


NICTA


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

Australian Government

Department of Broadband, Communications and the Digital Economy

Australian Research Council

NICTA Funding and Supporting Members and Partners

Australian National University

UNSW THE UNIVERSITY OF NEW SOUTH WALES

NSW GOVERNMENT | Trade & Investment

State Government Victoria

THE UNIVERSITY OF MELBOURNE

THE UNIVERSITY OF SYDNEY

Queensland Government

Griffith UNIVERSITY

QUT Queensland University of Technology

THE UNIVERSITY OF QUEENSLAND AUSTRALIA

Analyse drahtloser Netzwerke mittels Formaler Methoden

Peter Höfner

Past and Present: Protocol Development

- "Rough Consensus and Running Code" (Trial and Error)
  - start with a good idea
  - build a protocol out of it (implementation)
    - run tests (over several years)
    - find limitations, flaws, etc.
    - fix problems
  - build a new version of the protocol
    - start testing again
  - at some point, people agree on an RFC (standard)

Beauvais Cathedral
(~300 years to build, at least 2 collapses)

Future: Protocol Development

- Is there a method which is more reliable and cost-efficient?
- Is there a way to compare different protocols?

- **New methods required**
  (or finetune/extend existing ones)

"The original design was so boldly conceived that it was found structurally impossible to build."

Problems

- Standards (IETF RFCs) are not precise
  - written in English
  - ambiguous (sometimes incomplete)
  - no formal specification

Why Formal Specification?

Why Formal Specification?

Problems

- Standards (IETF RFCs) are not precise
  - written in English
  - ambiguous (sometimes incomplete)
  - no formal specification
- Compliant implementations
  - have different behaviours
  - are not compatible
  - have serious flaws
- Traditional evaluation techniques: simulation and test-bed
  - expensive
  - limited to (a small number of) specific scenarios
  - error found after years of evaluation
  - barely offer any guarantee for properties such as route discovery

Formal Methods for Mesh Networks

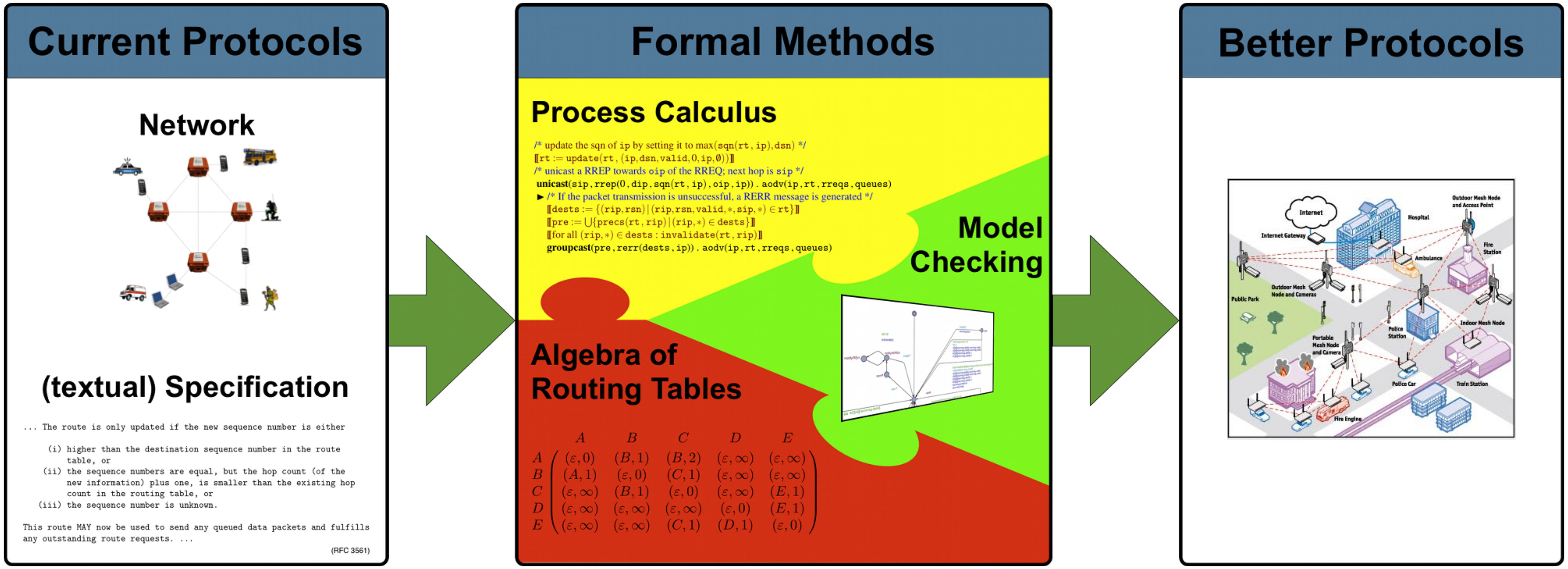
NICTA

- Goal
  - model, analyse, verify 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”

Formal Methods for Mesh Networks

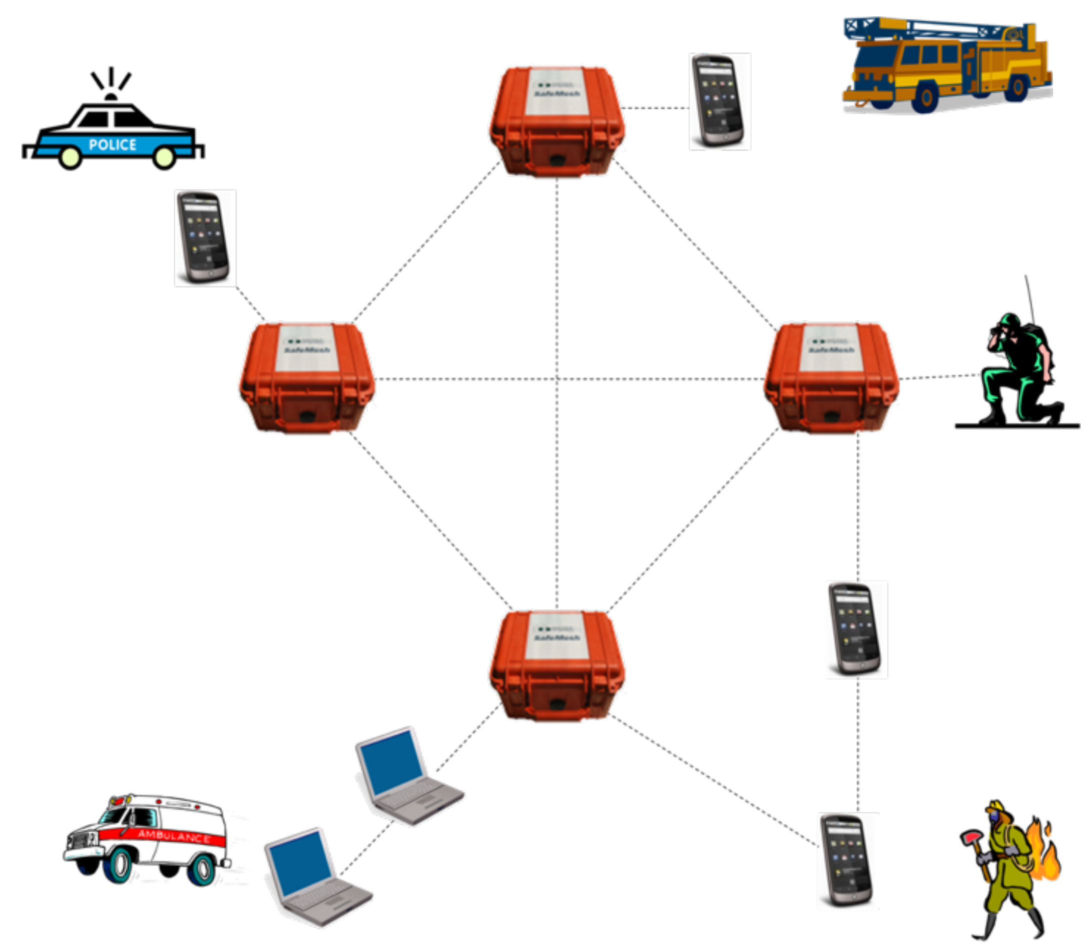
NICTA

- Main Methods used so far
  - process algebra
  - model checking
  - routing algebra



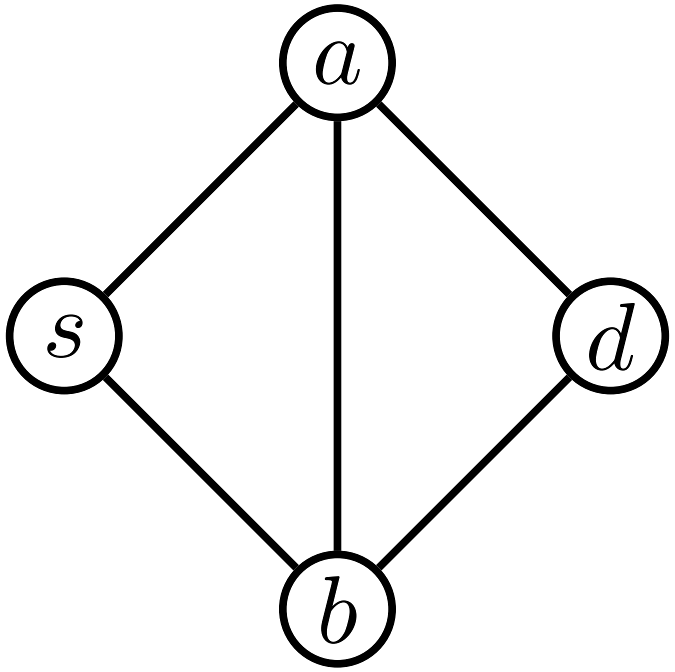
Wireless Mesh Networks

- 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



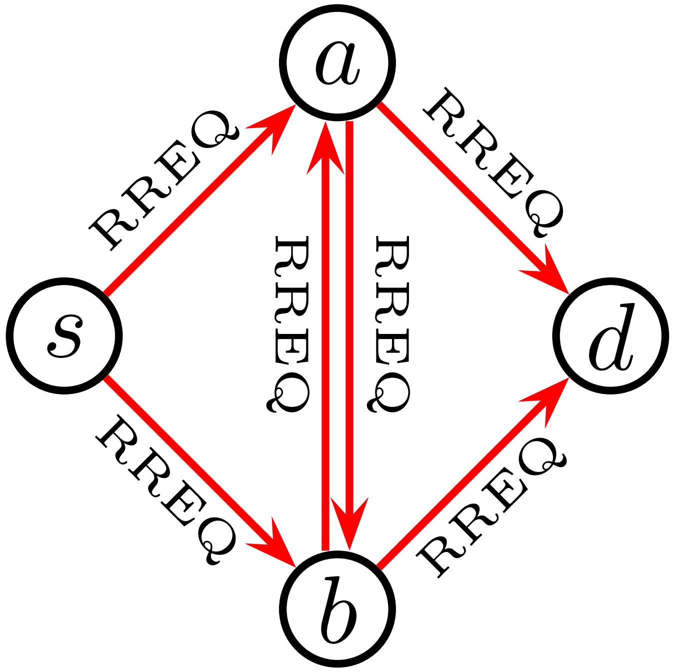
Case Study: AODV

- 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
    GROUPCAST RERR



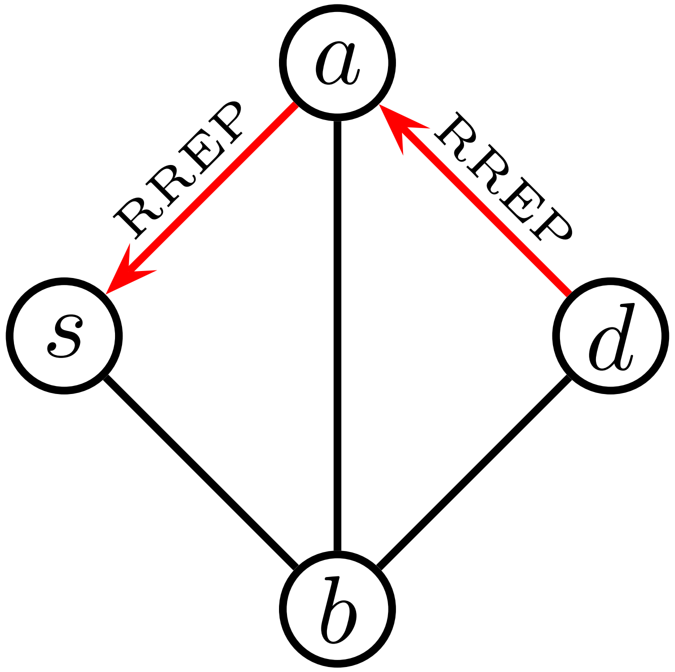
Case Study: AODV

- 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
    GROUPCAST RERR



Case Study: AODV

- 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
    GROUPCAST RERR



Ad Hoc On-Demand Distance Vector Protocol

NICTA

- Properties of AODV
  - route correctness
  - loop freedom
  - route discovery
  - packet delivery

Ad Hoc On-Demand Distance Vector Protocol

- Properties of AODV
  - route correctness ✓
  - loop freedom ✓ (at least for some interpretations)
  - route discovery ✗
  - packet delivery ✗

Process Algebra

NICTA

```
+ [ (oip, rreqid) ∉ rreqs ]      /* the RREQ is new to this node */
  /* update the route to oip in rt */
  [[rt := update(rt, (oip, osn, valid, hops + 1, sip, ∅))]]
  /* update rreqs by adding (oip, rreqid) */
  [[rreqs := rreqs ∪ {(oip, rreqid)}]]
  (
    [ dip = ip ]      /* this node is the destination node */
      /* update the sqn of ip by setting it to max(sqn(rt, ip), dsn) */
      [[rt := update(rt, (ip, dsn, valid, 0, ip, ∅))]]
      /* unicast a RREP towards oip of the RREQ; next hop is sip */
      unicast(sip, rrep(0, dip, sqn(rt, ip), oip, ip)) . AODV(ip, rt, rreqs, queues)
      ▶ /* If the packet transmission is unsuccessful, a RERR message is generated */
        [[dests := {(rip, rsn) | (rip, rsn, valid, *, sip, *) ∈ rt}]]
        [[pre := ⋃{precs(rt, rip) | (rip, *) ∈ dests}]]
        [[for all (rip, *) ∈ dests : invalidate(rt, rip)]]
        groupcast(pre, rerr(dests, ip)) . AODV(ip, rt, rreqs, queues)
    + [ dip ≠ ip ]      /* this node is not the destination node */
      (
        [ dip ∈ aD(rt) ∧ dsn ≤ sqn(rt, dip) ∧ sqn(rt, dip) ≠ 0 ]      /* valid route to dip that is
        fresh enough */
          /* update rt by adding sip to precs(rt, dip) */
          [[r := addpre(σ_route(rt, dip), {sip}); rt := update(rt, r)]]
```

Process Algebra

- Desired Properties
  - guaranteed broadcast
  - conditional unicast
  - data structure
- Inspired by
  - π- Calculus
  - ω- Calculus
  - (LOTOS)

Structure of WMNs

NICTA

- User
  - Network as a “cloud”
- Collection of nodes
  - connect / disconnect / send / receive
  - “parallel execution” of nodes
- Nodes
  - data management
    - data packets, messages, IP addresses ...
  - message management (avoid blocking)
  - core management
    - broadcast / unicast / groupcast ...
  - “parallel execution” of sequential processes

Nodes (Sequential Process Expressions)

- Syntax of sequential process expressions

$$\begin{array}{rcl} SP & ::= & X(exp_1, \ldots, exp_n) \mid [\varphi]SP \mid [\![\mathtt{var} := exp]\!]SP \mid SP + SP \mid \\ & & \alpha.SP \mid \mathbf{unicast}(dest, ms).SP \blacktriangleright SP \\ \alpha & ::= & \mathbf{broadcast}(ms) \mid \mathbf{groupcast}(dests, ms) \mid \mathbf{send}(ms) \mid \\ & & \mathbf{deliver}(data) \mid \mathbf{receive}(\mathtt{msg}) \end{array}$$

Snippet of AODV

```
+ [ (oip, rreqid) ∉ rreqs ]      /* the RREQ is new to this node */
  /* update the route to oip in rt */
  [[rt := update(rt, (oip, osn, valid, hops + 1, sip, ∅))]]
  /* update rreqs by adding (oip, rreqid) */
  [[rreqs := rreqs ∪ {(oip, rreqid)}]]
  (
    [ dip = ip ]      /* this node is the destination node */
      /* update the sqn of ip by setting it to max(sqn(rt, ip), dsn) */
      [[rt := update(rt, (ip, dsn, valid, 0, ip, ∅))]]
      /* unicast a RREP towards oip of the RREQ; next hop is sip */
      unicast(sip, rrep(0, dip, sqn(rt, ip), oip, ip)) . AODV(ip, rt, rreqs, queues)
      ► /* If the packet transmission is unsuccessful, a RERR message is generated */
        [[dests := {(rip, rsn) | (rip, rsn, valid, ∗, sip, ∗) ∈ rt}]]
        [[pre := ⋃{precs(rt, rip) | (rip, ∗) ∈ dests}]]
        [[for all (rip, ∗) ∈ dests : invalidate(rt, rip)]]
        groupcast(pre, rerr(dests, ip)) . AODV(ip, rt, rreqs, queues)
    + [ dip ≠ ip ]      /* this node is not the destination node */
      (
        [ dip ∈ aD(rt) ∧ dsn ≤ sqn(rt, dip) ∧ sqn(rt, dip) ≠ 0 ]      /* valid route to dip that is
        fresh enough */
          /* update rt by adding sip to precs(rt, dip) */
          [[r := addpre(σ_route(rt, dip), {sip}); rt := update(rt, r)]]
```

Case Study

- AODV Routing Protocol
- Achievements
  - full concise specification of AODV (RFC 3561) (without time)
  - verified/disproved properties
    - route discovery
    - packet delivery
    - loop freedom
      - first (correct) proof
      - disproved loop freedom for variants of AODV (as implemented in at least 3 open source implementations)
      - analysed more than 5000 interpretations
  - found several ambiguities, mistakes, shortcomings
  - found solutions for some limitations

Ambiguities and Loop Freedom

1.	**Updating the Unknown Sequence Number in Response to a Route Reply**	
1a.	the destination sequence number (DSN) is copied from the RREP message (Sect 6.7)	decrement of sequence numbers and loops
1b.	the routing table is not updated when the information inside is "fresher" (Sect. 6.1)	loop free
2.	**Updating with the Unknown Sequence Number (Sect. 6.5)**	
2a.	no update occurs	loop free, but opportunity to improve routes is missed.
2b.	overwrite any routing table entry by an update with an unknown DSN	decrement of sequence numbers and loops
2c.	use the new entry with the old DSN	loop free
3.	**More Inconclusive Evidence on Dealing with the Unknown Sequence Number (Sect. 6.2)**	
3a.	update when *incoming* sequence number is unknown	supports Interpretations 2b or 2c above
3b.	update when *existing* sequence number is unknown	decrement of sequence numbers and loops
3c.	update when no *existing* sequence number is known	supports Interpretation 2a above
4.	**(Dis)Allowing Self-Entries**	
4a.	allow self-entries	loop free if used with appropriate `invalidate`
4b.	disallow self-entries; if self-entries would occur, ignore mess.	loop free
4c.	disallow self-entries; if self-entries would occur, forward	loop free
5.	**Storing the Own Sequence Number**	
5a.	store sequence number as separate value	loop free
5b.	store sequence number inside routing table	excludes non-trivial self-entries (4b–c)
6.	**Invalidating Routing Table Entries in Response to a RERR message**	
6a.	copy DSN from RERR message (Sect. 6.11)	decrement of sequence numbers and loops (when allowing self-entries (Interpretation 4a))
6b.	no action if the DSN in the routing table is larger than the one in the RERR mess. (Sect. 6.1 & 6.11)	loops (when allowing self-entries)
6c.	take the maximum of the DSN of the routing table and the one from the RERR message	loops (when allowing self-entries)
6d.	take the maximum of the increased DSN of the routing table and the one from the RERR mess.	loop free

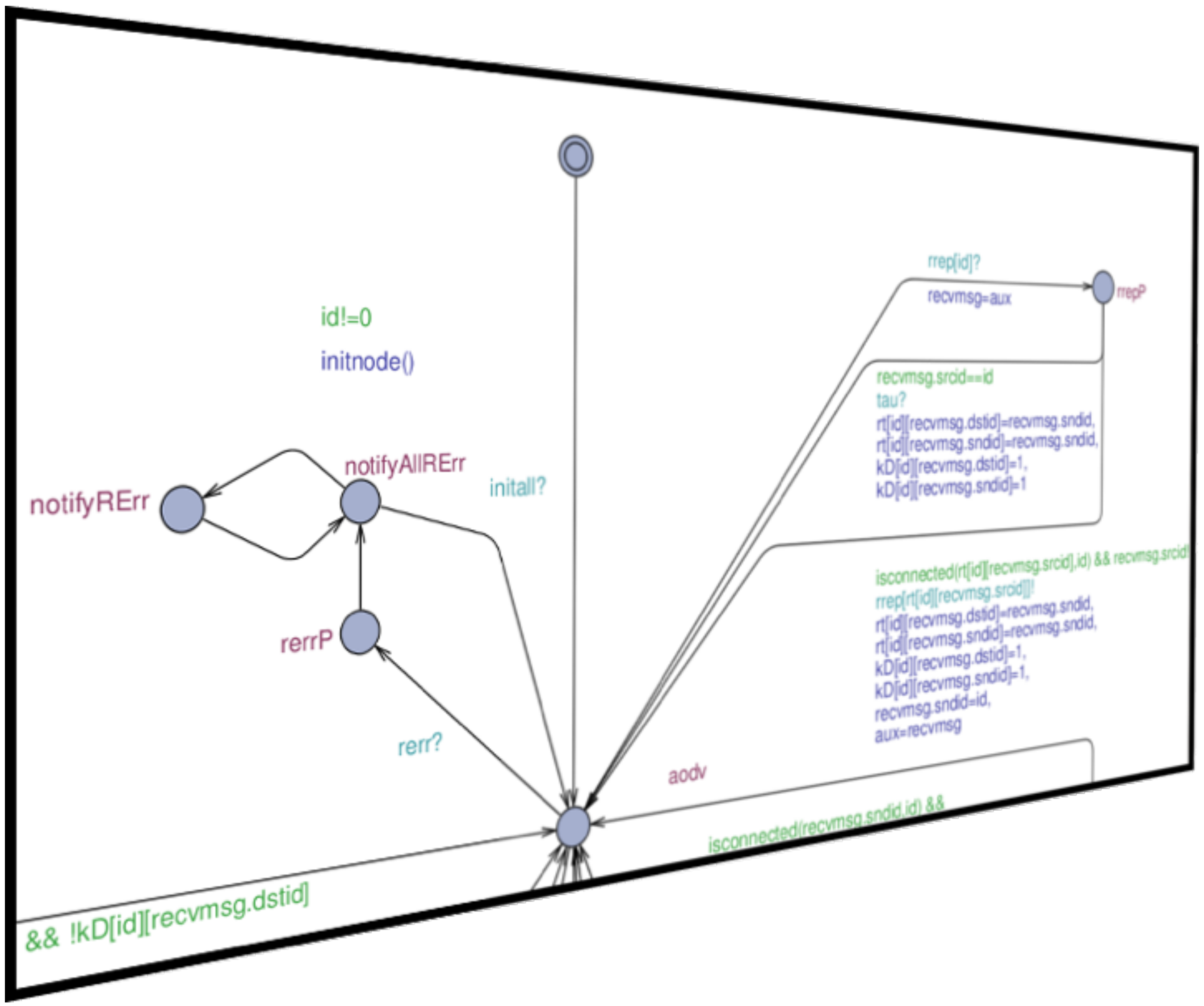
Table 2: Analysis of Different Interpretations of the RFC 3561 (AODV)

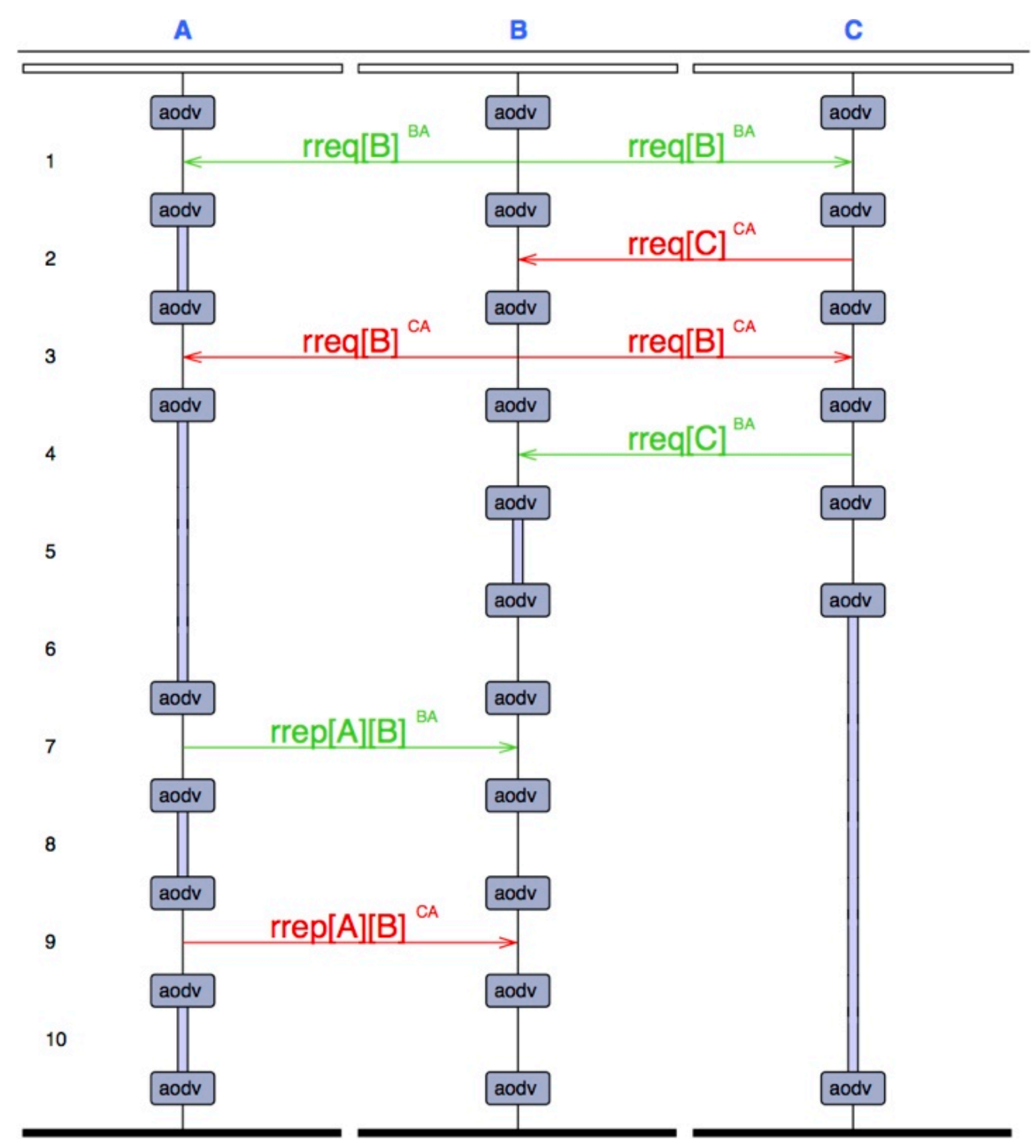
Ambiguities and Loop Freedom

- proof modularity (different invariants)
  - 5068 interpretations (240 are loop free and "correct")
    - 432 are "reasonable" (112 are loop free and "correct")
  - even some interpretations we never thought about

- simulation and test-bed experiment would be separate for each scenario

Model Checking





Model Checking

- Model checking routing algorithms
  - executable models
- Complementary to process algebra
  - find bugs and typos in model of process algebra
  - check properties of specification applied to particular topology
  - easy adaption in case of change
  - automatic verification
- Achievements
  - implemented process algebra specification of AODV
  - found/replayed shortcomings

UPPAAL Model Checker

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

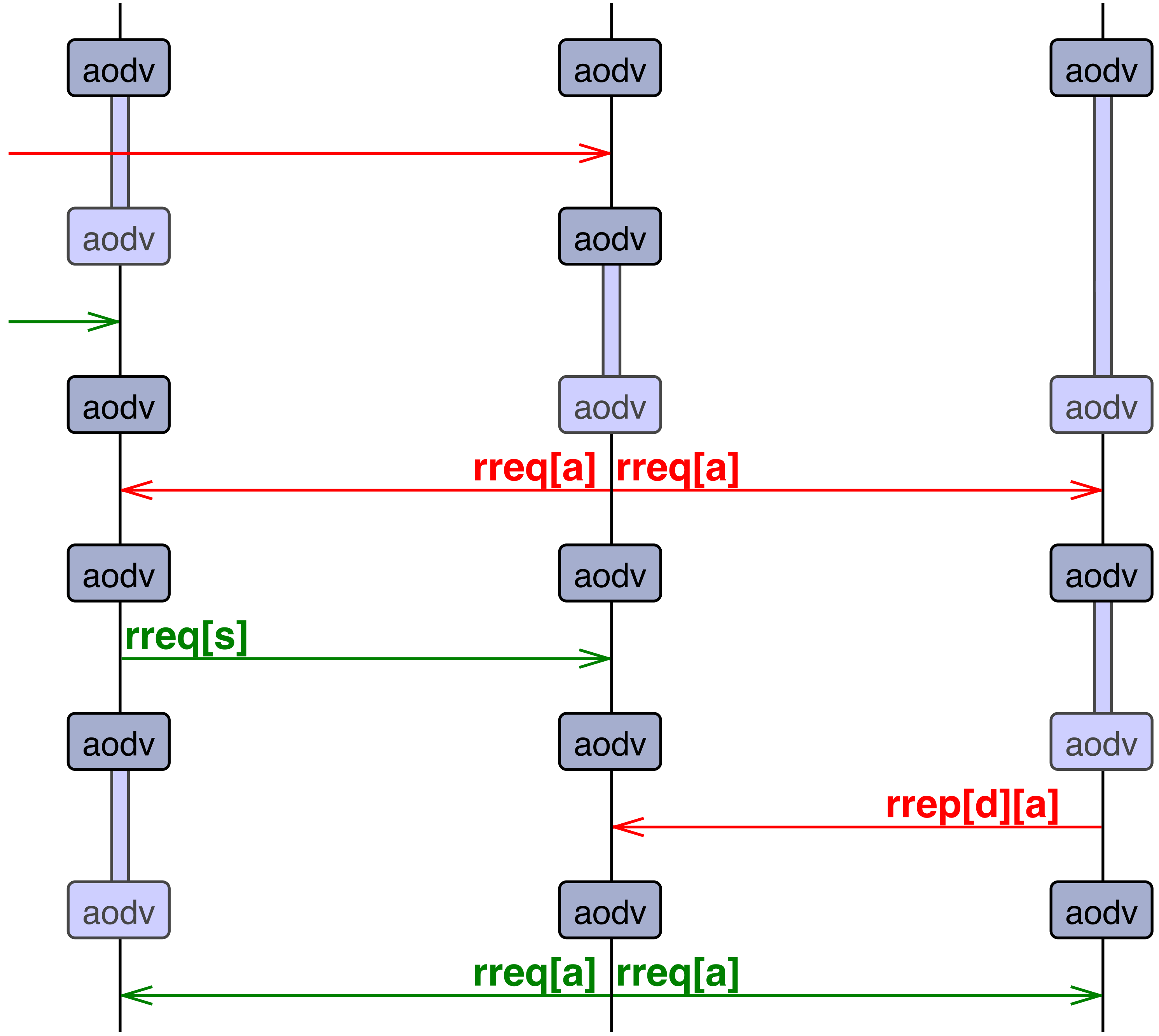
- Synchronisation mechanisms
  - binary handshake synchronisation (unicast communication)
  - broadcast synchronisation (broadcast communication)
- Common data structures
  - arrays, structs, ...
  - C-like programming language
- Provides mechanisms for time and probability

Experiments Set-Up

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

Results: Route Discovery (2004)

- Route discovery fails in a linear 3-node topology



Results: Route Discovery

- exhaustive search
  (potential failure in route discovery)
  - static topology: 47.3%
  - dynamic topology (add link): 42.5%
  - dynamic topology (remove link): 73.7%
- AODV repeats route request
- Other solution: forward route reply

Routing Algebra

$$
\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array}
\begin{array}{c}
\begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{pmatrix}
(\epsilon, 0) & (B, 1) & (B, 2) & (\epsilon, \infty) & (\epsilon, \infty) \\
(A, 1) & (\epsilon, 0) & (C, 1) & (\epsilon, \infty) & (\epsilon, \infty) \\
(\epsilon, \infty) & (B, 1) & (\epsilon, 0) & (\epsilon, \infty) & (E, 1) \\
(\epsilon, \infty) & (\epsilon, \infty) & (\epsilon, \infty) & (\epsilon, 0) & (E, 1) \\
(\epsilon, \infty) & (\epsilon, \infty) & (C, 1) & (D, 1) & (\epsilon, 0)
\end{pmatrix}
\end{array}
$$

Routing Algebra – Elements, Operators

- Matrices over routing table entries

$$
\begin{array}{c}
 \\ A \\ B \\ C \\ D \\ \vdots
\end{array}
\begin{array}{c}
\begin{array}{ccccc} \quad A \quad & \quad B \quad & \quad C \quad & \quad D \quad & \quad \ldots \end{array} \\
\begin{pmatrix}
(\_,0) & (B,1) & (B,2) & (\_,\infty) & \\
(A,1) & (\_,0) & (C,1) & (\_,\infty) & \ldots \\
(\_,\infty) & (B,1) & (\_,0) & (\_,\infty) & \\
(\_,\infty) & (\_,\infty) & (\_,\infty) & (\_,0) & \\
 & \vdots & & & \ddots
\end{pmatrix}
\end{array}
$$

routing table of A

“routes” to B

- standard matrix operations
- further abstraction possible (semirings, test, domain, modules ...)

Routing Algebra – Elements, Operators

- Routing table entries (no sequence number so far)

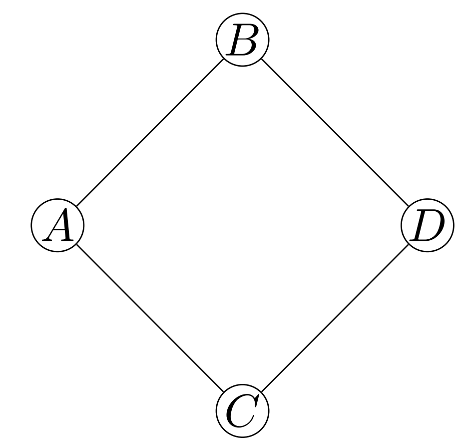
$$(\texttt{nhip}\,,\,\texttt{hops})$$

- Choice: $(A,5) + (B,2) = (B,2)$
- Multiplication: $(A,5) \cdot (B,2) = (A,7)$
  - – destination and source must coincide

- idea: back to Backhouse, Carré, Griffin, Sobrinho

Example

- A route request is broadcast



$$\underbrace{\begin{pmatrix} (\_,0) & (B,1) & (C,1) & (\_,\infty) \\ (A,1) & (\_,0) & (\_,\infty) & (D,1) \\ (A,1) & (\_,\infty) & (\_,0) & (D,1) \\ (\_,\infty) & (B,1) & (C,1) & (\_,0) \end{pmatrix}}_{\text{topology}} \bullet \underbrace{\begin{pmatrix} (\_,0) & (\_,\infty) & (\_,\infty) & (\_,\infty) \\ (\_,\infty) & (\_,\infty) & (\_,\infty) & (\_,\infty) \\ (\_,\infty) & (\_,\infty) & (\_,\infty) & (\_,\infty) \\ (\_,\infty) & (\_,\infty) & (\_,\infty) & (\_,\infty) \end{pmatrix}}_{\text{sender}} \bullet \underbrace{\begin{pmatrix} (\_,0) & (B,1) & (\_,\infty) & (\_,\infty) \\ (\mathbf{D},\mathbf{3}) & (\_,0) & (\_,\infty) & (\_,\infty) \\ (A,1) & (\_,\infty) & (\_,0) & (D,1) \\ (C,2) & (\_,\infty) & (C,1) & (\_,0) \end{pmatrix}}_{\text{routing table}}$$

$$= \underbrace{\begin{pmatrix} (\_,0) & (B,1) & (\_,\infty) & (\_,\infty) \\ (\mathbf{A},\mathbf{1}) & (\_,0) & (\_,\infty) & (\_,\infty) \\ (A,1) & (\_,\infty) & (\_,0) & (D,1) \\ (C,2) & (\_,\infty) & (C,1) & (\_,0) \end{pmatrix}}_{\text{updated routing table}}$$

Sent Messages

- sending messages

$$a + p \cdot b \cdot q \cdot (1 + c)$$

- by distributivity

$$a \;+\; p \cdot b \cdot q \;+\; p \cdot b \cdot q \cdot c$$

  snapshot, 1-hop connection learnt, content sent
- broadcast, unicast, groupcast are the same (modelled by different topologies)
- Kleene star models flooding the network (modal operators terminate flooding)

Conclusion/Future Work

- So far concentrated on AODV
  - well known
  - IETF standard
- Extend formal methods to other protocols
  - OSLR, DYMO, ...
  - CAN and other communication protocols
  - Open Flow
- Add further necessary concepts
  - time
  - probability (links, measurements)
  - define quality of protocols

Questions

NICTA

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