A Timed Process Algebra for Wireless Networks with an Application in Routing

Emil Bres, Rob van Glabbeek, Peter Höfner

April 2016

Flashback (ESOP 2012)

- "A Process Algebra for Wireless Mesh Networks"
- Summary
  - New Process Algebra Developed
    - language for formalising specs of network protocols
    - key features
      - guaranteed broadcast
      - conditional broadcast
      - data handling
  - Achievements
    - full concise specification of AODV (RFC 3561) (without time)
    - formally verified loop freedom (without timeouts)
    - found several ambiguities, mistakes, shortcomings

Flashback (ESOP 2012)

- "A Process Algebra for Wireless Mesh Networks"
- Summary
  - New Process Algebra Developed
    - language for formalising specs of network protocols
    - key features
      - guaranteed broadcast
      - conditional broadcast
      - data handling
  - Achievements
    - full concise specification of AODV (RFC 3561) (without time)
    - formally verified loop freedom (without timeouts)
    - found several ambiguities, mistakes, shortcomings

Time

- The Need for Time
  - motivated by network protocols
    - timeouts
    - (regularly) scheduled tasks
    - …

- Discrete vs. Continous
  - in practise discrete time is sufficient

Timed Process Algebra: T-AWN

Design Decisions

- Intranode Computations
  - sending messages between nodes takes many microseconds
  - time for intranode computations can be neglected
    (but could be added in the same spirit as time for message sending)
- Guaranteed Message Receipt
  - messages sent ***will*** be received when in transmission range
  - failure of route discovery is imperfection of the protocol
- Input Enabledness
  - models have to be able to receive messages
    (consequence of guaranteed receipt)
- T-AWN Syntax
  - protocol designers should not bother with timing issues
  - **same syntax as AWN**

Syntax: Sequential Processes

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

- Key Features
  - guaranteed broadcast
  - conditional broadcast
  - data handling

T-AWN in Use: AODV

```
RREQ(hops , rreqid , dip , dsn , dsk , oip , osn , sip , ip , sn , rt , rreqs , store) ≝
[[exp_rreqs(rreqs , now)]]
(
   [ (oip , rreqid , ∗) ∈ rreqs ]        /* the RREQ has been received previously */
      AODV(ip , sn , rt , rreqs , store)        /* silently ignore RREQ, i.e., do nothing */
   + [ (oip , rreqid , ∗) ∉ rreqs ]        /* the RREQ is new to this node */
      [[rt := update(rt , (oip, osn, kno, val, hops + 1, sip, ∅, now + ACTIVE_ROUTE_TIMEOUT))]]
      [[rt := setTime_rt(rt , oip , now + 2 · NET_TRAVERSAL_TIME − 2 · (hops + 1) · NODE_TRAVERSAL_TIME)]]
      [[rreqs := rreqs ∪ {(oip, rreqid, now + pathdiscoverytime)}]]        /* update rreqs */
      (
         [ dip = ip ]        /* this node is the destination node */
            [. . .]

         + [ dip ≠ ip ]        /* this node is not the destination node */
            (
               /* valid route to dip that is fresh enough */
               [dip ∈ vD(rt) ∧ dsn ≤ sqn(rt ,dip) ∧ sqnf(rt ,dip) = kno]
                  /* update rt by adding precursors */
                  [[rt := addpreRT(rt , dip , {sip})]]
                  [[rt := addpreRT(rt , oip , {nhop(rt , dip)})]]
                  /* unicast a RREP towards the oip of the RREQ */
                  unicast(nhop(rt , oip),
                           rrep(dhops(rt , dip) , dip , sqn(rt , dip) , oip , σ_time(rt , dip) − now , ip) .
                     AODV(ip , sn , rt , rreqs , store)
                  ▶ /* If the transmission is unsuccessful, a RERR message is generated */
                     [. . .]        /* update local data structure */
                     groupcast(pre , rerr(dests , ip)) . AODV(ip , sn , rt , rreqs , store)
               + [dip ∉ vD(rt) ∨ sqn(rt ,dip) < dsn ∨ sqnf(rt ,dip) = unk]        /* no fresh route */
                  /* no further update of rt */
                  broadcast(rreq(hops+1 , rreqid , dip , max(sqn(rt , dip) , dsn) , dsk , oip , osn , ip))
                  .
                  AODV(ip , sn , rt , rreqs , store)
```

Time Passing

- Time Passes iff
  - message sending (durational action)
  - ready to receive or synchronise (e.g. $\mathbf{send}(ms)$), but some synchronisation partner is not ready
    – implemented by wait-actions

$$P \xrightarrow{\mathrm{w}} \quad \text{iff} \quad P \overset{\mathbf{rcv.}}{\not\longrightarrow} \wedge P \overset{\mathbf{send}}{\not\longrightarrow} \wedge P \overset{\mathbf{other}}{\not\longrightarrow}$$

$$P \xrightarrow{\mathrm{wr}} \quad \text{iff} \quad P \xrightarrow{\mathbf{rcv.}} \wedge P \overset{\mathbf{send}}{\not\longrightarrow} \wedge P \overset{\mathbf{other}}{\not\longrightarrow}$$

T-AWN Processes

- Each AWN process, seen as a T-AWN process, can be (weakly) simulated by the AWN process

  AWN specifications can be analysed w.r.t. time

- A T-AWN process always admits a transition, *independently* of the outside environment.

  No time deadlocks

Theoretical Results

- Process algebra is isomorphic to one without data structure
  - a process for every substitution instance
  - resulting algebra is in (infinitary) de Simone format
  - generates same transition system
    (up to strong bisimulation)
- Hence strong bisimulation and other semantic equivalences are congruences
- Both parallel operators are associative
  (follows by a meta result of Cranen, Mousavi, Reniers)

$$\frac{P_k \xrightarrow{\mathrm{wr}} P'}{\overset{\infty}{\underset{i=k}{\triangle}} P_i \xrightarrow{\mathrm{wr}} \overset{\infty}{\underset{i=k+1}{\triangle}} P_i} \qquad \frac{P_k \xrightarrow{a} P'}{\overset{\infty}{\underset{i=k}{\triangle}} P_i \xrightarrow{a} P'}$$

Case Study: AODV

AODV

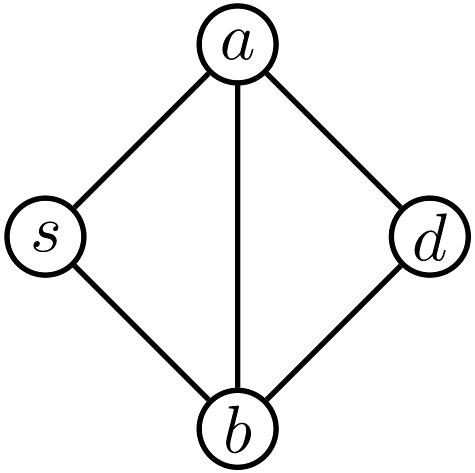
- Ad-hoc On-Demand Distance Vector Routing Protocol
  - routing protocol for wireless mesh networks (wireless networks without wired backbone)

  - Ad hoc (network is not static)
  - On-Demand (routes are established when needed)
  - Distance (metric is hop count)
  - Vector (routing table has the form of a vector)

- Developed 1997-2001 by Perkins, Beldig-Royer and Das (University of Cincinnati)
- basis of IEEE 802.11s

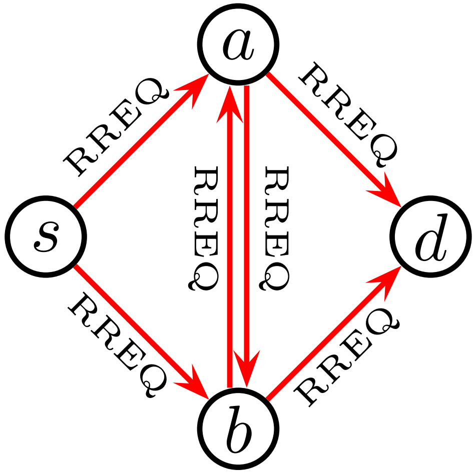
Intuition

- Main Mechanism
  - if route is needed
    BROADCAST RREQ
  - if node has information about destination
    UNICAST RREP
  - if unicast fails or link break is detected
    GROUPCAST RERR
- performance improvement via
  intermediate route reply



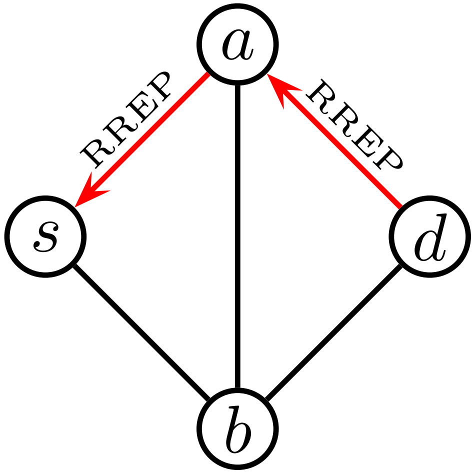
Intuition

- Main Mechanism
  - if route is needed
    BROADCAST RREQ
  - if node has information about destination
    UNICAST RREP
  - if unicast fails or link break is detected
    GROUPCAST RERR
- performance improvement via
  intermediate route reply



Intuition

- Main Mechanism
  - if route is needed
    BROADCAST RREQ
  - if node has information about destination
    UNICAST RREP
  - if unicast fails or link break is detected
    GROUPCAST RERR
- performance improvement via
  intermediate route reply



Formalisation

- (untimed) model
  - full specification of AODV (IETF Standard)
    - around 5 types and 30 functions
    - around 120 lines of specification (in contrast to 40 pages English prose)
    - no ambiguities, no underspecification, no contradictions
- timed model
  - extended with timeouts (routing table entry expiration and deletion)

RREQ Handling

```
RREQ(hops , rreqid , dip , dsn , dsk , oip , osn , sip , ip , sn , rt , rreqs , store) ≝
[[exp_rreqs(rreqs , now)]]
(
   [ (oip , rreqid , ∗) ∈ rreqs ]        /* the RREQ has been received previously */
      AODV(ip , sn , rt , rreqs , store)        /* silently ignore RREQ, i.e., do nothing */
   + [ (oip , rreqid , ∗) ∉ rreqs ]        /* the RREQ is new to this node */
      [[rt := update(rt , (oip, osn, kno, val, hops + 1, sip, ∅, now + ACTIVE_ROUTE_TIMEOUT))]]
      [[rt := setTime_rt(rt , oip , now + 2 · NET_TRAVERSAL_TIME − 2 · (hops + 1) · NODE_TRAVERSAL_TIME)]]
      [[rreqs := rreqs ∪ {(oip, rreqid, now + pathdiscoverytime)}]]        /* update rreqs */
      (
         [ dip = ip ]        /* this node is the destination node */
            [. . .]

         + [ dip ≠ ip ]        /* this node is not the destination node */
            (
               /* valid route to dip that is fresh enough */
               [ dip ∈ vD(rt) ∧ dsn ≤ sqn(rt ,dip) ∧ sqnf(rt ,dip) = kno ]
                  /* update rt by adding precursors */
                  [[rt := addpreRT(rt , dip , {sip})]]
                  [[rt := addpreRT(rt , oip , {nhop(rt , dip)})]]
                  /* unicast a RREP towards the oip of the RREQ */
                  unicast(nhop(rt , oip),
                           rrep(dhops(rt , dip) , dip , sqn(rt , dip) , oip , σ_time(rt , dip) − now , ip) .
                     AODV(ip , sn , rt , rreqs , store)
                  ▶ /* If the transmission is unsuccessful, a RERR message is generated */
                     [. . .]        /* update local data structure */
                     groupcast(pre , rerr(dests , ip)) . AODV(ip , sn , rt , rreqs , store)
               + [ dip ∉ vD(rt) ∨ sqn(rt ,dip) < dsn ∨ sqnf(rt ,dip) = unk ]        /* no fresh route */
                  /* no further update of rt */
                  broadcast(rreq(hops+1 , rreqid , dip , max(sqn(rt , dip) , dsn) , dsk , oip , osn , ip))
                  .
                  AODV(ip , sn , rt , rreqs , store)
```

Two Models of AODV

- The timed version of AODV is a proper extension of the untimed version [ESOP12].

  If all timing constants are set to ∞, then the (T-AWN) transition systems of both versions of AODV are weakly bisimilar.

Loop Freedom

- Loop Freedom of (untimed) AODV
  - 5184 possible interpretations due to ambiguities
  - 5006 of these readings of the standard contain loops
    (1 “default” variant has been proven loop free;
    the remaining 177 are loop free, adapting the default proof)
  - 3 out of 5 open-source implementations contain loops

- Loop Freedom of the default reading of (timed) AODV
  - by meta theory:
    if all timing constants are set to ∞, then loop free
  - however, it contains *time loops*

Time Loops

- Premature Route Expiration (Deletion)

  - node C had route to D before;
    B uses this information
  - B stores information longer than C
  - can be avoided by not deleting information (invalidating is still fine)

- Premature Route Expiration is the ***only*** cause of loops

Premature Route Expiration

- "Trivial" Cases
  - messages spend an inordinate amount of time in the in-queue (usually does not occur)

    Assumption 1: the transmission time of a message plus the period it spends in the queue of incoming messages of the receiving node is bounded by `NODE_TRAVERSAL_TIME`

  - early deletion before reply arrives

    Assumption 2: the period a RREQ travels through the network is bounded by `NET_TRAVERSAL_TIME`

Premature Route Expiration

- 5 lines of our formal specification can yield premature route expiration and hence ***routing/time loops***
  - in contrast to the RFC
  - in contrast to the main paper of AODV (13,000 citations)
  - in contrast to common belief

(adapting the (untimed) invariant proof revealed these problems)

- possible fixes lead to loop freedom
  - skip the 5 lines (may change the intention of AODV)
  - change lines
    (add condition, change time outs, …)

Conclusion and Outlook

Conclusion

- expanded process algebra for wireless networks
  - unique set of features
  - used for protocol analysis of industrial size
  - qualitative analysis, e.g. loop freedom, packet delivery

Outlook

- model, analyse and compare other protocols (e.g. B.A.T.M.A.N, OLSR)
  - what does it mean that protocol A is better than B
- quantitative analysis
  - for example “how long does it take until a packet is delivered”
- add probability
  - model unreliable links (quantitative analysis)
  - model “probabilistic protocols” such as CMSA

Thank you

www.data61.csiro.au

Related Work

Process algebra			Message loss	Type of broadcast		Connectivity model		
CBS	[88]	'91	enforced synchr.	global broadcast				symmetric
$b\pi$	[22]	'99	enforced synchr.	subscription-based broadcast				symmetric
CBS#	[74]	'06	enforced synchr.	local bc.	dynamic top.	$n[P,S]$	op. sem.	symmetric
CWS	[69]	'06	enforced synchr.	local bc.	static topology	$n[P]^{c}_{l,r}$	node	symmetric
CMAN	[40]	'07	lossy broadcast	local bc.	dynamic top.	$\lfloor p \rfloor^{\sigma}_{l}$	node	symmetric
CMN	[66]	'07	lossy broadcast	local bc.	dynamic top.	$n[P]^{\mu}_{l,r}$	node	symmetric
ω	[94]	'07	lossy broadcast	local bc.	dynamic top.	$P:G$	node	symmetric
RBPT	[34]	'08	lossy broadcast	local bc.	dynamic top.	$[\![P]\!]_{l}$	op. sem.	asymmetric
$bA\pi$	[42]	'09	lossy broadcast	local bc.	dynamic top.	$\lfloor p \rfloor_{l}$	network	asymmetric
$b\psi$	[9]	'11	lossy broadcast	local bc.	dynamic top.	P	op. sem.	asymmetric
AWN	here	'11	enforced synchr. with guar. receipt	local bc.	dynamic top.	$ip{:}P{:}R$	node	asym./sym.