Modelling and Verification of Protocols for Wireless Networks

(Lecture 2)

Peter Höfner

(Lecture at University of Twente, Jan/Feb 2017)

www.data61.csiro.au

last update: Jan 22, 2017

Contents of this Lecture

What should you have learnt

- What is a Structural Operational Semantics
  - recap ?!?
- Semantics of (untimed) AWN
  - intuition
  - level-based approach
  - detailed sos-rules

AWN: A Layered Approach

AWN: A brief Recap

- AWN (Algebra for Wireless Networks)
  - key features
    - local broadcast
    - prioritised unicast
    - data structure
    - dynamic topologies
    - …

Structure of Wireless Networks (2)

- Users
  - 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

The Process Algebra AWN

Structural Operational Semantics

Structural Operational Semantics (1)

Overview

- one of the main methods for defining the meaning in description languages
- *system behaviour* is represented
  - by a closed term built from a collection of operators
  - the behaviour of a process is given by its collection of (outgoing) transitions
- formal definitions to be found at the webpage

Structural Operational Semantics (2)

Example

The following fragment of CCS (Calculus of Communicating Systems) has the constant 0, unary operators $a.$ for $a \in \mathrm{Act}$, binary operators $+$ and $\|$, and the SOS rules below, one for every $\alpha \in \mathrm{Act}$ and $a \in A$ Here $\mathrm{Act} = A \dot\cup \{\tau\}$ (actions) and $A = N \dot\cup \overline{N}$ with N a set of names and $\overline{N} = \{\overline{a} \mid a \in N\}$ the set of co-names.

$$\frac{x_1 \xrightarrow{\alpha} y_1}{x_1 + x_2 \xrightarrow{\alpha} y_1} \qquad \frac{x_2 \xrightarrow{\alpha} y_2}{x_1 + x_2 \xrightarrow{\alpha} y_2} \qquad a.x_1 \xrightarrow{\alpha} x_1$$

$$\frac{x_1 \xrightarrow{\alpha} y_1}{x_1 \| x_2 \xrightarrow{\alpha} y_1 \| x_2} \qquad \frac{x_2 \xrightarrow{\alpha} y_2}{x_1 \| x_2 \xrightarrow{\alpha} x_1 \| y_2} \qquad \frac{x_1 \xrightarrow{a} y_1 \quad x_2 \xrightarrow{\overline{a}} y_2}{x_1 \| x_2 \xrightarrow{\tau} y_1 \| y_2}$$

A Language for Sequential Processes

syntax

- defining equation

- sequential process expressions

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

A Language for Sequential Processes
syntax

- defining equation

$$X(\texttt{var}_1, \ldots, \texttt{var}_n) \stackrel{def}{=} p \ ,$$

- sequential process expressions

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

A Language for Sequential Processes

syntax

- defining equation

$$X(\mathtt{var}_1, \ldots, \mathtt{var}_n) \stackrel{def}{=} p\ ,$$

process name

- sequential process expressions

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

A Language for Sequential Processes

syntax

- defining equation

$$X(\mathtt{var}_1, \ldots, \mathtt{var}_n) \stackrel{def}{=} p \ ,$$

process name

variables

- sequential process expressions

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

A Language for Sequential Processes

syntax

- defining equation

$$X(\texttt{var}_1, \ldots, \texttt{var}_n) \stackrel{def}{=} p\ ,$$

process name

variables

sequential process expressions

- sequential process expressions

$$\begin{array}{rcl} SP & ::= & X(\mathit{exp}_1, \ldots, \mathit{exp}_n) \ \mid\ [\varphi]SP \ \mid\ [\![\texttt{var} := \mathit{exp}]\!]SP \ \mid\ SP + SP \ \mid \\ & & \alpha.SP \ \mid\ \mathbf{unicast}(\mathit{dest}, \mathit{ms}).SP \blacktriangleright SP \\ \alpha & ::= & \mathbf{broadcast}(\mathit{ms}) \ \mid\ \mathbf{groupcast}(\mathit{dests}, \mathit{ms}) \ \mid\ \mathbf{send}(\mathit{ms}) \ \mid \\ & & \mathbf{deliver}(\mathit{data}) \ \mid\ \mathbf{receive}(\texttt{msg}) \end{array}$$

A Language for Sequential Processes

mandatory types & valuation function

- mandatory types
  - data structure always contains the types

$$\mathtt{DATA},\ \mathtt{MSG},\ \mathtt{IP} \text{ and } \mathcal{P}(\mathtt{IP}) \text{ of}$$

    – application layer data,
    – messages,
    – IP addresses—or any other node identifiers
    – sets of IP addresses
- sequential process expressions
  - seq. process expression p together with a *valuation* ξ *determines state*
  - values $\xi(\mathtt{var})$ associated to variables $\mathtt{var}$

A Language for Sequential Processes

structural operational semantics (1)

$$\xi, \textbf{broadcast}(ms).p \xrightarrow{\textbf{broadcast}(\xi(ms))} \xi, p$$

$$\xi, \textbf{groupcast}(dests, ms).p \xrightarrow{\textbf{groupcast}(\xi(dests),\xi(ms))} \xi, p$$

$$\xi, \textbf{unicast}(dest, ms).p \blacktriangleright q \xrightarrow{\textbf{unicast}(\xi(dest),\xi(ms))} \xi, p$$

$$\xi, \textbf{unicast}(dest, ms).p \blacktriangleright q \xrightarrow{\neg\textbf{unicast}(\xi(dest))} \xi, q$$

$$\xi, \textbf{send}(ms).p \xrightarrow{\textbf{send}(\xi(ms))} \xi, p$$

$$\xi, \textbf{deliver}(data).p \xrightarrow{\textbf{deliver}(\xi(data))} \xi, p$$

$$\xi, \textbf{receive}(\texttt{msg}).p \xrightarrow{\textbf{receive}(m)} \xi[\texttt{msg} := m], p \qquad (\forall m \in \texttt{MSG})$$

$$\xi, [\![\texttt{var} := exp]\!]p \xrightarrow{\tau} \xi[\texttt{var} := \xi(exp)], p$$

A Language for Sequential Processes

structural operational semantics (1)

just names

$$\xi, \mathbf{broadcast}(ms).p \xrightarrow{\mathbf{broadcast}(\xi(ms))} \xi, p$$

$$\xi, \mathbf{groupcast}(dests, ms).p \xrightarrow{\mathbf{groupcast}(\xi(dests),\xi(ms))} \xi, p$$

$$\xi, \mathbf{unicast}(dest, ms).p \blacktriangleright q \xrightarrow{\mathbf{unicast}(\xi(dest),\xi(ms))} \xi, p$$

$$\xi, \mathbf{unicast}(dest, ms).p \blacktriangleright q \xrightarrow{\neg\mathbf{unicast}(\xi(dest))} \xi, q$$

$$\xi, \mathbf{send}(ms).p \xrightarrow{\mathbf{send}(\xi(ms))} \xi, p$$

$$\xi, \mathbf{deliver}(data).p \xrightarrow{\mathbf{deliver}(\xi(data))} \xi, p$$

$$\xi, \mathbf{receive}(\mathtt{msg}).p \xrightarrow{\mathbf{receive}(m)} \xi[\mathtt{msg} := m], p \qquad (\forall m \in \mathtt{MSG})$$

$$\xi, [\![\mathtt{var} := exp]\!]p \xrightarrow{\tau} \xi[\mathtt{var} := \xi(exp)], p$$

A Language for Sequential Processes

structural operational semantics (2)

A Language for Sequential Processes

structural operational semantics (2)

$$\frac{\xi, p \xrightarrow{a} \zeta, p'}{\xi, p + q \xrightarrow{a} \zeta, p'} \qquad \frac{\xi, q \xrightarrow{a} \zeta, q'}{\xi, p + q \xrightarrow{a} \zeta, q'}$$

A Language for Sequential Processes

structural operational semantics (2)

$$\frac{\xi, p \xrightarrow{a} \zeta, p'}{\xi, p + q \xrightarrow{a} \zeta, p'} \qquad \frac{\xi, q \xrightarrow{a} \zeta, q'}{\xi, p + q \xrightarrow{a} \zeta, q'} \qquad \frac{\xi \xrightarrow{\varphi} \zeta}{\xi, [\varphi]p \xrightarrow{\tau} \zeta, p} \qquad (\forall a \in \mathrm{Act})$$

A Language for Sequential Processes

structural operational semantics (2)

$$\frac{\xi, p \xrightarrow{a} \zeta, p'}{\xi, p+q \xrightarrow{a} \zeta, p'} \quad \frac{\xi, q \xrightarrow{a} \zeta, q'}{\xi, p+q \xrightarrow{a} \zeta, q'} \quad \frac{\xi \xrightarrow{\varphi} \zeta}{\xi, [\varphi]p \xrightarrow{\tau} \zeta, p} \qquad (\forall a \in \mathrm{Act})$$

$$\frac{\emptyset[\mathtt{var}_i := \xi(exp_i)]_{i=1}^{n}, p \xrightarrow{a} \zeta, p'}{\xi, X(exp_1, \ldots, exp_n) \xrightarrow{a} \zeta, p'} \ (X(\mathtt{var}_1, \ldots, \mathtt{var}_n) \stackrel{def}{=} p) \ \ (\forall a \in \mathrm{Act})$$

Example Flooding

Process 1 Flooding

```
FLOOD(ip , m , b , store) def=
  (receive(ms) .
     /* check message format and distill contents */
     [ ms = msg(ip′, m′) ]
       (
         [ store(ip′) = m′ ]        /* message handled before */
           FLOOD(ip,m,b,store)
         + [ store(ip′) ≠ m′ ]        /* new message */
           ⟦store(ip′) = m′⟧
           broadcast(ms) .
           FLOOD(ip,m,b,store)
       ))
  + [ b = false ]        /* message not yet send */
     broadcast(msg(ip, m)) . FLOOD(ip,m,true,store)
```

A Language for Parallel Processes

syntax & sos-rules

- syntax

$$PP \;::=\; \xi, SP \;\mid\; PP \langle\!\langle\, PP\,,$$

- structural operational semantics (sos)

$$\frac{P \xrightarrow{a} P'}{P \langle\!\langle\, Q \xrightarrow{a} P' \langle\!\langle\, Q} \quad (\forall a \neq \mathbf{receive}(m))$$

$$\frac{Q \xrightarrow{a} Q'}{P \langle\!\langle\, Q \xrightarrow{a} P \langle\!\langle\, Q'} \quad (\forall a \neq \mathbf{send}(m))$$

$$\frac{P \xrightarrow{\mathbf{receive}(m)} P' \qquad Q \xrightarrow{\mathbf{send}(m)} Q'}{P \langle\!\langle\, Q \xrightarrow{\tau} P' \langle\!\langle\, Q'} \quad (\forall m \in \mathtt{MSG})$$

Example Flooding, including Queue

A Language for Networks
syntax & sos-rules

- syntax

$$N ::= [M] \qquad\qquad M ::= \quad ip : PP : R \quad | \quad M \| M$$

- structural operational semantics (sos)

A Language for Networks

syntax & sos-rules

- syntax

$$N ::= [M] \qquad\qquad M ::= \quad ip : PP : R \quad | \quad M \| M$$

unique identifier

- structural operational semantics (sos)

A Language for Networks

syntax & sos-rules

- syntax

$$N ::= [M] \qquad\qquad M ::= \quad ip : PP : R \quad | \quad M \| M$$

unique identifier

parallel process expression

- structural operational semantics (sos)

A Language for Networks

syntax & sos-rules

- syntax

$$N ::= [M] \qquad\qquad M ::= \quad ip : PP : R \quad | \quad M \| M$$

unique identifier

parallel process expression

nodes within transmission range

- structural operational semantics (sos)

A Language for Networks

syntax & sos-rules

- syntax

$$N ::= [M] \qquad\qquad M ::= \quad ip : PP : R \quad \mid \quad M \| M$$

unique identifier

parallel process expression

nodes within transmission range

- structural operational semantics (sos)

$$\frac{P \xrightarrow{\mathbf{broadcast}(m)} P'}{ip:P:R \xrightarrow{R:\mathbf{*cast}(m)} ip:P':R} \qquad \frac{P \xrightarrow{\mathbf{groupcast}(D,m)} P'}{ip:P:R \xrightarrow{R\cap D:\mathbf{*cast}(m)} ip:P':R}$$

$$\frac{P \xrightarrow{\mathbf{unicast}(dip,m)} P' \qquad dip \in R}{ip:P:R \xrightarrow{\{dip\}:\mathbf{*cast}(m)} ip:P':R} \qquad \frac{P \xrightarrow{\neg\mathbf{unicast}(dip)} P' \qquad dip \notin R}{ip:P:R \xrightarrow{\tau} ip:P':R}$$

A Language for Networks

syntax & sos-rules

- structural operational semantics (cont'd)

$$\frac{P \xrightarrow{\textbf{deliver}(d)} P'}{ip:P:R \xrightarrow{ip:\textbf{deliver}(d)} ip:P':R} \qquad \frac{P \xrightarrow{\textbf{receive}(m)} P'}{ip:P:R \xrightarrow{\{ip\}\neg\emptyset:\textbf{arrive}(m)} ip:P':R}$$

$$\frac{P \xrightarrow{\tau} P'}{ip:P:R \xrightarrow{\tau} ip:P':R} \qquad ip:P:R \xrightarrow{\emptyset\neg\{ip\}:\textbf{arrive}(m)} ip:P:R$$

A Language for Networks

syntax & sos-rules

- structural operational semantics (cont'd)

$$\frac{P \xrightarrow{\mathbf{deliver}(d)} P'}{ip\!:\!P\!:\!R \xrightarrow{ip\,:\,\mathbf{deliver}(d)} ip\!:\!P'\!:\!R} \qquad \frac{P \xrightarrow{\mathbf{receive}(m)} P'}{ip\!:\!P\!:\!R \xrightarrow{\{ip\}\neg\emptyset\,:\,\mathbf{arrive}(m)} ip\!:\!P'\!:\!R}$$

$$\frac{P \xrightarrow{\tau} P'}{ip\!:\!P\!:\!R \xrightarrow{\tau} ip\!:\!P'\!:\!R} \qquad ip\!:\!P\!:\!R \xrightarrow{\emptyset\neg\{ip\}\,:\,\mathbf{arrive}(m)} ip\!:\!P\!:\!R$$

$$ip : P : R \xrightarrow{\mathbf{connect}(ip,ip')} ip : P : R \cup \{ip'\} \qquad ip : P : R \xrightarrow{\mathbf{disconnect}(ip,ip')} ip : P : R - \{ip'\}$$

A Language for Networks

syntax & sos-rules

- structural operational semantics (cont'd)

$$\frac{P \xrightarrow{\textbf{deliver}(d)} P'}{ip\!:\!P\!:\!R \xrightarrow{ip\,:\,\textbf{deliver}(d)} ip\!:\!P'\!:\!R} \qquad \frac{P \xrightarrow{\textbf{receive}(m)} P'}{ip\!:\!P\!:\!R \xrightarrow{\{ip\}\neg\emptyset\,:\,\textbf{arrive}(m)} ip\!:\!P'\!:\!R}$$

$$\frac{P \xrightarrow{\tau} P'}{ip\!:\!P\!:\!R \xrightarrow{\tau} ip\!:\!P'\!:\!R} \qquad ip\!:\!P\!:\!R \xrightarrow{\emptyset\neg\{ip\}\,:\,\textbf{arrive}(m)} ip\!:\!P\!:\!R$$

$$ip : P : R \xrightarrow{\textbf{connect}(ip,ip')} ip : P : R \cup \{ip'\} \qquad ip : P : R \xrightarrow{\textbf{disconnect}(ip,ip')} ip : P : R - \{ip'\}$$

$$ip : P : R \xrightarrow{\textbf{connect}(ip',ip)} ip : P : R \cup \{ip'\} \qquad ip : P : R \xrightarrow{\textbf{disconnect}(ip',ip)} ip : P : R - \{ip'\}$$

A Language for Networks

syntax & sos-rules

- structural operational semantics (cont'd)

$$\frac{P \xrightarrow{\mathbf{deliver}(d)} P'}{ip\!:\!P\!:\!R \xrightarrow{ip\,:\,\mathbf{deliver}(d)} ip\!:\!P'\!:\!R} \qquad \frac{P \xrightarrow{\mathbf{receive}(m)} P'}{ip\!:\!P\!:\!R \xrightarrow{\{ip\}\neg\emptyset\,:\,\mathbf{arrive}(m)} ip\!:\!P'\!:\!R}$$

$$\frac{P \xrightarrow{\tau} P'}{ip\!:\!P\!:\!R \xrightarrow{\tau} ip\!:\!P'\!:\!R} \qquad ip\!:\!P\!:\!R \xrightarrow{\emptyset\neg\{ip\}\,:\,\mathbf{arrive}(m)} ip\!:\!P\!:\!R$$

$$ip : P : R \xrightarrow{\mathbf{connect}(ip,ip')} ip : P : R \cup \{ip'\} \qquad ip : P : R \xrightarrow{\mathbf{disconnect}(ip,ip')} ip : P : R - \{ip'\}$$

$$ip : P : R \xrightarrow{\mathbf{connect}(ip',ip)} ip : P : R \cup \{ip'\} \qquad ip : P : R \xrightarrow{\mathbf{disconnect}(ip',ip)} ip : P : R - \{ip'\}$$

$$\frac{ip \notin \{ip', ip''\}}{ip : P : R \xrightarrow{\mathbf{connect}(ip',ip'')} ip : P : R} \qquad \frac{ip \notin \{ip', ip''\}}{ip : P : R \xrightarrow{\mathbf{disconnect}(ip',ip'')} ip : P : R}$$

A Language for Networks

syntax & sos-rules

$$\frac{M \xrightarrow{R:*\mathbf{cast}(m)} M' \quad N \xrightarrow{H\neg K:\mathbf{arrive}(m)} N'}{M \| N \xrightarrow{R:*\mathbf{cast}(m)} M' \| N'} \left(\begin{array}{c} H \subseteq R \\ K \cap R = \emptyset \end{array}\right) \qquad \frac{M \xrightarrow{H\neg K:\mathbf{arrive}(m)} M' \quad N \xrightarrow{R:*\mathbf{cast}(m)} N'}{M \| N \xrightarrow{R:*\mathbf{cast}(m)} M' \| N'} \left(\begin{array}{c} H \subseteq R \\ K \cap R = \emptyset \end{array}\right)$$

$$\frac{M \xrightarrow{H\neg K:\mathbf{arrive}(m)} M' \quad N \xrightarrow{H'\neg K':\mathbf{arrive}(m)} N'}{M \| N \xrightarrow{(H\cup H')\neg(K\cup K'):\mathbf{arrive}(m)} M' \| N'}$$

$$\frac{M \xrightarrow{ip:\mathbf{deliver}(d)} M'}{M \| N \xrightarrow{ip:\mathbf{deliver}(d)} M' \| N} \qquad \frac{N \xrightarrow{ip:\mathbf{deliver}(d)} N'}{M \| N \xrightarrow{ip:\mathbf{deliver}(d)} M \| N'} \qquad \frac{M \xrightarrow{\tau} M'}{M \| N \xrightarrow{\tau} M' \| N} \qquad \frac{N \xrightarrow{\tau} N'}{M \| N \xrightarrow{\tau} M \| N'}$$

$$\frac{M \xrightarrow{\mathbf{connect}(ip,ip')} M' \quad N \xrightarrow{\mathbf{connect}(ip,ip')} N'}{M \| N \xrightarrow{\mathbf{connect}(ip,ip')} M' \| N'} \qquad \frac{M \xrightarrow{\mathbf{disconnect}(ip,ip')} M' \quad N \xrightarrow{\mathbf{disconnect}(ip,ip')} N'}{M \| N \xrightarrow{\mathbf{disconnect}(ip,ip')} M' \| N'}$$

$$\frac{M \xrightarrow{\mathbf{connect}(ip,ip')} M'}{[M] \xrightarrow{\mathbf{connect}(ip,ip')} [M']} \qquad \frac{M \xrightarrow{\mathbf{disconnect}(ip,ip')} M'}{[M] \xrightarrow{\mathbf{disconnect}(ip,ip')} [M']} \qquad \frac{M \xrightarrow{R:*\mathbf{cast}(m)} M'}{[M] \xrightarrow{\tau} [M']} \qquad \frac{M \xrightarrow{\tau} M'}{[M] \xrightarrow{\tau} [M']}$$

$$\frac{M \xrightarrow{ip:\mathbf{deliver}(d)} M'}{[M] \xrightarrow{ip:\mathbf{deliver}(d)} [M']} \qquad \frac{M \xrightarrow{\{ip\}\neg K:\mathbf{arrive}(\mathtt{newpkt}(d,dip))} M'}{[M] \xrightarrow{ip:\mathbf{newpkt}(d,dip)} [M']}$$

Example Flooding, 2 Node Topology

AWN: Some results

- both parallel operations are associative
- the outer one is commutative

- the process algebra is blocked (hence requires input-enabled processes)

- result follow from meta theory by de Simone

AWN: Some results

- both parallel operations are associative
- the outer one is commutative

- the process algebra is blocked (hence requires input-enabled processes)

$$\frac{P \xrightarrow{\textbf{receive}(m)}\!\!\!\!\!\!/\;\;\;}{ip{:}P{:}R \xrightarrow{\{ip\}\neg\emptyset : \textbf{arrive}(m)} ip{:}P{:}R}$$

- result follow from meta theory by de Simone

References

- Rob van Glabbeek. Structural Operational Semantics - The main definitions. available at the webpage
- A. Fehnker, R.J. van Glabbeek, P. Höfner, A. McIver, M. Portmann and W.L. Tan: *A Process Algebra for Wireless Mesh Networks.* In H. Seidl (ed.), Programming Languages and Systems (ESOP'12), Lecture Notes in Computer Science 7211, 295–315, Springer, 2012.
  doi: 10.1007/978-3-642-28869-2_15
- A. Fehnker, R.J. van Glabbeek, P. Höfner, M. Portmann, A. McIver and W.L. Tan: *A Process Algebra for Wireless Mesh Networks used for Modelling, Verifying and Analysing AODV.* Technical Report 5513, NICTA. 2013.
  arXiv: CoRR abs/1312.7645