Performance evaluation of the Trickle algorithm

van Bavel, S.E.

Award date:
2014

TU/E

BACHELOR FINAL PROJECT

Performance evaluation of the Trickle algorithm

Author:
Sjoerd VAN BAVEL

Supervisors:
Marko BOON
Thomas MEYFROYT

29 April 2014

CONTENTS

CHAPTER

ONE

INTRODUCTION

In this project we study the performance of the Trickle algorithm introduced in [1]. Using Monte Carlo Simulations we determine its performance in terms of propagation speed and hop count when propagating software as well as transmission rates in the periods nothing is being propagated. Our simulations are validated using results from [5]. Results are obtained by comparing idealized network topologies with a data set of street lights in the city of Eindhoven. More results are obtained by taking into account more parts of the devices and comparing the results to [5].

1.1 TRICKLE

A wireless sensor network consists of Wi-Fi enabled devices (nodes) which operate autonomously and without human intervention. Because of new knowledge or changing conditions of the network the software on the devices may need to be updated. This means that new software must be propagated over the network. This is done with the Trickle algorithm by propagating from nodes to neighbouring nodes. One of the advantages of these devices is that they can remain unattended for a long time in different conditions with different instructions. This benefit is enhanced when the devices can operate on battery power, this is the reason that Trickle was designed to propagate and maintain code with as few transmissions as possible.

The algorithm uses its nodes to form a 'polite gossip' network. This means that the information, referred to as 'gossip', is propagated to the neighbours of one node at a time. We assume the entire network is connected, so eventually the information reaches every node. It operates in a way where every once in a while, a node gossips its software version. All nodes want to have the latest 'gossip'; if a node hears newer 'gossip', it revise its software, if it hears older 'gossip', it responds with its own version shortly after. If a node hears its own 'gossip' one or more times it will not gossip its own 'gossip' for a while. To prevent 'gossip' from not being heard a node will not start speaking when a different node is already speaking within hearing range. Interference is the main reason the intervals of our 'gossips' are randomized instead of simply passing our 'gossip' on when it arrives.

1.2 RESEARCH GOALS

The goal of this project is to obtain results on the performance of the Trickle algorithm and how this depends on the topology of the network. We are interested in the propagation speed, the time it takes until the entire network is updated, and the hop count, the number of nodes the information passes through to reach a node. To obtain these performance measures we will simulate the Trickle algorithm on different networks using Monte Carlo simulation.

First, results from simple networks consisting of nodes on a simple line or grid will be compared with known theoretical solutions. Subsequently, we will simulate more complex and general networks. Finally, to test how network layouts affects the propagation speed, we do this by running simulations on a data set with street lights in the city of Eindhoven.

We now list the research questions we will be focusing on for this research:

1. How do the transmission rates of the algorithm change with the size and topology of the network or the settings of the algorithm?
2. How are the propagation speed and the hop count of the network affected by different topologies?
3. How accurate are the results of [5] when the processes on the MAC and physical layers are taken into account?

These layers will be discussed in more detail later in this report.

CHAPTER

TWO

DETAILED DESCIPTION OF THE TRICKLE ALGORITHM

As mentioned earlier, Trickle is an algorithm for propagating and maintaining software updates. The original algorithm describes the actions that take place on the application layer, our model goes beyond that and also tries to take the actions on the MAC and physical layer into account. These are the layers where the scheduling of transmissions and the actual transmitting take place.

Each node operates on 3 mostly independent layers, the application layer, the MAC layer and the physical layer. On the application layer the algorithm for scheduling transmissions takes place. We start by describing assumptions we have made for this model, subsequently we describe the algorithms for the different layers. Then we define some states the network can be in, these states will later be used to define performance measures like propagation speed. We show how the transitions between the different states are defined for a single node. We finish with some examples.

2.1 APPLICATION LAYER

The algorithm on the application layer is the algorithm introduced and described in [1]. Each node has three variables it tracks: a counter c, its current cycle length τ and possibly an upcoming transmission time t. Additionally there are five more parameters which are constant: k, the number of times an update must be heard before a transmission is not scheduled, τ_{max}, the largest cycle length, τ_{min}, the smallest cycle length, and η, the portion of the cycle which is listen-only.

The algorithm consists of the following set of rules:

Event	Action
t expires	transmit if $c < k$.
τ expires	double τ if $\tau < \tau_{max}$, $c = 0$ and pick a new t.
Receive current version	increment c.
Receive different version	if τ is larger than τ_{min}: $\tau = \tau_{min}$ and pick a new t.

Here t is always picked uniformly from $(\eta\tau, \tau)$.

The parameter η is a listen-only parameter, introduced to prevent the so-called short-listen problem [1]. A fraction η of the cycle is made listen-only, the transmission time t will not be scheduled in this listen-only period. In [1] η is always $\frac{1}{2}$, the generalisation to η was introduced in [5]. τ increases each cycle until $\tau \geq \tau_{max}$ so fewer transmissions are being made when no new information has entered the network for a while.

When this algorithm dictates a transmission, a transmission with the current version number is scheduled in the node its MAC layer.

2.2 MAC AND PHYSICAL LAYER

The MAC layer handles the broadcasts it receives from the application layer. The MAC layer also has three properties: Q_{max}, d_{max} and s. The MAC layer stores all transmissions in a queue. If the current size of the queue of transmissions is below Q_{max}, it adds the new transmission to the queue, if the queue is full, i.e. the size is Q_{max}, the transmission is dropped.

The MAC layer works with exponential backoffs to pick times for transmissions. For an exponential backoff the MAC layer picks a number of time steps from $\{0,1,\ldots,2^d-1\}$ where d is the minimum of d_{max} and the number of previous attempts to broadcast plus one. The length of one time step is s, and lasts 54μ seconds. So an amount of time between $[0, s(2^d-1)]$ is chosen. After this amount of time the MAC layer attempts to start the broadcast. If no transmission can be started, i.e. a neighbor is currently transmitting, d is increased and another exponential backoff is created with a larger expected amount of time. Otherwise the MAC layer starts the transmission by sending a transmission to the physical layer which starts to transmit. When a new transmission is send to the MAC layer a exponential backoff is set if the queue is empty.

When no neighbor is currently transmitting the transmission follows almost directly since the initial backoff, which is very short, is the only delay. If the queue is not empty when the transmission finishes, another backoff is created for a new transmission to start.

The physical layer is the physical receiver and transmitter of the node. When it receives a broadcast from another node it is processed by the application layer. It starts broadcasting when asked by the MAC layer and reports back when it finishes.

2.3 NODE STATES

A node has three possible states, listening, receiving and transmitting. We define *transmitting* as the period a node is broadcasting software. *Receiving* is when one or more neighbors are currently broadcasting. A node is *listening* when it does neither.

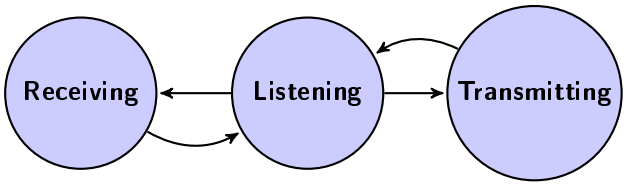


Figure 2.3.1: The three states and their transmissions

Note that in Figure 2.3.1 there are no links between receiving and transmitting. These are prevented because a node will not broadcast when another node is already broadcasting.

2.4 NETWORK STATES

For the entire network we define three different states: propagating, saturated and maintenance mode. The different states differ in the current versions and the cycle lengths of the nodes in the network. We define the network to be *propagating* when some nodes have a newer version than others and the new version is being transmitted over the network. The network is *saturated* when every node has the newest version but not all nodes have reached their maximum cycle length. And the final state is called *maintenance*, when all nodes have the same version and all nodes have τ_{max} as their cycle length.

The maintenance mode is defined like this, because it is a stable state in which the network ends when no new software is introduced for a while. When a node (a seed) is injected with a new version, the network is propagating until every node is updated and the network is saturated. After a while, it will then go to maintenance mode again.

The time it takes for the network to go from entering propagating mode and going to saturated mode is called the end-to-end delay, one of the performance measures of interest.

2.5 INTERVAL SKEW

If all the nodes were in sync with each other all cycle resets would happen at the same time. The difference between this situation and the asynchronous situation is called the interval skew. The interval skew is defined as the differences in time until the next cycle resets for the different nodes in maintenance mode. Since all nodes share the same cycle length in maintenance mode (τ_{max}), the times remain the same in respect to each other. If all nodes have their cycle resets simultaneously, the network would be called synchronized. When the network is not synchronized, the differences between the cycle resets and any given node will be somewhere between 0 and τ_{max}. The interval skew changes when the network state changes to propagating and nodes are reset when new software versions are introduced. In most cases it can be assumed that the interval skew is uniformly distributed. However, when a network starts synchronized the interval skew will depend on the hop count of the previous propagation waves. In other words: nodes with a similar distance to the source will tend to have cycle resets close to each other.

Using our simulation we will check whether this property affects the performance measures and whether we need to take this into account with the rest of the simulations. We will check this by simulating types of interval skew with the same parameters and comparing the results.

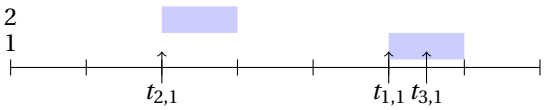
2.6 EXAMPLES

To get an impression of how this algorithm works we show how the algorithm works out in 2 different simulations.

2.6.1 SITUATION 1

We have a network in maintenance mode with 3 nodes in a single cell network, i.e. all three nodes are neighbours of each other. k is set at 2. Note: the time it takes to transmit a broadcast, M, is very large for example purposes.

This is the timeline:

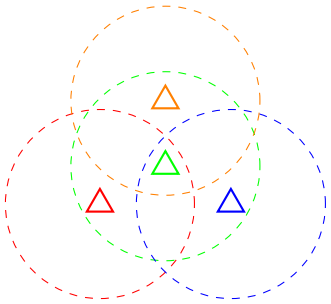


The blue line signifies when a node is transmitting (since all nodes are connected only one node can be transmitting at a time). Now we explain what happens at each moment.

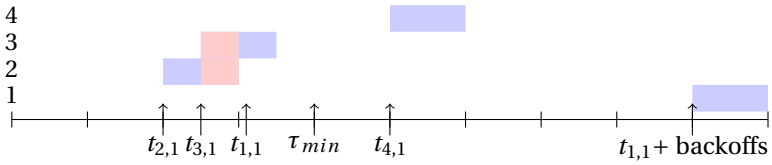
Time	Summary
$t = 0$	Node 1 is being updated. It sets a time $t_{1,1}$ between $[\tau_{min}\eta, \tau_{min}]$. Nodes 2 and 3 have already set times between $[\tau_{max}\eta, \tau_{max}]$ with τ_{max} an interval size larger than τ_{min}.
$t = t_{2,1}$	Node 2 starts its transmission.
$t = t_{2,1} + M$	Node 2 ends its transmission. Node 1 realises its version is more recent than that of node 2 so it resets if $\tau > \tau_{min}$. This is not the case so it does nothing. Node 3 also hears the broadcast, its version is equal to its own so it increments its c by one.
$t = t_{1,1}$	Node 1 starts its transmission,
$t = t_{3,1}$	Node 3 is scheduled to transmit and $c < k$ so a transmission is scheduled. Node 1 is already transmitting so node 3 does not transmit and the transmission is delayed a few times.
$t = t_{1,1} + M$	Node 1 ends its broadcast. Nodes 2 and 3 are updated and reset their period to τ_{min} and pick new broadcast times from $[\tau_{min}\eta, \tau_{min}]$.

2.6.2 SITUATION 2

Situation: 4 nodes, nodes 2 to 4 are on a circle around node 1. All are within range of node 1 but their mutual distances are so big they are out of each others reach, see the next picture



The events take place in the following order. The colored bars indicate whether a certain node is broadcasting. Note that the red color in the broadcasts only indicate interference at node 1. The broadcast may still reach other nodes without causing interference, as long as they do not also overlap with other broadcasts.



Time	Summary
$t = 0$	Node 1 is updated. It resets τ to τ_{min} and picks $t_{1,1}$ from $[\tau_{min}\eta, \tau_{min}]$. Nodes 2 to 4 have already set times t and have cycle lengths of τ_{max}.
$t = t_{2,1}$	Node 2 starts transmitting.
$t = t_{3,1}$	Node 3, out of reach of node 2, starts transmitting.
$t = t_{2,1} + M$	Node 2 ends its broadcast, the signal was interfered so node 1 does not receive anything.
$t = t_{1,1}$	Node 1 wants to transmit, but node 3 is still transmitting within range so node 1 delays the transmission a few times.
$t = t_{3,1} + M$	Node 3 ends its broadcast but because of interference by node 2 it was not received correctly by node 1.
$t = \tau_{min}$	Node 1 has finished its cycle so it increases its cycle length to $2\tau_{min}$ and picks a time $t_{1,2}$ from $[2\tau_{min}\eta, 2\tau_{min}]$.
$t = t_{4,1}$	Node 4 starts transmitting. During this transmission the backoff set by node 1 ends. The transmission still cannot be made so it is extended again.
$t = t_{4,1} + M$	Node 4 ends transmitting, there has been no interference so the signal came through to node 1. Node 1 receives a version number lower than its own so node 1 resets its τ to τ_{min}, removes the scheduled transmission at $t_{1,2}$ and picks a new time a from $[\tau_{min}\eta, \tau_{min}]$ and sets an event at the current time $+a = t_{1,3}$.
$t = t_{1,1}$+ backoffs	Node 1 finally starts transmitting the transmission scheduled at $t_{1,1}$.
$t = t_{1,1}$+ backoffs $+M$	Node 1 ends transmitting, nodes 2 to 4 receive the signal and update their version number. All reset τ to τ_{min} and pick new times t from $[\tau_{min}\eta, \tau_{min}]$.

2.7 LITERATURE OVERVIEW

We have conducted a literature review focusing on the following 3 questions:

- Have any other adaptations of the Trickle algorithm been made?
- Has any previous research been done on attributes of the Trickle algorithm?
- How has Trickle been applied to real world solutions?

2.7.1 ADAPTATIONS OF THE ALGORITHM

The Trickle algorithm originated in [1] as an algorithm for code propagation and maintenance. Since then it has been found so useful, that it has become a basic mechanism in wireless network technologies and used in

numerous protocols and systems according to [2]. While the protocol was designed for propagating information, it can also be used for data gathering where all individual nodes are able to generate data and distribute this data through the network. The same paper describes how the techniques in Trickle are also applied for route maintenance and neighbor discovery in different types of networks.

Adjustments to the algorithm have been made to take into account the number of neighbors of a node. A broadcast of a node with a lot of neighbors is probably more effective than a broadcast from a node with fewer neighbors [4]. Another algorithm, DIP ([3]), was designed building on Trickle, but requires fewer transmissions to determine whether an update is needed.

2.7.2 PERFORMANCE MEASURES

On the attributes of the Trickle algorithm some research has previously been done by Meyfroyt [5]. His work is particularly relevant because it uses the same extended version of the Trickle algorithm as we do. In [5] theoretical solutions are obtained for line and grid networks. The provided solutions will later be used to check our own simulations. The behavior of the original algorithm was studied in more papers. In [8] an analytical model is developed for the behavior for the time the algorithm takes to leave propagation mode and in [7] a model for the message count in steady state is presented.

2.7.3 TRICKLE APPLIED IN REAL WORLD SITUATIONS

On applications of the Trickle algorithm and the networks it allows to create the paper [10] reports extensively. Wireless sensor networks allow for applications which would not be allowed with regular wired networks since they are much more expensive. Moore's law says hardware will become less expensive but this does not apply to the cost of wires for a wired network and the installation costs of a wired network. The paper mentions a low of different applications: home applications, video surveillance monitoring, monitoring energy consumption and building applications like fire detection and sprinkler networks. Also very interesting are industrial applications like controlling a sensor grid of complex machines while reporting to a central control room and urban applications like gathering meteorological data and dimming street lights when less light is required.

In [9] it is described how wireless sensor networks are deployed around a volcano to gather data on volcanic events. An array of 16 nodes each equipped with seismological sensors. The obtained data was transmitted through a multihop network to an observatory, where over the course of 3 weeks 230 events where gathered.

CHAPTER

THREE

IMPLEMENTATION

In this chapter the implementation is discussed. In order to obtain the propagation speeds, hop counts and transmission rates of the algorithm on different networks we will use Monte Carlo simulation in a Java program. This program is written with very large networks in mind.

3.1 VARIABLES

In this section we introduce all variables needed to simulate the algorithm.

3.1.1 NODES

Each node indicates its own version by an integer V. Each node keeps track of a list of neighbors S which it is able to reach by broadcasting software. To track the states a node can be in, some additional variables have to be defined. When a node starts transmitting, it tracks its own state by a boolean T which is turned to true, T set to false when the node finishes transmitting. A transmission takes a predetermined M seconds.

When a node receives the announcement a neighbor starts broadcasting, it increments the integer Re by one. Consequently, when a neighbor stops broadcasting the node decreases Re by one. A node is in the state of receiving when the integer Re is larger than zero. Since every transmission has to start before it ends, i.e. an increment will always precede a decrease, and Re is initialised at zero, Re is always greater than or equal to zero.

3.1.2 INTERFERENCE

Since interference is defined simplified for this simulation, i.e. if and only if a node receives 2 overlapping broadcasts both broadcasts are lost, it can be tracked by a simple boolean I. This is turned true when more than one neighbor is currently transmitting and is only turned false when the node goes in listening state. This may take longer than it takes for the nodes to finish if more than 2 broadcasts overlap. As long as I is true, no broadcasts can be received.

3.1.3 EVENTS

At any given time every node has a cycle reset scheduled and may have more events scheduled. There is also a limited set of seed events where a node will be updated. The algorithm knows 5 types of events that have to be executed at a certain time:

- Starting a broadcast
- Ending a broadcast
- End backoff
- Resetting a cycle
- Update node (seed)

Events are objects in the simulation which can be one of these 5 types of events. All these events are stored in a single eventqueue. The attributes of an event are the time it must execute, its type and the node where it will take place.

3.1.4 PSEUDOCODE

All actions a node can execute are:

- Start transmitting
- End transmitting
- End backoff
- Start receiving
  Hearing a neighbor start a broadcast
- Stop receiving
  Hearing a neighbor end a broadcast
- Resetting a cycle
- Getting an update
  Receiving a new version from a seed

Most of these actions are activated by events and are activated by the eventqueue. Some are activated by neighbors starting or ending a transmission. We now describe the process of each action in pseudocode:

Start transmitting (Trickle Timer)

```
if c ≥ k then do nothing
    (doing nothing)
else Schedule broadcast
    if Q.size < Q_max then
        add V to Q if Q.size > 1 then We are still waiting for the current transmission or backoff to finish
            (doing nothing)
        else create exponential backoff
            d = 1
            Pick random integer a from [0, 2^d − 1]
            d = d + 1
            b = s ∗ a
            Set event End backoff at t_current + b
        end
    else do nothing
    end
end
```

End transmitting

```
CV = Q.nexttransmission
for every neighbor in S do
    neighbor.stopreceiving(CV)
end
if Q.size > 0 then create exponential backoff
    d = 1
    Pick random integer a from [0, 2^d − 1]
    d = d + 1
    b = s ∗ a
    Set event End transmitting at t_current + b
else
    T = false
end
```

End backoff

if *Re*> 0 **then**
  Pick random integer a from $[0, 2^d - 1]$
  $b = s * a$
  if $d < d_{max}$ **then**
    $d = d + 1$
  end
  Set event End transmitting at $t_{current} + b$
end
else Start transmitting
  $d = 1$
  for *all neighbors in S* **do**
    neighbor.startreceiving
  end
  T = true
  Set event End transmitting at $t_{current} + M$
end

Start receiving

Re = Re + 1
if *Re*> 1 **then**
  I = true
end

Stop receiving

Input: version number i
if I = *false* **then**
  if $i > V$ **then**
    $V = i$
    $c = 0$
    $\tau = \tau_{min}$
    Pick random a from $[\eta * \tau, \tau]$
    $t = t_{current} + a$
    Update event Start transmitting to t
    Update event Reset cycle to $t_{current} + \tau$
  else if $i < V$ **then**
    if $\tau > \tau_{min}$ **then**
      $c = 0$
      $\tau = \tau_{min}$
      Pick random a from $[\eta * \tau, \tau]$
      $t = t_{current} + a$
      Update event Start transmitting to t
      Update event Reset cycle to $t_{current} + \tau$
    end
  else $i = V$
    $c = c + 1$
  end
else do nothing
end
Re = Re - 1
if *Re* = *0* **then**
  I = false
end

Reset cycle

if $\tau < \tau_{max}$ **then**
  $\tau = \min(2\tau, \tau_{max})$
end
Pick random t from $[\eta * \tau, \tau]$
Set event Start transmission at $t_{current} + t$
Set event Reset cycle at $t_{current} + \tau$

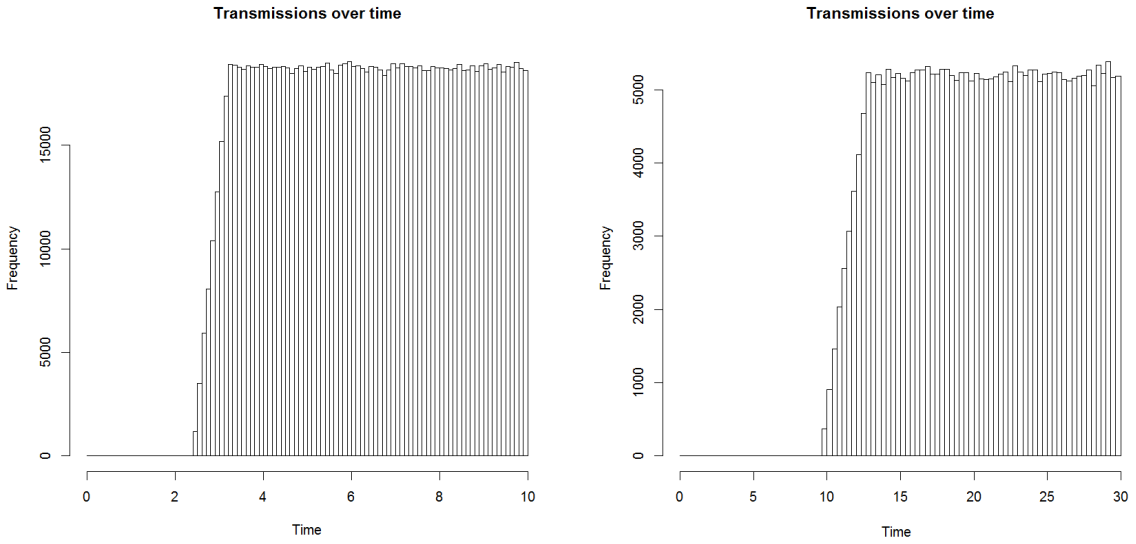
Get updated

Input: version number i
if $i > V$ **then**
  $V = i$
  $\tau = \tau_{min}$
  $c = 0$
  Pick random a from $[\eta * \tau, \tau]$
  $t = t_{current} + a$
  Update event Start transmitting to t
  Update event Reset cycle to $t_{current} + \tau$
end

With every update event the eventqueue will delete the previous event of this type by this node if present and create a new event with the given time. Here the current time is given by $t_{current}$.

3.2 INITIALISATION

Initializing all the different events as well as the counters directly would be complex to calculate directly. We can avoid this by letting the network spent a few cycles. We assume the network starts in maintenance mode and generate an interval skew. Then we can generate all first cycle resets at the start. After 2 cycles all broadcast moments are initialised and the counters are set. From simulations it follows that from this point the rate of transmissions stays constant so we assume the network enters a steady state at that point, see Figure 3.2.1. Since the network is in maintenance mode, cycle lengths do not increase and the initiation skew remains equal to what it was at the start.



(a) $n = 3$ and $\tau_{max} = 1.6$.

(b) $n = 10$ and $\tau_{max} = 6.4$.

Figure 3.2.1: Networks are lines and simulated using 10 000 runs.

3.3 TEST CASES

In this section we will introduce some network types. These networks we will later use to test the Trickle algorithm in our simulations.

3.3.1 SINGLE CELL

In a single cell network all nodes are connected with each other. This means that in a network with n nodes every node has $n-1$ neighbors. This type of network is interesting when looking for the behavior of the algorithm in maintenance mode.

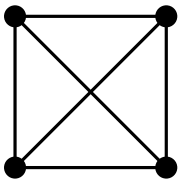


Figure 3.3.1: Nodes in a single cell where every node is connected to all other nodes.

3.3.2 LINE NETWORKS

A different network is in a line network. Visually this means all the nodes are on a line. We have n nodes spaced 1 step apart. Combined with a broadcast range R each node has a maximum of $2R$ neighbors, R to the left and R to the right. This can be less if the node is at the beginning or the end of the network.

Figure 3.3.2: Nodes on a line.

3.3.3 GRID NETWORKS

The 2-dimensional variant of a line network is a grid network. Here, nodes are still 1 step apart but in 2 dimensions. The neighbors of a node are the nodes within a circle with diameter $2R$.

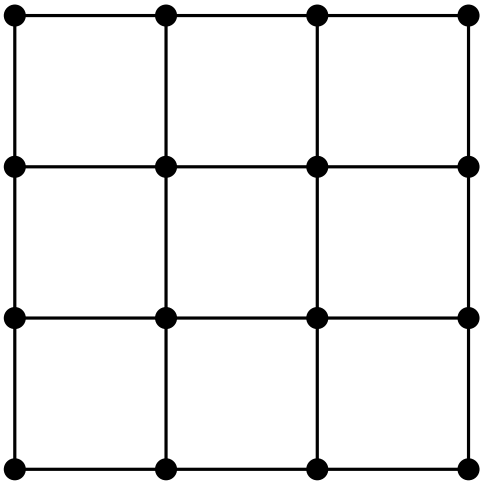


Figure 3.3.3: Nodes placed in a grid.

3.3.4 REAL-LIFE NETWORKS

To get a realistic image of what situations may occur in a real-life situation we have obtained a data set from a real-life situation. We have obtained a data set containing the locations of all street lights of the city of Eindhoven.

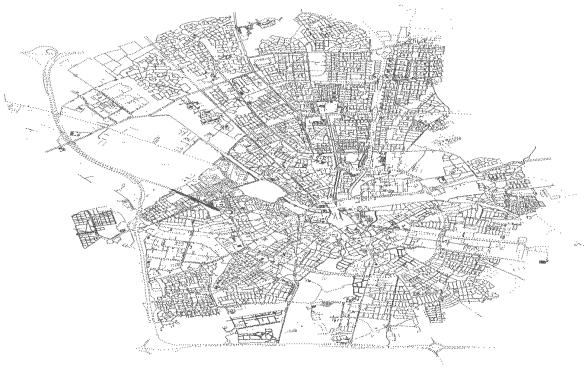


Figure 3.3.4: The street lights of the city of Eindhoven

3.4 VALIDATION

In other to validate our simulation before we obtain results for our research questions we will compare our simulation results with theoretical results obtained in [5] and [6]. The main results that can be compared are as follows:

1. In a single-cell network, i.e. a network where all nodes can reach each other, in maintenance mode the expected number of transmissions is approximated from below as $n \to \infty$ by
$$\frac{k}{\eta}. \tag{3.4.1}$$

2. In a multi-cell network, i.e. a network where not all nodes can hear each other, given by a $n \times n$ grid where nodes have a broadcasting range R in maintenance mode the number of broadcasts per interval scales as
$$\mathcal{O}\left(\frac{n^2}{R^2}\right). \tag{3.4.2}$$

3. In a multi-cell network in propagation mode, consisting of nodes placed on a line, if $\eta = \frac{1}{2}$ and $k = 1$, we have the following results:

   (a) The mean number of hops can be approximated by:
   $$\frac{3n}{2R+1}. \tag{3.4.3}$$

   (b) The mean time until network consistency is approximated by:
   $$\frac{3n}{2R+1}\left(M + \tau_{min}\left(\frac{1}{2} + \frac{R+1-\Sigma_{j=1}^{R+1}\frac{1}{j}}{R(R+1)}\right)\right). \tag{3.4.4}$$

We will simulate these cases in our own simulation and compare and discuss the results briefly. To obtain these results we have simulated the networks with the following variables (unless specified otherwise):

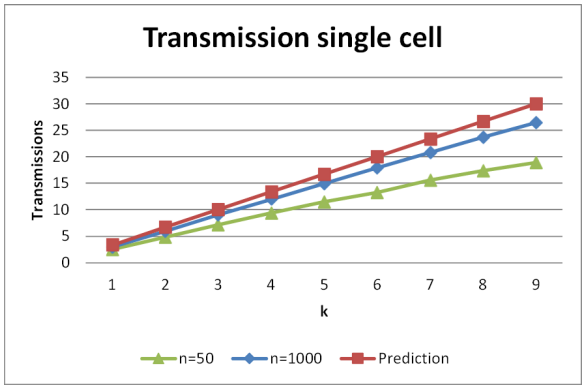
Time	Value
M	0
s	0.000054
τ_{min}	1
τ_{max}	16
d_{max}	8
Q_{max}	4

Note that with these variables we do not simulate a realistic network but it does replicate the conditions in the research done by Meyfroyt in [5].

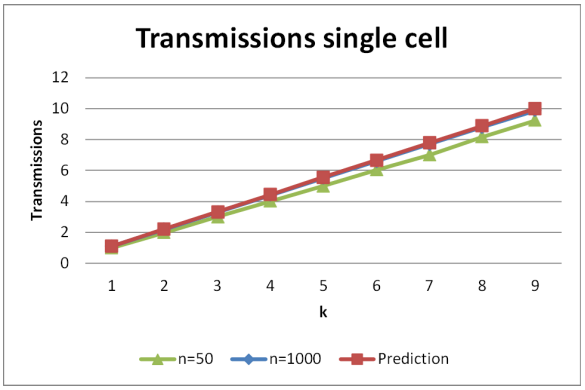
3.4.1 SINGLE CELL NETWORK

The first validation question was how many transmissions take place in a single cell network in maintenance mode, where we varied k and η, and whether this matched up with (3.4.1).

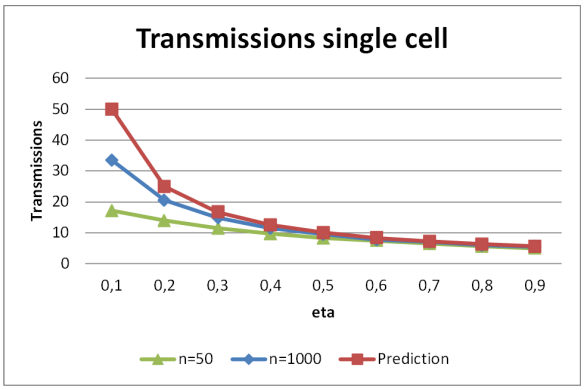
For these results we have done simulations with 50 nodes and 1000 nodes, the results are shown in Figure 3.4.1. Since (3.4.1) describes a limit for very large values of n we expected the results to improve for larger values of n which we see. For large values of η the results were spot on. For low values of η the results were much below the expected value but they improved much with the larger network size, this was expected since this is an asymptotic result.



(a) $\eta = 0.3$, data is shown in table 6.1.1.



(b) $\eta = 0.9$, data is shown in table 6.1.2.



(c) $k = 5$, data is shown in table 6.1.3.

Figure 3.4.1: Single cell networks generated with 1000 runs.

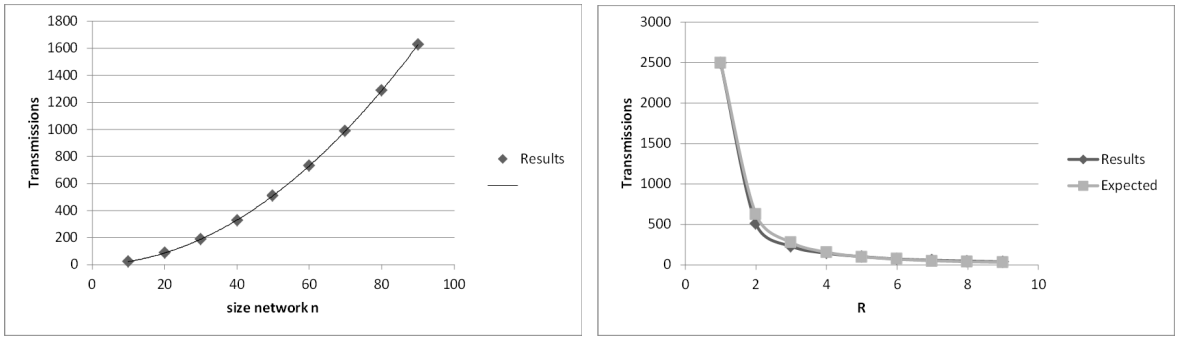
3.4.2 MULTI CELL NETWORK

The second validation question was how the results scale for large values of n and R. For this we have run a simulation on a grid network of n by n nodes.

The results are shown in Figure 3.4.2. In Figure 3.4.2a a quadratic trend line is shown, indicating that the growth of the results is as expected. In Figure 3.4.2b the predictions of $1/R^2$ is shown. This also confirms that the transmission rate per interval scales as $\mathcal{O}(\frac{n^2}{R^2})$.

3.4.3 PROPAGATION MODE

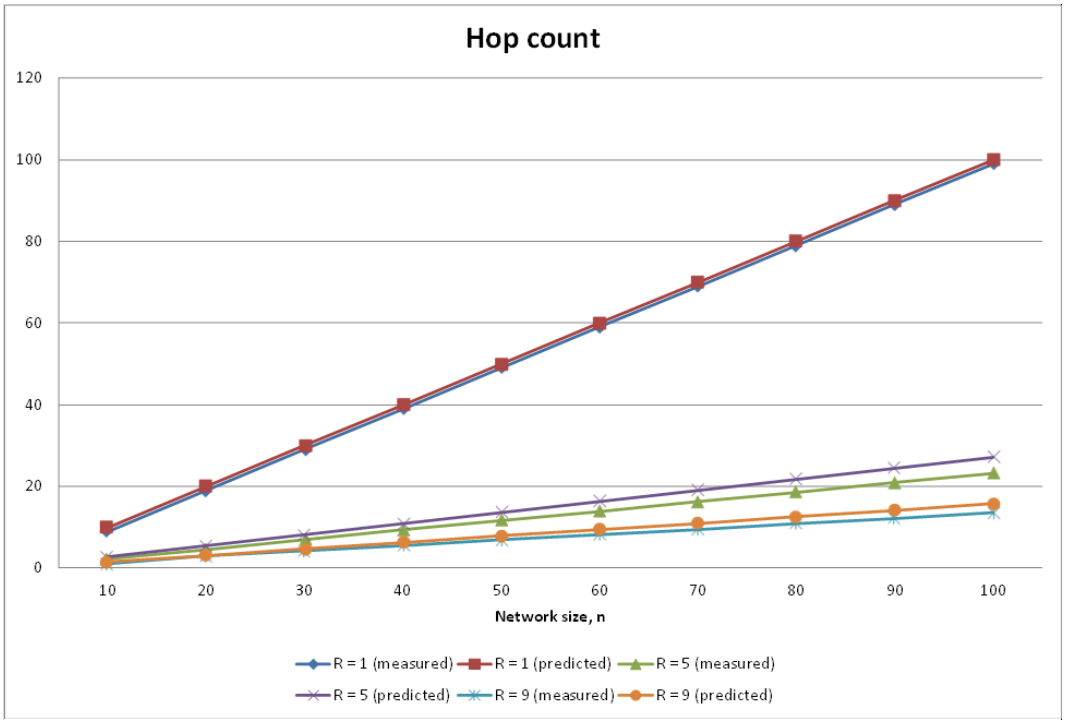
For the propagation mode we have 2 tests, (3.4.3) and (3.4.4): The first test is whether the hop count behaves as expected and second test is whether the propagation speed is as expected. We have run a simulation on a line network where we set a seed on the leftmost node and measure the time and number of hops it takes to get to the rightmost node. We show the hop count and the propagation speed. In Figure 3.4.4 we see the hop count and propagation speed of our simulations as well as the expectations. We see the results match the expected values exept for the divergence for large values of η. We have no explanation for this.



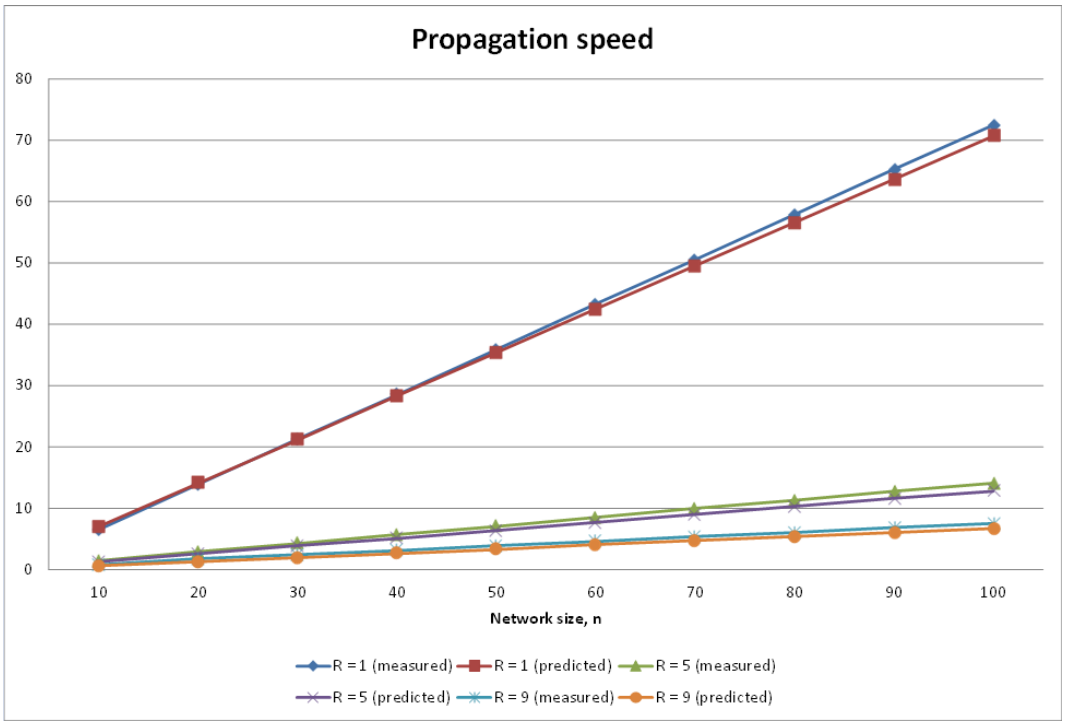
(a) $k = 1$, data is shown in table 6.1.6.

(b) $k = 5$, data is shown in table 6.1.7.

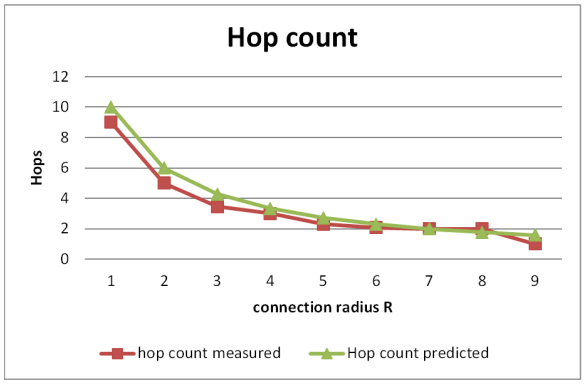
Figure 3.4.2: Multi cell grid networks over 100 cycles.



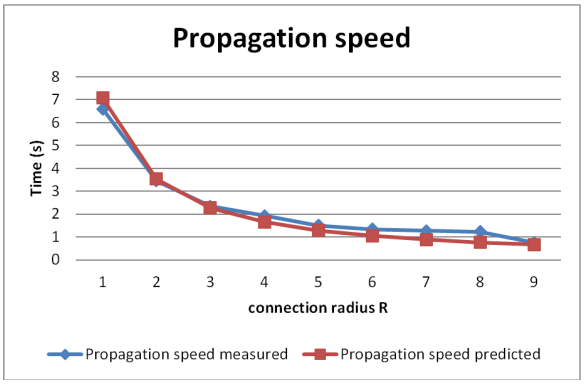
(a) Data is shown in table 6.1.4.



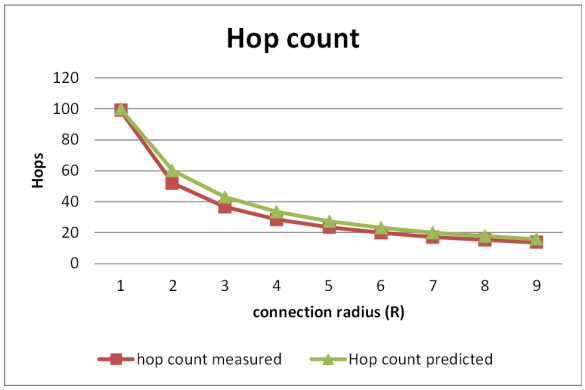
(b) Data is shown in table 6.1.5.



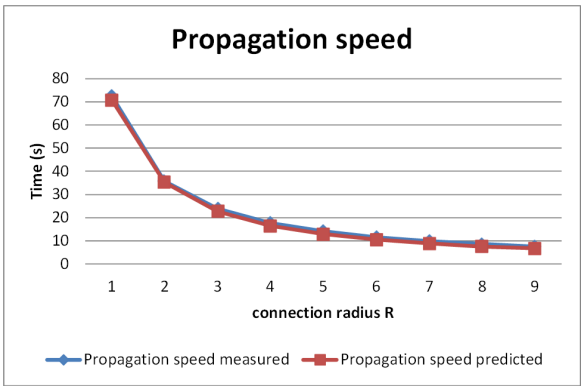
(a) $n = 10$, data is shown in table 6.1.8.



(b) $n = 10$, data is shown in table 6.1.8.



(c) $n = 100$, data is shown in table 6.1.9.



(d) $n = 100$, data is shown in table 6.1.9.

Figure 3.4.4: Results of the validation on propagation speed and hop count.

CHAPTER

FOUR

RESULTS

Like we did for our validation simulations we have kept most parameters constant. Unless specified otherwise the values used in the simulation are:

Time	Value
M	0.01
s	0.000054
τ_{min}	1
τ_{max}	16
d_{max}	8
Q_{max}	4

In our simulations we have seen that the Trickle algorithm stops working properly for large values of M in our networks. The number of scheduled transmissions per interval goes up to the number of nodes and the nodes are continuously transmitting. This leads to all nodes scheduling their transmissions because they do not receive any transmissions increasing their counters c. If the algorithm breaks down like this the value of d_{max} and Q_{max} do not make much of a difference. If the algorithm works as intended, i.e. for small values of M where the size of the queue never grows large and no transmissions are lost. Therefore we have chosen to take to keep these parameters constant.

First we will look at research question 1, how do the transmission rates change when we take different topologies into account? For this we want to compare our data set of Eindhoven with a grid network. We set the size of the grid network on 224 by 224 so the size of this network is about the same size of our data set (50176 nodes in the data set versus 50553 nodes in the grid network). Our data set of Eindhoven came with just the points of street lights but without information on the scale of the network. We have matched our values of R so that the average number of neighbours of each network roughly the same too. From this we picked 3 different couples of radii to focus on in our simulations. The network radii are:

City network		Grid network	
R	mean number of neighbours	R	mean number of neighbours
0.0049	505.01	11.0	503.44
0.006	740.63	13.1	741.31
0.007	990.43	16.0	1013.87

These networks are similar in their average number of neighbours but they still have very different topologies. If we define center nodes as nodes with the maximum number of neighbours, i.e. their range does not touch any of the edges of the network, we see that the grid network consist mostly of center nodes with a much smaller group of edge nodes with less neighbours which drag the average number down to an average comparable to the city network. In the city network the distribution of the number of neighbours has more nodes with a number of neighbours closer to the average.

In Figure 4.0.1 the results are shown from our comparison of a grid network and the city network. The results of the different simulations in Figure 4.0.1b are shown separately in Figure 4.0.2. We see the algorithm

shows very similar results in response to changes in k and η except the city networks shows many more broadcasts for all values of k. This can be explained by the different distributions of numbers of neighbours for an arbitrary node. The center nodes in the grid network are much more connected. Therefore they receive more transmission which increases the probability a transmission is suppressed.

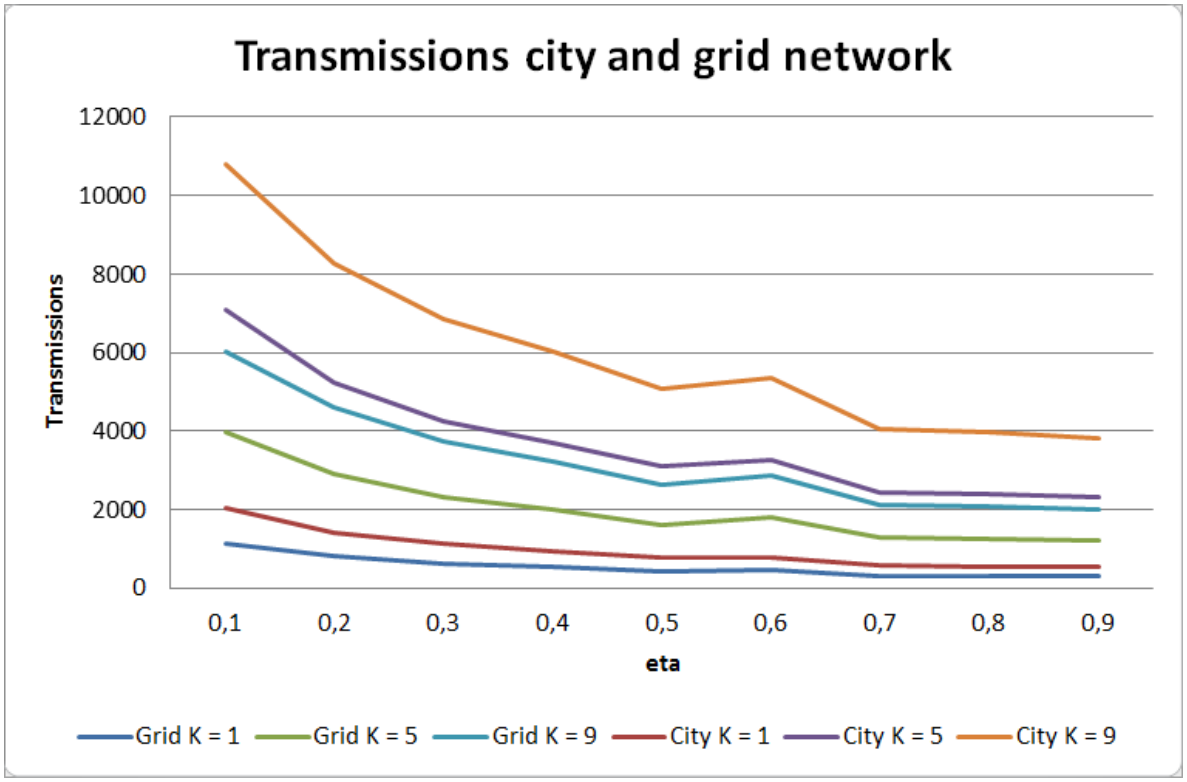
To answer research question 2, how are the propagation speed and the hop count affected by different topologies? We would have liked to compare the hop count and propagation speed of the data set and grid network like we did for transmission rates. In our previous tests we were working with line networks where we could take the first and last node to compare their distance, in the city dataset we have no concept of a first and last node in the city network. Therefore we can not make a direct prediction with any known results.

For research question 3: How accurate are the results of [5] when the processes on the MAC and physical layers are taken into account? We have repeated the tests we have done before for our validation with larger values of M. As discussed previously we do not take d_{max} and Q_{max} into account. We revisit the tests we have done before for our validation in the order we have done them before.

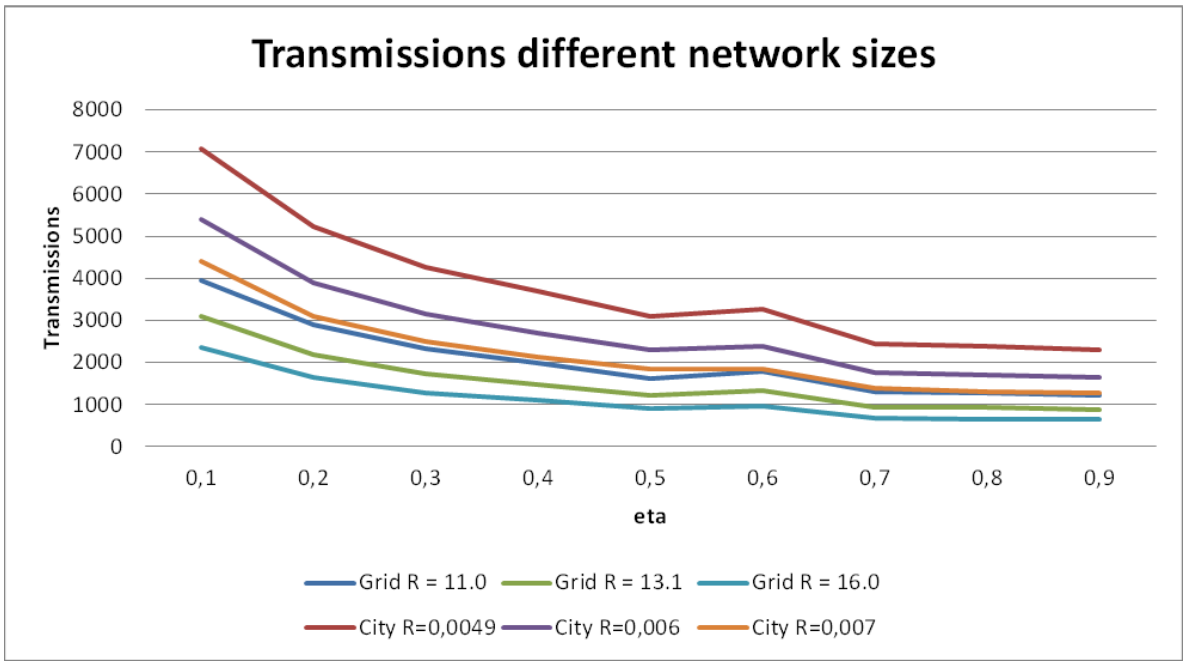
We have simulated a single cell network of 1000 nodes. The results are shown in Figure 4.0.3. We see that the algorithm stops working for large values of M. In Figure 4.0.4 we see the time of each cycle that is spent transmitting. We see that the network is continuously broadcasting for $M = 1$ or $\frac{\tau_{max}}{16}$. Note that this is not a permanent situation for high values of η since no new transmissions are scheduled when $\eta\tau_{max} > kM$. The network is heavily dependent on the way the network is initialized. We expect the network to stabilize after a long time if no new transmissions are scheduled, i.e. if $\eta\tau_{max} > kM$. For smaller values of M as shown in Figure 4.0.3a we see the network behaves the same as we have seen in our validation.

To test propagation mode we have simulated a line network of 100 nodes for different values of M. The results are shown in Figure 4.0.5. The results for the hop count are shown in Figure 4.0.5a. As we have seen before with the validation tests, our results on the hop count are a little bit below what is predicted. This is caused because the prediction was based on the case where $k = 1$ and in the simulation $k = 5$ was used. However the value of M has no significant effect on the hop count.

The results for the propagation speed are shown in Figure 4.0.5a. The prediction shown in this graph is made for a constant $\eta = 0.5$, other values of η are not taken into account. If we look at the place where $\eta = 0.5$ we see the value is a bit below the predicted value. The hop count was also below the expected value so this is to be expected. We see the results for values of η behaves linearly with η.

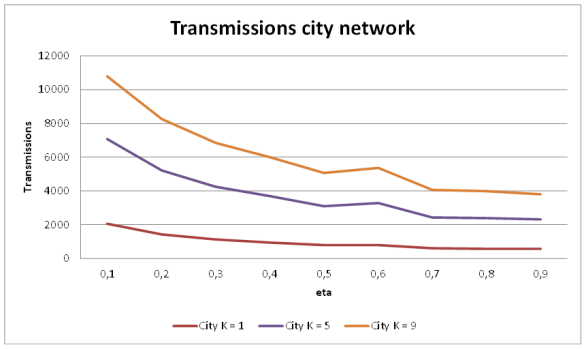


(a) Data is shown in table 6.2.3.

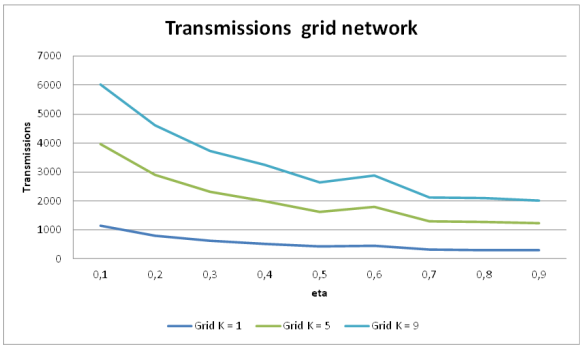


(b) Data is shown in table 6.2.4.

Figure 4.0.1: The transmission rates of our simulation of the city network and grid network.

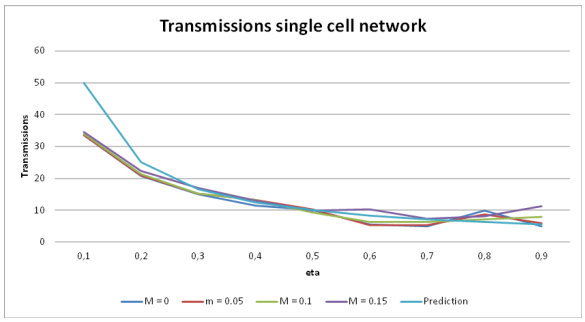


(a) Simulated city network.

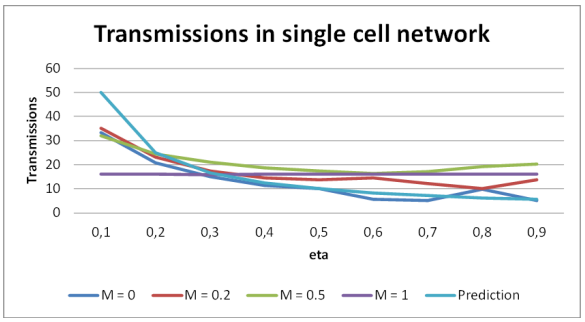


(b) Simulated grid network.

Figure 4.0.2: Transmission rates measured in city and grid networks. Data is shown in 6.2.3.



(a) Data is shown in table 6.2.5.



(b) Data is shown in table 6.2.6.

Figure 4.0.3: Transmission rates of a single cell network.$n = 1000$ and $k = 5$.

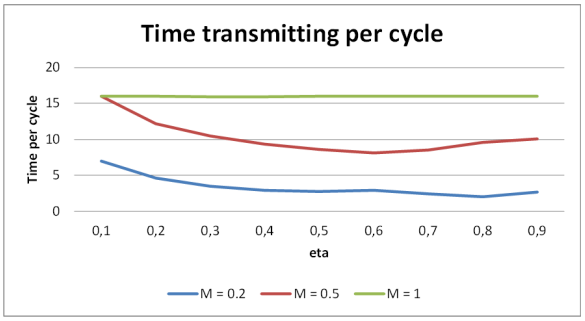
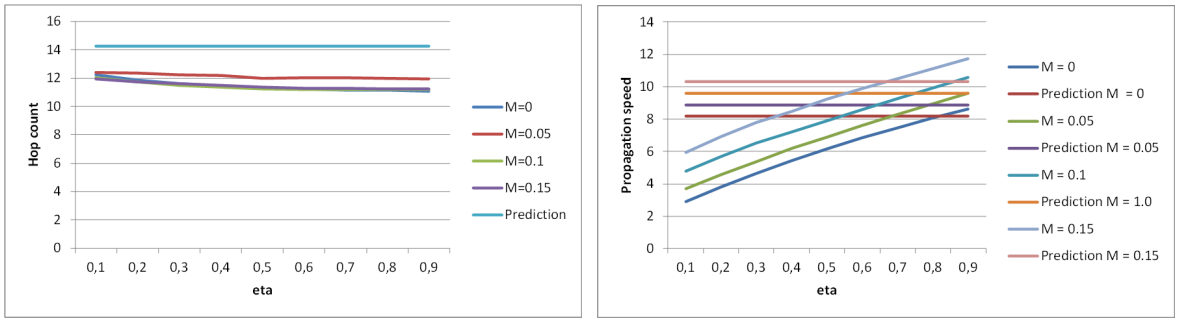


Figure 4.0.4: Time spend transmitting each cycle. $n = 1000$ and $k = 5$. Data is shown in 6.2.7.



(a) $k = 5$, data is shown in table 6.2.1.

(b) $k = 5$ and $\eta = 0.5$. Data is shown in table 6.2.2

Figure 4.0.5: Results of propagation speed and hop count in a line network.

CHAPTER

FIVE

CONCLUSIONS AND RECOMMENDATIONS

We have seen the topology of the network is highly important for the behaviour of the algorithm. We have seen that networks with the same number of nodes with the same average number of neighbours still produce very different results. Apart from the scale the behavior of the transmission rates for different values of k and η stayed roughly the same.

Our second research question remains unanswered because we could not do a comparison of this data set and a known network. We were unable to find a way to make a fair comparison between the different networks.

When we compared the results from [5] in situations with transmissions lengths larger than 0 we noticed some predictions holding up and some failing. Most notably the algorithm stops working as intended for large values of M, around $M = \frac{\tau_{max}}{16}$ we see the nodes are filling up all available bandwidth. If $\eta\tau_{max} > kM$ holds we expect the network to find a stable point after a long time.

The hop counts remain mostly unaffected by the changing transmission rates. Finally we have seen the prediction for propagation speed hold up for different transmission rates. The propagation speeds behave linearly to different values of η.

5.1 RECOMMENDATIONS

In order to be able to make a better comparison for city networks a good idea would probably be to look for different networks with a distribution of the numbers of neighbours that is closer to our data set.

The way we worked with hop count and propagation speed, i.e. the amount of hops and the time it takes to traverse from a given node to another given node, is hard to measure in a large dataset with no additional information. Future research could focus on the average time spent in propagation mode from different starting nodes.

As mentioned previously: from the results on the propagation speed for different values of η it might be likely a simple formula can be found to predict the behavior in propagation mode for general η.

CHAPTER

SIX

EXTENDED DATA

6.1 RESULTS OF VALIDATION

k	1	2	3	4	5	6	7	8	9
$n = 50$	2	4.82	7.12	9	11	13.26	16	17.39	18.93
$n = 1000$	2.96	5.95	9.01	11.98	14.88	17.9	20.77	23.64	26.44
Predicted	3.33333	6.66666	10	13.3333	16.6666	20	23.3333	26.6666	30

Table 6.1.1: Transmission rates measured in single cell networks in maintenance mode. This is the data used in Figure 3.4.1a. $\eta = 0.3$.

k	1	2	3	4	5	6	7	8	9
$n = 50$	1	2	3	4.01	5	6.04	7.01	8.17	9.25
$n = 1000$	1.1	2.2	3.3	4.4	5.5	6.6	7.7	8.8	9.88
Predicted	1.11	2.22	3.33	4.44	5.56	6.67	7.78	8.89	10

Table 6.1.2: Transmission rates measured in single cell networks in maintenance mode. This is the data used in Figure 3.4.1b. $\eta = 0.9$.

η	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
$n = 50$	17	13.9	11	9.66	8.23	7	6.43	5.6	5
$n = 1000$	33.48	20.54	14.88	11.52	9.5	7.75	7	6	5.5
Predicted	50	25	16.66	12.5	10	8.33	7.14	6.25	5.56

Table 6.1.3: Transmission rates measured in single cell networks in maintenance mode. This is the data used in Figure 3.4.1c. $k = 5$.

6.2 RESULTS OF SIMULATIONS

N	10	20	30	40	50	60	70	80	90	100
Hop count $R = 1$ measured	9	19	29	39	49	59	69	79	89	99
Hop count $R = 1$ predicted	10.00	20.00	30.00	40.00	50.00	60.00	70.00	80.00	90.00	100.00
Hop count $R = 5$ measured	2	5	7	9	12	14	16	19	21	23
Hop count $n = 5$ predicted	2.73	5.45	8.18	10.91	13.64	16.36	19.09	21.82	24.55	27.27
Hop count $R = 9$ measured	1	3	4	6	7	8	10	11	12	14
Hop count $n = 1$ predicted	1.58	3.16	4.74	6.32	7.89	9.47	11.05	12.63	14.21	15.79

Table 6.1.4: Hop counts measured in line networks. This is the data used in Figure 3.4.3a.

n	10	20	30	40	50	60	70	80	90	100
Hop count $R = 1$ measured	7	14	21	29	36	43	51	58	65	72
Hop count $R = 1$ predicted	7	14	21	28	35	42	49	57	64	71
Hop count $R = 5$ measured	2	3	4	6	7	9	10	11	13	14
Hop count $n = 5$ predicted	1	3	4	5	6	8	9	10	12	13
Hop count $R = 9$ measured	0.7483	2	2	3	4	5	5	6	7	8
Hop count $R = 1$ predicted	0.66989	1	2	3	3	4	5	5	6	7

Table 6.1.5: Propagation speed measured in line networks. This is the data used in Figure 3.4.3b.

n	10	20	30	40	50	60	70	80	90
Transmission rate average	22.91	85.86	187.95	328.17	511.4	732.55	990.94	1287.38	1626.59

Table 6.1.6: Transmission rates measured in grid networks. This is the data used in Figure 3.4.2a.

R	1	2	3	4	5	6	7	8	9
Transmission rate average	2500.06	509.68	230.52	143.57	101.92	72.32	57.38	46.06	37.89
Transmission rate expected	2500	625	277.78	156.25	100	69.44	51.02	39.06	30.86

Table 6.1.7: Transmission rates measured in grid networks. This is the data used in Figure 3.4.2b.

R	1	2	3	4	5	6	7	8	9
Hop count measured	9	5.01	3.46	3	2.30	2.08	2.00	2	1
Hop count predicted	10	6	4.29	3.33	2.73	2.31	2	1.76	1.58
Propagation speed measured	6.60	3.45	2.34	1.93	1.51	1.34	1.27	1.24	0.75
Propagation speed predicted	7.07	3.54	2.27	1.65	1.29	1.05	0.88	0.76	0.67

Table 6.1.8: Hop counts and propagation speed measured in line networks of 10 nodes. This is the data used in Figure 3.4.4a and Figure 3.4.4b.

R	1	2	3	4	5	6	7	8	9
Hop count measured	99	51.92	36.67	28.48	23.31	19.75	17.17	15.18	13.62
Hop count predicted	100	60	42.86	33.33	27.27	23.08	20	17.65	15.79
Propagation speed measured	72.46	35.70	23.91	17.82	14.13	11.66	9.90	8.59	7.57
Propagation speed predicted	70.71	35.36	22.73	16.50	12.86	10.49	8.84	7.63	6.70

Table 6.1.9: Hop counts and propagation speed measured in line networks of 100 nodes. This is the data used in Figure 3.4.4c and Figure 3.4.4d.

η	M=0	M=0.05	M=0.1	M=0.15	Prediction
0,1	12,22	12,42	12,01	11,97	14.29
0,2	11,85	12,36	11,76	11,75	14.29
0,3	11,63	12,25	11,49	11,60	14.29
0,4	11,44	12,19	11,37	11,52	14.29
0,5	11,28	12,00	11,25	11,37	14.29
0,6	11,25	12,02	11,21	11,30	14.29
0,7	11,17	12,02	11,19	11,28	14.29
0,8	11,15	11,98	11,22	11,26	14.29
0,9	11,10	11,93	11,22	11,24	14.29

Table 6.2.1: Hop counts measured in a line network of 1000 nodes, $R = 10$ and $k = 5$. This is the data used in Figure 4.0.5a.

η	M = 0	Prediction M = 0	M = 0.05	Prediction M = 0.05	M = 0.1	Prediction M = 1.0
0.1	2.89	8.18	3.70	8.89	4.80	9.61
0.2	3.79	8.18	4.57	8.89	5.71	9.61
0.3	4.64	8.18	5.37	8.89	6.53	9.61
0.4	5.42	8.18	6.19	8.89	7.20	9.61
0.5	6.15	8.18	6.90	8.89	7.89	9.61
0.6	6.86	8.18	7.60	8.89	8.59	9.61
0.7	7.45	8.18	8.29	8.89	9.26	9.61
0.8	8.09	8.18	8.94	8.89	9.91	9.61
0.9	8.62	8.18	9.60	8.89	10.58	9.61

η	M = 0.15	Prediction M = 0.15
0.1	5.95	10.32
0.2	6.92	10.32
0.3	7.77	10.32
0.4	8.48	10.32
0.5	9.24	10.32
0.6	9.90	10.32
0.7	10.49	10.32
0.8	11.11	10.32
0.9	11.73	10.32

Table 6.2.2: Propagation speeds measured in a line network of 1000 nodes, $R = 10$ and $k = 5$. This is the data used in Figure 4.0.5b.

η	Grid $k = 1$	City $k = 1$	Grid $k = 5$	City $k = 5$	Grid $k = 9$	City $k = 9$
0.1	1149.81	2041.62	3956.91	7078	6026.49	10783.5
0.2	802.71	1428.74	2892.69	5221.85	4622.63	8280.2
0.3	629.36	1119.12	2322.57	4250.62	3732	6844.86
0.4	524.52	934.69	1990.75	3681.33	3241.52	6013.63
0.5	429.48	796	1613.09	3108.38	2629.82	5067.22
0.6	458.25	788.7	1787.15	3273.98	2881.32	5362.52
0.7	321.56	601.4	1291.03	2432.43	2124.62	4047.85
0.8	309.68	562.58	1269.05	2378.24	2087.73	3973.78
0.9	304.65	556.47	1223.47	2305.17	2003.65	3822.67

Table 6.2.3: Transmission rates measured in the dataset and grid networks. The ranges are $R = 11$ in the grid network and 0.049 in the city dataset. This is the data used in Figure 4.0.1a and Figure 4.0.2.

η	Grid $R = 11.0$	City $R = 0.0049$	Grid $R = 13.1$	City $R = 0.006$	Grid $R = 16.0$	City $R = 0.007$
0.1	3956.91	7078	3082.06	5412.02	2369.05	4393.2
0.2	2892.69	5221.85	2185.36	3890.39	1634.93	3105.68
0.3	2322.57	4250.62	1735.3	3143.41	1283.95	2489.77
0.4	1990.75	3681.33	1474.01	2708.07	1091.42	2136.39
0.5	1613.09	3108.38	1209.68	2311.12	915.45	1849.54
0.6	1787.15	3273.98	1334.22	2379.99	968.96	1837.7
0.7	1291.03	2432.43	937.66	1758.76	682.55	1380.34
0.8	1269.05	2378.24	921.37	1702.35	660.79	1304.78
0.9	1223.47	2305.17	886.45	1655.28	640.91	1273.67

Table 6.2.4: Transmission rates measured in the dataset and grid networks. This is the data used in Figure 4.0.1b.

η	$M = 0$	$M = 0.05$	$M = 0.1$	$M = 0.15$	Prediction
0.1	33.46	33.55	33.97	34.48	50
0.2	20.68	20.95	21.32	22.26	25
0.3	14.98	15.23	15.30	16.95	16.67
0.4	11.54	13.22	13.20	13.08	12.5
0.5	10.00	10.25	9.31	9.88	10
0.6	5.62	5.41	6.33	10.24	8.33
0.7	5.03	5.40	6.24	7.36	7.14
0.8	9.80	8.66	7.18	8.09	6.25
0.9	5	5.99	7.92	11.23	5.56

Table 6.2.5: Transmission rates measured in single cell networks of 1000 nodes where $k = 5$. This is the data used in Figure 4.0.3a.

η	$M = 0$	$M = 0.2$	$M = 0.5$	$M = 1$	Prediction
0.1	33.42	35.07	32.00	16	50
0.2	20.71	23.26	24.39	16	25
0.3	14.97	17.51	20.98	15.93	16.67
0.4	11.45	14.58	18.64	15.95	12.5
0.5	10.00	13.81	17.29	16	10
0.6	5.63	14.57	16.25	15.99	8.33
0.7	5.03	12.17	17.02	16	7.14
0.8	9.78	10.13	19.16	16	6.25
0.9	5	13.64	20.18	16	5.56

Table 6.2.6: Transmission rates measured in single cell networks of 1000 nodes where $k = 5$. This is the data used in Figure 4.0.3b.

η	$M = 0$	$M = 0.2$	$M = 0.5$	$M = 1$
0.1	0	7.0132	15.9975	16
0.2	0	4.651	12.193	16
0.3	0	3.5022	10.491	15.93
0.4	0	2.915	9.3215	15.95
0.5	0	2.7612	8.6435	16
0.6	0	2.9136	8.124	15.99
0.7	0	2.4342	8.51	16
0.8	0	2.026	9.582	16
0.9	0	2.7274	10.088	16

Table 6.2.7: Time spend transmitting per cycle. This is calculated by multiplying the data from 6.2.6 by the transmission rate. Measured in single cell networks of 1000 nodes where $k = 5$. This is the data used in Figure 4.0.4.

BIBLIOGRAPHY

[1] Patel, N., Culler, D., & Shenker, S. (2003). Trickle: A self regulating algorithm for code propagation and maintenance in wireless sensor networks (pp. 15-28). Computer Science Division, University of California. http://www.cs.berkeley.edu/~pal/pubs/trickle-techreport.pdf

[2] Levis, Philip, et al. (2008). The emergence of a networking primitive in wireless sensor networks. Communications of the ACM, 51(7), 99-106.

[3] Lin, K., & Levis, P. (2008, April). Data discovery and dissemination with dip. In Proceedings of the 7th international conference on Information processing in sensor networks (pp. 433-444). IEEE Computer Society.

[4] Kulkarni, S. S., & Wang, L. (2005, June). MNP: Multihop network reprogramming service for sensor networks. In Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on (pp. 7-16). IEEE.

[5] Meyfroyt, T. M. M. (2013). Modeling and analyzing the Trickle algorithm. http://alexandria.tue.nl/extra1/afstversl/wsk-i/meyfroyt2013.pdf

[6] Meyfroyt, T. M. M., Borst & S. C. Boxma, O.J. (2014). Data dissemination performance in large-scale sensor networks.

[7] Kermajani, H., Gomez, C., & Arshad, M. (2012). Modeling the Message Count of the Trickle Algorithm in a Steady-State, Static Wireless Sensor Network. Communications Letters, IEEE, 16(12), 1960-1963.

[8] Becker, M., Kuladinithi, K., & Görg, C. (2012). Modelling and Simulating the Trickle Algorithm. In Mobile Networks and Management (pp. 135-144). Springer Berlin Heidelberg.

[9] Werner-Allen, G., Lorincz, K., Ruiz, M., Marcillo, O., Johnson, J., Lees, J., & Welsh, M. (2006). Deploying a wireless sensor network on an active volcano. Internet Computing, IEEE, 10(2), 18-25.

[10] Watteyne, T., & Pister, K. S. (2011). Smarter cities through standards-based wireless sensor networks. IBM Journal of Research and Development, 55(1.2), 7-1.

CHAPTER

SEVEN

INDEX OF VARIABLES

The network has the following properties:

Symbol	Summary
n	The number of nodes in the network.
N	A symmetric boolean $n * n$ matrix where index (i, j) indicates whether node i and node j are connected.
τ_{min}	The minimum cycle length.
τ_{max}	The maximum cycle length.
η	The fraction of the cycle that is waiting period.
k	An integer specifying how many previously heard transmissions it takes to suppress a broadcast
M	The length of one broadcast.
s	The length of the smallest timestep (54 μs).
d_{max}	The maximum number the backoff is extended.
Q_{max}	The maximum number of planned transmissions.
Seeds	A list of nodes and times where and when seed updates will take place.

The network has the following variables:

Symbol	Summary
Nodes	The set of nodes forming the network.
Eventqueue	The queue containing all the events.
Currenttime	The current time of the running simulation.

And each node keeps track of the following variables:

Symbol	Summary
S	A list of all the neighbors this node has.
τ	the current cycle length.
c	A counter, for every transmission with no newer or older software the node increments c by one.
t	The time for the next broadcast.
V	The current software version.
Re	An integer, the amount of neighbors currently transmitting.
T	A boolean that is true when the node is currently transmitting.
I	A boolean that is true when interference is currently occurring.
d	A parameter for the exponential backoff, this integer increases every time a transmission is delayed.
Q	The list of planned transmissions.