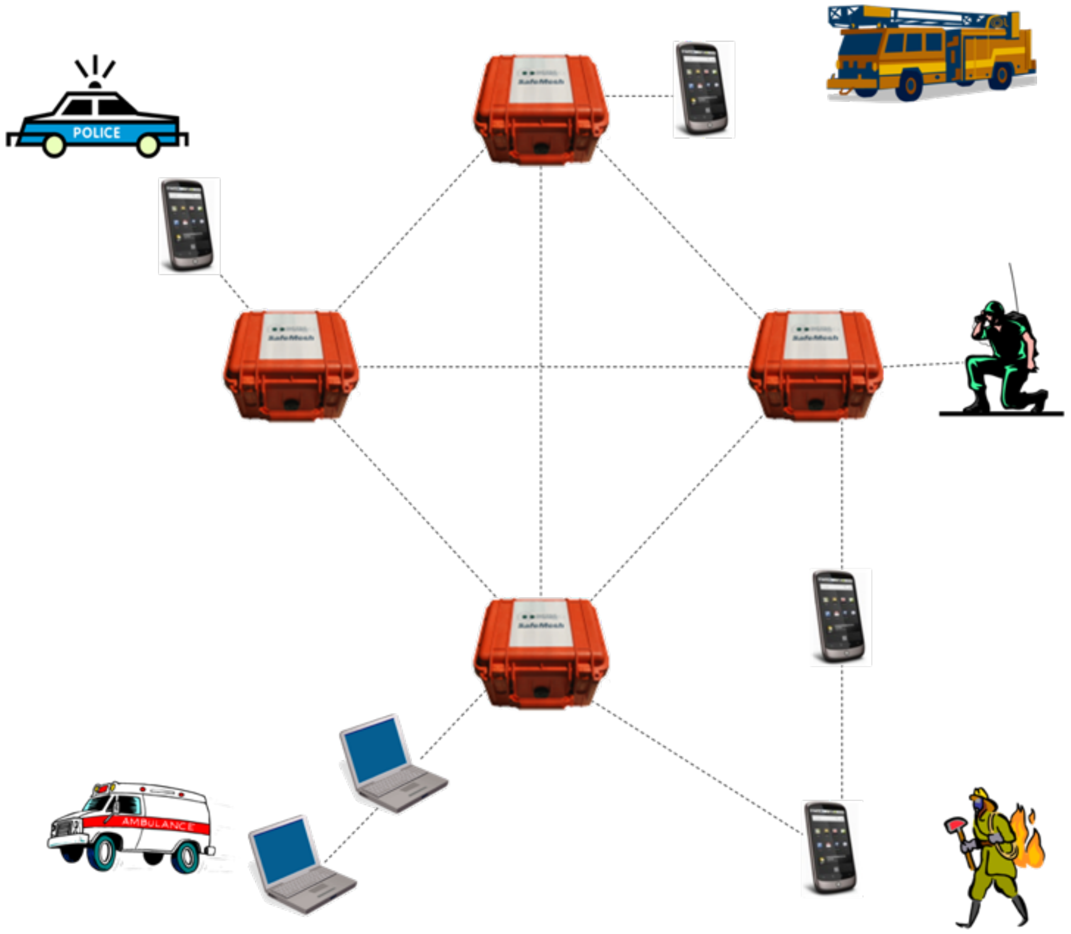
A. Fehnker, R. van Glabbeek, P. Höfner,
A. McIver, M. Portmann, W.-L. Tan

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Members
NICTA Partners

What is the Problem?

- Wireless Mesh Networks (WMNs)
  - key features: mobility, dynamic topology, wireless multihop backhaul
  - quick and low cost deployment
- Applications
  - public safety
  - emergency response, disaster recovery
  - transportation
  - mining
  - smart grid
  - ...
- Limitations in reliability and performance



Formal Methods for Mesh Networks

- Goal
  - model, analyse, verify, improve and increase the performance of wireless mesh protocols
  - develop suitable formal methods techniques

- Benefits
  - more reliable protocols
  - finding and fixing bugs
  - better performance
  - proving correctness
  - reduce “time-to-market”

Ad Hoc On-Demand Distance Vector Protocol

- Routing protocol for WMNs

- Ad hoc (network is not static)
- On-Demand (routes are established when needed)
- Distance-Vector

- Developed 1997-2001 by Perkins, Beldig-Royer and Das (University of Cincinnati)
- RFC by the IETF MANET working group
- basis of upcoming IEEE 802.11s

Ad Hoc On-Demand Distance Vector Protocol

NICTA

- AODV control messages
  - route request (RREQ)
  - route reply (RREP)
  - route error message (RERR)

- Main Mechanism
  - if route is needed
    BROADCAST RREQ
  - if node has information about a destination
    UNICAST RREP
  - if unicast fails or link break is detected
    SEND RERR

UPPAAL Model Checker

- Well established model checker
- Uses networks of timed automata
- Has been used for protocol verification

- Synchronisation mechanisms
  – binary handshake synchronisation (unicast communication)
  – broadcast synchronisation (broadcast communication)
- Common data structures
  – arrays, structs, ...
  – C-like programming language
- Provides mechanisms for time and probability

Modelling AODV in UPPAAL

- Systematically derived from process-algebraic model models all parts of the official specification (except time)
- Allows interplay
- Increases trust

- Process algebra AWN
  - developed specifically for WMN routing protocols
  - easily readable
  - three necessary features:
    data structures, local broadcast, conditional unicast

Process Algebra AWN

Table 1 Excerpt of AWN spec for AODV. A few cases for RREQ handling.

```
AODV(ip,sn,rt,rreqs,store) ≝
/*depending on the message on top of the message queue, the node calls different processes*/
. . .
[ msg = rreq(hops, rreqid, dip, dsn, oip, osn, sip) ∧ (oip, rreqid) ∈ rreqs ]
    /*silently ignore RREQ, i.e. do nothing, except update the entry for the sender*/
    ⟦rt := update(rt, (sip, 0, val, 1, sip))⟧ .   /*update the route to sip*/
    AODV(ip,sn,rt,rreqs,store)
+ [ msg = rreq(hops, rreqid, dip, dsn, oip, osn, sip) ∧ (oip, rreqid) ∉ rreqs) ∧ dip = ip ]
    /*answer the RREQ with a RREP*/
    ⟦rt := update(rt, (oip, osn, val, hops + 1, sip))⟧   /*update the routing table*/
    ⟦rreqs := rreqs ∪ {(oip, rreqid)}⟧   /*update the array of already seen RREQ*/
    ⟦sn := max(sn, dsn)⟧   /*update the sqn of ip*/
    ⟦rt := update(rt, (sip, 0, val, 1, sip))⟧   /*update the route to sip*/
    unicast(nhop(rt,oip),rrep(0,dip,sn,oip,ip)) .
    AODV(ip,sn,rt,rreqs,store)
+ [ msg = rreq(hops, rreqid, dip, dsn, oip, osn, sip) ∧(oip, rreqid) ∉ rreqs) ∧ dip ≠ ip ∧
     (dip ∉ vD(rt) ∨ sqn(rt,dip) < dsn ∨ sqnf(rt,dip) = unk) ]
    /*forward RREQ*/
    ⟦rt := update(rt, (oip, osn, val, hops + 1, sip))⟧   /*update routing table*/
    ⟦rreqs := rreqs ∪ {(oip, rreqid)}⟧   /*update the array of already seen RREQ*/
    ⟦rt := update(rt, (sip, 0, val, 1, sip))⟧   /*update the route to the sender*/
    broadcast(rreq(hops + 1,rreqid,dip,max(sqn(rt, dip), dsn),oip,osn,ip)) .
    AODV(ip,sn,rt,rreqs,store)
+ [ rreq(hops, rreqid, dip, dsn, oip, osn, sip) ∧ . . . ]
    . . .
```

UPPAAL Model

NICTA

Table 2 Excerpt of UPPAAL model. A few cases for RREQ handling.

```
...
aodv -> aodv {
guard nextmsg()==RREQ && rreqs[msglocal[0].oip][msglocal[0].rreqid];
sync  tau[ip]?;
assign sipupdate(), deletemsg();  },
aodv -> aodv {
guard nextmsg()==RREQ&&!rreqs[msglocal[0].oip][msglocal[0].rreqid]&&msglocal[0].dip==ip;
sync  rrep[ip][oipnhop()]!;
assign updatert(msglocal[0].oip,msglocal[0].osn,1,msglocal[0].hops+1,msglocal[0].sip),
       rreqs[msglocal[0].oip][msglocal[0].rreqid]=1,
       sn=max(sn,msglocal[0].dsn),
       sipupdate(),
       msgglobal=createrep(0,msglocal[0].dip,sn,msglocal[0].oip,ip), deletemsg();  },
aodv -> aodv {
guard nextmsg()==RREQ&&!rreqs[msglocal[0].oip][msglocal[0].rreqid]&&msglocal[0].dip!=ip
       && (!rt[msglocal[0].dip].flag || msglocal[0].dsn>rt[msglocal[0].dip].dsn
       || rt[msglocal[0].dip].dsn==0);
sync  rreq[ip]!;
assign updatert(msglocal[0].oip,msglocal[0].osn,1,msglocal[0].hops+1,msglocal[0].sip),
       rreqs[msglocal[0].oip][msglocal[0].rreqid]=1,
       sipupdate(),
       msgglobal=createreq(msglocal[0].hops+1,msglocal[0].rreqid,msglocal[0].dip,
       max(msglocal[0].dsn, rt[msglocal[0].dip].dsn),msglocal[0].oip,msglocal[0].osn,ip),
       deletemsg();  },
...
```

UPPAAL Model – Nodes of a Network

- Each node is modelled by a timed-automaton
- Additional (local) data structure
  - routing table
  - unique name
  - ...
- Data sending via shared variables

UPPAAL Model – Topology

- Topology modelled by adjacency matrix
- Topology change by additional timed-automaton

- Synchronisation only if two nodes are connected

Experiments Set-Up

- Exhaustive search
  - different properties
  - all topologies up to 5 nodes (one topology change)
  - 2 route discovery processes
  - 17400 scenarios
  - variants of AODV (4 models)

- Larger topologies possible, but only for a few scenarios

Experiments – 3 Properties

- Route Discovery
  - if two nodes are connected, does AODV find a route?

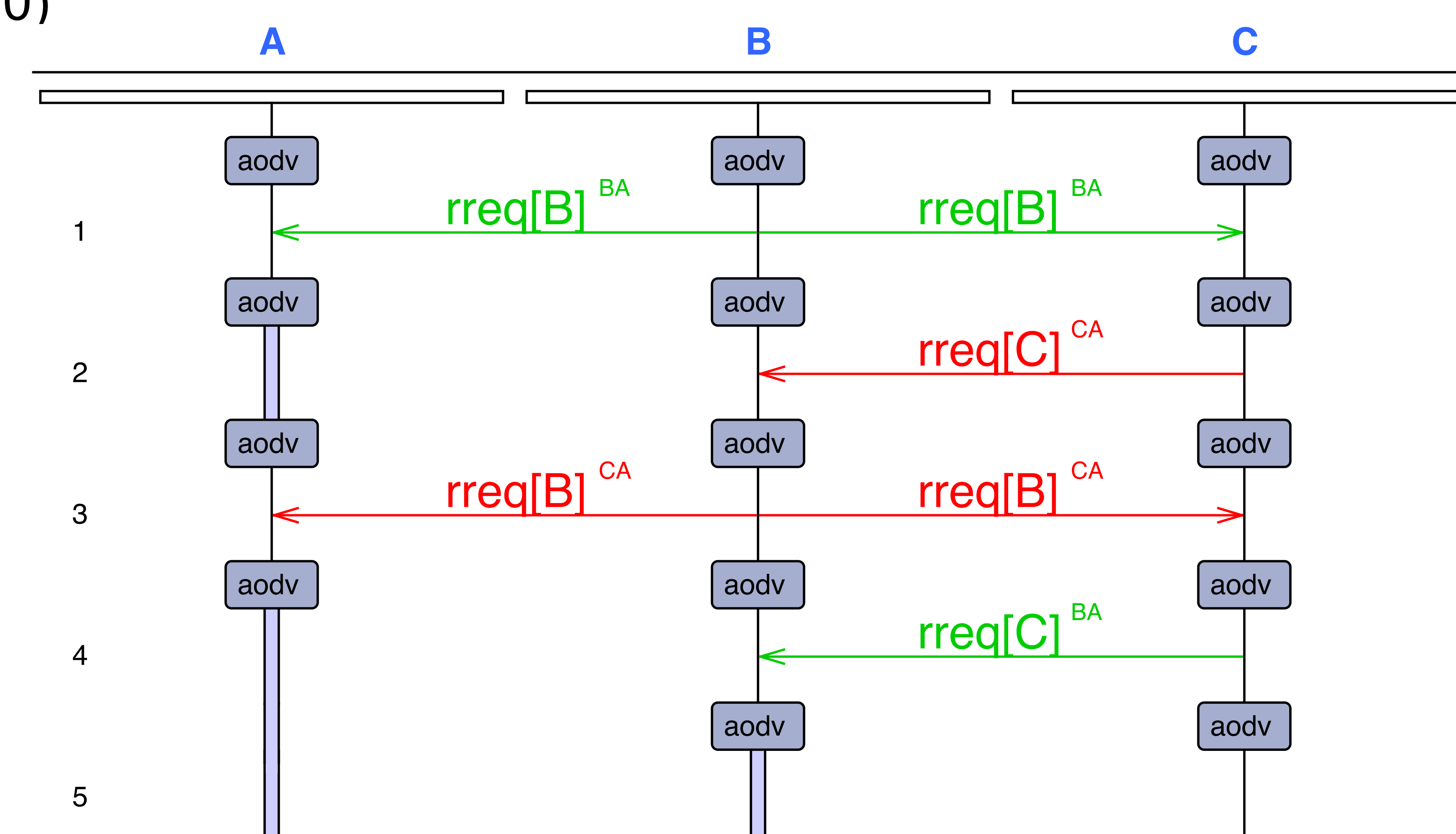
```
A[]((topology.final && emptybuffers()) imply
     (node(OIP).rt[DIP].nhop!=0))
```

- Route Optimality
  - no non-optimal route has been established after the protocol has been terminated

- Total Optimality
  - no non-optimal route found at all

Results

- Route discovery and route optimality do not hold
  - sanity check
  - found within seconds
  - shows power of model checking
    - route discovery (2004)
    - route optimality (2010)



Quantity Results

- Potential failure in route discovery
  - static topology: 47.3%
  - dynamic topology (add link): 42.5%
  - dynamic topology (remove link): 73.3%

- AODV repeats route request
- Other solution: Modify AODV
  - e.g., Forward Route Reply

4 Variants of AODV

NICTA

- Standard AODV
  – as reference
- Forwarding all route replies
  – increase the chance of route discovery
- Replying to improving requests
  – decrease number of sub-optimal routes
- Recovering from failed replies
  – further increase for route discovery
  – variant should considered with care

Experimental Results

		Property 1	Property 2	Property 3	Property 1& 2	all properties
static	model 1	52.7%	93.2%	50.7%	50.0%	13.5%
	model 2	100.0%	93.2%	47.5%	93.2%	47.5%
	model 3	100.0%	99.1%	47.5%	99.1%	47.5%
	model 4	100.0%	99.1%	47.5%	99.1%	47.5%

		Property 1	Property 2	Property 3	Property 1& 2	all properties
add link	model 1	57.5%	90.8%	49.1%	53.3%	18.1%
	model 2	100.0%	90.6%	46.2%	90.6%	46.2%
	model 3	100.0%	97.8%	46.2%	97.8%	46.2%
	model 4	100.0%	96.3%	46.2%	96.3%	46.2%

		Property 1	Property 2	Property 3	Property 1& 2	all properties
remove link	model 1	26.7%	90.5%	59.7%	26.2%	6.0%
	model 2	53.0%	89.4%	57.1%	51.2%	28.9%
	model 3	53.0%	93.1%	57.1%	52.8%	28.9%
	model 4	75.4%	94.0%	54.0%	73.8%	41.0%

UPPAAL Statistics

- Intel Core2 CPU 2.13GHz processor with 2GB RAM
- Uppaal 4.0.13.
- 70400 instances (17600 for each model)
- 4th variant (largest state space)
  - average of 9400 states
  - largest model has 475.000 states, median is 2.700
  - took on average 1.73 seconds, at most 81 seconds
- Larger topologies possible

- **An automated, systematic and rigorous analysis of reasonable rich routing protocols is feasible**

Further Work

NICTA

- Probabilistic/Statistical Model Checking
  - equip links and topology with probabilities
  - allows quantitative analysis

- Use process algebra AWN to analyse variants
  - e.g. loop freedom

- Add time such as time outs to AWN and UPPAAL-model

- Automatic translation from AWN to UPPAAL

NICTA

From **imagination** to **impact**