DATA 61

Modelling and Verification of Protocols for Wireless Networks

(Lecture 3)

Peter Höfner

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

www.data61.csiro.au

UNSW AUSTRALIA

CSIRO

last update: Jan 23, 2017

Contents of this Lecture

What should you have learnt

- A Case Study
  - how AWN can be used in “real life”

- Properties of Routing Protocols
  - liveness vs. safety properties
  - examples for routing
  - reachability vs. invariants

AODV (untimed)
A Case Study for using AWN

Case Study: AODV

$+$ [$(\texttt{oip},\texttt{rreqid}) \notin \texttt{rreqs}$] /* the RREQ is new to this node */
⟦$\texttt{rt} := \texttt{update}(\texttt{rt},(\texttt{oip},\texttt{osn},\texttt{kno},\texttt{val},\texttt{hops}+1,\texttt{sip},\emptyset))$⟧ /* update the route to oip in rt */
⟦$\texttt{rreqs} := \texttt{rreqs} \cup \{(\texttt{oip},\texttt{rreqid})\}$⟧ /* update rreqs by adding (oip, rreqid) */
(
[$\texttt{dip} = \texttt{ip}$] /* this node is the destination node */
⟦$\texttt{sn} := \max(\texttt{sn},\texttt{dsn})$⟧ /* update the sqn of ip */
/* unicast a RREP towards oip of the RREQ */
unicast(nhop(rt,oip),rrep(0,dip,sn,oip,ip)) . AODV(ip,sn,rt,rreqs,store)
▶ /* If the transmission is unsuccessful, a RERR message is generated */
⟦$\texttt{dests} := \{(\texttt{rip},\texttt{inc}(\texttt{sqn}(\texttt{rt},\texttt{rip}))) \mid \texttt{rip} \in \texttt{vD}(\texttt{rt}) \wedge \texttt{nhop}(\texttt{rt},\texttt{rip}) = \texttt{nhop}(\texttt{rt},\texttt{oip})\}$⟧
⟦$\texttt{rt} := \texttt{invalidate}(\texttt{rt},\texttt{dests})$⟧
⟦$\texttt{store} := \texttt{setRRF}(\texttt{store},\texttt{dests})$⟧
⟦$\texttt{pre} := \bigcup\{\texttt{precs}(\texttt{rt},\texttt{rip}) \mid (\texttt{rip},*) \in \texttt{dests}\}$⟧
⟦$\texttt{dests} := \{(\texttt{rip},\texttt{rsn}) \mid (\texttt{rip},\texttt{rsn}) \in \texttt{dests} \wedge \texttt{precs}(\texttt{rt},\texttt{rip}) \neq \emptyset\}$⟧
groupcast(pre,rerr(dests,ip)) . AODV(ip,sn,rt,rreqs,store)
$+$ [$\texttt{dip} \neq \texttt{ip}$] /* this node is not the destination node */
(
[$\texttt{dip} \in \texttt{vD}(\texttt{rt}) \wedge \texttt{dsn} \leq \texttt{sqn}(\texttt{rt},\texttt{dip}) \wedge \texttt{sqnf}(\texttt{rt},\texttt{dip}) = \texttt{kno}$] /* valid route to dip that is fresh enough */
/* update rt by adding precursors */
⟦$\texttt{rt} := \texttt{addpreRT}(\texttt{rt},\texttt{dip},\{\texttt{sip}\})$⟧
⟦$\texttt{rt} := \texttt{addpreRT}(\texttt{rt},\texttt{oip},\{\texttt{nhop}(\texttt{rt},\texttt{dip})\})$⟧
/* unicast a RREP towards the oip of the RREQ */
unicast(nhop(rt,oip),rrep(dhops(rt,dip),dip,sqn(rt,dip),oip,ip)) .

Case Study: AODV

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

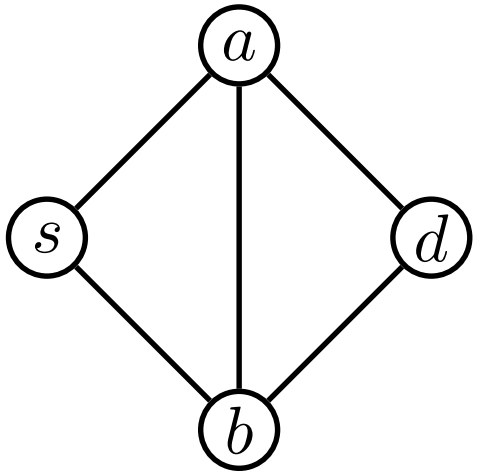
  - Ad hoc (network is not static)
  - On-Demand (routes are established when needed)
  - Distance (metric is hop count)

  - developed 1997-2001 by Perkins, Beldig-Royer and Das (University of Cincinnati)

  - one of the four protocols standardised by the IETF MANET working group (IEEE 802.11s)

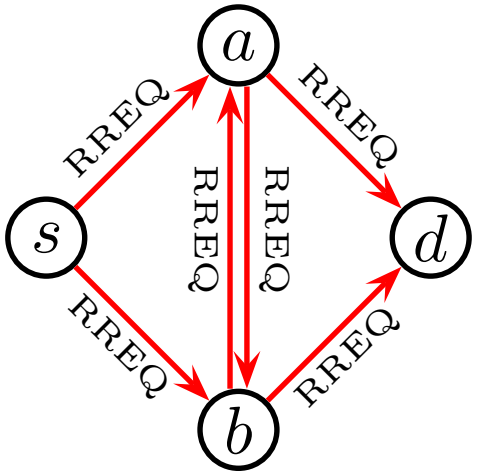
Case Study

- 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
  - performance improvement via intermediate route reply



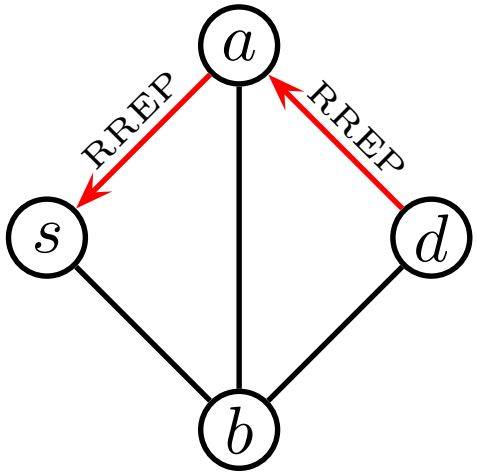
Case Study

- 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
  - performance improvement via intermediate route reply

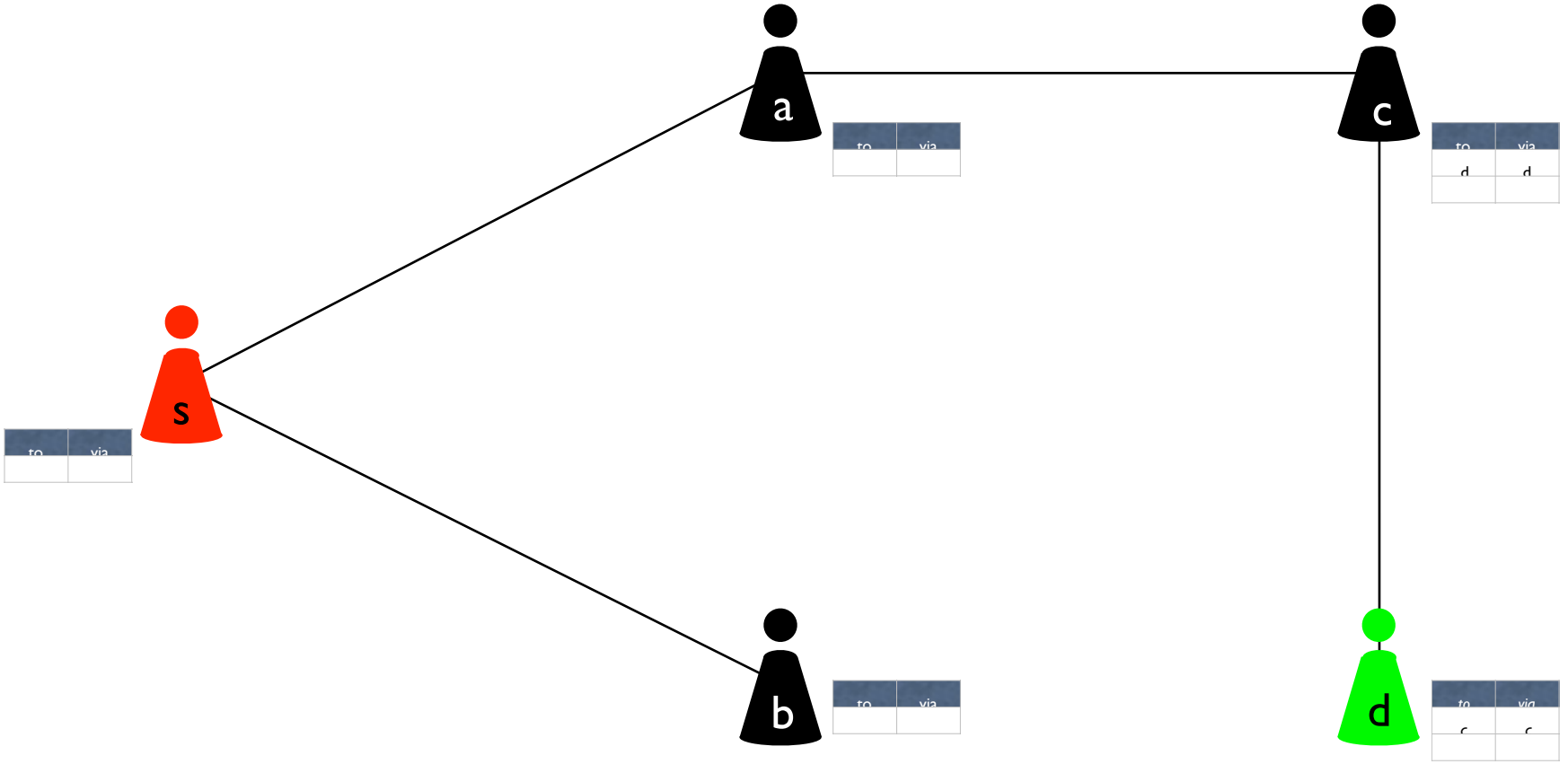


Case Study

- 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
  - performance improvement via intermediate route reply

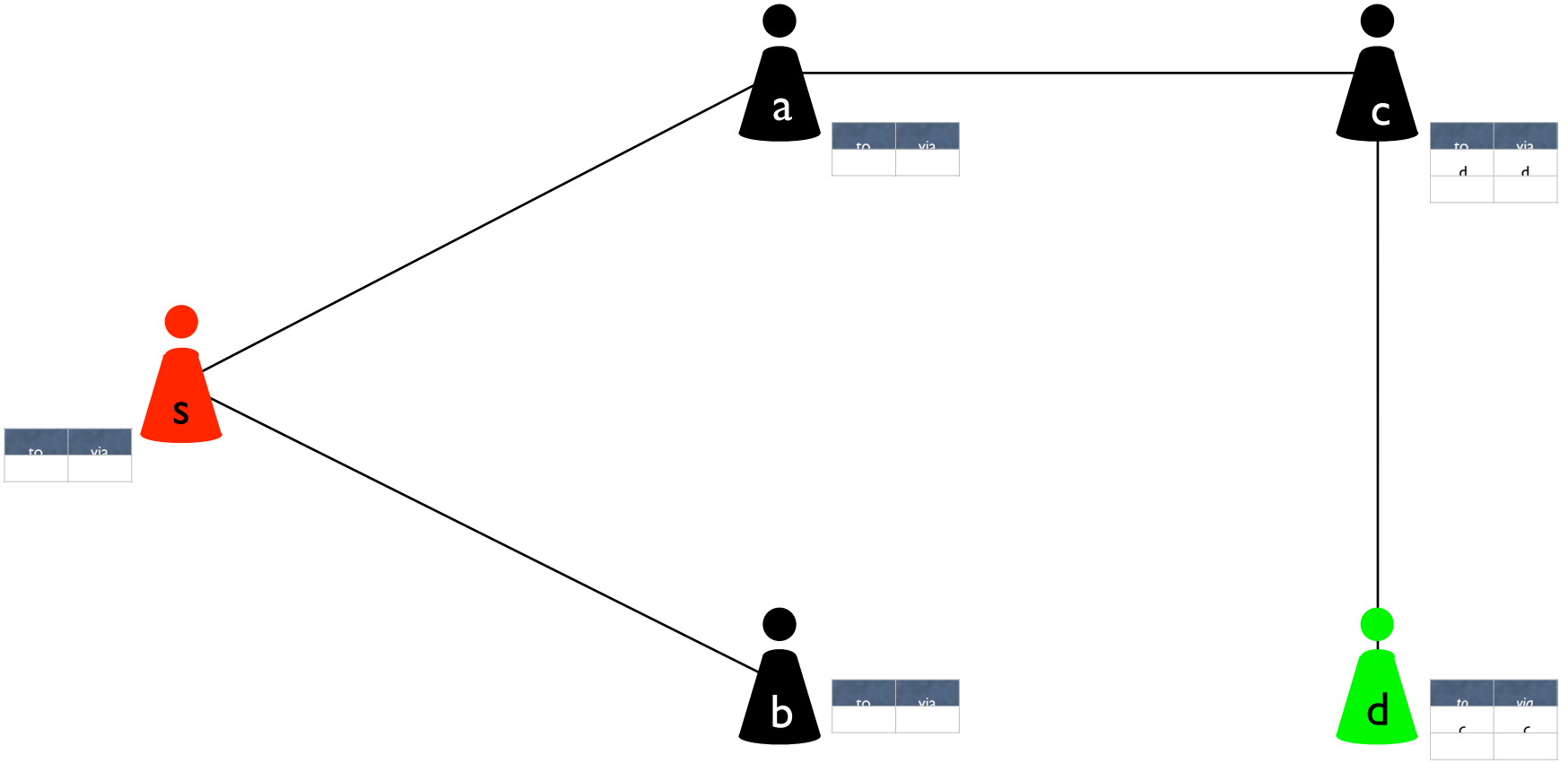


AODV - An Example

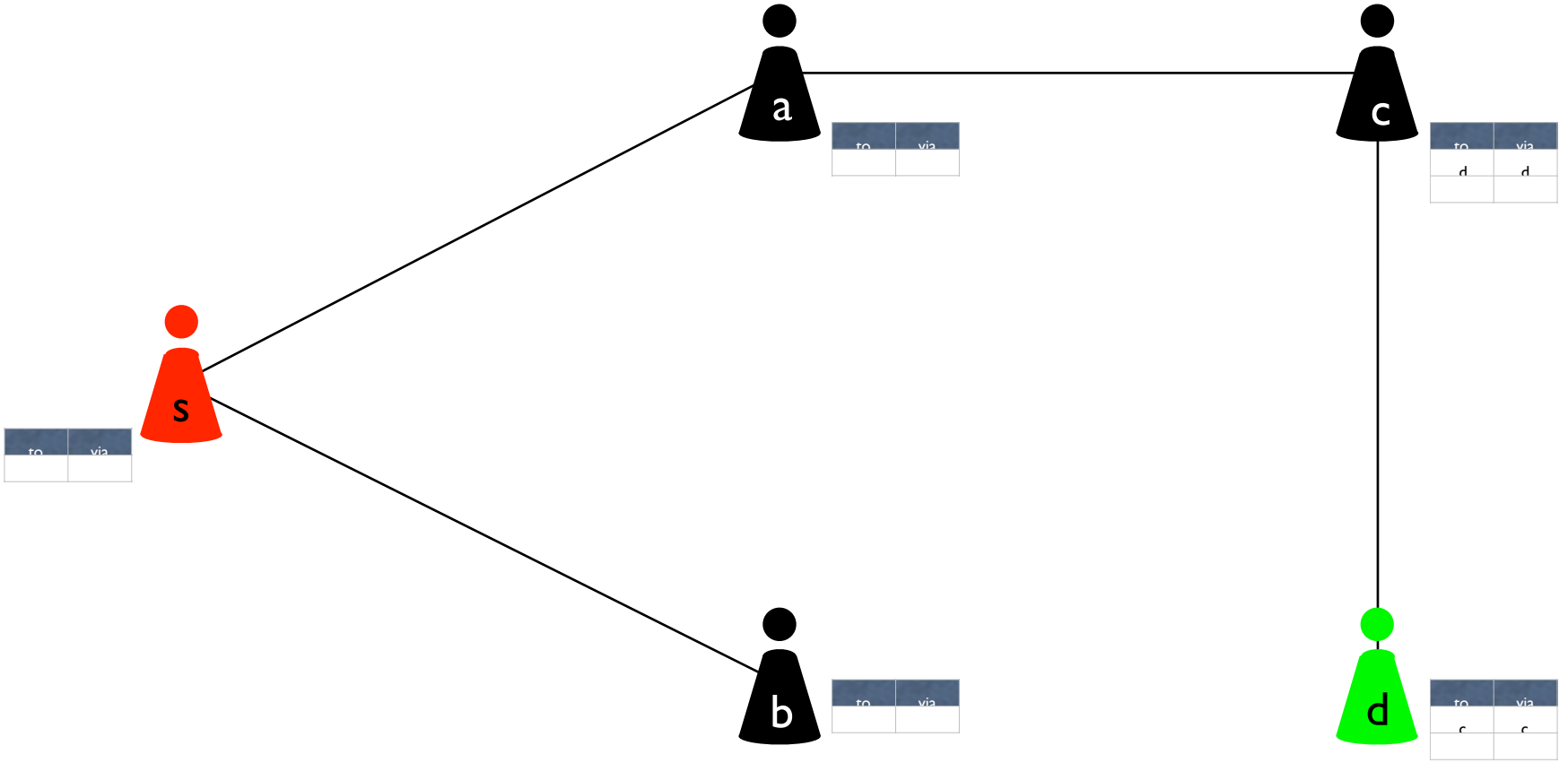


s is looking for a route to d

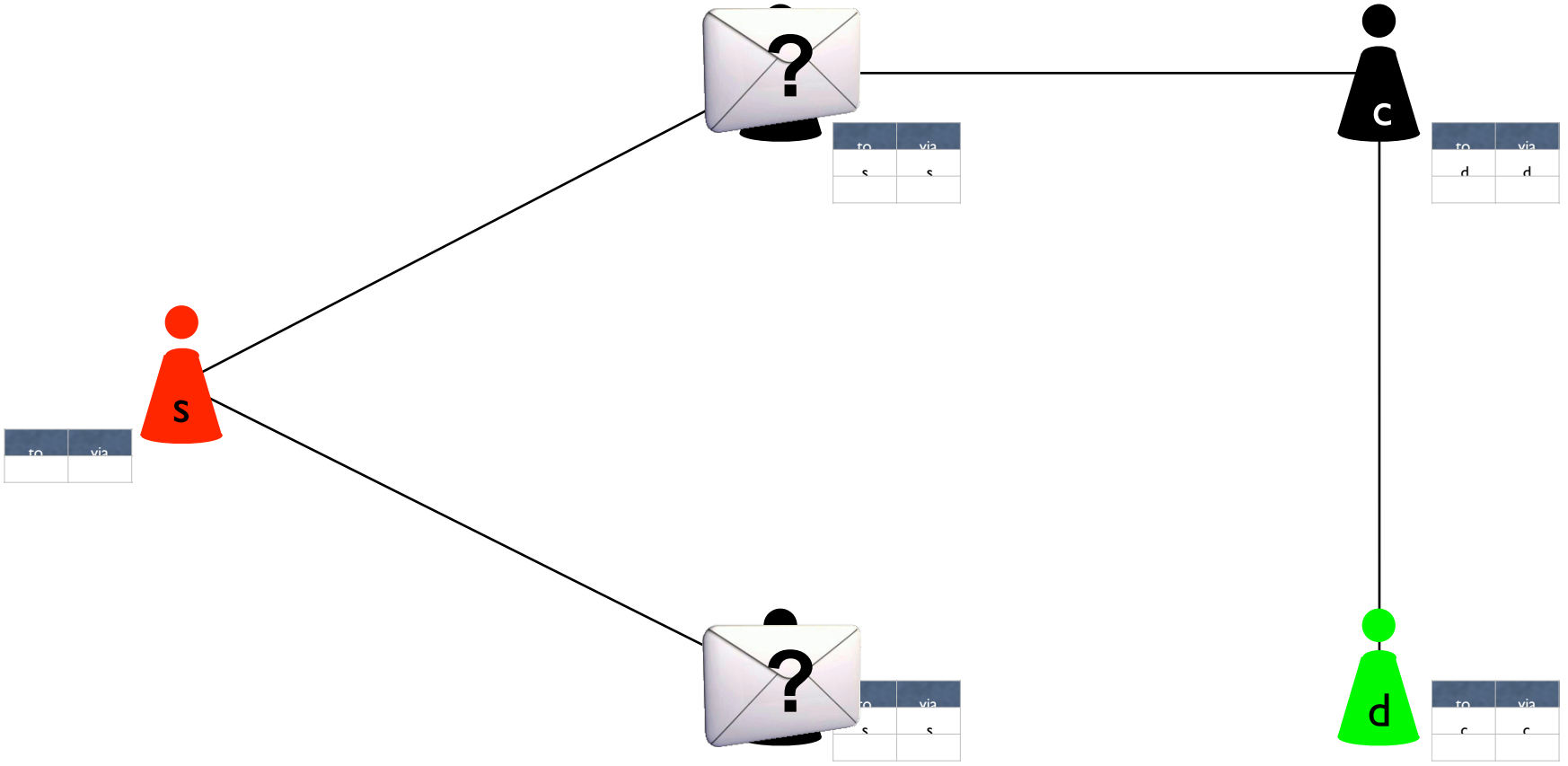
AODV - An Example



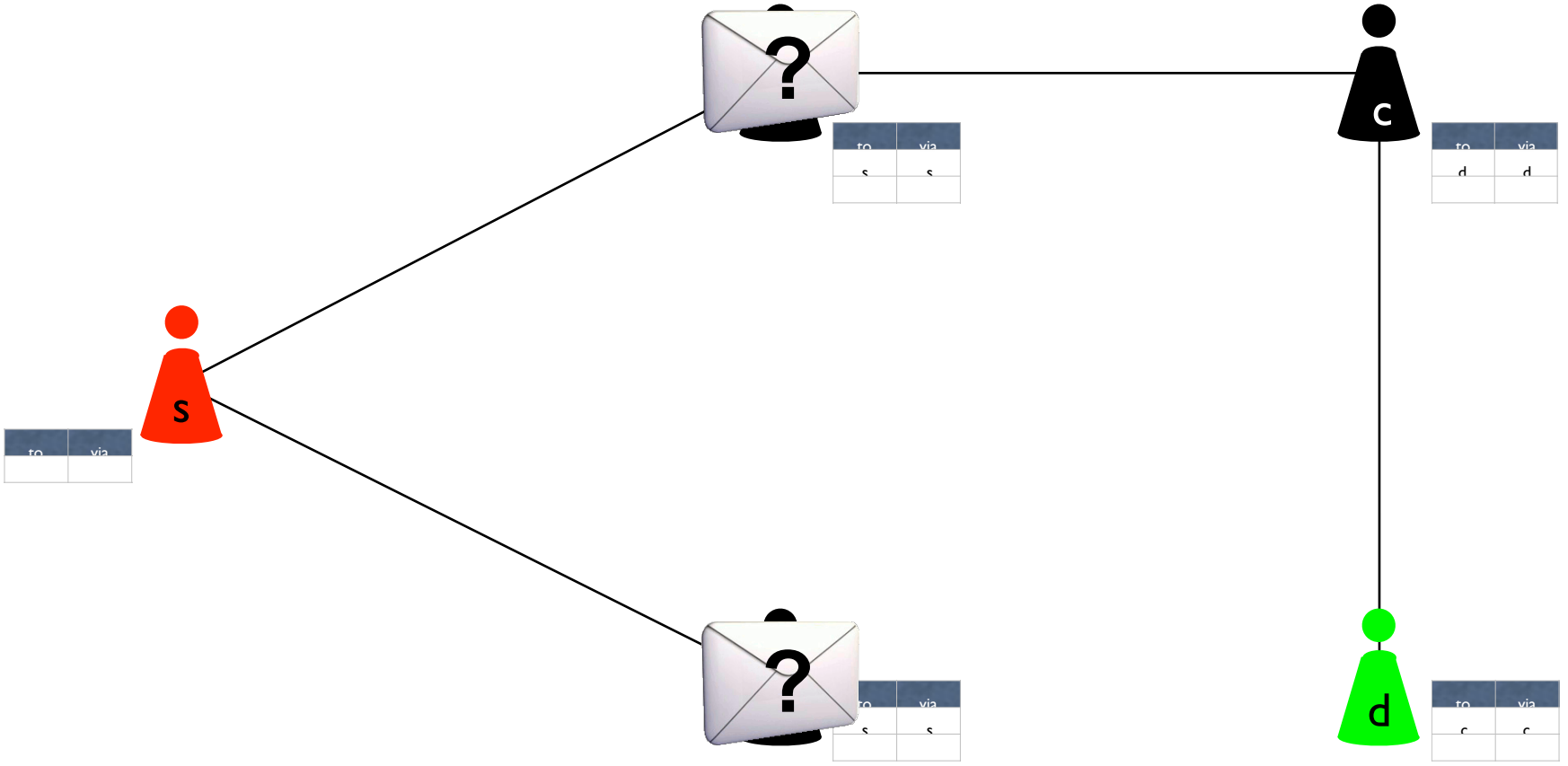
AODV - An Example



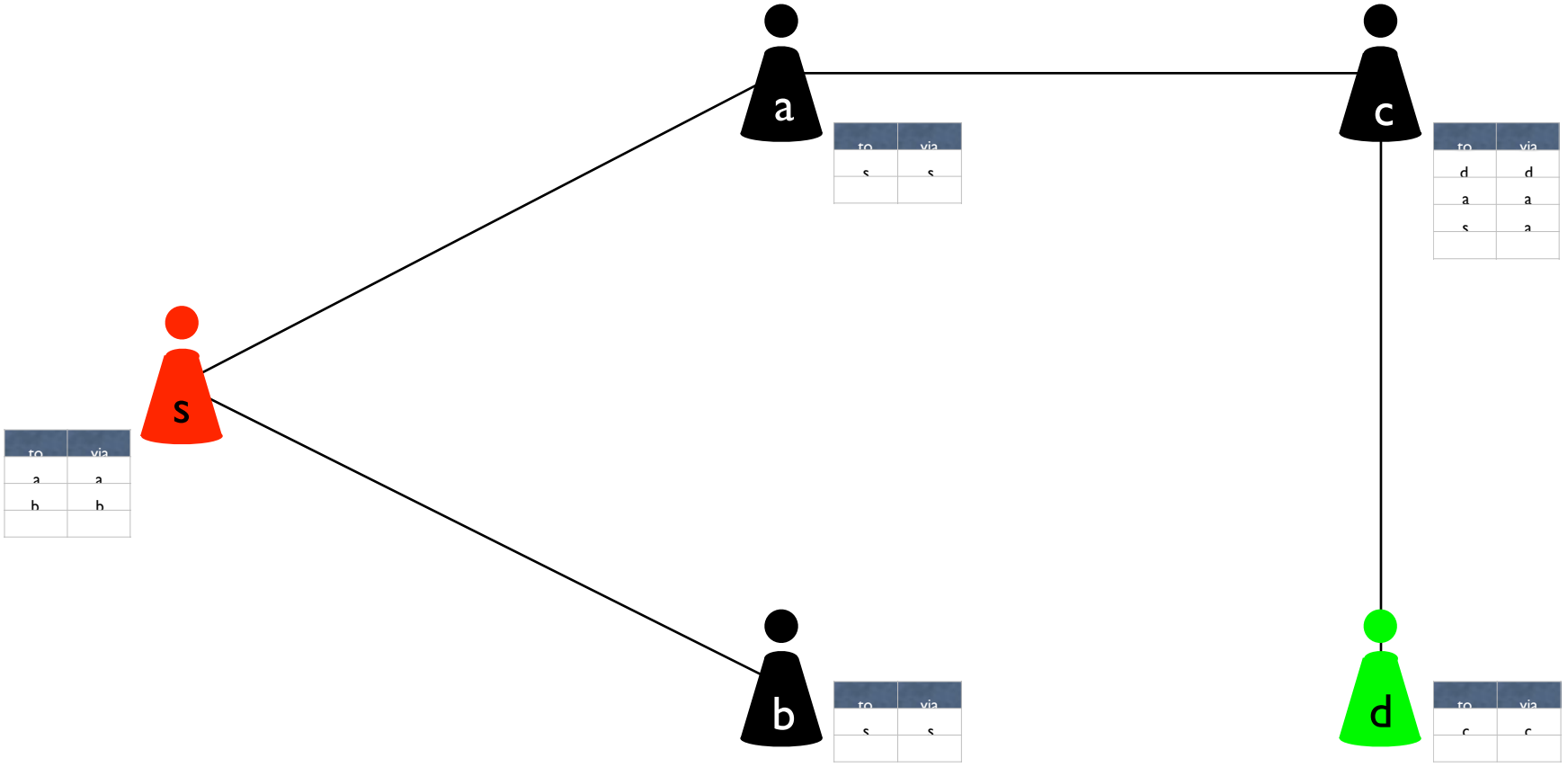
AODV - An Example



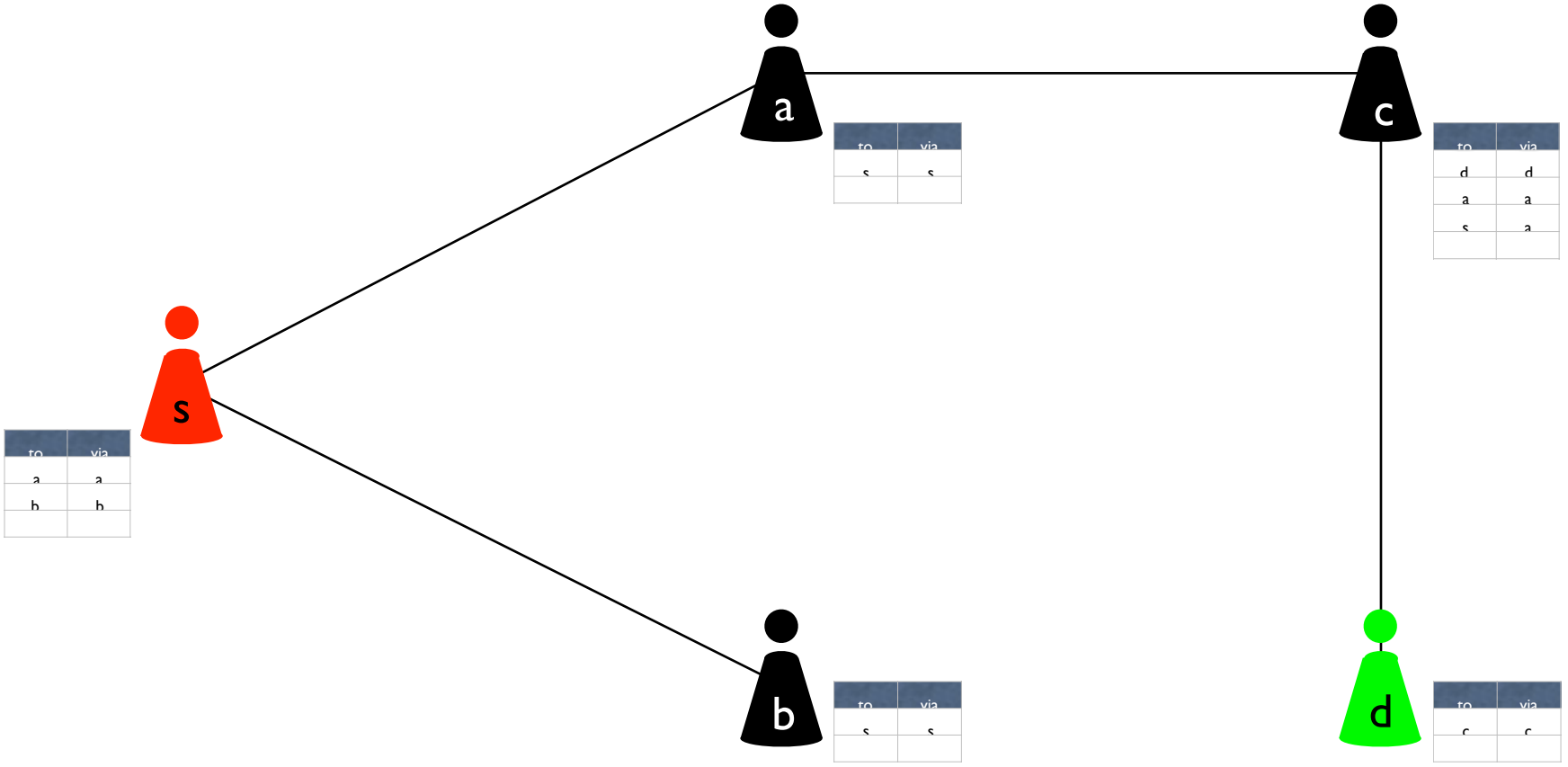
AODV - An Example



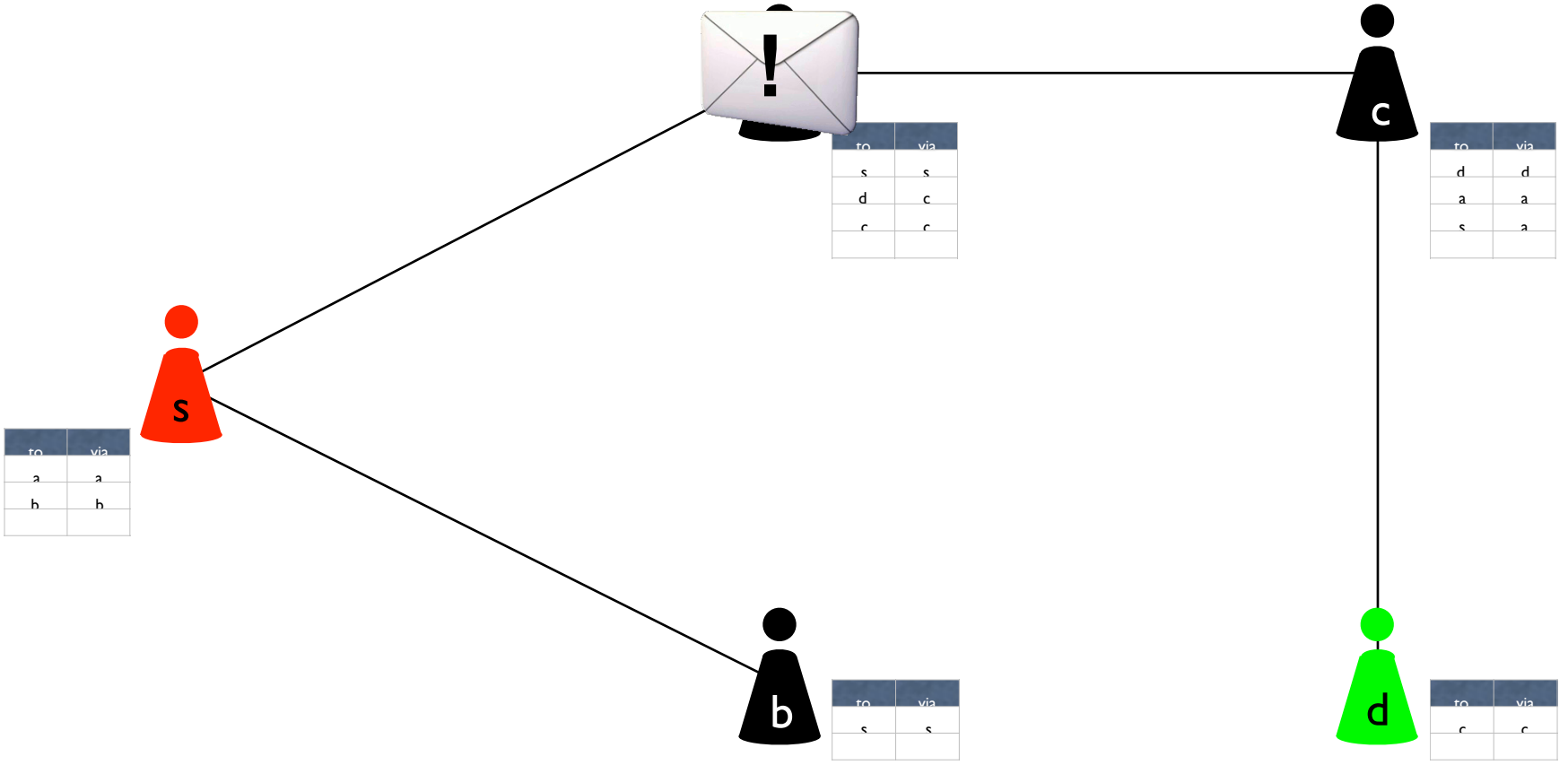
AODV - An Example



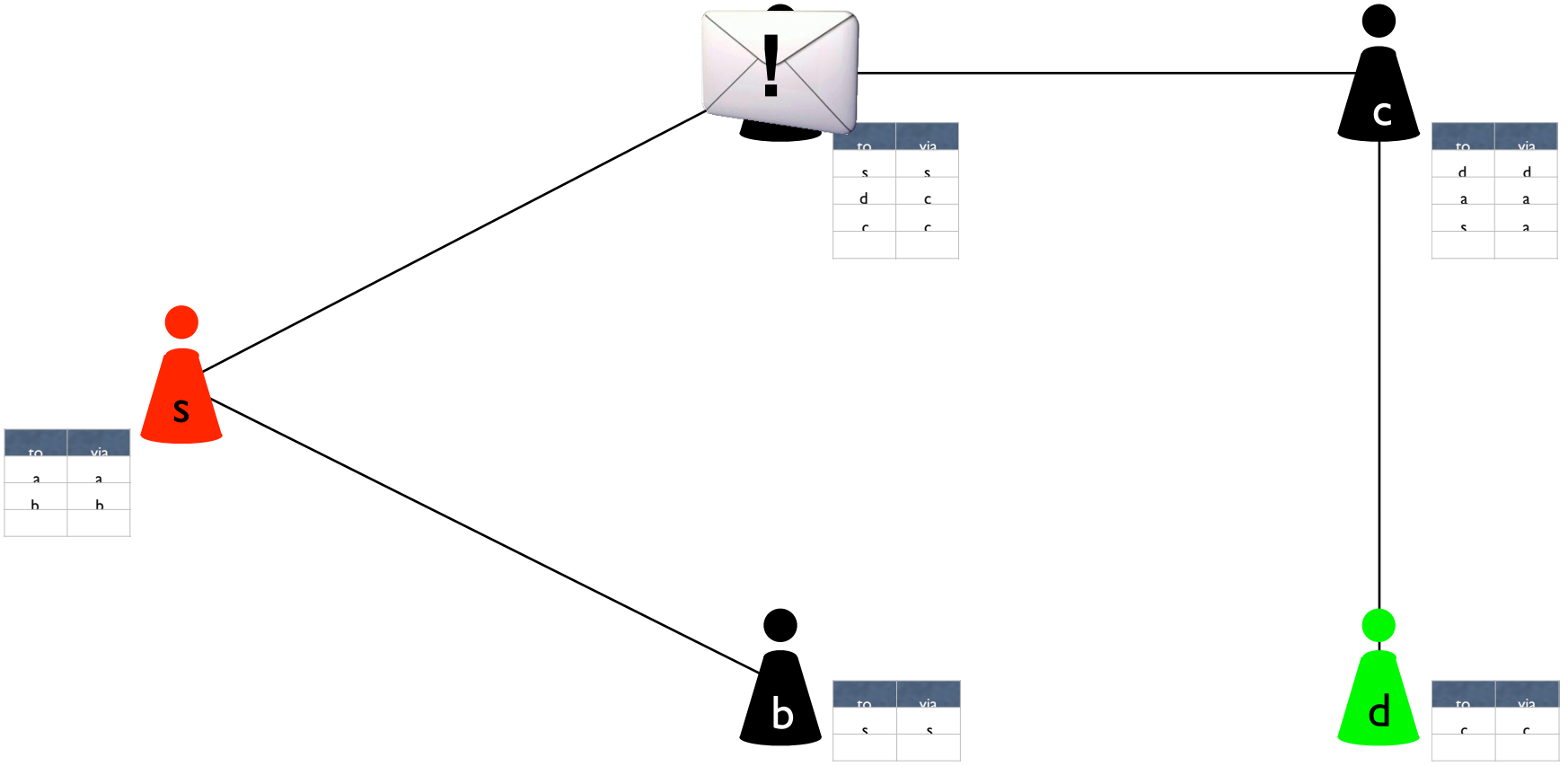
AODV - An Example



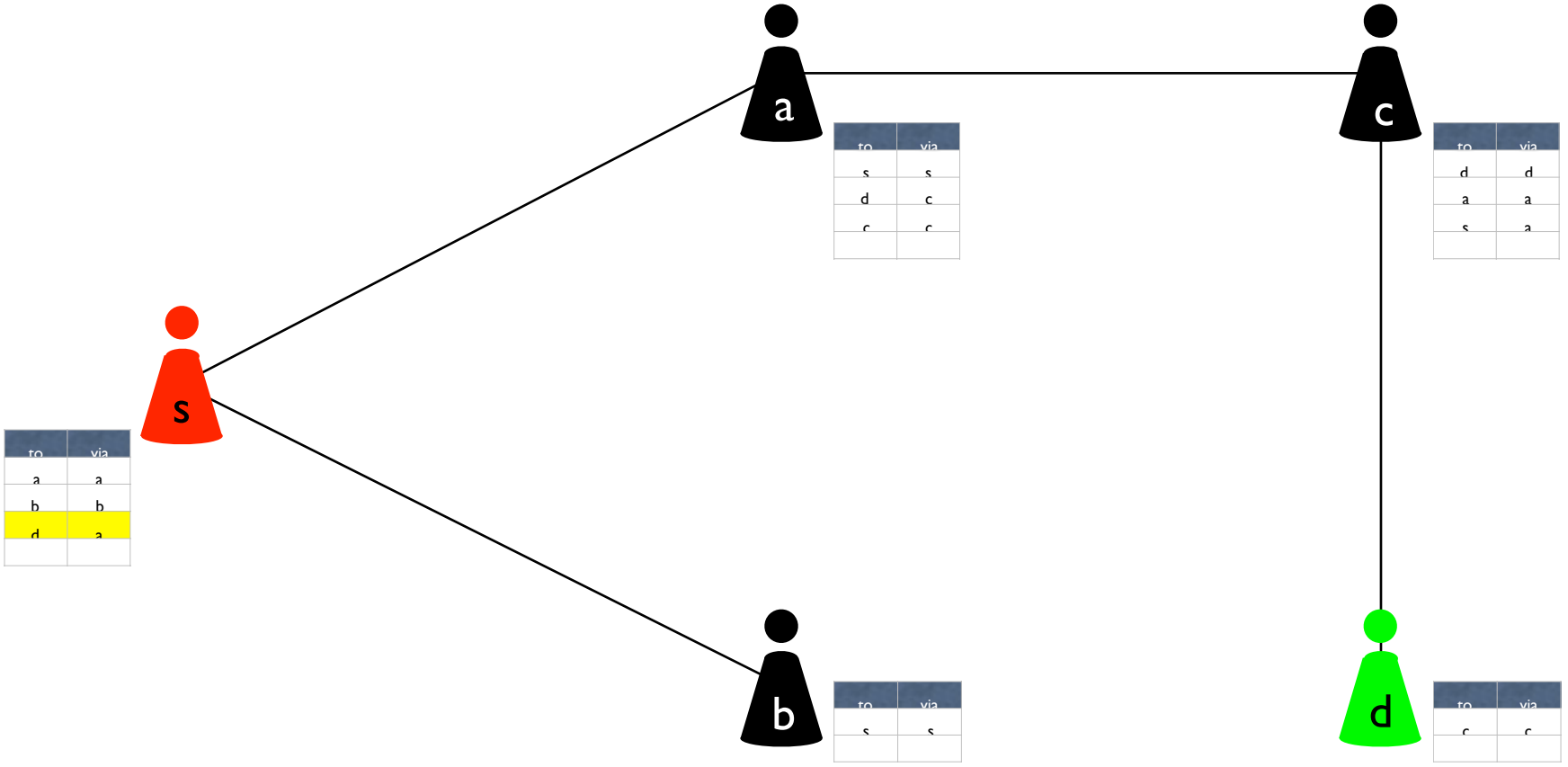
AODV - An Example



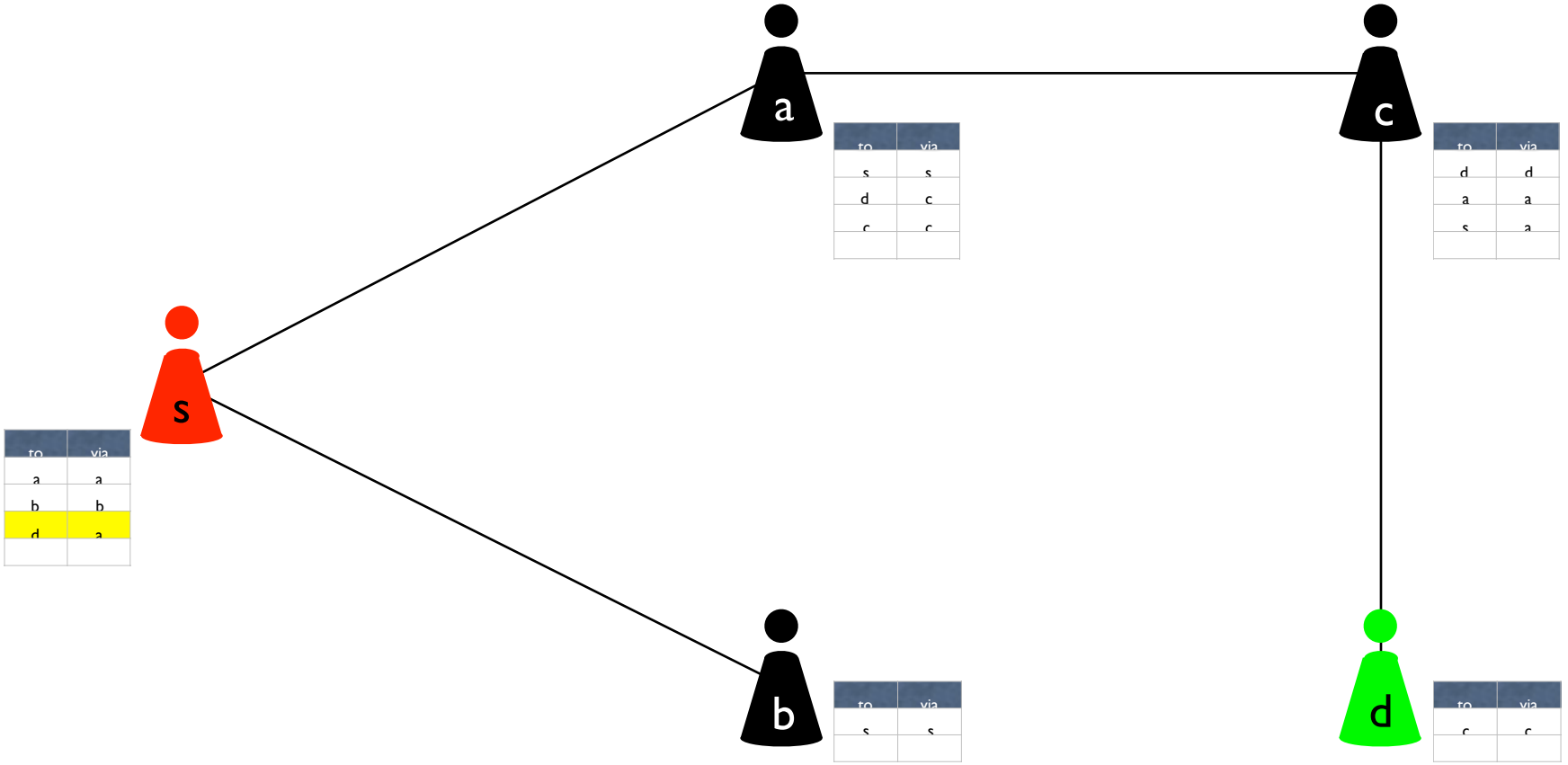
AODV - An Example



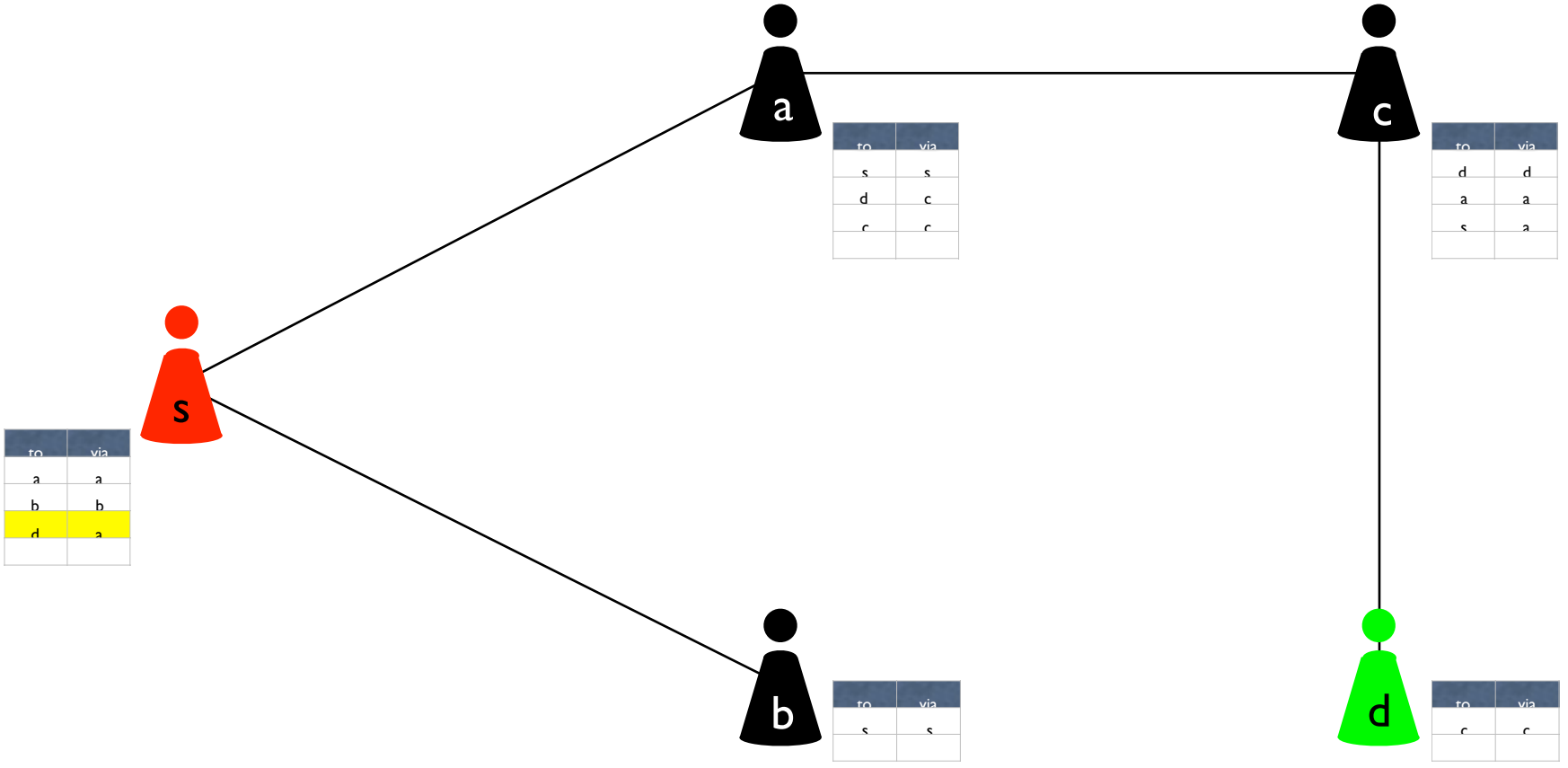
AODV - An Example



AODV - An Example



AODV - An Example



s has found a route to d

Case Study: AODV

- full specification of AODV (IETF Standard)
- specification details
  - around 5 types and 30 functions
  - around 120 lines of specification
    (in contrast to 43 pages English prose)

Main Process

Process 1 The basic routine

```
AODV(ip,sn,rt,rreqs,store) =def
 1.  receive(msg) .
 2.  /* depending on the message, the node calls different processes */
 3.  (
 4.     [ msg = newpkt(data,dip) ]        /* new DATA packet */
 5.        NEWPKT(data,dip , ip,sn,rt,rreqs,store)
 6.     + [ msg = pkt(data,dip,oip) ]     /* incoming DATA packet */
 7.        PKT(data,dip,oip , ip,sn,rt,rreqs,store)
 8.     + [ msg = rreq(hops,rreqid,dip,dsn,dsk,oip,osn,sip) ]     /* RREQ */
 9.        /* update the route to sip in rt */
10.        ⟦rt := update(rt,(sip,0,unk,val,1,sip,∅))⟧     /* 0 is used since no sequence number is known */
11.        RREQ(hops,rreqid,dip,dsn,dsk,oip,osn,sip , ip,sn,rt,rreqs,store)
12.     + [ msg = rrep(hops,dip,dsn,oip,sip) ]     /* RREP */
13.        /* update the route to sip in rt */
14.        ⟦rt := update(rt,(sip,0,unk,val,1,sip,∅))⟧
15.        RREP(hops,dip,dsn,oip,sip , ip,sn,rt,rreqs,store)
16.     + [ msg = rerr(dests,sip) ]     /* RERR */
17.        /* update the route to sip in rt */
18.        ⟦rt := update(rt,(sip,0,unk,val,1,sip,∅))⟧
19.        RERR(dests,sip , ip,sn,rt,rreqs,store)
20.  )
```

Main Process (cont'd)

21. $+$ [Let $\mathtt{dip} \in \mathtt{qD}(\mathtt{store}) \cap \mathtt{vD}(\mathtt{rt})$] /* send a queued data packet if a valid route is known */
22. $[\![\mathtt{data} := \mathtt{head}(\sigma_{queue}(\mathtt{store}, \mathtt{dip}))]\!]$
23. **unicast**$(\mathtt{nhop}(\mathtt{rt}, \mathtt{dip}), \mathtt{pkt}(\mathtt{data}, \mathtt{dip}, \mathtt{ip}))$.
24. $[\![\mathtt{store} := \mathtt{drop}(\mathtt{dip}, \mathtt{store})]\!]$ /* drop `data` from the `store` for `dip` if the transmission was successful */
25. $\mathtt{AODV}(\mathtt{ip}, \mathtt{sn}, \mathtt{rt}, \mathtt{rreqs}, \mathtt{store})$
26. ▶ /* an error is produced and the routing table is updated */
27. $[\![\mathtt{dests} := \{(\mathtt{rip}, \mathtt{inc}(\mathtt{sqn}(\mathtt{rt}, \mathtt{rip}))) \mid \mathtt{rip} \in \mathtt{vD}(\mathtt{rt}) \wedge \mathtt{nhop}(\mathtt{rt}, \mathtt{rip}) = \mathtt{nhop}(\mathtt{rt}, \mathtt{dip})\}]\!]$
28. $[\![\mathtt{rt} := \mathtt{invalidate}(\mathtt{rt}, \mathtt{dests})]\!]$
29. $[\![\mathtt{store} := \mathtt{setRRF}(\mathtt{store}, \mathtt{dests})]\!]$
30. $[\![\mathtt{pre} := \bigcup\{\mathtt{precs}(\mathtt{rt}, \mathtt{rip}) \mid (\mathtt{rip}, *) \in \mathtt{dests}\}]\!]$
31. $[\![\mathtt{dests} := \{(\mathtt{rip}, \mathtt{rsn}) \mid (\mathtt{rip}, \mathtt{rsn}) \in \mathtt{dests} \wedge \mathtt{precs}(\mathtt{rt}, \mathtt{rip}) \neq \emptyset\}]\!]$
32. **groupcast**$(\mathtt{pre}, \mathtt{rerr}(\mathtt{dests}, \mathtt{ip}))$. $\mathtt{AODV}(\mathtt{ip}, \mathtt{sn}, \mathtt{rt}, \mathtt{rreqs}, \mathtt{store})$
33. $+$ [Let $\mathtt{dip} \in \mathtt{qD}(\mathtt{store}) - \mathtt{vD}(\mathtt{rt}) \wedge \sigma_{p\text{-}flag}(\mathtt{store}, \mathtt{dip}) = \mathtt{req}$] /* a route discovery process is initiated */
34. $[\![\mathtt{store} := \mathtt{unsetRRF}(\mathtt{store}, \mathtt{dip})]\!]$ /* set request-required flag to `no-req` */
35. $[\![\mathtt{sn} := \mathtt{inc}(\mathtt{sn})]\!]$ /* increment own sequence number */
36. /* update `rreqs` by adding $(\mathtt{ip}, \mathtt{nrreqid}(\mathtt{rreqs}, \mathtt{ip}))$ */
37. $[\![\mathtt{rreqid} := \mathtt{nrreqid}(\mathtt{rreqs}, \mathtt{ip})]\!]$
38. $[\![\mathtt{rreqs} := \mathtt{rreqs} \cup \{(\mathtt{ip}, \mathtt{rreqid})\}]\!]$
39. **broadcast**$(\mathtt{rreq}(\mathbf{0}, \mathtt{rreqid}, \mathtt{dip}, \mathtt{sqn}(\mathtt{rt}, \mathtt{dip}), \mathtt{sqnf}(\mathtt{rt}, \mathtt{dip}), \mathtt{ip}, \mathtt{sn}, \mathtt{ip}))$. $\mathtt{AODV}(\mathtt{ip}, \mathtt{sn}, \mathtt{rt}, \mathtt{rreqs}, \mathtt{store})$

Route Reply

Process 5 RREP handling

```
RREP(hops,dip,dsn,oip,sip,ip,sn,rt,rreqs,store) =def
 1. [ rt ≠ update(rt,(dip,dsn,kno,val,hops+1,sip,∅)) ]      /* the routing table has to be updated */
 2.   ⟦rt := update(rt,(dip,dsn,kno,val,hops+1,sip,∅))⟧
 3.   (
 4.     [ oip = ip ]      /* this node is the originator of the corresponding RREQ */
 5.       /* a packet may now be sent; this is done in the process AODV */
 6.       AODV(ip,sn,rt,rreqs,store)
 7.     + [ oip ≠ ip ]      /* this node is not the originator; forward RREP */
 8.       (
 9.         [ oip ∈ vD(rt) ]      /* valid route to oip */
10.           /* add next hop towards oip as precursor and forward the route reply */
11.           ⟦rt := addpreRT(rt,dip,{nhop(rt,oip)})⟧
12.           ⟦rt := addpreRT(rt,nhop(rt,dip),{nhop(rt,oip)})⟧
13.           unicast(nhop(rt,oip),rrep(hops+1,dip,dsn,oip,ip)) .
14.             AODV(ip,sn,rt,rreqs,store)
15.           ▶ /* If the transmission is unsuccessful, a RERR message is generated */
16.             ⟦dests := {(rip,inc(sqn(rt,rip)))|rip ∈ vD(rt) ∧ nhop(rt,rip) = nhop(rt,oip)}⟧
17.             ⟦rt := invalidate(rt,dests)⟧
18.             ⟦store := setRRF(store,dests)⟧
19.             ⟦pre := ⋃{precs(rt,rip)|(rip,∗) ∈ dests}⟧
20.             ⟦dests := {(rip,rsn)|(rip,rsn) ∈ dests ∧ precs(rt,rip) ≠ ∅}⟧
21.             groupcast(pre,rerr(dests,ip)) . AODV(ip,sn,rt,rreqs,store)
22.         + [ oip ∉ vD(rt) ]      /* no valid route to oip */
23.           AODV(ip,sn,rt,rreqs,store)
24.       )
25.   )
26. + [ rt = update(rt,(dip,dsn,kno,val,hops+1,sip,∅)) ]      /* the routing table is not updated */
27.   AODV(ip,sn,rt,rreqs,store)
```

Case Study: AODV - Analysis

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

Case Study: AODV - Analysis

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

Case Study: Analysis

- Loop Freedom
  - invariant proof
    based on about 35 invariants, e.g.

> If a route reply is sent by a node ip_c, different from the destination of the route, then the content of ip_c's routing table must be consistent with the information inside the message.

$$N \xrightarrow{R:*\mathbf{cast}(\mathtt{rrep}(hops_c,dip_c,dsn_c,*,ip_c))}_{ip} N' \ \wedge \ ip_c \neq dip_c$$
$$\Rightarrow \quad dip_c \in \mathtt{kD}_N^{ip_c} \ \wedge \ \mathtt{sqn}_N^{ip_c}(dip_c) = dsn_c \ \wedge \ \mathtt{dhops}_N^{ip_c}(dip_c) = hops_c \ \wedge \ \mathtt{flag}_N^{ip_c}(dip_c) = \mathtt{val}$$

  - ultimately we defined quality on routes
    the quality strictly increases

$$dip \in \mathtt{vD}_N^{ip} \cap \mathtt{vD}_N^{nhip} \ \wedge \ nhip \neq dip \ \Rightarrow \ \xi_N^{ip}(\mathtt{rt}) \sqsubset_{dip} \xi_N^{nhip}(\mathtt{rt})$$

  - first rigorous and complete proof of loop freedom of AODV
    (for some interpretations)

Case Study: Analysis

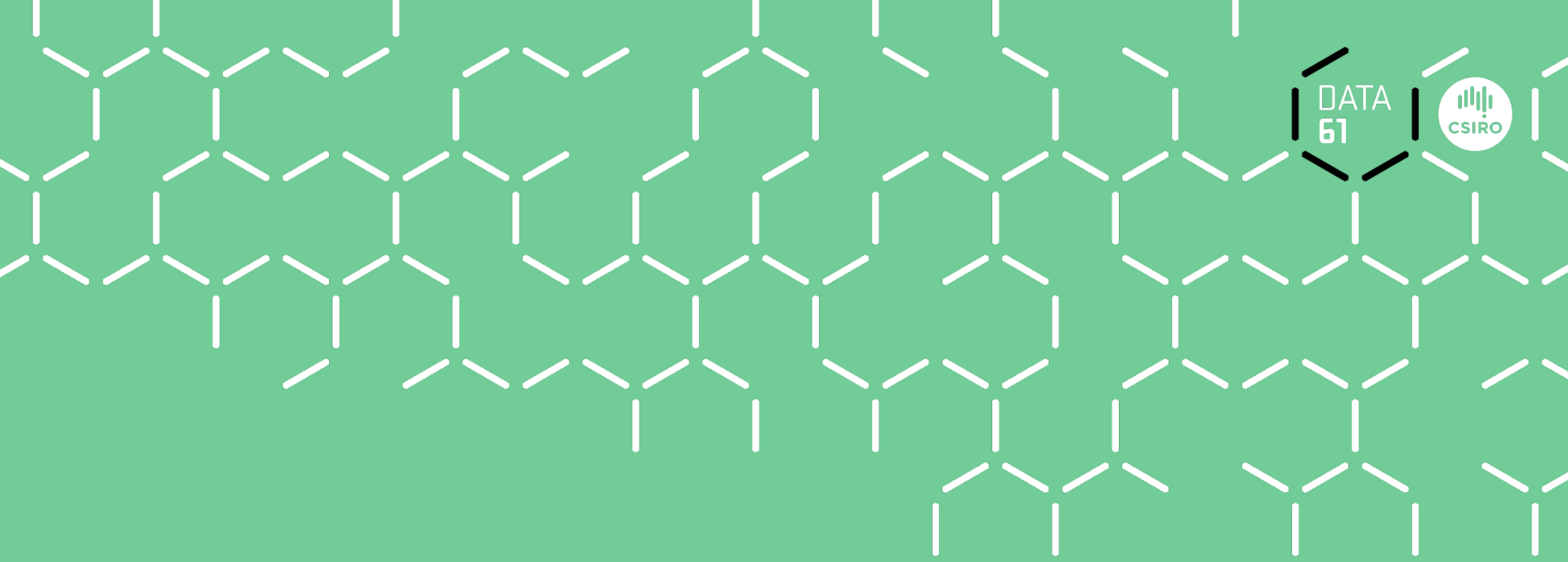
- Loop Freedom
  - 5184 possible interpretations due to ambiguities
  - 5006 of these readings of the standard contain loops
  - 3 out of 5 open-source implementations contain loops

- Found other shortcomings
  - e.g. non-optimal routing information
  - we proposed solutions and proved them correct

Computer-Aided Verification

- Model Checking
  - quick feedback for development
  - cannot be used for full verification (yet)

- (Interactive) Theorem Proving
  - Isabelle/HOL
  - replay proofs
    - proof verification
    - robust against small changes in specification



Properties of Routing Protocols

Safety & Liveness

- Safety

  something bad will never happen [Lamport77]

- Liveness

  something good will eventually happen [Lamport77]

Examples for Routing Protocols

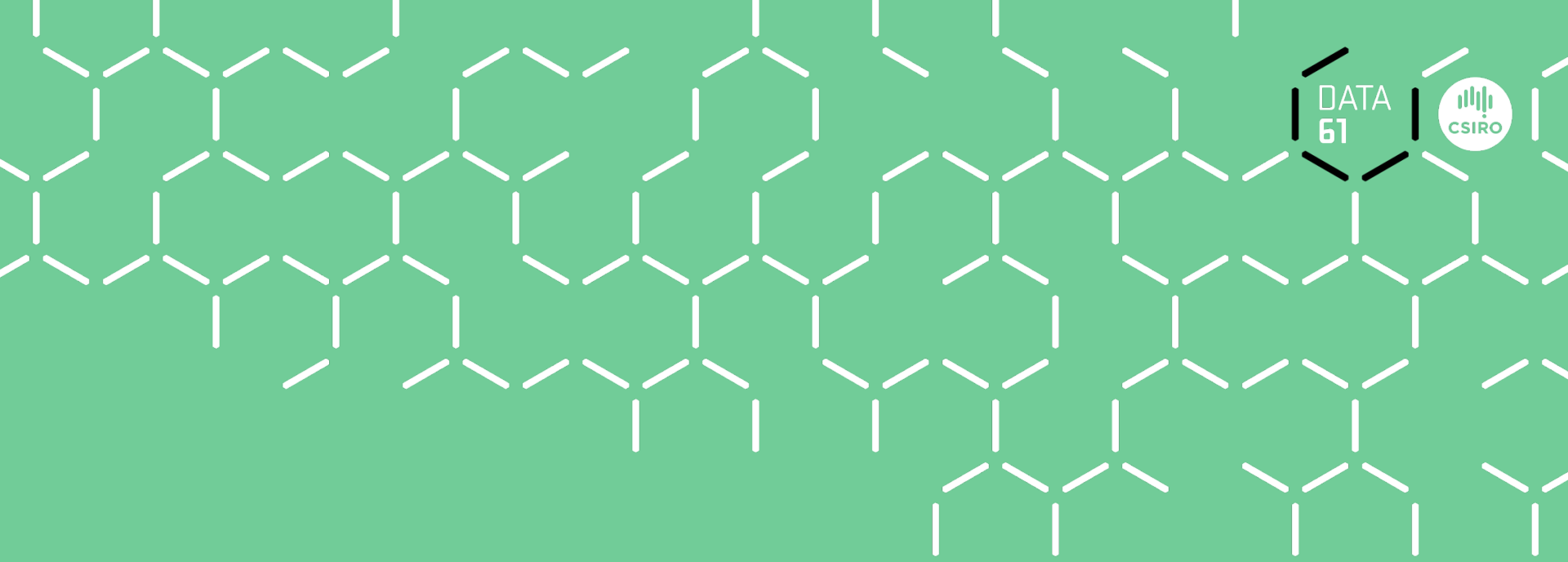
- route correctness
- loop freedom
- route discovery
- packet delivery

Proof principles

- Invariants
  - a condition that is true during the execution of a program, or during some portion of it
  - in Computation Tree Logic (CTL): $A\,G\;\varphi$ $(\text{same as } A\,\Box\;\varphi)$

- Reachability
  - a given program state can be reached
  - in CTL: $A\,F\;\varphi$ $(\text{same as } A\,\Diamond\;\varphi)$

Formalising Examples

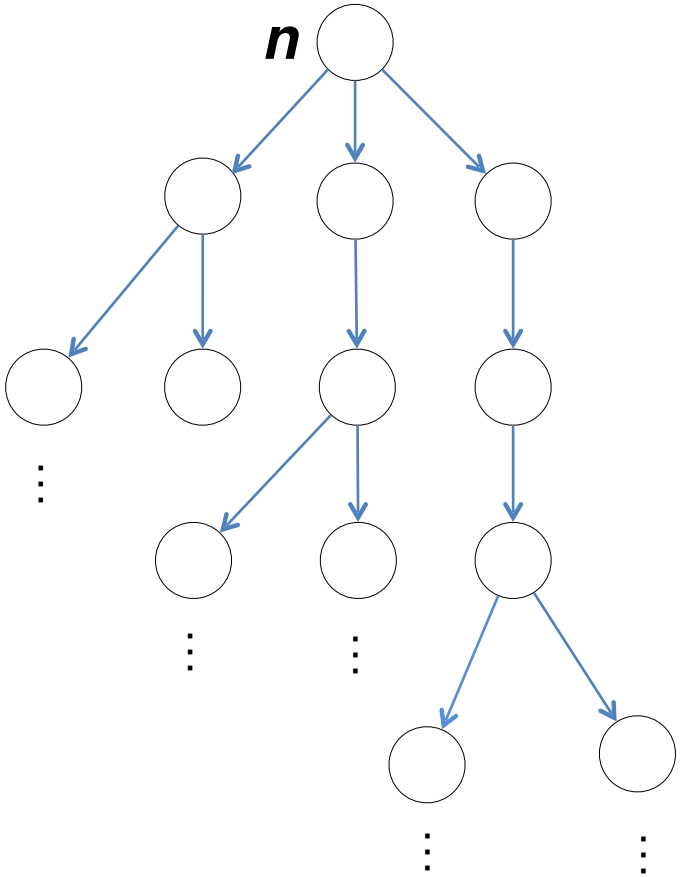


CTL (Computation Tree Logic)

Objective

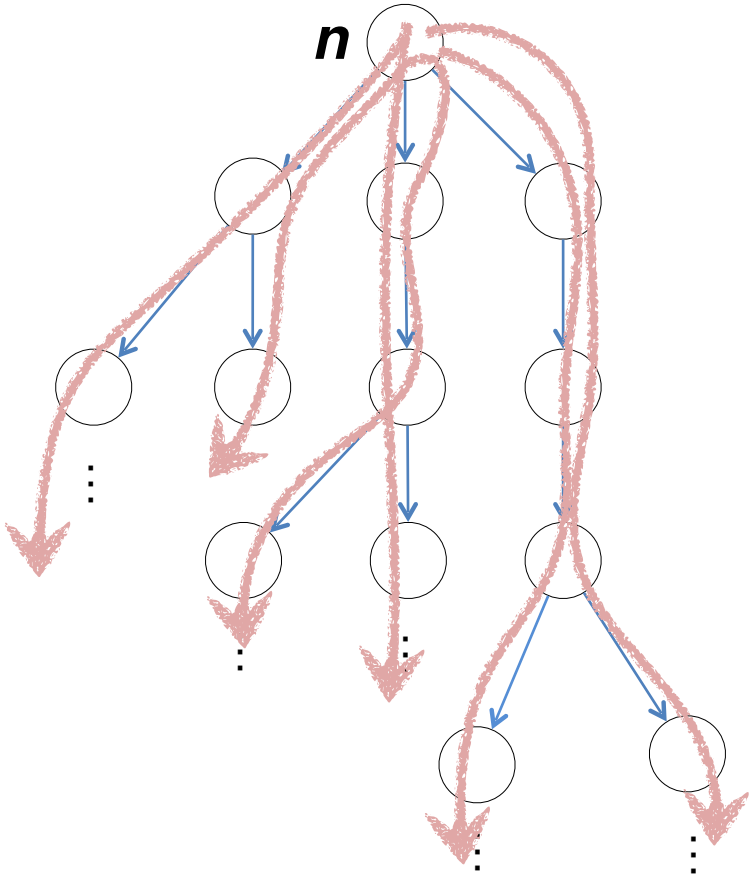
- Specify *branching time* properties: given a *computation tree*
- Define sets of trees (without enumerating …)
- All trees are infinite
- Setting: *branching time semantics* and tree

Properties of Trees



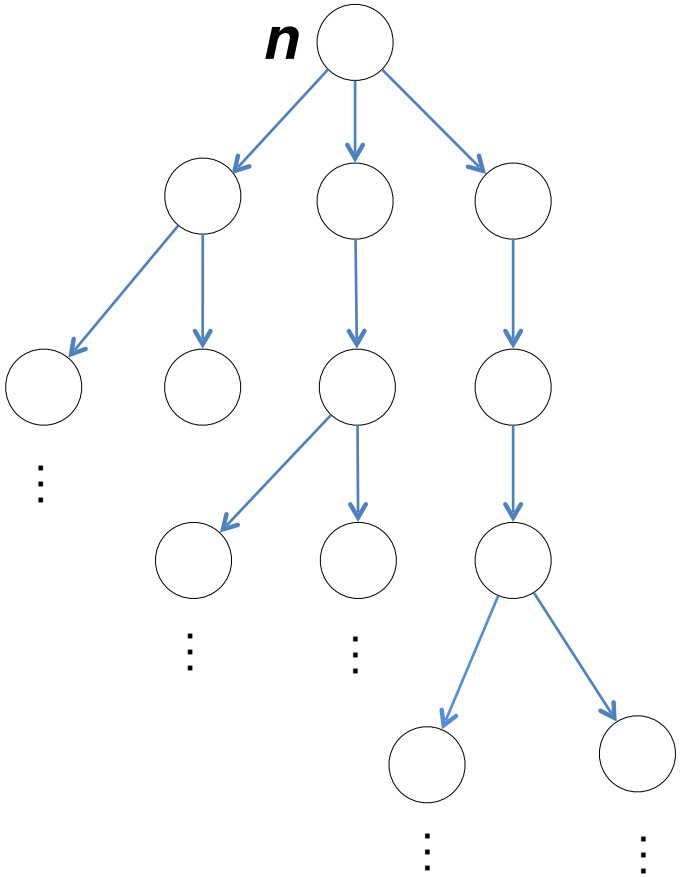
∀***n***-rooted path in the tree: operator A

Properties of Trees



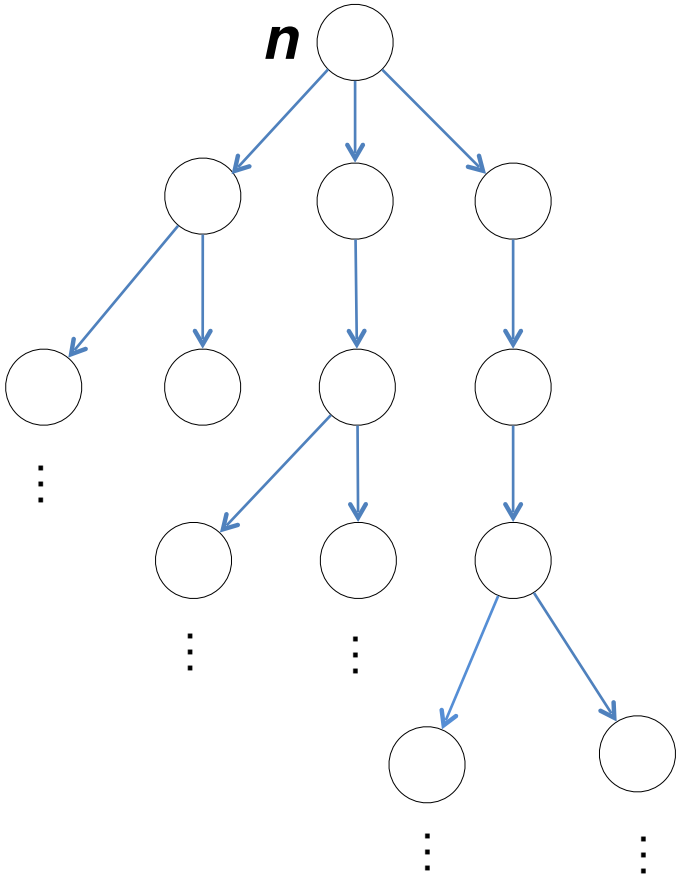
∀***n***-rooted path in the tree: operator A

Properties of Trees



∀***n***-rooted path in the tree: operator A

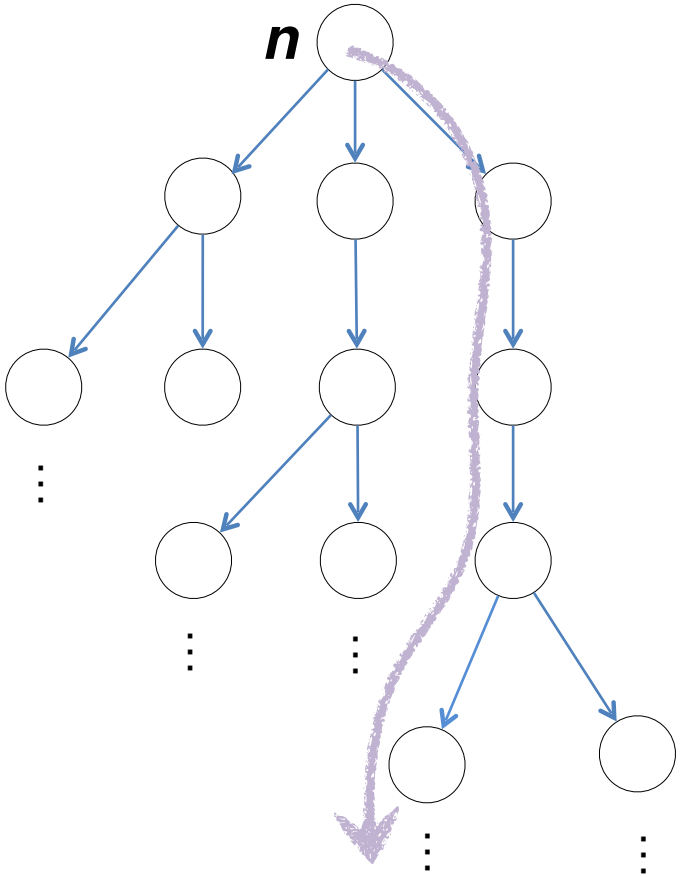
Properties of Trees



∀***n***-rooted path in the tree: operator A

∃***n***-rooted path in the tree: operator E

Properties of Trees



∀***n***-rooted path in the tree: operator A

∃***n***-rooted path in the tree: operator E

CTL – Syntax

- True is in CTL
- $\boldsymbol{x} \in \mathcal{P}$ is in CTL
- $\boldsymbol{P}, \boldsymbol{Q} \in$ CTL, $\neg \boldsymbol{P} \in$ CTL and $\boldsymbol{P} \wedge \boldsymbol{Q} \in$ CTL
- E X φ is in CTL if φ is in CTL
- A X φ is in CTL if φ is in CTL
- A $(\varphi_1$ UNTIL $\varphi_2)$ is in CTL if $\varphi_1, \varphi_2 \in$ CTL,
- E $(\varphi_1$ UNTIL $\varphi_2)$ is in CTL if $\varphi_1, \varphi_2 \in$ CTL,

CTL – Simple Semantics

State formulas only

- $s \models \mathrm{EX}\varphi$, iff $\exists s', s \longrightarrow s'$ and $s' \models \varphi$
- $s \models \mathrm{AX}\varphi$, iff $\forall s', s \longrightarrow s'$ and $s' \models \varphi$
- $s \models \mathrm{E}(\varphi_1 \text{ UNTIL } \varphi_2)$ iff

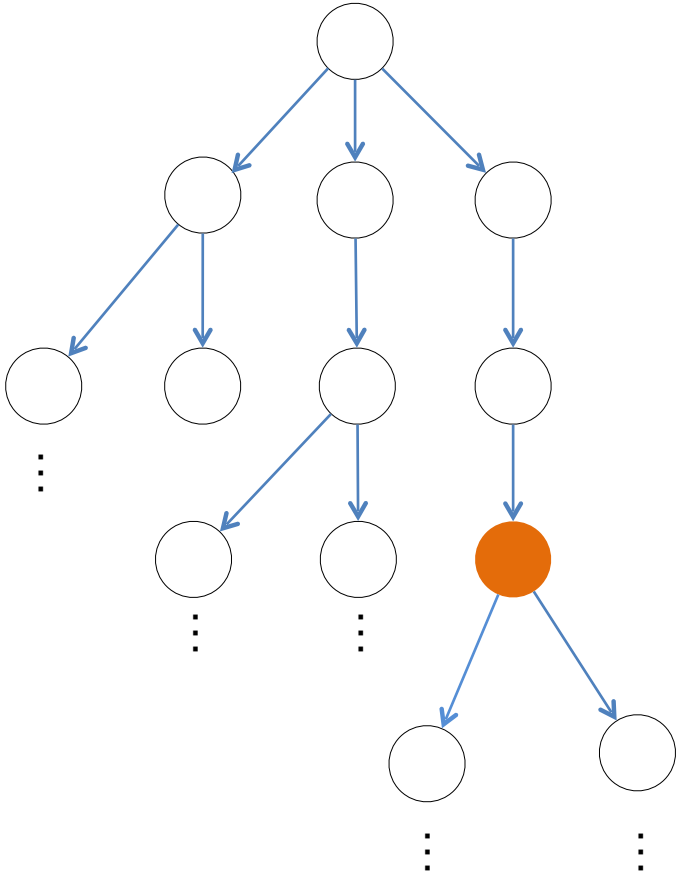
  $$\exists \rho = s_0 \cdots s_k \cdots \in \mathit{Runs}(s), \begin{cases} \exists k \geq 0, s_k \models \varphi_2 \\ \forall 0 \leq j < k, s_j \models \varphi_1 \end{cases}$$

- $s \models \mathrm{A}(\varphi_1 \text{ UNTIL } \varphi_2)$ iff

  $$\forall \rho = s_0 \cdots s_k \cdots \in \mathit{Runs}(s), \begin{cases} \exists k \geq 0, s_k \models \varphi_2 \\ \forall 0 \leq j < k, s_j \models \varphi_1 \end{cases}$$

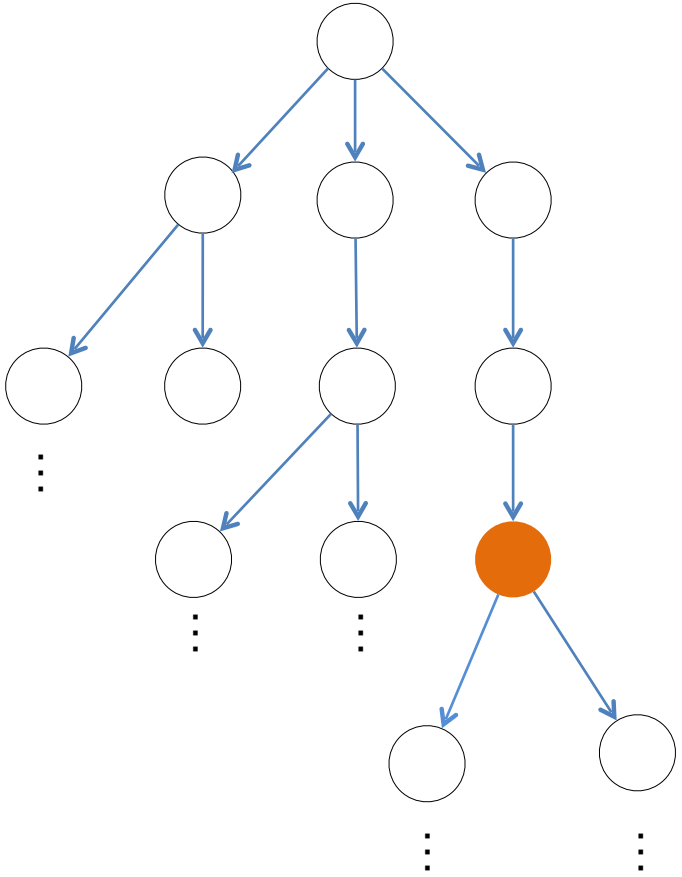
Useful Derived Operators (1)

E F ● ≡ E (***True*** UNTIL ●)

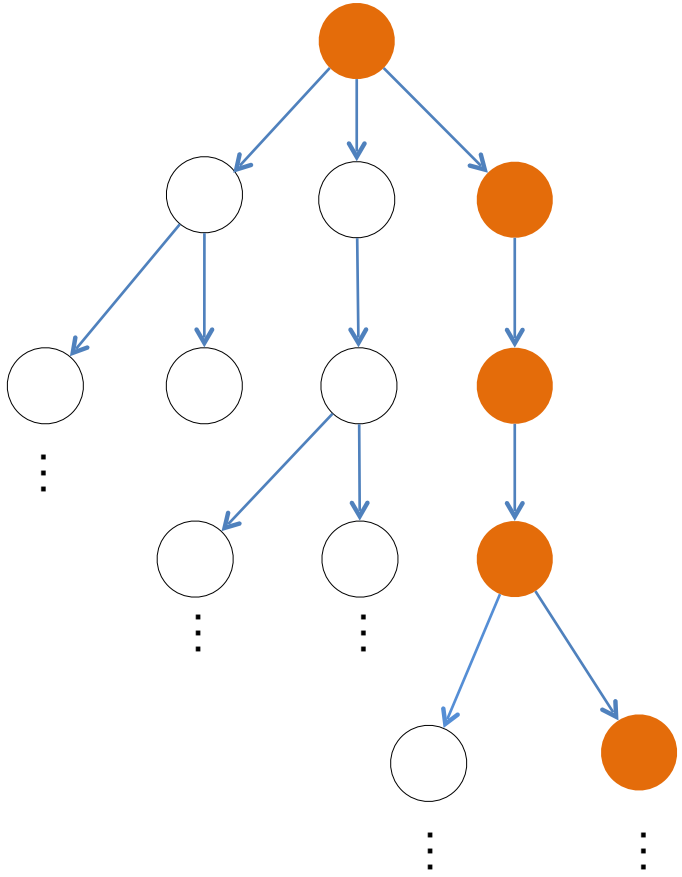


Useful Derived Operators (1)

E F ● ≡ E (***True*** UNTIL ●)

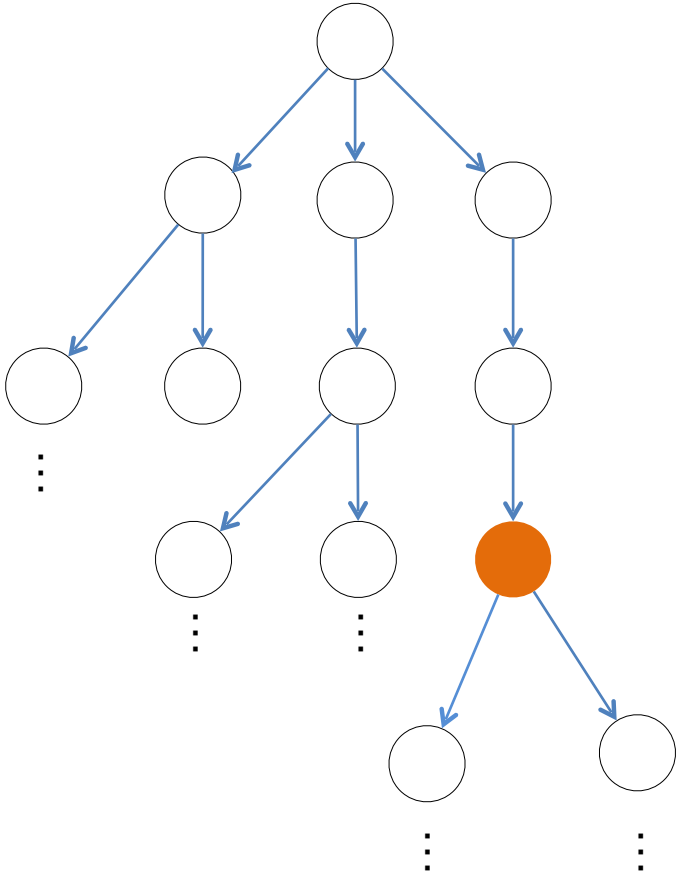


E G ● ≡ ¬A (***True*** UNTIL¬ ●)



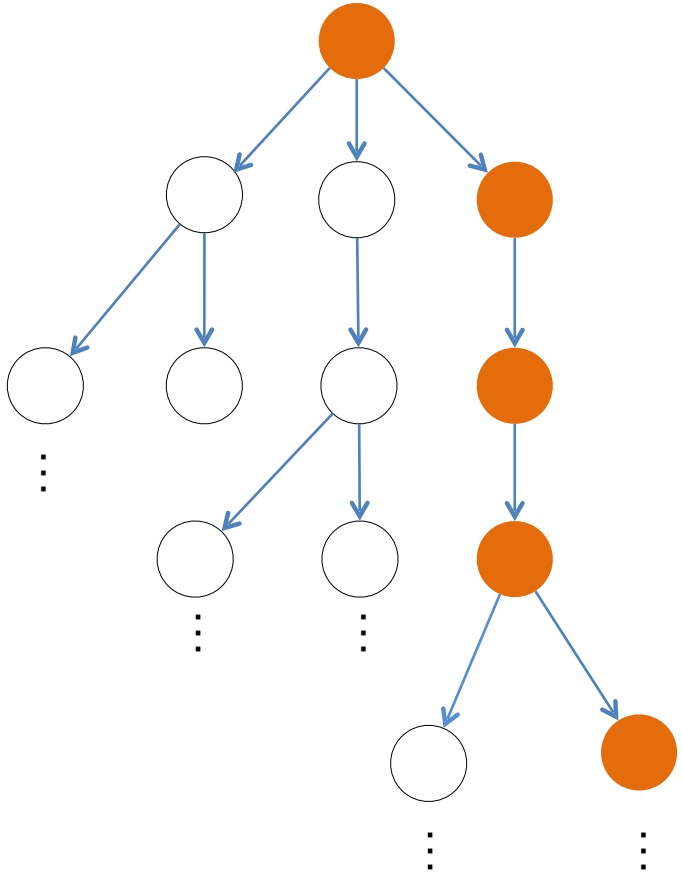
Useful Derived Operators (1)

E F ● ≡ E (***True*** UNTIL ●)



E G ● ≡ ¬A (***True*** UNTIL¬ ●)

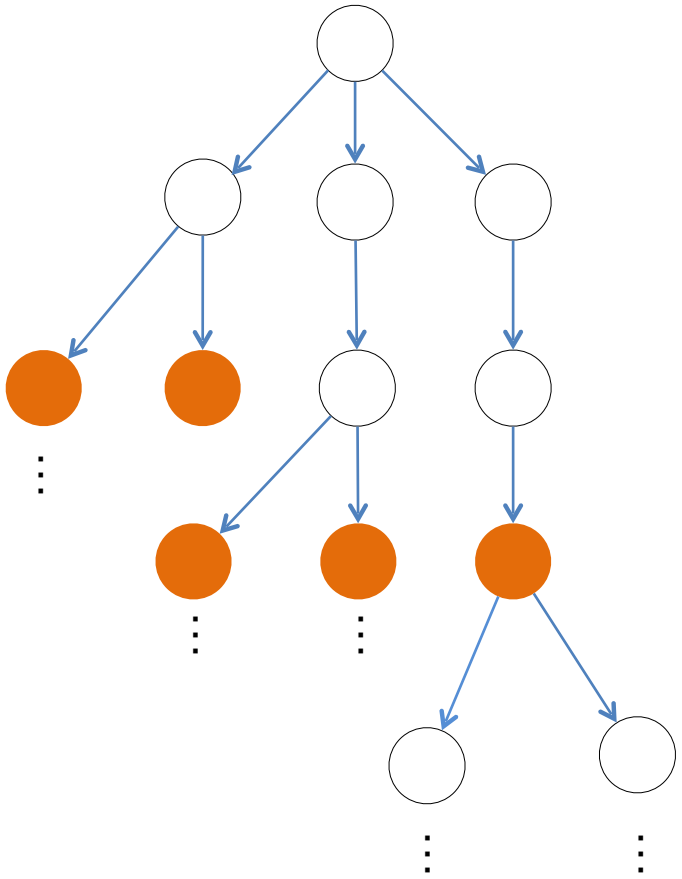
E G ● ≡ E ¬F¬ ●



Useful Derived Operators (2)

Useful Derived Operators (2)

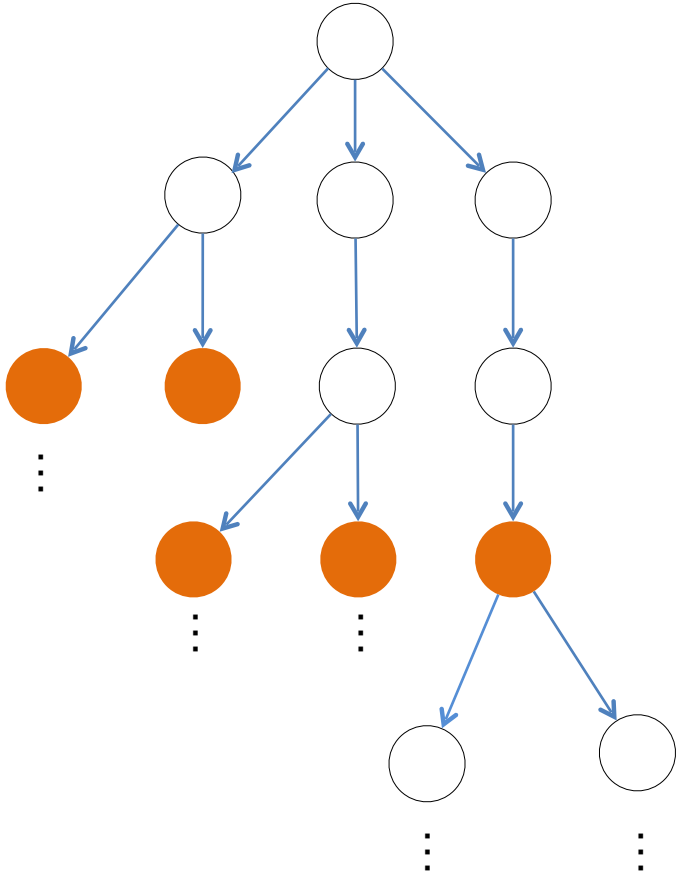
A F ● ≡ A (*True* UNTIL ●)



Useful Derived Operators (2)

A F ● ≡ A (***True*** UNTIL ●)

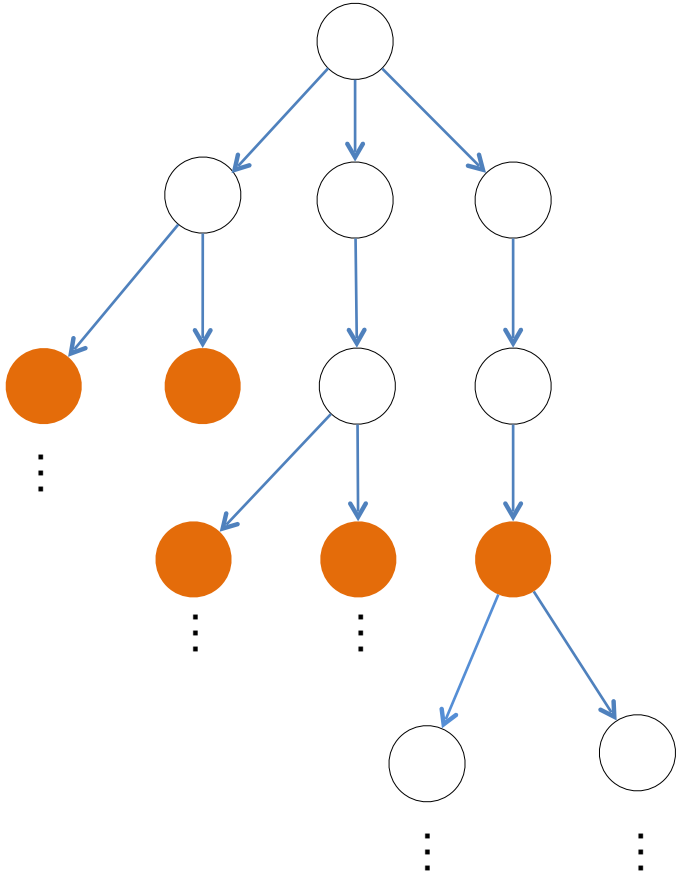
A F ● ≡ ¬E ¬F ●



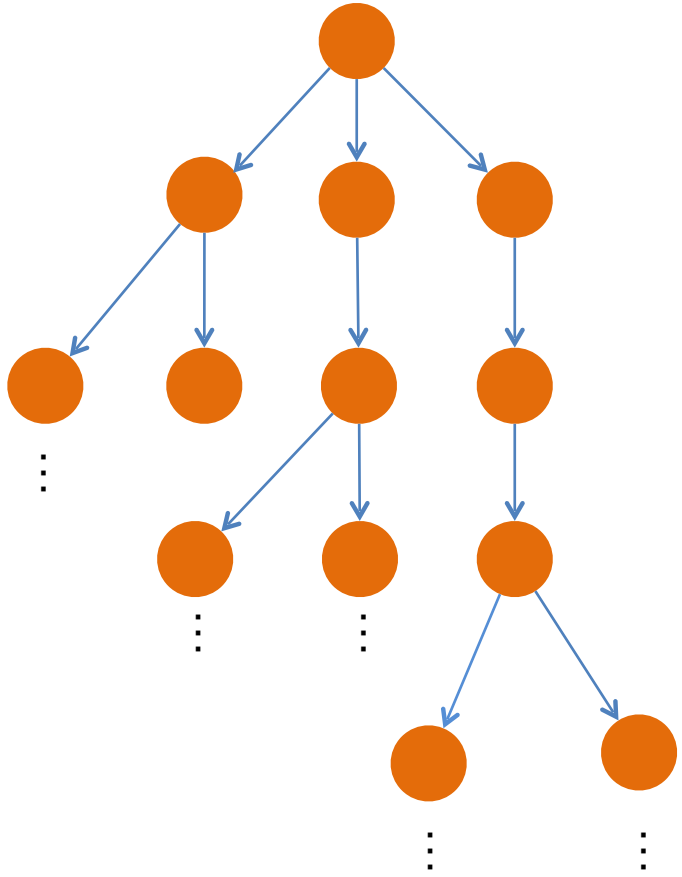
Useful Derived Operators (2)

A F ● ≡ A (***True*** UNTIL ●)

A F ● ≡ ¬E ¬F ●



A G ● ≡ ¬E F¬ ●



References

- 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
- R.J. van Glabbeek, P. Höfner, M. Portmann, W.L. Tan:
  Modelling and Verifying the AODV Routing Protocol. In Distributed Computing 29(4):279-315, Springer, 2016.
  arXiv: CoRR abs/1512.08867, doi: 10.1007/s00446-015-0262-7
- M. Huth and M. Ryan. *Logic in Computer Science* (2nd edition). Cambridge University Press, 2004.