Modelling, Verification, and Comparative Performance Analysis of the B.A.T.M.A.N. Protocol

Kaylash Chaudhary
School of Computing, Information, and Mathematical Sciences
University of the South Pacific

Ansgar Fehnker
Formal Methods and Tools
University Twente

Vinay Mehta
School of Computing, Information, and Mathematical Sciences
University of the South Pacific

This paper considers on a network routing protocol known as Better Approach to Mobile Ad hoc Networks (B.A.T.M.A.N.). The protocol serves two aims: first, to discover all bidirectional links, and second, to identify the best-next-hop for every other node in the network. A key element is that each node will flood the network at regular intervals with so-called originator messages.

This paper describes in detail a formalisation of the B.A.T.M.A.N. protocol. This exercise revealed several ambiguities and inconsistencies in the RFC. We developed two models. The first implements, if possible, a literal reading of the RFC, while the second model tries to be closer to the underlying concepts. The alternative model is in some places less restrictive, and rebroadcasts more often when it helps route discovery, and will on the other hand drop more messages that might interfere with the process.

We verify for a basic untimed model that both interpretations ensure loop-freedom, bidirectional link discovery, and route-discovery. We use simulation of a timed model to compare the performance and found that both models are comparable when it comes to the time and number of messages needed for discovering routes. However, the alternative model identifies a significantly lower number of suboptimal routes, and thus improves on the literal interpretation of the RFC.

1 Introduction

Since the introduction of wireless mobile ad hoc networks, many protocols have been used for making communication efficient and by finding the best possible routes from source to destination. The German "Freifunk" community developed the network routing protocol known as Better Approach to Mobile Adhoc Network B.A.T.M.A.N. as an alternative to OLSR. B.A.T.M.A.N. is a proactive protocol, for detecting all bidirectional links and identifying the best-next-hop for all other nodes. The protocol is table driven and maintains route information and updates by keeping information on received originator messages in a so-called *sliding window*. A node records how many messages from another node with a sequence number in a given range of recent sequence numbers were received via which neighbor.

This paper describes a model based on the RFC [12]. During the implementation of the protocol as an Uppaal model it turned out that a number of conditions were not unambiguously defined in the context of looping sequence number and a limited local view of a node. The formalisation will define those terms. The RFC also contained a few conditions that were inconsistent. To address this we developed two models: The literal interpretation of the RFC implements the protocol with a reading that is as close as possible to the text of the RFC. The alternative interpretation resolves inconsistencies in light of the underlying concepts of B.A.T.M.A.N.. For example, a literal reading of the description will exclude certain messages from being recorded in the sliding window, while the general description of the sliding window suggests that those messages should be recorded.

H. Hermanns, P. Höfner (Eds.): Models for
Formal Analysis of Real Systems (MARS 2017)
EPTCS 244, 2017, pp. 53–65, doi:10.4204/EPTCS.244.3

For both interpretations we verify a few basic properties for an untimed model and a small four node topology. This is mostly meant to help with debugging the models, and ensure e.g. that the bidirectional link check works as expected. This analysis show that both, the literal and alternative model, meet this minimum standard. We then use a timed model in a 17 node network to analyse the performance, in particular the quality of the chosen "best-next-hops". These are sometimes suboptimal and actually not the best next hop. This analysis shows that the alternative model reduces the occurrences of suboptimal "best-next-hops".

The performance of B.A.T.M.A.N. in comparison to OLSR has been analysed in a real environment in [10, 14]. A simulation of a real environment has been used to analyse B.A.T.M.A.N. in [7, 11]. Furlan simulated the performance of B.A.T.M.A.N. for a select type of network topologies [6]. These studies consider a particular implementation with specific hardware in a specific environment and do not analyse the routing algorithm in isolation. A formal analysis of B.A.T.M.A.N. was conducted by Cigno and Furlan in [2]. This analysis discovered that routing loops are possible, and proposed improvements that ensure loop freedom. This work was complemented by simulation studies and real measurements. While this work does not use model checking per se, it presents a formal model to study routing loops. In comparison, this paper presents a formal Uppaal model, and uses verification and simulation to study the quality of route discovery. Uppaal has been used perviously to study other protocols, such as LUNAR, OLSR, AODV, DYMO, and LMAC [1, 4, 5, 9, 15]

This paper is organised as follows. The next section describes the B.A.T.M.A.N. routing protocol and in particular the sliding window. Section 3 introduces the common data structures, Section 4 and 5 the handling and processing of OGMs. These section give rise to the two alterative models. Section 6 and 7 describe the templates and the verification and simulation results, respectively.

2 The B.A.T.M.A.N. protocol

The major objective of a network routing protocol is to discover routes available in a network. An active user group known as the German "Freifunk" community has been involved in the development of a network routing protocol known as Better Approach to Mobile Adhoc Network [12]. The RFC abbreviates the protocol as B.A.T.M.A.N., including the dots. The protocol is meant as an alternative to OLSR (Optimized Link State Routing Protocol), a proactive wireless routing protocol widely used in mobile ad hoc networks [3]. Instead of having each node compute routing tables that capture the complete network topology as OLSR does, B.A.T.M.A.N. nodes maintain only information on the neighboring nodes. This information is used to identify the best-next-hop [8]. It is then a desired property of B.A.T.M.A.N. that the best-next-hop will indeed realize the best route.

To maintain a fresh list of available neighbours, and to deal with changing topologies, nodes will send *originator messages* (OGMs) at regular intervals. OGMs are forwarded under certain circumstances to neighbouring nodes, first to correctly identify all available bidirectional links. Once these have been established a node selects a neighbour as best-next-hop to a destination, based on the number of OGMs from that destination that have passed through that neighbor within a given time frame, called the *sliding window*. The protocol drives on OGMs flooding the network and is phrased in terms of the origin of the OGMs. Once routes have established along bidirectional links, these originators will be the destination for packets to be send.

The phases of finding bidirectional links and finding the best-next-hop are however not clearly divided. They may overlap for different nodes, and nodes will iterate through these phases continually, as the sliding window moves, and bidirectional links expire.

Terminology

The following terminology is used by the B.A.T.M.A.N. routing protocol[12].

- **OGM**. *Originator messages*. A message send periodically by a node to announce the existence of a node. Each OGM contains the following information:
  - The *OID* (Originator ID). Once an OGM is created, the OID will remain unchanged, even when it is rebroadcasted.
  - The *SID* (Sender ID). A node that rebroadcasts an OGM will set the SID to its own ID.
  - A *sequence number*. According to the RFC it has a range 0 to $2^{16}-1$. The sequence number will be incremented if a node generates a new OGM. Once it exceeds the upper bound, it will start at 0 again.
  - The TTL (Time-to-live). The TTL decremented each time an OGM is rebroadcasted by another node. Once it reaches the value 1, the OGM will be dropped. The TTL is also a measure for the distance travelled.
  - A *uni-directional link flag*. Used to indicate that an OGM was received from a link that is presumed to be uni-directional.
  - A *direct-link-flag*. Used when a node rebroadcasts an OGM where the sender was also the originator.
- **Routing Table** Each node maintains a routing table with the following information:
  - **Bidirectional Sequence Number** The sequence number of the last self-originating OGM that was rebroadcasted by a neighbor.
  - **Sliding Window** The *sliding window* specifies for each originator a range of recent sequence numbers. OGMs with sequence number in this range will be recorded. A neighbour which rebroadcasted the most OGMs for an originator within the sliding window is considered the *best-next-hop* for that originator.
  - For each originator the **last TTL** and the **last sequence number**.

Example

Figure 1 gives a example routing table for node A in for a small network with four nodes. The bidirectional sequence number for originator B is in this example is 14, and the last TTL is 8. The sliding window for originator B encompasses sequence numbers 13 to 1. Within the window, A received an OGM from originator B, three times from Node B, twice from Node C and twice from Node D. This means that Node B is currently the best-next-hop for Node B.

Similarly we have for originator C – according to this example – that both Node B and Node C have broadcasted three times an OGM from C within the window, and Node D only once. This means that both B and C are considered to be the best-next-hop for originator C.

Suppose that Node A will receives an OGM rebroadcasted by Node B, with OID A, sequence number 2, and the direct-link-flag set to true. In this case the OGM is an echo of an OGM node A sent itself. From this follows that there exists a bidirectional link. The bidirectional link sequence number (BD SQN) for OID B will be changed to 2.

Suppose further that A receives an OGM, rebroadcasted by Node D, with OID C, and sequence number 12. The range of the sliding window for C was sequence number 6 to 10. After reception, the range of the sliding window will be 8 to 12. Data of sequence numbers 6 and 7 will be dropped, and

Routing Table for Node A - BEFORE

OID	BD SQN	TTL	SLIDING WINDOW					
B	14	8	SID↓ / SQN→	13	14	15	0	1
			B		1		1	1
			C		1	1		
			D	1		1		
C	3	7	SID↓ / SQN→	6	7	8	9	10
			B			1	1	1
			C	1	1	1		
			D				1	
D	9	8	SID↓ / SQN→	2	3	4	5	6
			B	1	1	1	1	
			C			1	1	1
			D		1	1		

Routing Table for Node A - AFTER

OID	BD SQN	TTL	SLIDING WINDOW					
B	2	8	SID↓ / SQN→	13	14	15	0	1
			B		1		1	1
			C		1	1		
			D	1		1		
C	3	6	SID↓ / SQN→	8	9	10	11	12
			B	1	1	1		
			C	1				
			D		1			1
D	9	8	SID↓ / SQN→	2	3	4	5	6
			B	1	1	1	1	
			C			1	1	1
			D		1	1		

Figure 1: Routing table for A before and after processing two OGMs from originators B and C.

D will be recorded as having sent an OGM with sequence number 12. After this update node B will have rebroadcasted 3 out the last 5 OGMs, Node C one, and Node D two. Node B will be the presumed best-next-hop for originator C. See Figure 1 for the new routing table after these updates.

3 Data Structures and Constants

This section describes the common data structures and data structures used by of the different Uppaal model. The subsequent section will describe the processes and conditions for handling OGMs.

Constant are the number of nodes `N`, the maximum sequence number `MAX_SQN`, the initial TTL `TTL_MAX`, and the size of the sliding window `WINDOW_SIZE`. We considered models with up to 25 nodes, a maximum sequence number of 15, an initial TTL of up to 10, and a window size between 2 and 8. The maximum sequence number of 15 is a lot smaller than the recommended value of $2^{16}-1(=65535)$. The reason is that we want to analyse the behaviour of the sliding window, including the behaviour when sequence numbers loop.

Datatypes have been defined for sequence numbers, node IDs, TTLs, and indices into arrays of the

Table 1 Datatype definitions for OGMs and routing table entries.

```
typedef struct {
   anyIdT oid;
   anyIdT sid;
   ttlT ttl;
   sqnT sqn;
   bool isDirect;
   bool isUnidirectional;
} OGM;
```

```
typedef struct {
   bool entries[WINDOW_SIZE];
} SlidingWindow;

typedef struct {
   sqnT biDirectionalSqn;
   sqnT sqn;
   ttlT ttl;
   SlidingWindow window[N];
} OriginatorInfo;
```

sliding window. Node IDs take values from 0 to $N-1$, and we reserve the node ID -1 for an invalid ID; it is used as padding where we had to use fixed sized arrays to implement variable sized buffers.

The datatype `OGM` in Table 1 is used for the OGMs. Nodes communicate by writing an OGM to a shared global variable `ogmglobal`, and synchronizing on a channel from array `broadcast chan SendOGM[N]`. Each node maintains a buffer of received, but not yet processed OGMs, `ogmlocal`, which is implemented as a fixed array of type `OGM`. The next OGM to be processed is `ogmlocal[0]`. The size of this buffer is chosen such to avoid buffer overflow – we use values up to 64. A node increments counter `buffererror` if a buffer overflow occurs.

Each node maintains an array `table` of size `N` of type `OriginatorInfo`. Each entry contains the bidirectional sequence number `biDirectionalSqn`, i.e. the last sequence number received from the originator `sqn`, the time-to-live `ttl`, and the sliding window `window`. The latter is an array with an boolean array of size `WINDOW_SIZE` for each node in the network. The last element in this array, refers to sequence number `sqn`, the second last element to `sqn-1`, etc. The model includes a number of functions to update the window for a given `oid` consistently, e.g. to move the entries, when the sequence increases. Of course this takes into account that the sequence number may loop.

4 Handling of Originator Messages

This section gives a detailed account of how the model relates to the B.A.T.M.A.N. RFC. A key to model the B.A.T.M.A.N. protocol is to faithfully represent Section 5.2 of the RFC. It describes how to classify different types of OGMs. This paper uses the numbering scheme from the RFC to refer to these rules.

Rule 5.2.1 This rule states to drop OGMs sent with a different version of the protocol. It is not included in the model, since we assume that all nodes run the same version.
Rule 5.2.2 This rule states to drop OGMs a node receives from its own transmitter. Since the automaton modelling a node cannot synchronize with itself, this scenario is excluded by the semantics of synchronisation in Uppaal. Note, this rule has nothing to do with dropping messages that are a rebroadcast of a message that has been sent earlier by this node. That is dealt with elsewhere.
Rule 5.2.3 This rule deals with nodes that have multiple transmission interfaces. It is not included, since the model only considers nodes with a single interface.
Rule 5.2.4 This rule deals with the bidirectional-link check. It is modeled by boolean function `bool`

`meetsRule5_2_4(OGM ogm)`. It returns true if `ogm.oid` is equals to the nodes own ID. In this case the bidirectional link information may have to be updated, as described in the next section.

Rule 5.2.5 This rule deals also with the bidirectional-link check. The OGM is dropped if the unidirectional-link flag `ogm.isUnidirectional` is set. This rule is modelled by boolean function `bool meetsRule5_2_5(OGM ogm)`.

Rule 5.2.6 This rule determines which OGMs should be used to update the sliding window. This update is referred to as *Process 5.4*, and will be explained later. *Rule 5.2.6* states to invoke *Process 5.4* if the OGM is received via a bidirectional link and contains, as the RFC state a "New Sequence Number (is NOT a duplicate)".

This apparently simple statement requires some interpretation, since the RFC does not define how to determine whether a sequence number is new. The model has to assume a local view on the network, and cannot refer to a global notion of "duplicate" or "new". We also have to take into account that sequence numbers may loop, and that OGMs may not arrive in the order they have been created.

Given an OGM from originator `ogm.oid` and sequence number `ogm.sqn`, and the most recently recorded sequence number `table[oid].sqn`. We modeled "new sequence number" to mean that the value of `ogm.sqn` is between the `table[ogm.oid].sqn` and `table[ogm.oid].sqn` plus half of the range of the sequence numbers, modulo the range.

The comment "NOT a duplicate" in brackets, might seem like a clarification, but complicates things further. Being not a duplicate is not necessarily the same as having a new sequence number. Within the context of the sliding window, a message with a sequence number in the range of the sliding window is not a duplicate, if for the same `ogm.oid` no OGM from the same sender `ogm.sid` with same sequence number `ogm.sqn` was received. These OGMs are not duplicates, but also not newer.

Recall that this rule is the precondition for *Process 5.4*. This process however clearly states that it should only use OGMs with a newer sequence number. It does not mention duplicates.

We decided to create two different models that explore the different alternatives. The first model applies a *literal interpretation*, i.e. *Rule 5.2.6* only applies if an OGM has a newer sequence number. This model implements the condition that was explicitly stated in the description of *Process 5.4*.

This interpretation, however seems to undermine the purpose of the sliding window, namely to collect information on OGMs with sequence numbers in the range of the sliding window. The *alternative interpretation* of *Rules 5.2.6* is to consider OEMs with a newer sequence number, or OGMs that are in range but not duplicates.

This rule is modelled by boolean function `bool meetsRule5_2_5(OGM ogm)`.

Rule 5.2.7 This rules deals with which OGMs will be rebroadcasted. Before rebroadcast it has to execute *Process 5.5* which will be described later. The first set of OGMs that have to be rebroadcasted, are those that have been received from a single hop neighbor, i.e. for which `ogm.oid` equals `ogm.sid`. This includes, but not exclusively, the OGMs that take part in the bidirectional link check.

In addition OGMs that satisfy the following will be rebroadcasted according to the RFC:

> *The OGM was received via a Bidirectional link AND via the Best Link AND is either not a duplicate or has the same TTL as the last packet which was not a duplicate (last TTL).*

The "bidirectional link" is the link between the receiving node and the sender of the OGM, `ogm.sid`. The specific condition for whether a node considers a link to be bidirectional will be discussed later.

To be the "Best Link" means that the sender `ogm.sid` is a best-next-hop for originator `ogm.oid`. This decision is based on the sliding window. We will discuss later how to determine whether a node is a best-next-hop.

To determine whether an OGM is a duplicate we use the same interpretation as for *Rule 5.2.6*. The sequence number is newer, or in range, and no OGM with the same `ogm.sid`, `ogm.oid`, and `ogm.sqn` has been received previously.

The last TTL is the value of `table[oid].ttl`. The short summary here does not quite align with the description later in the RFC that describes *Process 5.5*. We will discuss this in the next section. Also, while it is not explicitly mentioned in the RFC, the condition on the TTL can only be reasonably applied if the sequence number of OGM is in range. OGMs that are neither newer, nor in range should be dropped regardless of the TTL.

Note furthermore that this only tests for equality of the TTL of the OGM, with the last TTL. It disregards OGMs with an improved TTL, i.e. OGMs that arrived via a shorter route. This seems to contradict the purpose of the sliding window, namely to collect information to determine the best hops, and thus the best route. In contrast, the condition that the OGM should not be a duplicate does not consider the TTL at all. It means that OGMs will be rebroadcasted that are not a duplicates but have lower TTLs, i.e. OGMs that arrived via longer route.

Also here we decided to consider the alternatives. Both interpretations require that the link to the sender is bidirectional and received via a best-next-hop. The literal interpretation will require in addition that the sequence number is newer, or that the OGM is in range but not a duplicate, or in range and that its TTL equals the last TTL.

The alternative interpretation requires that OGM is newer, or that the OGM is in range **and** not a duplicate **and** also has TTL that is equal or better than the last TTL.

Finally, recall that *Rule 5.2.7* is a precondition for *Process 5.5*. The description of this process requires explicitly that the TTL is at least 2, and to drop it otherwise. We include this in *Rule 5.2.7*.

This complete rule is modelled by boolean function `bool meetsRule5_2_7(OGM ogm)`.

It should be noted that these rules, in particular *Rule 5.2.6* and *Rule 5.2.7* are not mutually exclusive. Either of those rules may apply to an OGM or both. The RFC does not specify explicitly what to if the latter is the case. To ensure correct computation of the best-next-hop it is necessary that OGMs that satisfy both rules, are used to update the sliding window, and are rebroadcasted.

It should also be noted that these rules are not complete; there can be OGMs to which none of these rules apply. The RFC does not mention explicitly what to do; it appears that these OGMs have to be silently dropped.

5 Processing of Originator Messages

The RFC describes three procedures to process OGMs, one for the bidirectional-link-check, one to update the neighbor ranking, and the last one to update an OGM before rebroadcast.

Process 5.3: Bidirectional Link Check This process checks if an OGM is a recent self-originating message, i.e. if `ogm.sid` is it own ID, the `ogm.isDirect` flag set, and the sequence number `ogm.sqn` equal to its current number `sqnNode`. If so it stores `ogm.sqn` in `table[oid].biDirectionalSqn`.

The description of *Process 5.3* also defines when a link is considered bidirectional: if the difference between the current sequence number `sqnNode` and the stored `table[oid].biDirectionalSqn` is less than some threshold `BI_LINK_TIMEOUT`. Unfortunately, the RFC does not specify values for `BI_LINK_TIMEOUT`; for the model we use values between 2 and 10 (half the size of the sliding window, and up to twice its size).

Process 5.4: Neighbor Ranking This process updates the sliding window. The update adds a new entry for sequence number `ogm.sqn` and originator `ogm.oid`. If the sequence number `ogm.sqn` is newer

than the last recorded sequence number `table[oid].sqn`, the sliding window will be shifted before that update. This means that the oldest entries will be deleted and new entries up to sequence number `ogm.sqn` added.

Note, that this process also defines a condition for updating the sliding window, even though *Rule 5.2.6* serves the same purpose. However, these rules are not the same. Here, the precondition is that an OGM has have a newer sequence number. As mentioned before, the literal model uses this condition as *Rule 5.2.6*. The *alternative model* relies on the alternative interpretation of *Rule 5.2.6*.

Note furthermore, that the RFC mentions that the *packet count* must be updated. However, a *packet count* has not been introduced nor used elsewhere in the RFC. We assume that it is used as synonym for the neighbor ranking based on the sliding window.

Finally, the RFC states that each node has to nominate one node from the set of nodes with the most OGMs in range to be the unique best-next-hop. This is how we implemented it for the literal model. In the alternative we leave this choice open; if two or more nodes happen to have the same ranking, they will all be considered a next-best-hop.

Process 5.5: Rebroadcasting An OGM has to be rebroadcasted when it satisfies *Rule 5.2.7*. Before rebroadcast the following fields of the OGM will be changed:

- The TTL `ogm.ttl` has to be decremented by one.
- If the sender is equal to the originator, i.e. if `ogm.sid` equals `ogm.oid`, then the `isDirect` flag will be set to true. This allows the originator - a direct neighbor - to distinguish between OGMs that were received directly from that neighbor, or via a detour. Note, that the RFC also mentions as requirement that the OGM will sent the OGM back to the sender/originator. Since our model uses a only broadcast, this is satisfied by default.
- The `isUniDirectional` flag has to be set to true, if the link to the sender `ogm.sid` does not satisfy the bidirectional-link requirement.

Note furthermore, that the RFC mentions that OGMs with a TTL of 1 (or 0 after decrementing) have to be dropped. In the model we incorporated this in Rule 5.2.7, which effectively drops the message.

6 Untimed Model

The untimed model serves to assure that the bidirectional link check works, and that routes are discovered, and whether routes are loop-free for a static topology. It uses the same template for each node `id`.

The template has only two control locations. The first, is a committed initial location, the second, labelled `Processing`, models normal processing of OGMs. The edge from the initial location to location `processing` initialises the sliding windows and other local variables.

From location `Processing` there are 8 self loops. They model sending and receiving of OGMs, and the processing of the OGMs:

- An edge that synchronizes on channel `SendOGM[oid]!`. This transaction creates an OGM, and copies it to the shared global variable `ogmGlobal`.
- An edge that synchronizes on channel `SendOGM[sid]?`, where `sid` is a valid sender ID. The guard `topology[sid]` ensures that sender and receiver are connected. This transaction calls a function `receiveOGM()`, which copies the OGM from global variable `ogmGlobal`, and appends it to the local buffer `ogmLocal` unless it satisfies *Rule 5.2.5*.

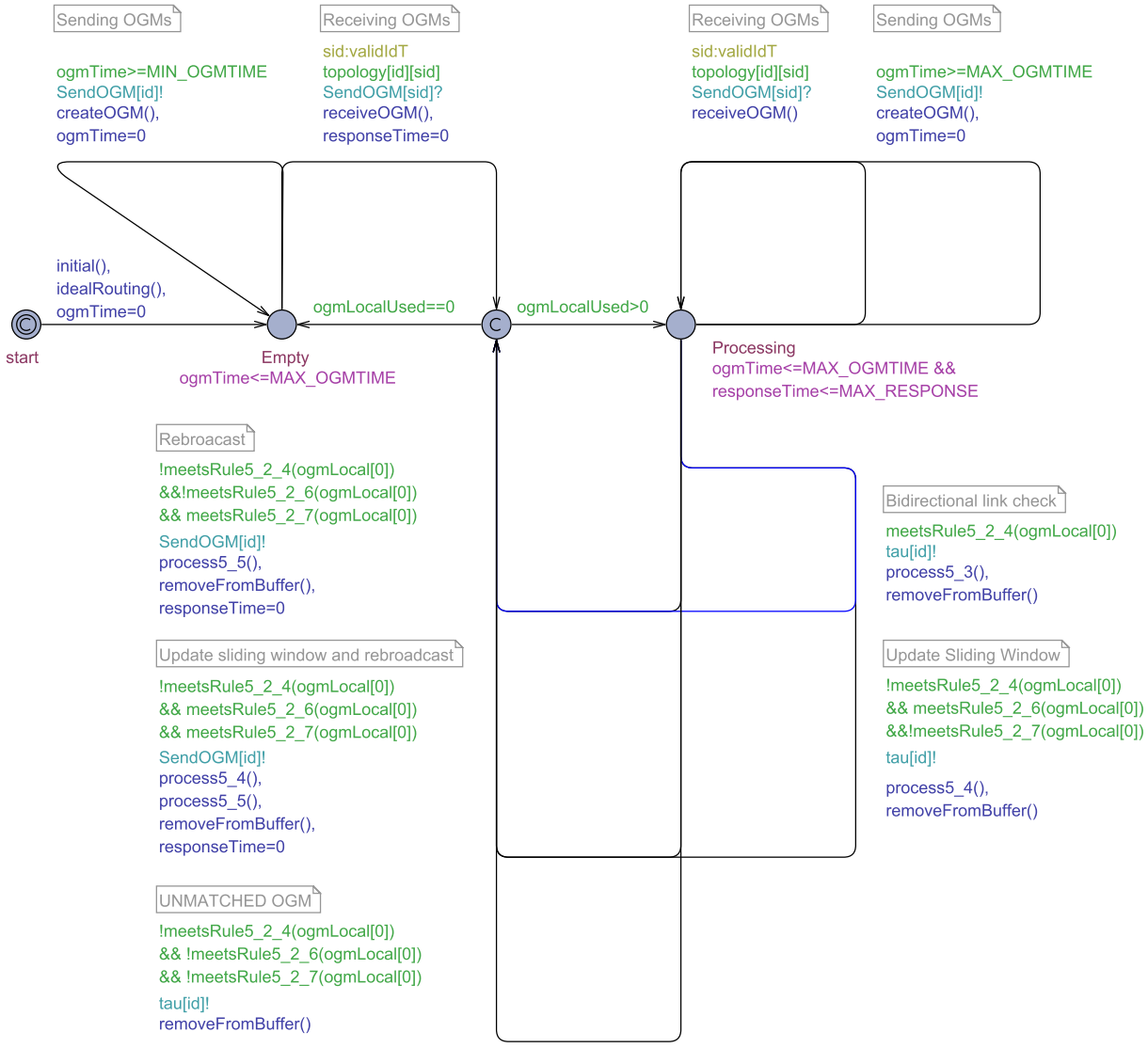

Figure 2: Timed model for simulation

- Five edges that model the processing of OGMs in the buffer. One for when *Rule 5.2.4* applies, one for when *Rule 5.2.6* applies but not *Rule 5.2.7*, one for when *Rule 5.2.7* applies but not *Rule 5.2.6*, one for when both apply, and one for when the buffer is not empty, and neither rule applies. If Rule 5.2.7 applies, the edge will synchronize on channel `SendOGM[oid]!`, i.e. it forwards an OGM. All others are labelled with broadcast channel `tau[id]`. These edges model a local update of the routing information. The label allows use to give those transition a higher priority – to reduce the number of states – while it will not affect the routes that will be discovered.

The model contains a few auxiliary functions to add and remove OGMs to the buffer, to manage the sliding window, and to process the OGMs. It also includes a number of function that are used to specify properties, e.g. `bool hasLoop()` which is used to check for loop freedom, or function `int countBidirectionalMisses()` which counts how many bidirectional links still need to be discovered.

Verification Results For the verification we limit ourself to a model with four nodes in a ring. Each nodes sends one OGM, and node 0 a second one. For this we check the following:

- `A[] !hasloop()`. Whether the routes have loops.
- `A[] (forall(id:validIdT) Node(id).OGMremain==0 && sumBuffer() ==0 ) imply countBidirectionalMisses() == 0`
  This checks that once all OGMs have been sent and processed each node other than 0 have discovered a route all bidirectional links have been discovered (no misses).
- `A[] ((forall(id:validIdT) Node(id).OGMremain==0&& sumBuffer() ==0 ) imply (forall(id:validIdT) (id==0 || Node(id).table[0].bestNext>-1))`
  This checks that once all OGMs have been sent and processed, each node other than 0 have discovered a route to node 0.

The model satisfies all of these. It should be noted that we show absence of loops for a small static network with only 4 nodes. In general, sequence number based protocols have been shown to not to be loop free [13], and [2] presents a specific scenario for B.A.T.M.A.N. with a changing topology.

The literal and alternative models behave are except for the implementation of the functions implementing *Rule 5.2.4* to *Rule 5.2.7* and *Processes 5.3* to *Process 5.5*, as discussed previously. There is only one notable difference in the data structure for the routing table. The literal model also includes a field `table[oid].nextBest`, since the literal interpretation requires a unique best-next hop for each originator.

Verification was performed on an Intel i5-5200 CPU 2.2Ghz processor with 8 GB RAM running Uppaal 4.1.19. Verification of these properties took 19 seconds for either model. Verifying these same properties for a 5 node network, with one extra central node, runs out of memory after about 600 seconds.

7 Timed Models

The timed model uses simulation to analyse and compare the performance of the two versions of the protocol; in particular how quickly the bi-directional link check occurs, how quickly routes are discovered, and the quality of the routes.

The timed models uses the same data-structures and functions as the untimed model, but they attach time bounds on when events can occur. The model uses clock `ogmTime` to ensure that any node sends an OGM once between `MIN_OGMTIME` and `MAX_OGMTIME`. The transition that model creation and sending of new OGMs includes guard `ogmTime>=MIN_OGMTIME`, and all control location invariant `ogmTime<=MAX_OGMTIME`.

The model uses clock `responseTime` to ensure that a node rebroadcasts within `MAX_RESPONSE` time units, while there is an OGM in the buffer. The model includes two control locations, `Empty` for when the buffer is empty, `Processing` for when the node processes OGMs. In the latter location there is the additional invariant `responseTime <=MAX_RESPONSE`.

All other transitions model internal updates and are labelled with urgent broadcast channel `tau[id]!`. This means these transition are taken as soon as they are enabled. See Figure 2 for a depiction of node template.

The model also includes a number of auxiliary functions to support analysis of the performance by simulation, such as function `countRouteMismatch()` which counts how many “best-next-hops” are actually suboptimal.

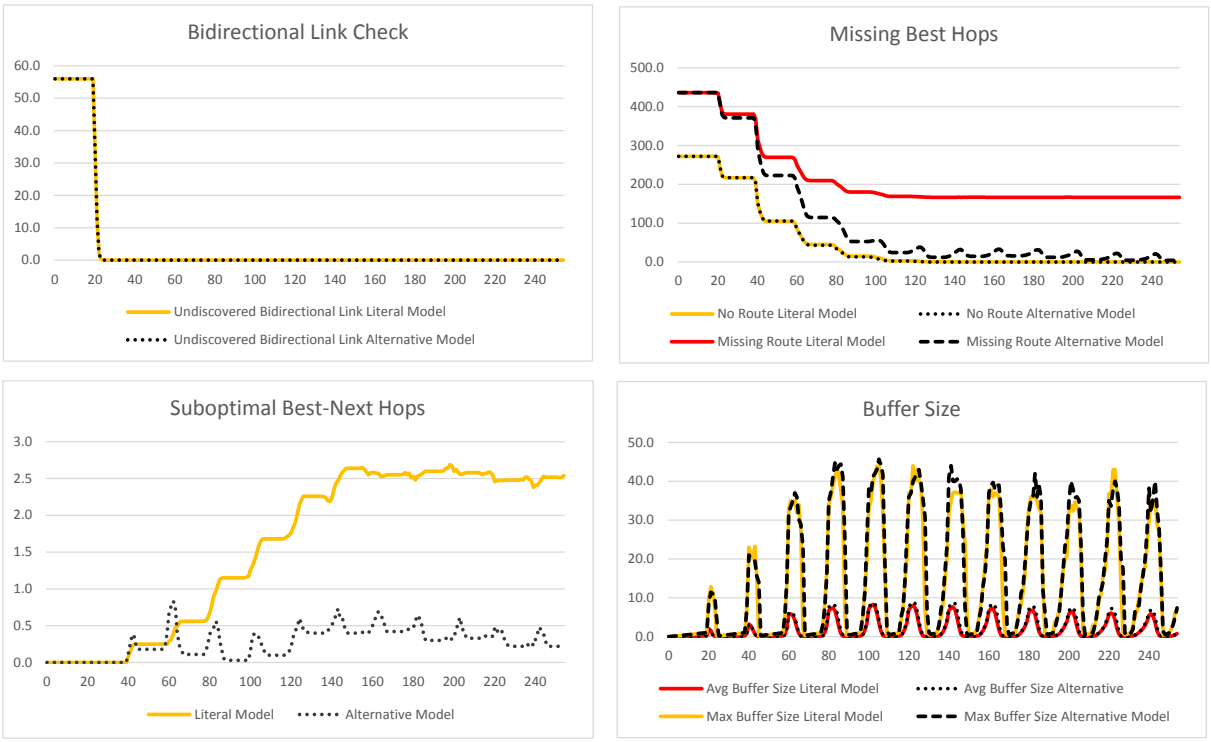

Figure 3: Simulation results. The results are averages over 100 runs, except for the maximum buffer size, which is the maximum over 100 runs.

Simulation Results. For the simulation we consider a 4 by 4 grid with 16 nodes, plus one additional node in the center. The one additional node is connected to the four central nodes of the grid. The purpose of the central node is to introduce irregularity to the grid. The model furthermore assumes that processing of OGMs takes at most 1 time unit. New OGMs are created once between 19 and 20 time units. The buffer for each node has maximum size 64. For these setting we ran a simulation of 100 runs up to time 255, which means it consider 12 rounds of OGMs being sent and processed. The results are depicted in Figure 3. Simulation of one run takes about 320 seconds for the literal model, and 330 second for the alternative model. The presented results are for 100 simulations each.

The number of undetected bidirectional links drops to zero after the first round of OGMs have been sent. There is no significant difference between the literal and alternative model, as the changes do not affect the bidirectional link check.

The results for the number of routes for which no “Best-next-hop” declines within 6 rounds to zero. This corresponds to the 6 hops of longest route in a 4 by 4 grid; from the one corner to the diagonally opposite corner. There is no significant difference between the two model in this respect.

There is a noticeable with respect to the number of potential best-next-hops. The literal model selects a designated best-next-hop. This has consequences further on, since a node only forwards OGMs that were received via a best-next-hop. The literal model will rebroadcast only OGM received via the designated best-next-hop. The alternative will rebroadcast OGMs received by any best-next-hop.

This then has further consequences when we consider the number of route errors. These are nodes that have been identified as best-next-hop, while they are not. In the literal model the number of routing errors at time 250, when the 12th round of OGMs has been processed, is 2.5. In the alternative model

this value is only 0.22. It appears 85% of all simulations of the literal model have at least one routing error at time 250, while this is true for only 17% of all runs in the alternative model. Since the alternative message rebroadcasts more OGMs from more best-next-hops, the chances that a suboptimal route wins the neighbor ranking is greatly diminished.

It should also be seen in light that a suboptimal "best-next-hop" in the literal model is the designated "best-next-hop". A node has no alternatives. The alternative model identifies all potential best-next-hops. Even if some are suboptimal, there will be optimal alternatives that have been identified as well.

The number of 2.5 route errors seems small, but for many routes there is no suboptimal "best-next-hop". For example, any route from the top left node to any node in the bottom right half of the grid, will pass through either of the two neighbors of the top left node. Both neighbors are optimal choices.

The changes in the alternative model suggest that there may be some overhead since it sends more OGMs from more best-next-hops. On the other hand the alternative model will only rebroadcast duplicates if they improve on the TTL, unlike the literal model. We see that the maximum over 100 runs, rises for both models up to 45 OGMs in the buffer. Over all 255 time units the average number of OGMs in the buffer of a node in the literal model is 1.7, while it is 2.0 for the alternative model. It appears that the alternative model introduces a noticeable, but small overhead.

8 Conclusion

This paper modelled on a network routing protocol known as Better Approach to Mobile Ad hoc Networks (B.A.T.M.A.N.). The formalisation of the protocol revealed several ambiguities and inconsistencies in the RFC. We developed two models, one implementing a literal reading of the RFC, the second model closer to the underlying concepts. The literal model e.g implemented the requirement that there can only be one best-next-hop, whereas the alternative model relaxed the condition to include all nodes with the same ranking. On the other hand, the alternative model does rebroadcast fewer OGMs with a suboptimal TLL, i.e. OGMs that arrived via suboptimal routes.

For a basic model we showed that both interpretations ensure loop-freedom, bidirectional link discovery, and route-discovery. This served mostly to debug the model. We then use simulation of a timed model to compare the performance of the literal and alternative model, and found that the alternative model identifies a significantly lower number of suboptimal routes, and thus improves on the literal interpretation of the RFC.

Future work should consider mobility in networks and check whether we can observe route discovery failures as has been predicted by Cigno and Furlan [2]. It would be interesting to see how the proposed alternative could be combined by suggestion for improvements made by these authors and others.

In general the process of formalisation has shown that it is important for an RFC to define all concepts, even those that seem obvious, such as "newer sequence number". It also shows that the RFC should clearly distinguish between the preconditions – *Rule 5.4* to *Rule 5.7* in this paper – and the processes itself. If a process formulates those conditions again, even if it is only meant as a clarification, it can be a source of inconsistencies and ambiguities.

References

[1] Peter Bulychev, Alexandre David, Kim Gulstrand Larsen, Marius Mikučionis, Danny Bøgsted Poulsen, Axel Legay & Zheng Wang (2012): *UPPAAL-SMC: Statistical model checking for priced timed automata*. *arXiv preprint arXiv:1207.1272*, doi:10.4204/EPTCS.85.1.

[2] Renato Cigno & Daniele Furlan: *Improving BATMAN Routing Stability and Performance*. Ph.D. thesis, PhD thesis, University of Trento, 2011.

[3] T. Clausen & Jacquet P.: *Optimized Link State Routing Protocol (OLSR), Network Working Group*. Available at http://www.tools.ietf.org/html/rfc3626. Accessed:2015-06-08.

[4] Ansgar Fehnker, Peter Höfner, Maryam Kamali & Vinay Mehta (2013): *Topology-based mobility models for wireless networks*. In: *International Conference on Quantitative Evaluation of Systems*, Springer, pp. 389–404, doi:10.1007/978-3-642-40196-1_32.

[5] Ansgar Fehnker, Rob Van Glabbeek, Peter Höfner, Annabelle McIver, Marius Portmann & Wee Lum Tan (2012): *Automated analysis of AODV using UPPAAL*. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, pp. 173–187, doi:10.1007/978-3-642-28756-5_13.

[6] Daniele Furlan (2011): *Analysis of the overhead of BATMAN routing protocol in regular torus topologies*. *University of Trento, Italy, Tech. Rep.* Available at https://downloads.open-mesh.org/batman/papers/OGMoverhead.pdf.

[7] Tobias Hardes (2015): *Performance Analysis and Simulation of a Freifunk Mesh Network in Paderborn using B.A.T.M.A.N Advanced*. Master's thesis, University of Paderborn. Available at http://thardes.de/wp-content/uploads/2016/03/thesis.pdf.

[8] Aleksandr Huhtonen (2004): *Comparing AODV and OLSR Routing Protocols*. Available at http://www.tml.tkk.fi/Studies/T-110.551/2004/papers/Huhtonen.pdf.

[9] Mojgan Kamali, Peter Höfner, Maryam Kamali & Luigia Petre (2015): *Formal Analysis of Proactive, Distributed Routing*. In: *Software Engineering and Formal Methods: 13th International Conference, SEFM 2015, York, UK, September 7-11, 2015. Proceedings*, Springer, pp. 175–189, doi:10.1007/978-3-319-22969-0_13.

[10] Elis Kulla, Masahiro Hiyama, Makoto Ikeda & Leonard Barolli (2012): *Performance Comparison of OLSR and BATMAN Routing Protocols by a MANET Testbed in Stairs Environment*. *Comput. Math. Appl.* 63(2), pp. 339–349, doi:10.1016/j.camwa.2011.07.035.

[11] Spyridon Marinis Artelaris (2016): *Performance evaluation of routing protocols for Wireless Mesh Networks*. Available at http://lnu.diva-portal.org/smash/get/diva2:903013/FULLTEXT01.pdf.

[12] Axel Neumann, Corinna Aichele, Marek Lindner & Simon Wunderlich (2008): *Better approach to mobile ad-hoc networking (BATMAN)*. *IETF draft*, pp. 1–24. Available at https://tools.ietf.org/html/draft-wunderlich-openmesh-manet-routing-00.

[13] Rob Van Glabbeek, Peter Höfner, Wee Lum Tan & Marius Portmann (2013): *Sequence numbers do not guarantee loop freedom: AODV can yield routing loops*. In: *Proceedings of the 16th ACM international conference on Modeling, analysis & simulation of wireless and mobile systems*, ACM, pp. 91–100, doi:10.1145/2507924.2507943.

[14] J. C. P. Wang, B. Hagelstein & M. Abolhasan (2010): *Experimental evaluation of IEEE 802.11s path selection protocols in a mesh testbed*. In: *2010 4th International Conference on Signal Processing and Communication Systems*, pp. 1–3, doi:10.1109/ICSPCS.2010.5709664.

[15] Oskar Wibling, Joachim Parrow & Arnold Pears (2004): *Automatized verification of ad hoc routing protocols*. In: *International Conference on Formal Techniques for Networked and Distributed Systems*, Springer, pp. 343–358, doi:10.1007/978-3-540-30232-2_22.