# Sequence Numbers Do Not Guarantee Loop Freedom —AODV Can Yield Routing Loops—

Rob van Glabbeek
NICTA, Australia
University of New South Wales, Australia
rvg@cs.stanford.edu

Peter Höfner
NICTA, Australia
University of New South Wales, Australia
Peter.Hoefner@nicta.com.au

Wee Lum Tan
NICTA, Australia
University of Queensland, Australia
WeeLum.Tan@nicta.com.au

Marius Portmann
NICTA, Australia
University of Queensland, Australia
marius@itee.uq.edu.au

## ABSTRACT

In the area of mobile ad-hoc networks and wireless mesh networks, sequence numbers are often used in routing protocols to avoid routing loops. It is commonly stated in protocol specifications that sequence numbers are sufficient to *guarantee* loop freedom if they are monotonically increased over time. A classical example for the use of sequence numbers is the popular Ad hoc On-Demand Distance Vector (AODV) routing protocol. The loop freedom of AODV is not only a common belief, it has been claimed in the abstract of its RFC and at least two proofs have been proposed. AODV-based protocols such as AODVv2 (DYMO) and HWMP also claim loop freedom due to the same use of sequence numbers.

In this paper we show that AODV is not a priori loop free; by this we counter the proposed proofs in the literature. In fact, loop freedom hinges on non-evident assumptions to be made when resolving ambiguities occurring in the RFC. Thus, monotonically increasing sequence numbers, by themselves, do *not* guarantee loop freedom.

## Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Routing protocols; Protocol verification; F.3.1 [**Specifying and Verifying and Reasoning about Programs**]: Invariants

## Keywords

AODV; loop freedom; process algebra; routing protocols; wireless mesh networks


*MSWiM* 2013 Barcelona, Spain

http://dx.doi.org/10.1145/2507924.2507943 ...$15.00.

## 1. INTRODUCTION

Wireless Mesh Networks (WMNs), which can be considered to include Mobile Ad-hoc Networks (MANETs), have gained considerable popularity and are increasingly deployed in a wide range of application scenarios, including emergency response communication, intelligent transportation systems, mining and video surveillance. They are self-organising wireless multi-hop networks that can provide broadband communication without relying on a wired backhaul infrastructure, a benefit for rapid and low-cost network deployment.

Highly dynamic topologies are a key feature of WMNs and MANETs, due to mobility of nodes and/or the variability of wireless links. This makes the design and implementation of robust and efficient routing protocols for these networks a challenging task, and a lot of research effort has gone into it.

Loop freedom is a critical property for any routing protocol, but it is particularly relevant and challenging for WMNs and MANETs. Descriptions as in [9] capture the common understanding of loop freedom: "*A routing-table loop is a path specified in the nodes' routing tables at a particular point in time that visits the same node more than once before reaching the intended destination.*" Packets caught in a routing loop, until they are discarded by the IP Time-To-Live (TTL) mechanism, can quickly saturate the links and have a detrimental impact on network performance. It is therefore critical to ensure that protocols prevent routing loops.

Sequence numbers, indicating the freshness of routing information, have been widely used to guarantee loop freedom, in particular for distance vector protocols such as DSDV [20], AODV [19], AODVv2 (formerly known as DYMO) [21] and HWMP [14]. These protocols claim to be loop free due to the use of monotonically increasing sequence numbers. For example, the AODV RFC states: AODV "*uses destination sequence numbers to ensure loop freedom at all times (even in the face of anomalous delivery of routing control messages), ...*" [19], and a similar claim is made in the IETF draft of AODVv2 [21]: "*AODVv2 uses sequence numbers to assure loop freedom [Perkins99].*"[1] A proof of loop freedom of AODV has been provided in [22]. Another, more recent

[1]Here, [Perkins99] is our reference [22].

proof is [26].[2] It is therefore a common belief that the use of sequence numbers in this context guarantees loop freedom.

However, while this use of sequence numbers can be an efficient approach to address the problem of routing loops, we show in this paper that sequence numbers by themselves do not guarantee loop freedom. We illustrate this using AODV as a running example.

We show that loop freedom can be guaranteed only if sequence numbers are used in a careful way, considering further rules and assumptions on the behaviour of the protocol. The problem is, as shown in the case of AODV, that these additional rules and assumptions are not explicitly stated in the RFC, and that the RFC has significant ambiguities in this regard. We demonstrate that routing loops can be created—while fully complying with the RFC, and making reasonable assumptions when the RFC allows different interpretations. As a consequence, which is a *key contribution* of this paper, we obtain that routing protocols using sequence numbers as described in [19] are not a priori loop free. We argue that the lack of precision and the corresponding ambiguity of the protocol definition in the RFC is a key problem here—and for RFCs in general. As *another contribution* we show details of several ambiguities and contradictions in the AODV RFC, and discuss which interpretations will lead to routing loops. A *third contribution* of this paper is an analysis of five key implementations of the AODV protocol, and a discussion of their corresponding loop freedom properties.

To address the problem of ambiguities and contradictions in RFCs, we argue for the benefit of more precise and formal approaches for the specification of protocols, which are sufficiently expressive to model real networks and protocols, while maintaining usability. As a *final contribution* we show how formal methods can be used to avoid ambiguities in RFCs and to guarantee properties such as loop freedom.

The remainder of the paper is organised as follows: in Section 2 we recapitulate the basic principles of the AODV protocol. We state what we mean by loop freedom and discuss the existing proofs of AODV's loop freedom. In Section 3, nearly all plausible readings of the AODV RFC, resolving its ambiguities and contradictions, are discussed. Each of the presented interpretations is analysed w.r.t. loop freedom. In Section 4 we show an example of a routing loop occurring in the AODV routing protocol, as a result of following a reasonable and plausible interpretation of the RFC. Having the various RFC interpretations and the loop example in mind, we then analyse five different key implementations of AODV w.r.t. routing loops. Finally, in Section 6, we sketch how formal methods—here in the form of process algebra—can be applied to model routing protocols and verify the presented results on routing loops. Before we conclude with a short discussion in Section 8, we present major related work in Section 7.

# 2. AODV

To prove that sequence numbers do not a priori guarantee loop freedom, we use AODV as an example. We expect that other routing protocols such as AODVv2 and HWMP behave similarly. AODV [19] is a popular routing protocol designed for MANETs, and is one of the four protocols currently standardised by the IETF MANET working group[3]. It also forms the basis of new WMN routing protocols, including the upcoming IEEE 802.11s wireless mesh network standard [14]. AODV is designed for wireless and mobile networks where links are particularly unreliable.

[2]We discuss problems with these proofs later in this paper.
[3]`http://datatracker.ietf.org/wg/manet/charter/`

## 2.1 Brief Overview

AODV is a reactive protocol, which means that routes are established only on demand. If a node $S$ wants to send a data packet to a node $D$, but currently does not know a route, it temporarily buffers the packet and initiates a route discovery process by broadcasting a route request (RREQ) message in the network. An intermediate node $A$ that receives the RREQ message creates a routing table entry for a route towards node $S$ referred to as a *reverse route*, and re-broadcasts the RREQ. This is repeated until the RREQ reaches the destination node $D$, or alternatively a node that knows a route to $D$. In both cases, the node replies by unicasting a corresponding route reply (RREP) message back to the source $S$, via a previously established reverse route. When forwarding RREP messages, nodes create a routing table entry for node $D$, called the *forward route*. When the RREP reaches the originating node $S$, a route from $S$ to $D$ is established and data packets can start to flow. Both forward and reverse routes are maintained in a routing table at every node—details are given below. In the event of link and route breaks, AODV uses route error (RERR) messages to notify the affected nodes: if a link break is detected by a node, it first invalidates all routes stored in the node's own routing table that actually use the broken link. Then it sends a RERR message containing the unreachable destinations to all (direct) neighbours using this route.

In AODV, a routing table consists of a list of entries—at most one for each destination—each containing the following information:

- the destination IP address;
- the destination sequence number;
- the sequence-number-status flag—tagging whether the recorded sequence number can be trusted;
- a flag tagging the route as being valid or invalid—this flag is set to invalid when a link break is detected;
- the hop count, a metric to indicate the distance to the destination;
- the next hop, an IP address that identifies the next (intermediate) node on the route to the destination;
- a list of precursors, a set of IP addresses of those 1-hop neighbours that use this particular route; and
- the lifetime (expiration or deletion time) of the route.

The destination sequence number constitutes a measure approximating the relative freshness of the information held—a higher number denotes newer information. The routing table is updated whenever a node receives an AODV control message (RREQ, RREP or RERR) or detects a link break.

During the life time of the network, each node not only maintains its routing table, it also stores its *own sequence number*. This number is used as a local "timer" and is incremented whenever a new route request is initiated.

Full details of the protocol are outlined in the request for comments (RFC) [19], the official specification of AODV.

## 2.2 Loop Freedom

The "naive" notion of loop freedom is a term that informally means that "a packet never goes round in cycles without (at some point) being delivered". This dynamic definition is too restrictive a requirement for AODV. There are situations where packets are sent in cycles, but which should not be considered "harmful". This can happen when the network topology keeps changing. The sense of loop freedom is much better captured by a static invariant, saying that *at any given time the collective routing tables of the nodes do not admit a loop.* Such a requirement does not rule out the dynamic loop alluded to above. However, in situations where the topology remains stable sufficiently long it does guarantee that packets will not keep going around in cycles.

## 2.3 Proofs Based On Sequence Numbers

As mentioned before, AODV "*uses destination sequence numbers to ensure loop freedom at all times*" [19, Page 2]. Moreover, it has been "proven" at least twice that AODV is loop free [22, 26].

In both papers a main argument is that messages relating to a particular route request are handled only once at every node; moreover that every route discovery process has a unique sequence number. The latter is guaranteed since a node's own sequence number is incremented whenever a new route discovery is initiated. Each sequence number stored in any routing table for destination *dip* is ultimately derived from *dip*'s own sequence number at the time such a route was discovered.

The proof sketch given in [22] uses the fact that when a loop in a route to a destination $Z$ is created, all nodes $X_i$ on that loop must have route entries for destination $Z$ with the same destination sequence number. "*Furthermore, because the destination sequence numbers are all the same, the next hop information must have been derived at every node $X_i$ from the same RREP transmitted by the destination $Z$*" [22, Page 11]. The latter is not true at all: some of the information could have been derived from RREQ messages, or from a RREP message transmitted by an intermediate node that has a route to $Z$. More importantly, the nodes on the loop may have acquired their information on a route to $Z$ from different RREP or RREQ messages, that all carried the same sequence number. This will be illustrated in our forthcoming loop example.

The proof of [26] was established in two steps: (i) a mathematical model of AODV was derived[4]; (ii) based on the derived model loop freedom was proven using the interactive theorem prover Isabelle [18].[5] In the model of [26] route replies generated by intermediate nodes [19, Sect. 6.6.2.] are not considered. Since this is a key feature of AODV (and, as we shall see, an essential ingredient in the creation of routing loops), the work of [26] cannot be said to pertain to the full protocol.

# 3. AMBIGUITIES IN THE AODV RFC

"*A Request for Comments (RFC) is a publication of the Internet Engineering Task Force (IETF) and the Internet Society [. . . ]. A RFC is authored by engineers and computer scientists in the form of a memorandum describing methods, behaviors, research, or innovations applicable to the working of the Internet and Internet-connected systems. [. . . ] The IETF adopts some of the proposals published as RFCs as Internet standards.*"[6] Not all RFCs are standards [13]. However, in the case of the AODV routing protocol, the RFC 3561 [19] is the de facto standard.

RFCs are typically written in English and are not equipped with a formal description language. This has the advantage that everybody can read any RFC. However, it holds the disadvantage that the RFC contains contradictions and ambiguities. Additionally, there may be unexpected situations occurring in real networks that are not described or anticipated by the RFC. In sum, this yields a variety of interpretations for every RFC; some interpretations being more plausible than others.

In this section, we discuss some of the ambiguities found in the specification of AODV. We catalogue and analyse the variants of AODV arising from interpretations consistent with the reading of the RFC and show which of them yields routing loops. Our analysis is based on a rigorous, formal, and mathematical approach [7]. A full and detailed analysis is given in [8]. Table 1 summarises the results; section numbers refer to [19].

One of the crucial aspects of AODV is the maintenance of routing tables. Hence the update of routing table entries with new information has to be performed carefully. Unfortunately, the RFC specification only gives hints how to update routing table entries; an exact and precise definition is missing.

**1. Updating the Unknown (Invalid) Sequence Number in Response to a Route Reply.** If a node receives a RREP message, it might have to update its routing table: "*the existing entry is updated only in the following circumstances: (i) the sequence number in the routing table is marked as invalid [. . . ]*" [19, Sect. 6.7]. Here it is relevant that, through the sequence-number-status flag, any sequence number can be marked as unknown or invalid. In the same section it is also stated what actions occur if a route is updated. Among others it is stated that "*the destination sequence number is marked as valid, [. . . ] and the [new] destination sequence number [in the routing table] is the Destination Sequence Number in the RREP message.*" [19, Sect. 6.7]. The interpretation that follows these lines literally is denoted 1a in Table 2. It can decrement sequence numbers, which immediately yields routing loops, as explained in [8, Sect. 8]. This update mechanism contradicts Sect. 6.1 of [19], which states that any information from an incoming AODV control message that carries a lower sequence number than the corresponding entry in the routing table MUST be discarded (Interpretation 1b). The routing loops resulting from following Sect. 6.7 strongly indicate that this contradiction in the RFC should be resolved in favour of Sect. 6.1.

**2. Updating with the Unknown Sequence Number.** Whenever a node receives a forwarded AODV control message from a 1-hop neighbour, it creates a new or updates an existing routing table entry to that neighbour. For example, "*[w]hen a node receives a RREQ, it first creates or updates a route to the previous hop without a valid sequence number*" [19, Sect. 6.5]. In case a new routing table entry is created, the sequence number is set to a default value (typically 0, as is done in implementations such as AODV-UU [2], AODV-UIUC [15] and AODV-UCSB [4]) and the sequence-

[4] In [26] the model is only given partially.

[5] Again only snippets of the proofs can be found in [26].

[6] `http://en.wikipedia.org/wiki/Request_for_Comments`

**Table 1: Analysis of different interpretations of the RFC 3561 (AODV)**

| | | |
|---|---|---|
| **1. Updating the Unknown (Invalid) Sequence Number in Response to a Route Reply** | | |
| 1a. | the destination sequence number (DSN) is copied from the RREP message (Sect 6.7) | may decrement sequence numbers, which causes loops |
| 1b. | the routing table is not updated when the information that it has is "fresher" (Sect. 6.1) | does not cause loops |
| **2. Updating with the Unknown Sequence Number (Sect. 6.5)** | | |
| 2a. | no update occurs | does not cause loops, but opportunity to improve routes is missed. |
| 2b. | overwrite any routing table entry by an update with an unknown DSN | may decrement sequence numbers, which causes loops |
| 2c. | use the new entry with the old DSN | does not cause loops |
| **3. (Dis)Allowing the Creation of Self-Entries in Response to a Route Reply** | | |
| 3a. | allow (arbitrary) self-entries | loop free only if used with Interpretations 4d or 4e below |
| 3b. | allow optimal self-entries only; store own sequence number in optimal self-entry | does not cause loops |
| 3c. | disallow self-entries; if self-entries would occur, ignore msg. | does not cause loops |
| 3d. | disallow self-entries; if self-entries would occur, forward | does not cause loops |
| **4. Invalidating Routing Table Entries in Response to a Route Error Message** | | |
| 4a. | always copy DSN from RERR message (Sect. 6.11) | may decrement sequence numbers, which causes loops (when allowing self-entries (Interpretation 3a)) |
| 4b. | only invalidate if the DSN in the routing table is smaller than or equal to the one from the RERR message (Sect. 6.1) | causes loops (when allowing self-entries) |
| 4c. | take the maximum of the DSN of the routing table and the one from the RERR message | causes loops (when allowing self-entries) |
| 4d. | take the maximum of the increased DSN of the routing table and the one from the RERR message | does not cause loops |
| 4e. | only invalidate if the DSN in the routing table is smaller than the one from the RERR message (Sect. 6.2) | does not cause loops |

number-status flag is set to `unk` to signify that the sequence number corresponding to the neighbour is unknown. But, what happens if there exists already a routing table entry? Following the quote above, the routing table has to be updated. Unfortunately, it is not stated how the update is done.[7] There are three reasonable options:
(2a) no update occurs. This interpretation is harmless with regard to routing loops, but misses an opportunity to improve some routes. It can be argued that the RFC rules out this option by including "or updates" in the quote above.
(2b) All information is taken from the incoming AODV control message; since that message formally does not contain a sequence number for the neighbour, the destination sequence number is set to value 0. Since this can decrease sequence numbers, routing loops might occur. Hence this interpretation must not be used.
(2c) The information from the routing table and from the incoming AODV control message is merged.[8] This interpretation does not give rise to loops.

**3. (Dis)Allowing the Creation of Self-Entries in Response to a Route Reply.** In any practical implementation, when a node sends a data packet to itself, the packet will be delivered to the corresponding application on the local node without ever involving a routing protocol and therefore without being "seen" by AODV or any other routing protocol. Because of this it seems that it does not make a difference whether any node using AODV stores routing table entries to itself.

In AODV, when a node receives a RREP message, it creates a routing table entry for the destination node if such an entry does not already exist [19, Sect. 6.7]. If the destination node happens to be the processing node itself, this leads to the creation of a self-entry. The RFC does not mention self-entries explicitly; it only refers to them at one location: "*A node may change the sequence number in the routing table entry of a destination only if: – it is itself the destination node [. . . ]*" [19, Sect. 6.1]. This points at least to the possibility of having self-entries. We have analysed various implementations of AODV and found that the Kernel-AODV [1], AODV-UIUC [15], AODV-UCSB [4] and AODV-ns2 implementations allow the creation of self-entries.

If arbitrary self-entries are allowed (Interpretation 3a in Table 2) this can, in combination with other plausible assumptions, yield routing loops, as we will show in the next section. However, storing only optimal self-entries in routing tables (Interpretation 3b) does not cause loops. For example, Kernel-AODV maintains the nodes' own sequence numbers in this way.

On the other hand, there are two possibilities to disallow self-entries: if a node receives a route reply and would create a self-entry, it silently discards the message (Interpretation 3c). This interpretation has the disadvantage that replies are lost. The alternative is that the node forwards the message without updating its routing table (3d). Both variants by themselves do not yield routing loops.

**4. Invalidating Routing Table Entries in Response to a Route Error Message.** If a node receives a RERR message, it might invalidate entries of its routing table based on information from this message. When invalidating routing table entries, destination sequence numbers should be "*copied from the incoming RERR*" [19, Sect. 6.11]. This is Interpretation 4a in Table 1. In particular, this part of the RFC prescribes the replacement of an existing destination sequence number in a routing table entry with one that may be strictly smaller, which contradicts Sect. 6.1 of the RFC.

[7] Section 6.2 of [19] further explains in which circumstances an update occurs. It does not resolve this ambiguity (cf. [8]).
[8] By taking the destination sequence number from the existing routing table entry and all other information from the AODV control message.

To make invalidation consistent with Sect. 6.1 of the RFC, one could use two possible variants instead. The first (4b), strictly following Sect. 6.1, invalidates only if the destination sequence number in the routing table is smaller than or equal to the destination sequence number provided by the incoming RERR message; it aborts otherwise. The second (4c) invalidates in all circumstances, but prevents a decrease in the destination sequence number by taking the maximum of the stored and the incoming number.[9] In the next section we will show that each of these three interpretations can yield routing loops, when used in conjunction with non-optimal self-entries.

There are two reasonable solutions to avoid routing loops in these circumstances. As a modification of Interpretation 4c, one can first increment the destination sequence number of the routing table by one and then use the maximum of this updated sequence number and the one from the RERR message (Interpretation 4d). Alternatively, one could invalidate only if the destination sequence number in the routing table is (strictly) smaller than the destination sequence number provided by the incoming RERR message (4e), and abort if it is larger or equal. The latter interpretation is at least consistent with Section 6.2 of the RFC, which states that "*The route is only updated if the new sequence number is either (i) higher than the destination sequence number in the route table, or [...]*".

# 4. AODV YIELDS LOOPS

In the previous section, we discussed some of the ambiguities found in the specification of AODV, and catalogued the variants of AODV that arise from interpretations consistent with the RFC. In the following we describe an example of a routing loop in AODV, based on a reasonable and plausible interpretation of the RFC, following Interpretations 3a and any out of 4a, 4b and 4c of AODV, in resolving the ambiguities from Table 1.

## 4.1 Creating Routing Loops

The given example (shown in Figure 1) consists of four parts: (1) First, a standard RREQ-RREP cycle occurs (Figures 1(a)–(b)); (2) Then, a node stores information about *itself* in its routing table (Figures 1(c)–(e)); Such information is called a *self-entry*. (3) Another standard RREQ-RREP cycle occurs (Figure 1(f)); (4) Finally, a link break in combination with another route discovery yields the loop (Figures 1(g)–(h)).

**1. Standard RREQ-RREP Cycle.** In Figure 1(a), we show the initial network topology, with the nodes' sequence numbers depicted inside the circles. Figure 1(b) shows node $D$ searching for a route to node $A$. We see that the sequence number for node $D$ is increased to 2 ("*The Originator Sequence Number in the RREQ message is the node's own sequence number, which is incremented prior to insertion in a RREQ.*" [19, Sect. 6.3]). Figure 1(b) also shows the nodes' routing tables. Each routing table entry (depicted as a 5-tuple) contains information about the destination, the destination's sequence number, the validity of the routing table entry, the hop count and the next hop (cf. Section 2.1). We do not show the sequence-number-status flag, the list of precursors in the routing table entries and the lifetime of a route as they constitute auxiliary information that is not critical to the loop example here. Due to the successful exchange of RREQ-RREP messages, nodes $D$ and $A$ create routing table entries to each other in their routing table.

**2. Self-Entries.** In Figure 1(c), we see node $S$ searching for a route to node $D$. In Figure 1(d), the link between nodes $S$ and $D$ goes up (e.g. due to node mobility) and node $S$ then searches for a route to node $X$. The route request message $\text{RREQ}_{S\to X}$ is forwarded by nodes $D$ and $A$ since they do not have any information on the destination node $X$. From the information contained in the $\text{RREQ}_{S\to X}$ message, a routing table entry to node $S$ is created in the routing tables of nodes $D$ and $A$. "*Then the node searches for a reverse route to the Originator IP Address [...] If need be, the route is created, or updated using the Originator Sequence Number from the RREQ in its routing table.*" [19, Sect. 6.5]. Node $S$ also receives the forwarded $\text{RREQ}_{S\to X}$ message from node $D$, and before silently discarding the message (since it is the originator of the RREQ message), updates its routing table to create an entry to node $D$. "*When a node receives a RREQ, it first creates or updates a route to the previous hop without a valid sequence number.*" [19, Sect. 6.5]. We use the value 0 for an unknown/invalid sequence number created in this manner, as is also done in implementations like AODV-UU [2], AODV-UIUC [15] and AODV-UCSB [4].

At some point, $\text{RREQ}_{S\to D}$ finally reaches node $A$ (Figure 1(e)). Since node $A$ has a valid routing table entry to node $D$, it generates an intermediate route reply message using the information from its routing table [19, Sect. 6.6 and 6.6.2]. The $\text{RREP}_{S\to D}$ message is unicast to node $D$, the next hop on the path towards node $S$. "*Once created, the RREP is unicast to the next hop toward the originator of the RREQ, as indicated by the routing table entry for that originator.*" [19, Sect. 6.6]. Node $D$ processes the $\text{RREP}_{S\to D}$ message, updates its routing table and forwards the message to node $S$, which establishes a route to node $D$. When updating its routing table, node $D$ creates a self-entry (following Interpretation 3a in Table 1) since the $\text{RREP}_{S\to D}$ message contains information about a route to node $D$ [19, Sect. 6.7].

**3. Standard RREQ-RREP Cycle.** In Figure 1(f), the link between nodes $S$ and $X$ goes up, while the link between nodes $D$ and $A$ goes down. Node $D$ also searches for a route to node $X$. Due to the successful exchange of RREQ-RREP messages, the routing tables of nodes $D$, $S$, and $X$ are updated accordingly.

**4. From Self-Entries to Loops.** The loop example continues with the link between nodes $S$ and $X$ going down (Figure 1(g)). In addition, node $D$ detects that its link to node $A$ is broken. Following from this, node $D$ initiates processing for a RERR message. "*A node initiates processing for a RERR message [...] if it detects a link break for the next hop of an active [(`val`)] route in its routing table while transmitting data*" [19, Sect. 6.11]. In this process, "*the node first makes a list of unreachable destinations consisting of the unreachable neighbor and any additional destinations [...] in the local routing table that use the unreachable neighbor as the next hop.*" [19, Sect. 6.11]. In addition, the routing table for node $D$ has to be updated for these unreachable destinations as follows: "*1. The destination sequence number of this route entry, if it exists and is valid, is incremented*

[9] Although this is not really a plausible reading of the RFC, Interpretation 4c can be seen as a compromise between Sections 6.11 and 6.1 of the RFC—it seems a natural way to avoid a decrement of destination sequence numbers and still take all information from the RERR message into account.

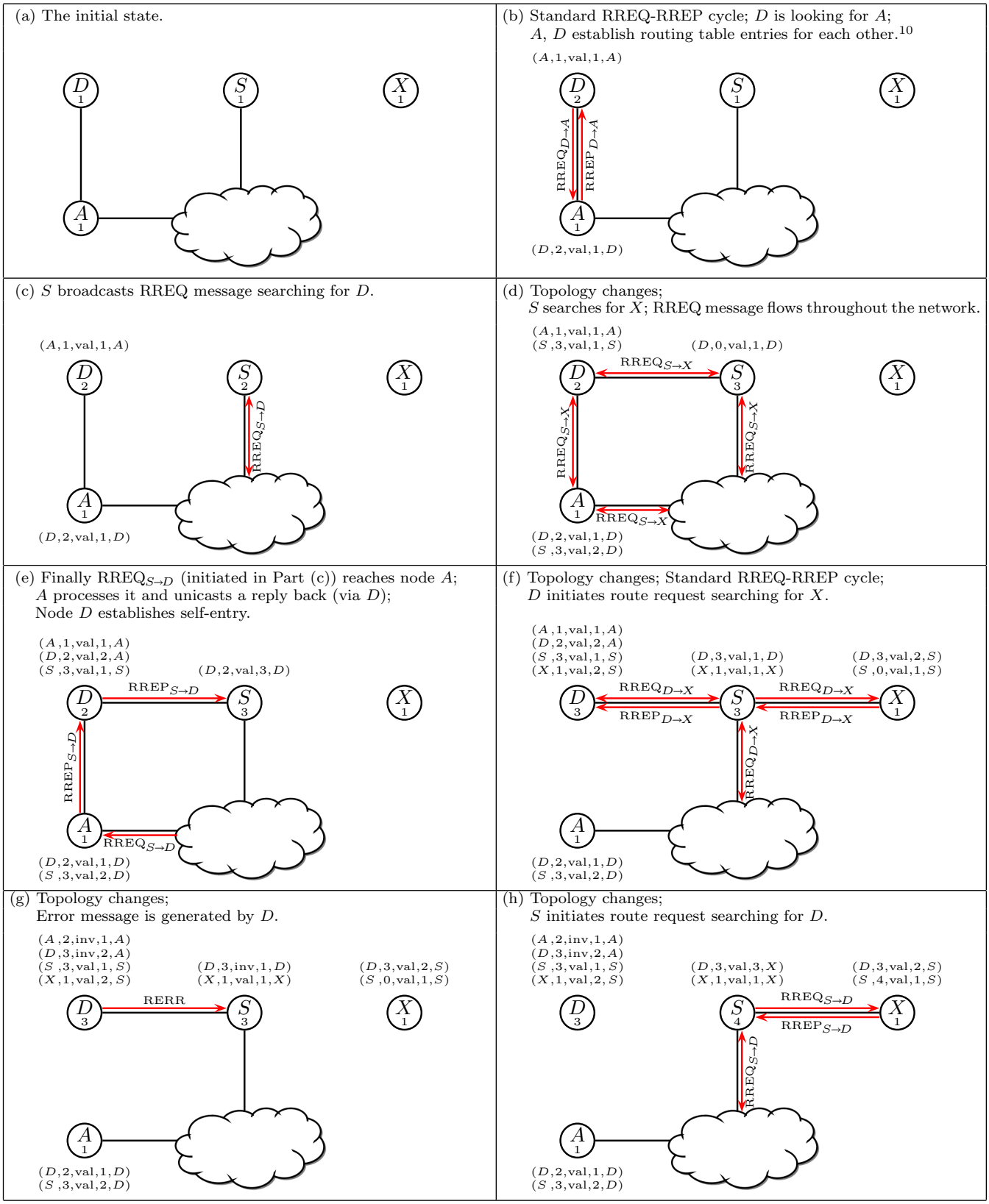


**Figure 1: Creating routing loops**

[10]Each routing table entry has the form (*destination*, *sequence number*, *validity*, *hop count*, *next hop*).

*[...]. 2. The entry is invalidated by marking the route entry as invalid*". [19, Sect. 6.11]. The result of this process is that the sequence numbers of routing table entries for the unreachable destination nodes $A$ and $D$ are increased, and the entries invalidated, as shown in the routing table of node $D$ in Figure 1(g).

The RERR message generated by node $D$ contains information about the unreachable destination nodes $A$ and $D$, taken from the routing table of node $D$. The message is sent to node $S$ since it is the precursor for the unreachable destination node $D$. Node $S$ receives the RERR message, and updates its routing table as follows: "*1. The destination sequence number of this routing entry [...] is copied from the incoming RERR. [...] 2. The entry is invalidated by marking the route entry as invalid*". [19, Sect. 6.11]. Therefore, the entry to node $D$ in node $S$'s routing table is updated to $(D, 3, \texttt{inv}, 1, D)$.[11]

In Figure 1(h), the link between nodes $S$ and $D$ goes down, while the link between nodes $S$ and $X$ goes up. Node $S$ also searches for a route to node $D$. Accordingly, the destination sequence number in the $\text{RREQ}_{S\to D}$ message is set to the value 3 since "*a previously valid route to the destination [...] is marked as invalid. [...] The Destination Sequence Number field in the RREQ message is the last known destination sequence number for this destination and is copied from the Destination Sequence Number field in the routing table.*" [19, Sect. 6.3]. The RREQ message is received by node $X$, which generates an intermediate RREP message since "*it has an active route to the destination, the destination sequence number in the node's existing route table entry for the destination is valid and greater than or equal to the Destination Sequence Number of the RREQ*" [19, Sect. 6.6]. Due to this, node $X$ unicasts a $\text{RREP}_{S\to D}$ message back to node $S$. Finally, node $S$ receives this message and updates its routing table, as shown in Figure 1(h).

A routing loop between nodes $S$ and $X$ for destination node $D$ is now established. When either of the nodes has a data packet to send to destination node $D$, the data packet will loop between the two nodes.

In this section, we have shown that AODV is not a priori loop free. The presented example can create loops if (i) sequence numbers are implemented in a way consistent with the RFC, (ii) self-entries are allowed, and (iii) destination sequence numbers are copied directly from RERR messages—even when this copying is only executed if it does not cause a decrement in the destination sequence number in the routing table. This shows that loop freedom hinges on non-evident assumptions to be made when interpreting the RFC and not only on monotonically increasing sequence numbers.

# 5. AODV IMPLEMENTATIONS

To show that our results are not only theoretically driven, but *do* occur in practice, we analyse five different open source implementations of AODV:

- *AODV-UU* [2] is an implementation of AODV, developed at Uppsala University.
  `http://aodvuu.sourceforge.net/`
- *Kernel-AODV* [1] is an implementation developed at NIST.
  `http://w3.antd.nist.gov/wctg/aodv_kernel/`
- *AODV-UIUC* [15] (Univ. of Illinois at Urbana-Champaign) is an implementation that is based on an early draft (version 10) of AODV.
  `http://sourceforge.net/projects/aslib/`
- *AODV-UCSB* [4] (Univ. of California, Santa Barbara) is another implementation that is based on an early draft (version 6) of AODV. `http://moment.cs.ucsb.edu/AODV/aodv-ucsb-0.1b.tar.gz`
- *AODV-ns2* is an AODV implementation in the ns2 network simulator, originally developed by the CMU Monarch project and improved upon later by S. Das and E. Belding-Royer (the authors of the AODV RFC [19]). It is based on an early draft (version 8) of AODV. It is frequently used by academic and industry researchers to simulate AODV.
  `http://ns2.sourcearchive.com/documentation/2.35~RC4-1/aodv_8cc-source.html`

Although these implementation behave differently, all of them *do* capture the main aspects of the AODV protocol, as specified in the RFC [19]. As we have shown in the previous sections, implementing the AODV protocol based on the RFC specification does not necessarily guarantee loop freedom: routing loops may occur when following either Interpretation 1a, Interpretation 2b or the combination of Interpretation 3a with any of 4a–c of Table 1. Therefore, we look at these five concrete AODV implementations to determine whether any of them is susceptible to routing loops. In particular, we examine the code of these implementations to see if routing loops can occur. Table 2 shows the results of this analysis: it indicates for each of the implementations which of the interpretations of Table 1 they follow and whether they can create routing loops or not.

We found that none of the five implementations makes use of the sequence-number-status flag, so a positive sequence number can never be marked as unknown. As a result, Ambiguity 1 of Table 1 does not arise.

In AODV-UU, self-entries are never created because a check is always performed on an incoming RREP message to make sure that the destination IP address is not the same as the node's own IP address. In terms of Table 1 it follows Interpretations 2c, 3c and 4a. It is shown in [8] that this interpretation of the RFC (avoiding self-entries) is loop free.

In Kernel-AODV, which follows Interpretations 2a, 3b and 4a of Table 1, an optimal self-entry (with hop count 0 and next hop being the node itself) is always maintained by every node in the network. The optimal self-entry is created during node initialisation. The node also maintains its own sequence number in this entry. Since the self-entry is already optimal, a node will never update the self-entry when processing any incoming RREP messages that contain information about itself. As such, a routing loop as described by the example in Section 4.1 will never occur.

AODV-UIUC follows the Interpretations 2b, 3a and 4a, whereas AODV-UCSB implements 2b, 3a and 4b. Due to Interpretation 3a, both implementations allow the occurrence of self-entries. These self-entries are not created during node initialisation, but generated based on information contained in received RREP messages.

The processing of RERR messages in AODV-UIUC and AODV-UCSB does not adhere to the RFC specification (or even the draft versions that these implementation are based upon). Due to this non-adherence, we are unable to recreate the routing loop example in Section 4.1. However, we note that if both AODV-UIUC and AODV-UCSB were

[11] Each of the Interpretations 4a, 4b and 4c in Table 1 will produce the same update in this scenario.

**Table 2: Analysis of AODV implementations**

| Implementation | Interpretation | Analysis |
|---|---|---|
| AODV-UU [2] | 2c, 3c, 4a | Loop free, since self-entries are explicitly excluded. |
| Kernel-AODV [1] | 2a, 3b, 4a | Loop free, due to optimal self-entries. |
| AODV-UIUC [15] | 2b, 3a, 4a | Yields loops, through decrement of sequence numbers, by use of Interpretation 2b. |
| AODV-UCSB [4] | 2b, 3a, 4b | Yields loops, through decrement of sequence numbers, by use of Interpretation 2b. |
| AODV-ns2 | 2a, 3a, 4b | Yields routing loops in the way described in Section 4, following plausible interpretations of the RFC w.r.t. Ambiguities 3 and 4. |

to strictly follow the RFC specification with respect to the RERR processing, loops would have been created.

Even though the routing loop example of Section 4.1 could not be recreated, both implementations allow a decrease of destination sequence numbers in routing table entries to occur, as a result of following Interpretation 2b. This contradicts the idea of monotonically increasing destination sequence numbers, and can give rise to routing loops in a straightforward way [8, Sect. 8.1].

In AODV-ns2, self-entries are allowed to occur in nodes and the processing of RERR messages follows the RFC specification. It follows Interpretations 2a, 3a and 4b. However in AODV-ns2, whenever a node generates a RREQ message, sequence numbers are incremented by two instead of by one as specified in the RFC. We have modified the code such that sequence numbers are incremented by one whenever a node generates a RREQ message, and are able to replicate the routing loop example of Section 4.1 in the ns2 simulator, with the results showing the existence of a routing loop between node $S$ and node $X$ in Figure 1(h). However, we note that even if the code remains unchanged and sequence numbers are incremented by two, AODV-ns2 can still yield loops; the example is very similar to the one presented and only varies in subtle details.

In sum, we discovered not only that three out of five AODV implementations can produce routing loops, but also that there are essential differences in various aspects of protocol behaviour. This is due to different interpretations of the RFC by the developers of the AODV implementations.

# 6. METHODOLOGY

In Section 3 we discussed several interpretations of the RFC. We also stated which of these interpretations are loop free and which are not. To show that there is a possible routing loop, a single example, like the one we have given in Section 4, is sufficient.

The statement that a particular interpretation is loop free is much harder to show. Looking at the specification of AODV and AODV-based protocols, loop freedom is claimed in the preamble; the justification is then given by a short informal statement. For example, AODV "*uses destination sequence numbers to ensure loop freedom at all times (even in the face of anomalous delivery of routing control messages), avoiding problems (such as "counting to infinity") associated with classical distance vector protocols.*" [19, Sect. 1]. As we have shown, these statements are not sufficient and not necessarily true. The only way to guarantee loop freedom for some interpretations and to analyse all (reasonable) interpretations in a systematic manner is by the use of formal modelling and analysis.

## 6.1 Formal Modelling and Analysis

Ideally, any specification is free of ambiguities and contradictions. Using English prose only, this is nearly impossible to achieve. Hence every specification should be equipped with a formal specification. The choice of an appropriate specification language is often secondary, although it has high impact on the analysis. The use of *any* formal language helps to avoid ambiguities and to precisely describe the intended behaviour. Examples for modelling languages are (i) the Alloy language, which is used to model Chord [24]; (ii) timed automata, which are the input language for the UPPAAL model checker, used by Chiyangwa, Kwiatkowska [5] and others [6] to reason about AODV; (iii) routing algebra as introduced by Griffin and Sobrinho [11] or (iv) AWN, a process algebra particularly tailored for (wireless mesh) routing protocols [7, 12].

The analysis yielding the results presented in this paper is based on a formal model using AWN. The reason why we chose this formal language is two-fold: on the one hand it is tailored for wireless protocols and therefore offers primitives such as **broadcast**; on the other hand, it defines the protocol in a pseudo-code that is easily readable by any network or software researcher/engineer. (The language itself is implementation independent). Table 3 presents the main primitives of AWN; the full specification of AODV in AWN can be found in [8].

**Table 3: AWN language (main primitives)**

| | |
|---|---|
| $X(exp_1, \ldots, exp_n)$ | process name with arguments |
| $P + Q$ | choice |
| $[\varphi]P$ | if statement: execute $P$ if condition $\varphi$ holds |
| $[\![\texttt{var} := exp]\!]P$ | assignment followed by $P$ |
| $\mathbf{broadcast}(ms).P$ | broadcast $ms$ followed by $P$ |
| $\mathbf{groupcast}(dests, ms).P$ | iterative unicast to all destinations $dests$ |
| $\mathbf{unicast}(dest, ms).P \blacktriangleright Q$ | unicast $ms$ to $dest$: if successful proceed with $P$; otherwise with $Q$ |
| $\mathbf{deliver}(data).P$ | deliver data to application layer |
| $\mathbf{receive}(msg).P$ | receive a message |
| $P \| Q$ | parallel composition of nodes |

Based on a formal specification one can now perform a careful analysis of the model to see if the model is consistent and if it satisfies properties such as loop freedom. Our analysis uses "classical" paper-and-pen verification techniques, and, in this end, guarantees the loop freedom of some interpretations of the AODV specification.

## 6.2 Using Formal Methods to Augment RFCs

Though we have not shown our proofs in the present paper—the purpose was to show that sequence numbers do not a priori guarantee loop freedom and that formal methods are needed—our analysis is based on a rigorous, formal and mathematical approach. As mentioned before, each interpretation of the RFC has been formalised in an unambiguous way.

We strongly believe that a "good" specification should consist of both a formal specification such as the one given in [12, 8] *and* an English description. The English text then serves the purpose of informing the reader about the main behaviour of the protocol and explains design decisions; the formal specification gives all the details, without allowing any ambiguities.

The IETF argues for the value of formal methods for specifying, analysing and verifying protocols.

> "*Formal languages are useful tools for specifying parts of protocols. However, as of today, there exists no well-known language that is able to capture the full syntax and semantics of reasonably rich IETF protocols.*" [IETF Web page[12]]

The quote is dated 1 Oct 2001; we believe that since then formal methods have advanced to such a state that they are now able to capture the full syntax and the full semantics of protocols and should be used for protocol specification and analysis.

# 7. RELATED WORK

Analysing and verifying routing protocols has a long tradition. Merlin and Segall [16] were amongst the first to use sequence numbers to guarantee loop freedom of a routing protocol. As we have pointed out, at least two proofs for AODV's loop freedom have been proposed [22, 26]. These proof attempts are discussed in Section 2.3. Besides, other researchers have used formal specification and analysis techniques to investigate the correctness of AODV; we point only at some examples.

A preliminary draft of AODV has been shown to be not loop free by Bhargavan et al. in [3]. Since then, AODV has changed to such a degree that the analysis of [3] has no bearing on the current version [19]. Furthermore, their loop had to do with timing issues (as is also the case in [10, 23]), whereas ours is time-independent. In the same paper they show the use of model checking on a draft of AODV, demonstrating the feasibility and value of automated verification of routing protocols. Using the model checker SPIN they were able to find/verify the routing loop in the preliminary draft. Musuvathi et al. [17] introduced the CMC model checker primarily to search for coding errors in implementations and used AODV as an example. Chiyangwa and Kwiatkowska [5] use the timing features of the model checker UPPAAL to study the relationship between the timing parameters and the performance of route discovery. A "time-free" model of AODV is exhaustively analysed by Fehnker et al. in [6], where variants of AODV, which yield performance improvements, are proposed and analysed. The presented model is based on the process algebra AWN introduced by the same authors in [7]; the complete model of AODV can be found in [8].

Next to the analysis of AODV with various formal methods, formal languages are also used for the specification and verification of other reasonably rich protocols. In [11], Griffin and Sobrinho use path algebras and algebraic routing to define a metarouting language that allows the definition of routing protocols. Their main interest lies in the combination of different algebras with applications in interdomain routing protocols such as BGP. Zave et al. provide abstractions for implementing the SIP protocol; the major elements of the language are presented in [25].

# 8. DISCUSSION & CONCLUSION

We have shown that, in contrast to common belief, sequence numbers do not guarantee loop freedom, even if they are increased monotonically over time and incremented whenever a new route request is generated. This was mainly achieved by analysing the Ad hoc On-demand Distance Vector (AODV) routing protocol. We have shown that AODV can yield routing loops.

Of course, one could argue that the given loop example does not occur in practice very often, since the sequence in which different requests have to be initiated and afterwards handled by different nodes might be really rare. However, it has been claimed that routing loops are avoided in *all possible scenarios* by the use of sequence numbers—this has been proven to be incorrect. Here, it does *not* matter how often this scenario occurs in real life. The fact that the example exists and can occur, makes the protocol flawed and disproves the fact that monotonically increasing sequence numbers are sufficient to guarantee loop freedom.

Next to that we have analysed several different interpretations of the AODV RFC. It turned out that several interpretations can yield unwanted behaviour such as routing loops. We also found that implementations of AODV behave differently in crucial aspects of protocol behaviour, although they all follow the lines of the RFC. This is often caused by ambiguities, contradictions or unspecified behaviour in the RFC. Of course a specification "*needs to be reasonably implementation independent*"[12] and can leave some decisions to the software engineer; however it is our belief that any specification should be clear and unambiguous enough to guarantee the same behaviour when given to different developers. As demonstrated, this is not the case for AODV, and likely not for many other RFCs provided by the IETF.

Our work confirms that RFCs written merely in a natural language contain ambiguities and contradictions. As a consequence, the various implementations depart in various ways from the RFC. Moreover, semi-informal reasoning is inadequate to ensure critical safety-properties like loop freedom. We believe that formal specification languages and analysis techniques—offering rigorous verification and analysis techniques—are now able to capture the full syntax and semantics of reasonably rich IETF protocols. These are an indispensable augmentation to natural language, both for specifying protocols such as AODV, AODVv2 and HWMP, and for verifying their essential properties.

*Acknowledgement.*

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

[12] `http://www.ietf.org/iesg/statement/pseudocode-guidelines.html`

# 9. REFERENCES


[1] Kernel AODV (ver. 2.2.2), NIST. `http://www.antd.nist.gov/wctg/aodv_kernel/` (accessed August 2, 2013).

[2] AODV-UU: An implementation of the AODV routing protocol (IETF RFC 3561). `http://sourceforge.net/projects/aodvuu/` (accessed August 2, 2013).

[3] K. Bhargavan, D. Obradovic, and C. Gunter. Formal verification of standards for distance vector routing protocols. *J. ACM*, 49(4):538–576, 2002.

[4] I. Chakeres and E. Belding-Royer. AODV routing protocol implementation design. In *24th International Conference on Distributed Computing Systems Workshops (WWAN'04)*, pages 698–703. IEEE Press, 2004.

[5] S. Chiyangwa and M. Kwiatkowska. A timing analysis of AODV. In *Formal Methods for Open Object-based Distributed Systems (FMOODS'05)*, volume 3535 of *LNCS*, pages 306–322. Springer, 2005.

[6] A. Fehnker, R. J. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. L. Tan. Automated analysis of AODV using UPPAAL. In C. Flanagan and B. König, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12)*, volume 7214 of *LNCS*, pages 173–187. Springer, 2012.

[7] 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, editor, *European Symposium on Programming (ESOP'12)*, volume 7211 of *LNCS*, pages 295–315. Springer, 2012.

[8] A. Fehnker, R. J. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. L. Tan. A process algebra for wireless mesh networks used for modelling, verifying and analysing AODV. Technical Report 5513, NICTA, 2013. `http://www.nicta.com.au/pub?doc=5513`.

[9] J. J. Garcia-Luna-Aceves. A unified approach to loop-free routing using distance vectors or link states. In *Symposium Proceedings on Communications, Architectures & Protocols (SIGCOMM '89)*, volume 19(4) of *ACM SIGCOMM Computer Communication Review*, pages 212–223. ACM Press, 1989.

[10] J. J. Garcia-Luna-Aceves and H. Rangarajan. A new framework for loop-free on-demand routing using destination sequence numbers. In *IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS)*, pages 426–435, 2004.

[11] T. Griffin and J. Sobrinho. Metarouting. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '05)*, volume 35(4) of *ACM SIGCOMM Computer Communication Review*, pages 1–12. ACM Press, 2005.

[12] P. Höfner, R. J. van Glabbeek, W. L. Tan, M. Portmann, A. McIver, and A. Fehnker. A rigorous analysis of AODV and its variants. In *Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM'12)*. ACM Press, 2012.

[13] C. Huitema, J. Postel, and S. Crocker. Not all RFCs are standards. RFC 1768 (Informational), 1995.

[14] IEEE P802.11s. IEEE draft standard for information technology—telecommunications and information exchange between systems—local and metropolitan area networks—specific requirements—part 11: Wireless LAN Medium Access Control (MAC) and physical layer (PHY) specifications-amendment 10: Mesh networking, July 2010.

[15] V. Kawadia, Y. Zhang, and B. Gupta. System services for ad-hoc routing: Architecture, implementation and experiences. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, MobiSys '03, pages 99–112. ACM, 2003.

[16] P. M. Merlin and A. Segall. A failsafe distributed routing protocol. *IEEE Transactions on Communications*, com-27(9), 1979.

[17] M. Musuvathi, D. Y. W. Park, A. Chou, D. R. Engler, and D. L. Dill. CMC: a pragmatic approach to model checking real code. In *Operating Systems Design and Implementation (OSDI'02)*, 2002.

[18] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[19] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing. RFC 3561 (Experimental), 2003. `http://www.ietf.org/rfc/rfc3561.txt`.

[20] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Conference on Communications, Architectures, Protocols & Applications (SIGCOMM '94)*, volume 24(4) of *ACM SIGCOMM Computer Communication Review*, pages 234–244. ACM Press, 1994.

[21] C. Perkins and I. Chakeres. Dynamic MANET on-demand (AODVv2) routing. Internet Draft (Standards Track), 2013. `http://tools.ietf.org/html/draft-ietf-manet-dymo-26`.

[22] C. Perkins and E. Royer. Ad-hoc On-Demand Distance Vector Routing. In *2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1999.

[23] H. Rangarajan and J. J. Garcia-Luna-Aceves. Making on-demand routing protocols based on destination sequence numbers robust. In *IEEE International Conference on Communications (ICC)*, volume 5, pages 3068–3072, 2005.

[24] P. Zave. Using lightweight modeling to understand CHORD. *SIGCOMM Comput. Commun. Rev.*, 42(2):49–57, 2012.

[25] P. Zave, E. Cheung, G. W. Bond, and T. M. Smith. Abstractions for programming SIP back-to-back user agents. In *International Conference on Principles, Systems and Applications of IP Telecommunications*, IPTComm '09, pages 11:1–11:12. ACM Press, 2009.

[26] M. Zhou, H. Yang, X. Zhang, and J. Wang. The proof of AODV loop freedom. In *Wireless Communications & Signal Processing (WCSP09)*, pages 1–5. IEEE Press, 2009.