Modeling AODV

(Ad hoc On-Demand Distance Vector Routing Protocol)
Why is a rigorous AODV model needed?

- C.E.Perkins/E.M.Belding-Royer/S.R.Das: *AODV* RFC 3561 de facto standard docu of 2003 is partly ambiguous, incomplete, contradictory – various implementations show different behavior on protocol relevant features (e.g. loops, route discovery, packet delivery, optimal routes)
  - a huge number of different interpretations of RFC 3561 are possible
- widely believed, performance relevant loop freedom claim (by Perkins and Royer in 1999) established only in 2013 for a ‘correct’ interpretation of AODV
  - after some wrong/partial proofs in the literature
- TR 5513 identifies also some performance relevant shortcomings of AODV and five key implementations
Goal of this lecture

- explain the functional core behavior of AODV for users and programmers, ‘from scratch’ and reliably, by stepwise developing an Abstract State Machine (ASM) model
  - reflecting a correct and complete understanding of the (core) requirements in the de facto standard document RFC 3561
  - but informed by the professional analysis in the NICTA TR 5513 as a prerequisite for a rigorous high-level analysis, long before coding
- performed in the NICTA TR 5513, in process algebraic terms, for different interpretations and implementations of the RFC 3561 wrt
  - loop freedom
  - route discovery
  - packet delivery
  and related correctness and performance relevant issues
Mobile Ad hoc Network (MANET) routing protocols

In MANETs every network agent

- can move independently to change its position
- can (try to) send messages to every ‘directly connected’ network node it knows (‘neighbor’)
- can (try to) broadcast messages to all its ‘neighbors’
- for (wireless) communication with any other network agent must ask a routing protocol to indicate a communication path to that destination

The routing protocol

- receives and elaborates route request messages, by forwarding them and generating reply messages once a route has been found
- receives and elaborates route reply messages by forwarding them back to the original requestor
- creates and propagates route error messages if some broken direct link is detected
Background and agent/protocol interaction structure

Background structure:
- network: graph \((\text{Agent}, \text{Link})\) with dynamic sets of nodes and edges
- determines for each \(a \in \text{Agent}\) a dynamic set \(\text{neighb}(a)\)

Agent/protocol interaction: when \(a\) \textit{WantsToCommunicateWith} \(d\)
- in case \(a\) \textit{KnowsActiveRouteTo}(\(d\)), it can right away \textit{StartCommunicationWith}(\(d\)), without entering \textit{WaitingForRouteTo}(\(d\)) (which is initialized by \textit{false})
- otherwise it must \textit{GenerateRouteReq}(\(d\)) and becomes \textit{WaitingForRouteTo}(\(d\)) until via the protocol run it eventually \textit{KnowsActiveRouteTo}(\(d\))
- to \textit{GenerateRouteReq}(\(d\)) only once per required communication, it is called only when \(a\) is not already \textit{WaitingForRouteTo}(\(d\)) — easily refinable to permit repeated route requests to a same \(d\)

Copyright CC BY–NC-SA 4.0
\textbf{PrepareComm} =

\begin{verbatim}
if WantsToCommunicateWith(destination) then
    if KnowsActiveRouteTo(destination) then
        STARTCOMMUNICATIONWITH(destination)
        WantsToCommunicateWith(destination) := false
        WaitingForRouteTo(destination) := false
    else
        if not WaitingForRouteTo(destination) then
            GENERATEROUTEREQ(destination)
            WaitingForRouteTo(destination) := true
\end{verbatim}

NB. \textit{WantsToCommunicateWith} is assumed to be set to \textit{true} only by the application program and to \textit{false} only by \textbf{PrepareComm}. 
Router components and \texttt{Aodv} program structure

The main \texttt{Aodv} program each node is equipped with:

\begin{align*}
\texttt{AodvSpec} &= \texttt{one of} \hspace{1cm} \text{-- main program} \\
\texttt{PrepareComm} \\
\texttt{Router} \\
\texttt{ProcessRouteReq} \\
\texttt{ProcessRouteRep} \\
\texttt{ProcessRouteErr} \\
\texttt{GenerateRouteErr} \\
\texttt{PropagateRouteErr}
\end{align*}

The \texttt{Router} consists of components to process request, reply and error msgs:

\begin{align*}
\texttt{Router} &= \texttt{one of} \\
\texttt{ProcessRouteReq} \\
\texttt{ProcessRouteRep} \\
\texttt{ProcessRouteErr} \\
\texttt{GenerateRouteErr} \\
\texttt{PropagateRouteErr}
\end{align*}
Each route table entry keeps agent $a$’s knowledge that:

- the destination $dest(entry)$ of the entry may be reachable
- in the direction as indicated by a neighbor node $nextHop(entry)$
- on a path of length $hopCount(entry)$ (distance to the destination)
- which can be considered as $Active$ (without $LinkBreak$)

To avoid loops in communication paths, each agent keeps a local request/-level counter:

- $localReqCount(a)$ and $curSeqNum(a)$, both initialized by 0
- $curSeqNum(a)$ possibly incremented when $a$ receives a new request to reach $a$, after a $LinkBreak$ which made an $Active$ path to an $InActive$

Each route table entry for $d = dest(entry)$ also records:

- as $destSeqNum(entry)$ the last known value of $curSeqNum(d)$
- by $known(entry) \in \{true, false\}$ whether this number is valid

NB. For $precursor(entry)$ info for route error handling see below
AODV data structure: route table entry attributes

\[ entryFor(d, RT) = \begin{cases} 
\text{entry} & \text{if forsome } \text{entry} \in RT \text{ dest(}\text{entry}) = d \\
\text{undef} & \text{else}
\end{cases} \]

KnowsActiveRouteTo(destination) \iff Active(entryFor(destination, RT))

\[ lastKnownDestSeqNum(d, RT) = \begin{cases} 
\text{destSeqNum(}\text{entry}) & \text{if forsome } \text{entry} \in RT \text{ dest(}\text{entry}) = d \\
\text{unknown} & \text{else}
\end{cases} \]

ValidDestSeqNum(entry) \iff known(entry) = \text{true}
**AODV data structure: RouteRequest**

Each route request msg $rreq \in RouteRequest$, when generated, records information about:

- the request destination $\text{dest}(rreq) \in Agent$
- the last known value $\text{destSeqNum}(rreq) \in \text{NAT} \cup \{\text{unknown}\}$ the request originator knows about the $\text{curSeqNum}(\text{dest}(rreq))$
- $\text{known}(rreq)$ indicating whether $\text{destSeqNum}(rreq)$ is reliable
- the request originator $\text{origin}(rreq) \in Agent$
- the originator’s $\text{originSeqNum}(rreq) \in \text{NAT}$
- the length $\text{hopCount}(rreq) \in \text{NAT}$ of the path the request traveled from its $\text{origin}(rreq)$ to its current $rreq$-sender
- the value $\text{localId}(rreq) \in \text{NAT}$ of $\text{localReqCount} + 1$ at the $\text{origin}(rreq)$ when the $rreq$ is generated

NB. Global identification of $rreq$ via the following equation:

$\text{globalId}(rreq) = (\text{localId}(rreq), \text{origin}(rreq))$
AODV data structure: RouteReply

Each route reply msg \( rrep \in RouteReply \), when generated, records information about:

- the destination \( d = \text{dest}(rrep) \in Agent \) of the detected route
- the value \( \text{destSeqNum}(rrep) \in NAT \) of \( \text{curSeqNum}(d) \)
  - as known at an intermediate node (not \( d \)), if \( rrep \) is generated there
  - or as updated when the destination node \( d \) generates \( rrep \)
- the request originator \( \text{origin}(rrep) \in Agent \) to whom the reply is addressed
- the length \( \text{hopCount}(rrep) \in NAT \) of the current route from the 
  \( rrep \)-sender to \( \text{dest}(rrep) \)

NB. We abstract from concerns about \( rrep \) lifetime, network traffic and performance properties.
Each route error msg $rerr \in \text{RouteError}$, sent by an agent $a$,

- indicates a set of destinations, together with their increased $\text{destSeqNum}$ value, which became unreachable via $a$, i.e. cannot be reached at present using $a$ as $\text{nextHop}$ of a route entry
- inactivates every route table $\text{entry}$ which uses the $rerr$ sender $a$ as $\text{nextHop}$ to any relevant unreachable destination communicated by $rerr$
- is forwarded along the $\text{precursor}$ chain
**GenerateRouteReq**(destination)

\[
\text{let } r = \text{new } (\text{RouteRequest}) \text{ in } \\
\begin{align*}
\text{dest}(r) & := \text{destination} \\
\text{destSeqNum}(r) & := \\
& \quad \text{lastKnownDestSeqNum}(\text{destination}, \text{RT}) \\
\text{if } \text{entryFor}(\text{destination}, \text{RT}) \neq \text{undef} \\
\text{then } \text{known}(r) & := \text{known}(\text{entryFor}(\text{destination}, \text{RT})) \\
\text{else } \text{known}(r) & := \text{false} \\
\text{origin}(r) & := \text{self} \quad \text{originSeqNum}(r) := \text{curSeqNum} + 1 \\
\text{hopCount}(r) & := 0 \quad \text{localId}(r) := \text{localReqCount} + 1 \\
\text{Broadcast}(r) & \quad \text{-- i.e. } \forall n \in \text{neighb} \; \text{do SEND}(r, \text{to } n) \\
\text{INCREMENT}(& \text{curSeqNum}) \quad \text{INCREMENT}(& \text{localReqCount}) \\
\text{Buffer}(r) & \quad \text{-- i.e. } \text{INSERT}(\text{globalId}(r), \text{ReceivedReq}) \\
\end{align*}
\]

NB. **Buffer**(r) helps to recognize whether r has been ‘seen’ already
**ProcessRouteReq**(*rreq*)

```plaintext
if Received(*rreq*) and *rreq* ∈ RouteRequest then
  if not AlreadyReceivedBefore(*rreq*) then -- *rreq* processed once
    BUFFER(*rreq*)
    if HasNewReverseRouteInfo(*rreq*) then
      BUILDREVERSEROUTE(*rreq*)
      seq
        if FoundValidPathFor(*rreq*)
          then GENERATEROUTEREPLY(*rreq*)
          else FORWARDREFRESHEDREQ(*rreq*)
      CONSUME(*rreq*)
```

**NB.** GENERATEROUTEREPLY sends *rreply* to nextHop in—possibly by BUILDREVERSEROUTE updated—entryFor(origin(*rreq*), RT), which could be different from sender(*rreq*).
A req where HasNewReverseRouteInfo(req) is false

AlreadyReceivedBefore(req) iff globalId(req) ∈ ReceivedReq
    // i.e. req has been BUFFERed when received for the first time

Such route req msgs are simply discarded. Nothing else happens.

Otherwise, if the req brings no new reverse route information to the RT of the receiving agent, the existing reverse route entry is kept unchanged and the protocol proceeds to either GENERATEROUTE_REPLY(rreq) or FORWARDREFRESHEDREQ(rreq).

HasNewReverseRouteInfo(req) iff req ∈ RouteRequest and
   ThereIsNoRouteInfoFor(origin(req), RT) or
   (ThereIsRouteInfoFor(origin(req), RT) and
    HasNewOriginInfo(req, RT))
When $\text{HasNewOriginInfo}(\text{req})$ to update the reverse route

$\text{HasNewOriginInfo}(\text{req}, \text{RT})$ iff

let $entry = entry\text{For}(\text{origin}(\text{req}), \text{RT})$

$\text{originSeqNum}(\text{req}) > \text{destSeqNum}(entry)$

or $\text{originSeqNum}(\text{req}) = \text{destSeqNum}(entry)$ and

$(\text{hopCount}(\text{req}) < \text{hopCount}(entry) \text{ or }$

not $\text{Active}(entry))$

NB. For comparison with $\text{destSeqNum}$bers, every natural number $n$ is stipulated to be better than $\text{unknown}$, formally $\text{unknown} < n$. 
**BuildReverseRoute(rreq) component**

if ThereIsRouteInfoFor(origin(rreq), RT) then

  UpdateReverseRoute(entryFor(origin(rreq), RT), rreq)

else ExtendReverseRoute(RT, rreq)

where

  UpdateReverseRoute(e, req) =
  destSeqNum(e) := originSeqNum(req) -- freshest destSeqNum
  known(e) := true     Active(e) := true
  nextHop(e) := sender(req)
  hopCount(e) := hopCount(req) + 1 -- maybe shorter path

  ExtendReverseRoute(RT, req) =

  let e = new (RT)
    dest(e) := origin(req)    precursor(e) := ∅
  UpdateReverseRoute(e, req)
Redirecting Reverse Route example

- Reverse routes for segments of \( rreq \)-path from \( a_1 \) to \( a_{n+1} \) are created.

- \( a_i \) receives a new \( req \) from \( a_1 \) to another destination and redirects the \( rreq \) reverse route at \( a_i \).\(^1\)

\(^1\) Figures © 2018 Springer-Verlag Berlin Heidelberg, reprinted with permission
The meaning of \texttt{FoundValidPathFor}(\texttt{req})

- either \texttt{req} is received by the destination node
- or an ‘intermediate’ node which \texttt{KnowsFreshEnoughRouteFor} the \texttt{dest(req)} receives the \texttt{req}:

\texttt{FoundValidPathFor}(\texttt{req}) \texttt{iff}

\[
\texttt{dest(req)} = \texttt{self} \texttt{ or } \texttt{KnowsFreshEnoughRouteFor(req, RT)}
\]

A ‘fresh enough’ route entry besides being \texttt{Active} must have a \texttt{ValidDestSeqNumber} that is not smaller than the \texttt{destSeqNumber} of the received route \texttt{request}:

\texttt{KnowsFreshEnoughRouteFor(req, RT) iff forsome entry } \in \texttt{RT}

\[
\texttt{dest(entry)} = \texttt{dest(req)} \texttt{ and } \texttt{ValidDestSeqNum(entry)}
\]

\[
\text{and } \texttt{destSeqNum(entry)} \geq \texttt{destSeqNum(req)}
\]

\[
\text{and } \texttt{Active(entry)}
\]
**ForwardRefreshedReq** \((r) = \)

let \(r' = \text{new} (RouteRequest)\)  

\[ \text{COPY}(\text{dest, origin, originSeqNum, localId, known, from } r \text{ to } r') \]

\[ \text{hopCount}(r') := \text{hopCount}(r) + 1 \]

\[ \text{destSeqNum}(r') := \max\{\text{destSeqNum}(r), \text{lastKnownDestSeqNum}(\text{dest}(r), \text{RT})\} \]

\[ \text{Broadcast}(r') \]

where  

\[ \text{COPY}(f_1, \ldots, f_n, \text{from arg to arg'}) = \]

\[ \text{forall } 1 \leq i \leq n \text{ do } f_i(\text{arg'}) := f_i(\text{arg}) \]
**GenerateRouteReply**$(rreq)$ at $dest$ or intermediate node

let $r = \text{new (RouteReply)}$

let $revEntry = \text{entryFor}(\text{origin}(rreq), RT)$

dest$(r) := \text{dest}(rreq)$

origin$(r) := \text{origin}(rreq)$

if $\text{dest}(rreq) = self$ then

    \text{hopCount}(r) := 0

    \text{destSeqNum}(r) := \max\{\text{curSeqNum}, \text{destSeqNum}(rreq)\}$

    \text{curSeqNum} := \max\{\text{curSeqNum}, \text{destSeqNum}(rreq)\}$

else let $fwdEntry = \text{entryFor}(\text{dest}(rreq), RT)$ -- at intermediate

    \text{hopCount}(r) := \text{hopCount}(fwdEntry) -- node

    \text{destSeqNum}(r) := \text{destSeqNum}(fwdEntry)$

    \text{PrecursorInsertion}$(\text{nextHop}(revEntry), fwdEntry)$

\text{Send}(r, \text{to} \text{nextHop}(revEntry)) -- maybe to sender$(rreq)$
The role of precursor nodes

Consider the case that a node \( a \) has an \( \text{entryFor}(d, RT) \) and by a \textit{rerror} msg, received by \( a \), node \( d \) is reported as unreachable. Then each \( \text{neighbor}(a) \) which has a route entry to \( d \) that uses \( a \) as \textit{nextHop} is called a \textit{precursor} of \( a \).

The \textit{precursor} set is recorded in \( \text{entryFor}(d, RT) \) so that such a \textit{rerror} msg can be propagated to its elements.

\begin{equation}
\text{PrecursorInsertion}(node, entry) = \text{Insert}(node, \text{precursor}(entry))
\end{equation}

NB. We leave ‘gratuitous’ replies as an exercise. Through gratuitous replies, a destination node obtains a reverse route to the request originator without having requested a route, namely in case an intermediate node answered the request. For this case one needs also a \text{PrecursorInsertion} of \( \text{nextHop}(fwdEntry) \) into \text{precursor}(revEntry).
**ProcessRouteRep**($rrep$)

if Received($rrep$) and $rrep \in \text{RouteReply}$ then
  if HasNewForwardRouteInfo($rrep$) then -- else just discard $rrep$
    BUILDFORWARDROUTE($rrep$)
  if MustForward($rrep$) then FORWARDREFRESHEDREP($rrep$)
  CONSUME($rrep$)

where  -- note symmetry to ProcessRouteReq

HasNewForwardRouteInfo($rep$) iff $rep \in \text{RouteReply}$ and
  ThereIsNoRouteInfoFor(dest($rep$), RT) or
  (ThereIsRouteInfoFor(dest($rep$), RT) and
   HasNewDestInfo($rep$, RT))

MustForward($rep$) iff
  origin($rep$) \neq \text{self} and Active(entryFor(origin($rep$), RT))
Meaning of $\text{HasNewDestInfo}$

$\text{HasNewDestInfo}(\text{rep}, \text{RT})$ \textbf{iff} \ -- \ rep has either

\[
\text{let } \text{entry} = \text{entryFor}(\text{dest}(\text{rrep}), \text{RT})
\]

\[
\text{destSeqNum}(\text{rep}) > \text{destSeqNum}(\text{entry}) \quad \text{-- better destSeqNum}
\]

or\  \ ( \text{destSeqNum}(\text{rep}) = \text{destSeqNum}(\text{entry})

\[
\text{and hopCount}(\text{rep}) + 1 < \text{hopCount}(\text{entry})) \quad \text{-- or shorter path}
\]

or\  \ ( \text{destSeqNum}(\text{rep}) = \text{destSeqNum}(\text{entry})

\[
\text{and Active(\text{entry}) = false) } \quad \text{-- or not Active(\text{entry})}
\]

NB. Remember $\text{unknown} < n$ for each $n = 0, 1, \ldots$
**BuildForwardRoute**\((rrep)\) with fresh info

if ThereIsRouteInfoFor\((dest(rrep), RT)\)
  then UpdateForwardRoute\((entryFor(dest(rrep), RT), rrep)\)
else ExtendForwardRoute\((RT, rrep)\)  -- create new entry

where

**UpdateForwardRoute**\((e, rep)\) =

\[
\begin{align*}
\text{destSeqNum}(e) & := \text{destSeqNum}(rep) \\
\text{known}(e) & := \text{true} \\
\text{nextHop}(e) & := \text{sender}(rep) \\
\text{hopCount}(e) & := \text{hopCount}(rep) + 1 \\
\text{Active}(e) & := \text{true} \\
\end{align*}
\]

**ExtendForwardRoute**\((RT, rep)\) =

let \(e = \text{new}(RT)\)
  dest\((e) := \text{dest}(rep)\)

**UpdateForwardRoute**\((e, rep)\)

**SetPrecursor**\((rep, e)\) = if MustForward\((rep)\) then

**Insert**\((\text{nextHop}(\text{entryFor}(\text{origin}(rep), RT)), \text{precursor}(e))\)
Upon forwarding \textit{rep}, only the \textit{hopCount} is updated:

\begin{verbatim}
FORWARD_REFRESHEDREP(rep) =
let rep' = new (RouteReply)
COPY(dest, destSeqNum, origin, \textbf{from rep to rep'})
hopCount(rep') := hopCount(rep) + 1
SEND(rep', \textbf{to} nextHop(entryFor(origin(rep), RT)))
\end{verbatim}
Redirecting Forward Route example (1)

- Destination $a_{n+1}$ answers $rreq$ before $req$ by $rrep$ with $destSeqNum = 0$

- $a_{n+1}$ broadcasts a new $rrq$ for destination $z$, $curSeqNum := 1$
- $a_{n+1}$ answers $req$ by $rep$ with $destSeqNum(rep) := 1$
- $a_i$ receives $rep$ before $rrep$ and establishes $entryFor(a_{n+1}, RT(a_i))$ with $destSeqNum 1$
- $rrep_{a_1,a_{n+1}}(0)$ is discarded at $a_i$
A new route request may be answered by $a_i$ as intermediate node establishing the dotted communication path ($\cdots$) from $a_1$ to $a_{n+1}$.

The communication path from $a_{n+1}$ to $a_1$ still goes along the reverse route established by $rreq_{a_1,a_{n+1}}[1]^2$. 

---

Figures (1)-(3) © 2018 Springer-Verlag Berlin Heidelberg, reprinted with permission
Generation of error messages

\textbf{GENERATEROUTEERR} =

\begin{align*}
&\text{let } \text{BrokenEntry} = -- \text{compute } \text{Active} \text{ entries with broken link} \\
&\{ \text{entry} \in \text{RT} \mid \text{LinkBreak}(\text{nextHop}(\text{entry})) \text{ and } \text{Active}(\text{entry}) \} \\
&\text{forall } \text{entry} \in \text{BrokenEntry} -- \text{if there are any} \\
&\quad \text{Active}(\text{entry}) := \text{false} -- \text{ Invalidate } \text{ entry} \\
&\quad \text{INCREMENT}(\text{destSeqNum}(\text{entry})) -- \text{and } \text{destSeqNum} \\
&\text{let } \text{rerr} = \{ (\text{dest}(e), \text{destSeqNum}(e) + 1) \mid \\
&\quad e \in \text{BrokenEntry} \text{ and } \text{precursor}(e) \neq \emptyset \} \\
&\quad \text{forall } a \in \text{precursor}(\text{entry}) \text{ SEND}(\text{rerr}, \text{to } a) \\
\end{align*}

NB. Stipulation \textit{unknown} + 1 = \textit{unknown}
\begin{align*}
\textbf{PropagateRouteErr}(rerr) \\
\textbf{if } Received(rerr) \text{ and } rerr \in RouteError \text{ then} \\
\text{let } UnreachDest = \{(d, s) \in rerr \mid \text{for some } entry \in RT \\
\quad d = \text{dest}(entry) \text{ and } \text{nextHop}(entry) = \text{sender}(rerr) \\
\quad \text{and } \text{Active}(entry) \text{ and } \text{destSeqNum}(entry) < s \} \\
\forall (d, s) \in UnreachDest \quad \text{let } entry = \text{entryFor}(d, RT) \\
\quad \text{Active}(entry) := \text{false} \\
\quad \text{destSeqNum}(entry) := s \\
\forall a \in \text{precursor}(entry) \quad \text{SEND}(rerr', \text{to } a) \\
\text{if } \text{WaitingForRouteTo}(d) \text{ then } \text{ReGenerateRouteReq}(d) \\
\text{CONSUME}(rerr) \\
\text{where } err' = \\
\quad \{(d', s') \in UnreachDest \mid \text{precursor}(\text{entryFor}(d', RT)) \neq \emptyset\} \\
\text{ReGenerateRouteReq}(d) = (\text{WaitingForRouteTo}(d) := \text{false})
\end{align*}
Conclusion: What else to do with a rigorous model?

- **system debugging** by rigorous model analysis
  - (dis)prove system properties of interest (e.g. using PVS, KIV,...)
    - for AODV: loop freedom and correctness proved, route discovery and packet delivery disproved for process algebra model in TR 5513 (for loop freedom using Isabelle)
  - identify shortcomings (here e.g. non-optimal routes)
  - testing/modelchecking of executable model refinements (e.g. in CoreASM, Asmeta, ...)

- **system evaluation** by comparison of implementations with the model
  - in TR 5513 done for five key AODV implementations, three of them shown to possibly produce routing loops

- **model reuse** for experimenting with system extensions/variations, prior to coding

- **documentation** for maintenance needs
References

- C.E. Perkins, E.M. Belding-Royer, S. Das: *Ad hoc On-Demand Distance Vector (AODV) Routing*

- A. Fehnker, R. van Glabbeek, P. Höfner, A. McIver, M. Portmann, W.L. Tan: *A process algebra for wireless mesh networks used for modelling, verifying and analysing AODV*

- E. Börger, A. Raschke: *Modeling Companion for Software Practitioners*
  – Springer 2018. (Ch.6.1)
    http://modelingbook.informatik.uni-ulm.de
It is permitted to (re-) use these slides under the CC-BY-NC-SA licence

https://creativecommons.org/licenses/by-nc-sa/4.0/

i.e. in particular under the condition that

- the original authors are mentioned
- modified slides are made available under the same licence
- the (re-) use is not commercial