Connecting Wireless Sensor Networks Using Gateway Nodes

Sjoerd Langkemper
Vrije Universiteit Amsterdam, The Netherlands

Abstract

Wireless sensor networks consist of many radio-equipped nodes which are limited in resources and power. Nodes that can be reached by radio, possibly through multiple hops, form one network. This master thesis presents a way to connect two or more networks together, using gateway nodes that have a wired connection with each other. This way, it appears to the nodes as if the two networks have a radio connection. We also show through simulations that the system we propose gives a good connection between the networks, but breaks the working of synchronous protocols.

Contents

Chapter 1

Introduction

1.1 Problem statement

Wireless sensor networks consist of many nodes which can communicate with each other over radio. A node is part of a network if it has a connection with at least one other node in that network. In this thesis, we propose a system to connect two wireless sensor networks, so that they form one network.

In the following sections we will further describe the characteristics of wireless sensor networks and what it means to connect them.

1.2 Domain

1.2.1 Wireless sensor networks

Wireless sensor networks consist of many nodes with limited resources. A typical node has 64 KB of memory and a processor which runs at 16 MHz. When designing software for wireless sensor nodes, these limited resources must be kept in mind.

These nodes communicate with each other using a radio, which has typically a range in the order of meters. Nodes that can reach each other by radio, possibly through multiple hops, are said to be in the same network.

Wireless sensor nodes typically run on batteries. This means that the node can use a limited amount of power, to avoid draining the batteries. When designing software for sensor nodes, power usage must be taken into account. This usually means doing as few as possible power-intensive tasks, such as radio transmissions or writing flash memory.

To further save battery power, wireless sensor nodes usually do not listen for messages all the time. Instead, they receive messages in a few milliseconds, process them and then go to sleep for several seconds to save power. One cycle of working and sleeping is called a *dutycycle*. Because messages can only be transferred if one node transmits and another node listens at the same time, the dutycycles of all nodes have to be the same time and take place at the same time. In other words, the dutycycles of all nodes in a network are synchronized.

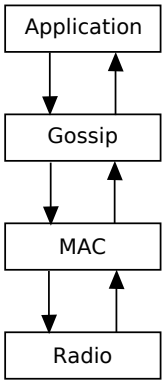

Figure 1.1: Protocol stack in a MyriaNed node.

1.2.2 MyriaNed

The idea for this project originated at a consortium of companies, named DevLab. The companies are working with a particular wireless sensor network, called MyriaNed. All of the companies in this consortium want to experiment with a large network, eventually consisting of up to 10,000 nodes. They can achieve this by connecting their networks in order to make a virtual big network to experiment with.

The MyriaNed nodes are equipped with a processor and a radio. The processor runs various layers of software, as shown in fig. 1.1. The radio layer controls the radio hardware to send out data. On top of this runs the MAC layer.

The MAC layer corresponds to the data link layer in the OSI model [Osi96]. It offers a service that can broadcast and receive messages. An important part of this is to control the timing of the dutycycles. If the MAC layer would just pass messages to the radio layer as it got them, chances are that the message will be sent out but no other nodes is listening for it. Therefore, the MAC layer synchronizes its dutycycle with the surrounding nodes. When all nodes do this, the whole network will eventually be synchronized.

When considering two separate networks, the dutycycles of both will not be at the same time. When connecting them by bringing them in each other's radio range, the MAC layer will try to synchronize both networks. However, in this thesis we consider the situation where we connect the two networks while they are not in each other's range. This means that another way has to be found to synchronize the two networks, or the fact that the two networks are not synchronized has to be taken into account when passing messages.

On top of the MAC layer runs the gossip layer, which corresponds to the transport layer [Osi96]. Whereas the MAC layer can only send messages to the direct neighbours, the gossip layer distributes messages over the whole network. Furthermore, it notifies the application layer only of new messages it sees, not of all messages. We will look further into the gossip protocol in chapter 3.

On top of the gossiping layer is the application layer. The application layer

implements the task of the network, which usually consists of reading sensor data, processing it and passing it to the gossip layer.

1.3 Problem definition

1.3.1 Motivation

The companies participating in DevLab want to experiment with a large wireless sensor network. The goal is to let the network consist of 10,000 nodes. Instead of constructing such a network as a whole, the DevLab community has chosen to make several smaller networks and connect these with each other. Each company then has a subset of a large network.

An advantage of this is that distributing the network among the companies makes it easy to share costs and responsibility. Because 10,000 nodes are expensive, the companies split the costs of the network. This means they purchase a number of nodes, which form part of the network. They can use the nodes to their liking, but they are also responsible for managing and repairing the nodes. Since the company has easy physical access to the nodes, managing and using the nodes is easy. This distributed approach makes it easy for every participating company to access the network, which is an advantage both when using the network and when managing it.

1.3.2 Goal

Our goal is to connect the separate networks at the different companies so that they will form one big network. Let's simplify the problem by saying we want to connect two networks. These two networks have no radio connection to each other. Nodes in one network cannot send a message to any node in the other network, and vice versa. We would like to change that: if a path exists from any node in one network to any node in the other network, the two are connected. Then, all nodes can reach each other and thus form one network. Thus, to connect the two networks, we need to make it possible for at least two nodes to send messages to each other.

To connect the two networks, it must be possible for messages to travel either way between the two networks. In other words, a one-way connection is not enough, because this makes part of the network unreachable from certain nodes. In principle, two one-way connections suffice, one in either way. However, it is attractive to use one bidirectional connection, because this means only one connection has to be made instead of two. Therefore, we will use a bidirectional connection to connect the two networks.

1.3.3 Transparency

Apart from just connecting the networks, we want to connect them transparently. The primary use of the network is to experiment with it and explore its characteristics. Because of this, the joined networks must behave as one network.

From the viewpoint of a node, the two situations must not be distinguishable. From the information it can gather from the network, it must not be able to tell whether it runs on a network which consist of multiple connected parts.

From the viewpoint of the user of the network, any protocol which can be run on a normal network should be able to run on a connected network. Furthermore, it should give the same results.

The system which connects the two networks may use a number of nodes. When connecting two networks of 10 nodes to each other, the resulting network may have more than 20 nodes. It is desirable that the resulting network acts like a normal network of the same size. Because the parts that are connected already are like a normal network, this task lies especially on the nodes introduced by the connecting system. It is desirable that these nodes have similar characteristics as normal nodes in terms of number of neighbours, message loss, speed, sensor data, etc. If this is the case, any measurement done on the connected network will differ little from the results of a normal network. Otherwise, the system may influence measurements done on the network. If the system loses a large number of messages, for example, any measurement of message loss in the large network will give inaccurate results.

1.3.4 Limitations

Above we stated that two connected networks should act as one normal network. However, in some ways this is just not possible. Consider for example a situation in which the two connected networks are far apart (at least outside radio range) and each node carries a position sensor. In a large network, one could assume that the distance between two nodes is at most the range of the radio. In a connected network, this no longer holds. This means that the connection is not transparent anymore: protocols which assume a limited distance between nodes will break and nodes can easily detect the connection between the two networks by reading sensor data of their neighbours.

Even if the nodes are not position-aware, the application may depend on the nodes being close to each other. For example, measuring light to determine whether it is day or night works in a normal network, but not when it is spread out over hundreds of miles, or even on two sides of a mountain.

1.3.5 Other applications

Our system makes it possible to connect wireless sensor networks to each other, in order to experiment with them. However, the system can also be used for other purposes. In this section, we look into some alternative uses.

For example, the system can be used to connect a PC with a remote network. This makes it possible to retrieve data from a remote wireless sensor network. Although our system can be used for this, retrieving data from a remote network can be done in a much simpler manner. When retrieving data from a network, transparency is not needed. A simple radio gateway would suffice and the network would not change at all.

When a client wants to retrieve data from multiple networks, it is possible to connect all these networks and query them once. However, a simpler solution exists. After all, the networks can be queried apart from each other. In this case, the client can connect to each network to retrieve data. The client does not

send data itself, it has little timing constraints and the networks do not need to know of each other. This makes the situation much simpler. In section 2.1 we will look into some related work which makes it possible to query multiple networks.

When several networks have the same function but are geographically dispersed, it may be useful to connect them. [Hac06] gives an example: Wireless sensor networks can be used in cargo tracking applications. It is conceivable that one wants to track cargo in several warehouses that are distant enough that no radio connection between them is possible. In this case, the cargo tracking networks in several warehouses can be combined, so that information on all cargo can be retrieved from any warehouse.

An advantage over one network instead of several is that management becomes easier [Dai04]. For example, by connecting many networks with the same task, a code update has to be inserted in just one network instead of several.

Instead of connecting nodes in separate networks, it is also possible to connect two places in one network. In this case, a *wormhole* is created between two locations in the network. This can speed up propagation because it makes the distance between the two locations much shorter. Contrary to the uses above, transparency is needed for this particular use. After all, creating a wormhole is just like connecting two networks together, except that it happens to be the same network.

[Hu03] and others have discovered that a wormhole can be used to do a security attack on wireless sensor networks. An attacker can set up a wormhole and allow route requests through it. This way, all nodes will send their data through the wormhole, since that is the fastest connection. The attacker can then read, manipulate or discard the messages.

Although this security attack has little use when using wormholes in a normal way, research related to this security attack can be interesting. For example, [Wan04] proposes a way to visualize wormholes in wireless sensor networks. Originally a way to defend against the described security attack, it can also be used as a monitoring tool when using wormholes to speed up a network.

Chapter 2

Related work

2.1 Data retrieval

As described in the previous section, retrieving data from a remote network is much simpler than connecting two networks. However, in both cases an interface is needed to pass data from the wireless sensor network elsewhere. Furthermore, some systems exist that retrieve data from multiple networks, which can be an alternative to our system in some cases. Therefore, we will look into research on data retrieval.

Suppose a client is interested in data from a sensor network. The client wants to make a connection over a network (e.g., the Internet) to retrieve the data. This is not straightforward, because the sensor network may use other protocols and needs a gateway to connect to the internet. There are basically three ways to implement this:

- a proxy in the sensor network stores data that the client may query
- the sensor network runs TCP/IP and the client can connect directly to any node
- a protocol gateway makes it possible to translate messages between the two networks

A proxy is part of the wireless sensor network and stores data it receives in a database. When a client connects, it can query the database. This has as advantage that data is directly accessible, because the proxy has already aggregated the data. However, the proxy would need more resources and more power to fulfill its function [Dun04, May06].

If the sensor network runs TCP/IP, the client can make a direct connection to any node in the network and retrieve data this way. We will look further into this in section 2.2.

When using a protocol gateway, messages are translated between the two networks. At the gateway, messages are simply forwarded, but the underlying protocols are changed. When the client wants to send something to the sensor network, for example, the gateway strips the TCP/IP headers and sends the message in a format the sensor network can understand.

The easiest form of this is simply wrapping sensor network messages in TCP/IP messages, as is done in [Dai04]. Messages from the sensor network are put in a TCP/IP message in whole. This means translating between protocols is now simply sending and receiving TCP/IP messages. No knowledge of the protocol used in the sensor network is necessary. [Dai04] claims this can be used to connect multiple wireless networks, but does not specify this any further. We will use this method to pass data between the two networks, because it is simple and we need the messages passed unaltered.

By expanding the systems described above, it is possible to retrieve data from multiple networks in an easy way. Several systems allow a client to query multiple networks in a transparent way: the topology and number of networks is hidden from the client. In some cases, this may be an alternative to connecting the networks with the system proposed in this thesis.

In [Shn04], [Abe06] and [Lei06], systems are proposed which make it possible to query multiple networks for information. By distributing the query to several networks and combining the results, it looks to the client as if it was querying one network.

The GSN system proposed in [Abe06] for example, makes it possible to query a set of wireless sensor networks. It proposes an abstraction method for sensor data, so that sensor data can be queried no matter what the underlying implementation is. Furthermore, it proposes a sensor network registry, in which networks can register themselves. When a query is done, it is sent to all registered networks, making the distinction between the separate networks invisible for the user. Hourglass [Shn04] and VIP bridges [Lei06] work in the same way in that they abstract the query process and send the query to all networks.

In the aforementioned systems, the nodes are able to answer basic queries. In contrast, Agimone [Hac06] does not assume the nodes are capable of this but makes it possible to run software agents on any node. These agents can query sensors and store their results in tuple spaces. These agents can migrate to other nodes, optionally retaining program status. They can also migrate to other networks. By passing through a number of nodes, they can aggregate data from these nodes. This data can be stored in one of the tuple spaces, making it possible for other agents to read it. This allows for a powerful method of aggregating data from multiple networks. However, since Agimone uses the wireless sensor network in an unconventional way, programming new programs or porting existing programs may be hard.

2.2 TCP/IP overlay

As shown in [Dun03], it is possible for nodes with limited resources to run the TCP/IP protocol. If the sensor network runs TCP/IP, it is straightforward to connect several networks to each other. Messages can be sent as easy to nodes in the local network as over the Internet to another wireless sensor network. This also means that a client can access individual nodes, making data retrieval more flexible. Furthermore, the gateway nodes would be very simple, only translating from wired to wireless medium. Finally, it would be possible to profit from existing software and research on TCP/IP [May06].

However, TCP/IP has a large overhead and behaves badly over wireless

links [Dun04]. Furthermore, TCP/IP had an address-centric paradigm: the client must know the host that contains the data it wants. However, wireless sensor networks are data-centric: the client wants to query the whole network on its data. Instead of getting the temperature at a specific node, one may want to query where the nodes are where the temperature exceeds a specific threshold [Zúñ03].

Currently, MyriaNed has no routing or addressing scheme, making the implementation of TCP/IP hard. Because of this, its bad performance and its address-centric paradigm, using TCP/IP on the sensor networks would not be a viable solution.

Chapter 3

Gossiping

Although the system that is proposed in this paper does not require it, we assume that the protocol that runs on the nodes is a *gossip* protocol. A gossip protocol is a type of *epidemic protocol*, named that way because information in an epidemic protocol spreads very much like an infection.

3.1 Epidemic protocols

In an epidemic protocol, each node regularly chooses another node and exchanges information with it. This typically means that a node transfers all of its data to the other node and adds anything it gets to its own data. Because all nodes keep passing information to each other, all nodes get all information eventually [Pit87]. As some nodes may receive the same data multiple times, epidemic protocols are not the most efficient in terms of resource usage. However, this also makes them fault-tolerant. Epidemic protocols scale well because of their decentralized nature and are used in database replication, peer-to-peer networks and routing protocols [Dem87, Vou03, Haa02].

3.2 Traditional gossip

Gossip distinguishes itself from other epidemic protocols by keeping the coverage of the information into account when spreading messages: when a node receives a new piece of information, it becomes a "hot rumor". It tells other nodes about the new information, but it will stop when it finds that many nodes already know the rumor. The terminology is not precise: epidemic protocols which do not have this property are sometimes also called gossip protocols.

The goal of a gossip protocol is to spread a message to the whole network. Instead of guaranteeing that the message arrives at every node, gossip protocols only give a high chance of spreading a message, due to their probabilistic nature. This also makes them much simpler and robust.

In [Haa02], an epidemic protocol is laid out that spreads a message to most nodes in a network. Like all epidemic protocols, it has a probabilistic behaviour. When a node gets a message, it forwards this message to all its neighbours with some probability p. Because not all nodes forward the message, the protocol is

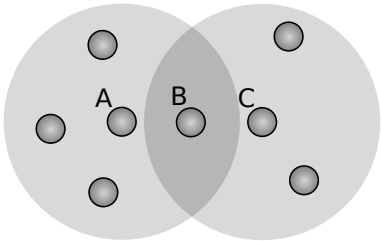

Figure 3.1: In some cases, broadcasting is more efficient than routing.

more efficient than simply flooding the network. With the right parameters for this protocol, it can be shown that there is a high chance that most of the nodes receive the message most of the time.

Instead of spreading one message, the newscast protocol [Vou03, Jel02] focuses on spreading several news messages over the network. Every node keeps a cache with messages. Once in a while, it selects a random node and does a cache exchange with it: it gets the other's cache and merges the cache with its own. Then, it sends its own cache, so the other node can do the same. By exchanging messages this way, the messages get to an increasing number of nodes.

A simple way of selecting a random node in a network works by keeping a list of all nodes, and selecting randomly from this list. This would not scale well to large networks, because constructing and maintaining the list would be a lot of work. Therefore, the nodes in the newscast protocol keep only a subset of nodes. This subset is changed when two nodes exchange their caches, so that the messages are spread to as much nodes as possible.

3.3 Wireless gossip

Nodes in a wireless sensor network are unreliable, as well as the wireless links between them. If many nodes fail, the network must still perform its task. With this in mind, gossip protocols are well suited to run on wireless sensor networks. Gossip protocols are decentralized, making it no problem when nodes fail. Furthermore, they are resistant to link failure. In wireless sensor networks, it is desirable to spread the usage of resources equally over all nodes. If some nodes are doing more work, their batteries drain faster, eventually disabling them. Therefore, a decentral protocol like gossip is well suited for this environment.

As we described in the previous section, some gossip protocols route messages to random nodes in the network. This is inefficient in wireless sensor networks. Consider for example the setting in fig. 3.1. Suppose node A randomly picks node C. These nodes do not have a direct radio link, they depend on node B to route their messages. A sends its message to B, which in turn sends it to C.

In this setting, all neighbours of node A and all neighbours of node B have received the message, but ignored it because it was not meant for them. All of these nodes must be listening, because A may send something to them.

Since listening takes a considerable amount of power, this method is not very efficient.

A better option is for A to simply broadcast the message instead of sending it to some remote node. If A broadcasts it, all neighbours of A get the message. This is much more efficient: every node which overhears the message stores it. The next round, each of the neighbours of A may broadcast the message again. This way, the message will propagate through the whole network.

In the newscast protocol, amongst others, the nodes *exchange* information. This means that when two nodes communicate, they receive each other's data. The alternative would be that a node only sends data to another node and receive nothing in return. Some protocols depend on the communication to be bidirectional. For example, consider an algorithm which counts the number of nodes in the network. At the start, one of the nodes has a value of 1 as data, the others have 0. Subsequently, each node randomly picks another node, they exchange their data value and both store the average value of the two. The data value of each node would converge to $1/n$, where n is the number of nodes in the network.

When nodes do not have this one-to-one communication, this protocol breaks. If communication is one-way, the nodes can no longer determine the average of the two data values. This means that the network size cannot be measured with this particular protocol.

As described earlier, we want to make use of broadcasting because it is efficient in terms of resource usage. With broadcasting, it is not possible to establish one-on-one bidirectional communication. Therefore, protocols as the one we described above will not work.

3.4 MyriaNed gossip

In the MyriaNed gossip protocol, all nodes have a cache. The cache can store a limited amount of messages. Every dutycycle, a node chooses a random message from its cache and broadcasts it. As its neighbours do the same, it will receive a number of messages. Of these messages, it discards those it already has in its cache. It randomly picks one of the remaining messages and puts it in the cache. If the cache is full, it overwrites a random other message. Fig. 3.2 shows the protocol in pseudocode.

The MyriaNed gossip protocol uses broadcasting as an efficient way to spread messages. By broadcasting random cache entries, messages get to most nodes in a reasonable amount of time.

```
itemToSend = random item from cache
radio.broadcast(itemToSend)

receiveBuffer = radio.receive()
for each item in receiveBuffer {
  if item is in cache {
    delete item from receiveBuffer
  }
}

item = random item out of receiveBuffer
place = random place in cache
cache[place] = item
```

Figure 3.2: Main loop of simulation program in pseudocode.

Chapter 4

Topology

4.1 Network interface

Our goal is to connect two networks, so that a node in one network can reach a node in another network, possibly through multiple hops. To do this, data has to be transferred between the networks. Since the networks are not in radio range, normal nodes cannot do this. A special kind of node is needed, which we will call a *gateway node*. In addition to a radio, a gateway node has an interface with which it can make a long-range connection to a gateway node in another network. We will call this connection a *tunnel*. A pair of gateway nodes connected with a tunnel is called a *gateway*.

In our setup, the tunnel consists of several steps: a normal node is equipped with a USB interface and connected to a PC. This PC makes a TCP/IP connection to another PC, which also has a gateway node. Messages from each gateway node are sent directly to the other gateway node, without altering them. The PCs are currently connected by a local network, but may connect over the Internet in the future.

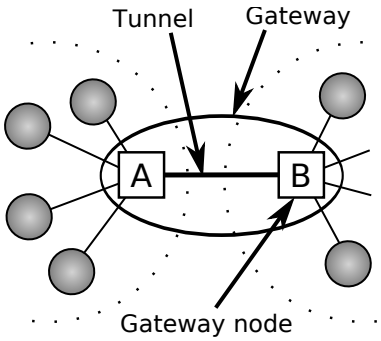

Figure 4.1: A gateway consists of two gateway nodes connected by a tunnel.

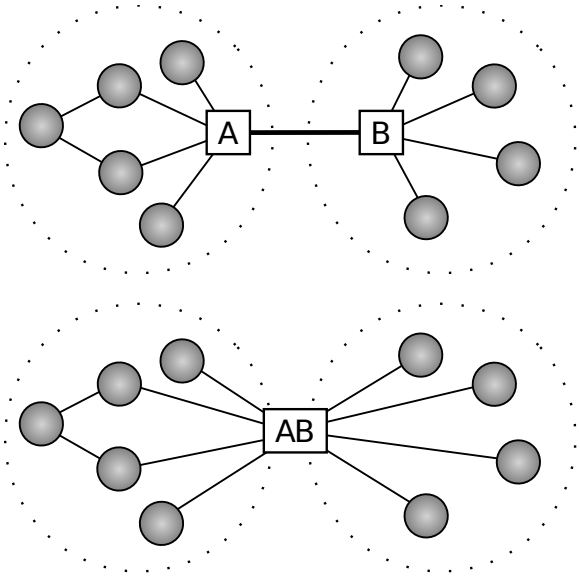

Figure 4.2: Two networks connected with gateway nodes. Above the actual topology, below the conceptual situation.

4.2 Connection methods

4.2.1 Neighbouring gateway nodes

When connecting two gateway nodes, there are several ways they can be fit in the topology of both networks. A straightforward solution would be to let the two gateway nodes be neighbours. Instead of a radio link, the gateway nodes would use the tunnel to communicate with each other. Anything sent over the radio is also sent over the tunnel. Anything received from the tunnel is handled as if it came from the radio.

Letting the gateway nodes be neighbours would work, but it would not connect the networks very tightly. As we stated earlier, the gossip protocol picks one received message at random and puts it in its cache. If the gateway node has most of its neighbours in its own network, it will more likely pick one of these messages. Therefore, its own network gets an advantage.

For example, because gateway node A in fig. 4.2 has four neighbours in its own network, it has a 4/5 chance of using a message from its own network and a 1/5 chance of using a message from gateway node B. Because each gateway node gives its own network an advantage, messages are not passed efficiently.

It is possible to alter the gossip protocol to compensate for this, by giving precedence to the messages coming from the other gateway node. However, this would add another version of the gossip protocol which has to be maintained. It would make replacing the gossip protocol on all nodes difficult, because now not all nodes have the same protocol anymore. Furthermore, by letting the gateway nodes behave differently, they are no longer transparent.

Another disadvantage of this approach is that this requires the tunnel between the gateway nodes to simulate a wireless link, for the situation to be like

a large wireless network. The tunnel could have different properties than a radio link. For example, a radio link could be unreliable while the tunnel loses no messages at all. Any measurement of message loss on the network will give inaccurate results, unless the tunnel simulates a radio link. This would mean simulating collisions, RF noise and all other kinds of artifacts introduced by radio. Doing this accurately is difficult.

4.2.2 Joined gateway nodes

Instead, we chose to let the two gateway nodes emulate one node. As can be seen in fig. 4.2, the two gateway nodes pretend to be one normal node. This conceptual node AB has both the neighbours of A and the neighbours of B. This means that any message that is received on either node A or node B is passed to the gossip protocol layer as if it came from the radio. Any message that the gossip protocol sends is sent out over the radio by both gateway node A and gateway node B.

Above, we explained that gateway nodes have a small chance of passing messages when they are neighbours of each other. If the two gateway nodes emulate one node, chances of passing messages are higher. Provided that the two gateway nodes have approximately the same number of neighbours, both networks are handled equally. The chances for AB to send out a message from either network are about the same. After all, the gossip protocol stores a random message from all its neighbours. If these neighbours are evenly spread across two networks, both networks are selected equally often.

Fig. 4.3 shows the result of a simulation. Simulation details are further explained in section 8.1. The dotted line shows the situation in which the gateway nodes are neighbours, as explained in section 4.2.1. The full line shows the results for the situation where the two gateway nodes emulate one node. The x-axis shows time, the y-axis the percentage of nodes which have a certain message. A steeper curve means propagation is faster, because more nodes get the message in less time. As can be seen, propagation is faster when the two gateway nodes emulate one node.

As we wrote earlier, the gateway has to be transparent. The gateway emulates a part of a network, namely a node with its connections. If it does this well, it means that the gateway is not distinguishable from a normal node. Our goal is thus to emulate one node as precisely as possible. This is feasible with the design explained above, except for a time delay we describe further in chapter 6.

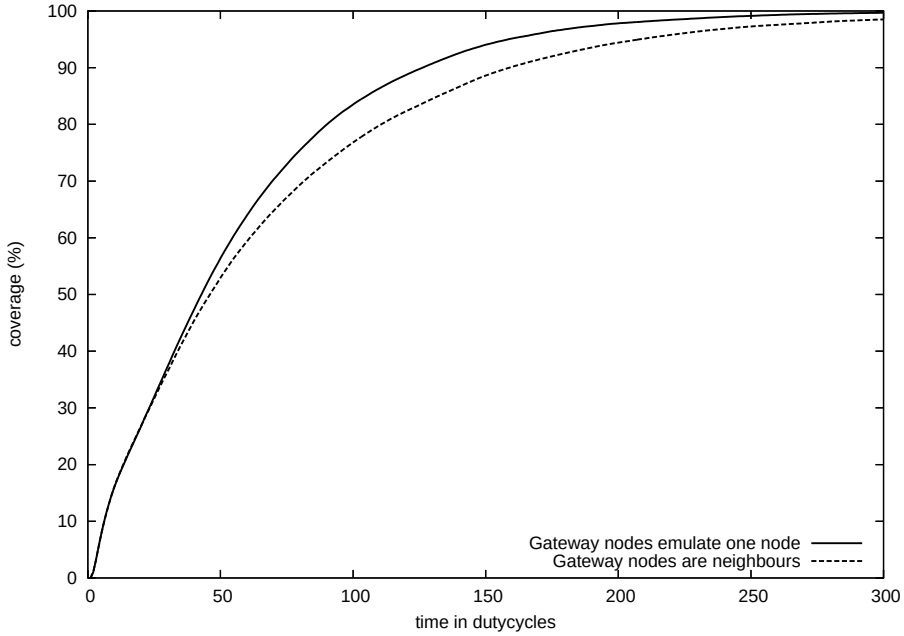

Figure 4.3: Propagation in networks where the gateway nodes are neighbours (bottom line) and where the gateway nodes emulate one node (top line).

Chapter 5

System architecture

5.1 Software placement

Since we want the two gateway nodes to emulate a node, they have to run the application and gossip protocol that also runs on the other nodes. A normal node simply runs the software on its own processor. However, our emulated node is split in two, making it possible to choose on which node to run the software.

5.1.1 Both gateway nodes

One possibility is to run the software on both gateway nodes. However, this will introduce a problem. Remember that for the two gateway nodes to emulate one node, both have to send out the same data. This means that when we run the software on both gateway nodes, both programs must have the same output.

Since most gossip protocols choose messages randomly, two instances of the same protocol will not output the same data. In other words, the software is not deterministic. Therefore, it is not possible to run the software on both gateway nodes.

5.1.2 One of the gateway nodes

The software can be run on one of the gateway nodes. This needs little changes to the software, since the gateway node is much like a normal node: it already runs the software a normal node does. Because the gossip layer and application are the same on gateway nodes and normal nodes, they are easy to maintain and replace.

In this case, one of the gateway nodes runs the software, using the messages from both gateway nodes. We will explain in more detail how this works in section 5.4.

This is the simplest solution and our design indeed has the software on one of the gateway nodes. However, there are other possibilities which have interesting sides, which we will look into next.

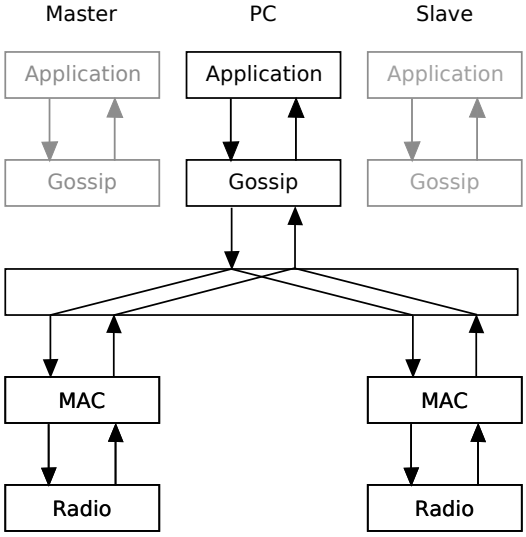

Figure 5.1: Layered overview of an implementation with the gossip protocol and the application run on a PC.

5.1.3 One computer per gateway

Instead of running the software on one of the nodes, it is also possible to let a computer run the software. In this case, all messages are sent to the computer, it runs the gossip protocol and application and the results are sent back to the nodes. This way, the gateway nodes are used only to transmit and receive messages. This makes the gateway nodes simpler, as can be seen in fig. 5.1.

This method would need an extra computer, but in many cases a computer is already present somewhere in the system. In our case, this is one of the PCs that is connected to the gateway nodes. Although running the application on the PC gives more flexibility, the application and gossip layer must first be ported to the PC. In addition, any sensors and sensor drivers need to be ported to the PC. This means that two versions of the software have to be maintained. Because of this, we chose to run the software on one of the gateway nodes.

Another option would be to use a simulator or virtual machine to run the application on the PC without altering it. Although this is much more complicated than simply running the application on the node, it may prove useful in the future. On average, the gateway has twice as many neighbours as a normal node, because it consists of two nodes. This means that twice as much memory is needed to store all received messages. Furthermore, more memory is needed to buffer messages waiting to be transferred to the other gateway node. This could mean that an application runs fine on normal nodes, but fails to run on the gateway because of memory shortage. Obviously, this would break the working of the gateway. In this case, running the application on the PC can be a solution, as can be using hardware with more memory.

5.1.4 Central server

Another option is to use a central server to run the software for all gateways. For example, if three gateways are used to connect four networks, the server would run three instances of the software.

As this method also depends on the software running on a computer, it has the same advantages and disadvantages as described above. Apart from that, the centralized approach makes the system easy to maintain, because only one host has to be maintained. If the software changes, only the software on the central server needs to be updated. At the same time, this centralized approach introduces a single point of failure. This contradicts the nature of a wireless sensor networks, which continue to run even if many nodes fail. A central server may also introduce scalability issues. However, this is not likely because the amount of data transferred by the gateways is minimal and the server has far more resources than a node.

Another problem is when the application makes use of sensor data: because all the gateways have their application running on the server, they all make use of the same or similar sensor data. In a situation where sensor data is aggregated, this may give wrong results. Consider for example a situation where a network is used to monitor the temperature in five freezers. In these freezers, the temperature is somewhere around -18° C and each freezer contains 10 nodes. They are then connected with 10 gateways, which have their software running on a server in a server room where it is 30° C. Reading the average of these measurements will then give a result of -10° C instead of -18° C. A solution to this can be to put no sensors on the gateway nodes. However, the application has to be modified to handle this situation. Furthermore, equipping as many nodes as possible with a sensor reduces the cost per sensor of the network. It is therefore attractive to run the application in the monitored area, so that sensor readings are accurate.

5.2 Master placement

The application is run on only one of the gateway nodes, creating an inequality between them. One of the gateway nodes is the *master*. The master runs the gossip protocol and the application. The other gateway node is the *slave*. The slave only transfers messages from the master to the radio and vice versa.

For the working of the gateway, it is of little influence which of the gateway nodes becomes master and which becomes slave. At the application level it may have a bigger impact. Since the two gateway nodes may have different locations and sensor readings, the application may behave differently according to on which gateway node it is run.

If the tunnel has asymmetrical properties, the choice of which node becomes master may be more important. Consider for example that there is much message loss from the slave to the master, but not the other way around. In this case, messages from the slave to the master are lost. Of course this has as effect that propagation in that way suffers, but it also means that the propagation from the master to the slave becomes better. If no messages from the slave get to the master, it will only send its own messages to the slave. This means that the slave will receive less of the messages it already had. However, in this case

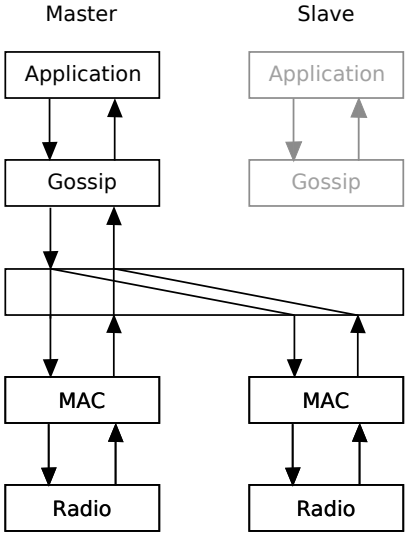

Figure 5.2: Master-slave communication is done between the gossip layer and the MAC layer.

the transparency of the gateway is lost, because it no longer reliably emulates one node.

5.3 Implementation layer

The system should be implemented below the gossip layer, because only one instance of the gossip layer is run, as we described at the start of this chapter. Implementing the communication below the MAC layer is not practical. The MAC layer synchronizes its dutycycles with other nodes. Using one MAC layer for two nodes would need very tight time synchronization. We chose to put the communication between the gossip layer and the MAC layer, as shown in fig. 5.2. This abstracts from the timing issues of the MAC layer, while making it possible to use another gossip protocol.

5.4 Protocol description

The gateway thus works as follows. Each dutycycle, the slave receives messages over the radio from its neighbours. These messages are sent to the master. In addition to these messages, the master also receives messages from its own neighbours. It passes all these messages, both from its own neighbours and from the slaves neighbours, to the gossip protocol. Anything the gossip protocol sends is both broadcast over the radio and sent to the slave. The slave, in turn, broadcasts these messages.

In other words, the slave transfers messages from the radio to the master and vice versa. The master combines the slaves messages with its own and

Master:

```
radioMessages = radio.receive()
slaveMessages = tunnel.receive()
allMessages = merge(radioMessages, slaveMessages)
messageToSend = gossip(allMessages)
radio.send(messageToSend)
tunnel.send(messageToSend)
```

Slave:

```
messageToSend = tunnel.receive()
radio.send(messageToSend)
slaveMessages = radio.recv()
tunnel.send(slaveMessages)
```

Figure 5.3: Working of the gateway in psuedocode.

uses these as the input for the gossip protocol and the application. It sends the output over the radio and to the slave.

Fig. 5.3 shows this process in pseudocode. The code for both the master and the slave is executed each dutycycle. The tunnel is assumed to allow asynchronous communication.

By sending the slaves messages over the tunnel, the gossip protocol gets the messages from both the neighbours of the master and the neighbours of the slave. Any message that is sent by the gossip protocol will also be sent to all these neighbours. This is exactly the situation as it would be when the master and the slave were one node.

5.5 Tunnel properties

As described earlier, the communication is done between the MAC layer and the gossip layer. Therefore, the messages that are transferred over the tunnel do contain information from the gossip layer and above, but not from the MAC layer or below. When these messages are sent over the tunnel, message boundaries must be preserved, so that messages that follow close after each other are not concatenated into one message.

The tunnel passes messages between the two gateway nodes. If a message is lost in the tunnel, the propagation of messages between the two networks suffers. Either the input to the gossip protocol is disturbed, or the output is not sent over the radio. In our analogy where the gateway is one node, this would correspond to message loss inside the node. As this does not usually happen and because message loss has a bad effect on performance, we want the tunnel to be as reliable as possible.

Chapter 6

Timing issues

6.1 Delay sources

It is possible that a delay occurs in the communication between the two gateway nodes. This means messages are transferred slower, which could affect the propagation in the network. In this section, we will look into the various reasons why a delay can appear and what the effects of it are.

6.1.1 Dutycycle offset

Each of the two networks has a dutycycle, in which messages are transferred and the application is run. However, these dutycycles may not start at the same time. This means that the messages from the slave can arrive at any point in the dutycycle of the master. This and other factors are the cause of delays introduced by the gateway, which cause messages to be processed or sent out several dutycycles later than in a normal node.

This offset between dutycycles can have implications on the delay the gateway introduces. Fig. 6.1b shows a situation where the offset between dutycycles gives a delay. The figure shows the communication between the slave and master over time, where the arrows indicate when messages are sent and received. As can be seen, the message from the master arrives just after the dutycycle of the slave. In this case, the message of the master is delayed one dutycycle.

This delay occurs when the message from the master arrives just after the dutycycle of the slave. Had it arrived earlier, then the slave could process the message. Had it arrived later, the master could process new messages from the slave. In both cases, messages go back and forth within one dutycycle.

This delay only happens when the processing and transfer of messages of the master and the slave overlap. There is a small chance of this happening, because the handling of messages takes a lot less time (milliseconds) compared to the dutycycle (seconds). Therefore, there is a bigger chance of a message arriving while a gateway node is sleeping than when it is active.

This does not hold when the gateway connects two places in the same network. Because the two gateway nodes are in the same network, their dutycycles are at the same time. This means the delay explained above will always happen. Because this is not the primary use of the gateway, we did not investigate

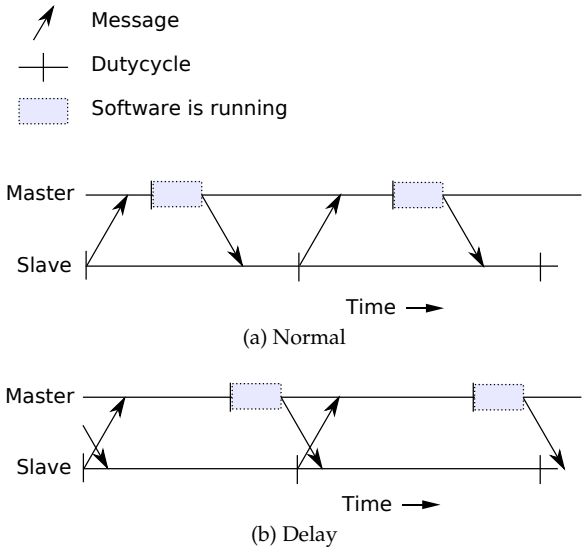

Figure 6.1: Above a normal situation, below a situation where a delay occurs.

this further.

Synchronizing the gateway nodes so that the dutycycles have a specific offset partially solves the problem. However, synchronizing the dutycycles is not trivial and it would not totally solve the problem, because the transfer delays cannot be known in advance. Furthermore, a delay of one dutycycle has little influence on a network, as we shall show later. Therefore, we do not synchronize the dutycycles.

Another solution could be to run the application at a different moment. In the situation in fig. 6.1, the application is run in the dutycycle of the master. However, it can also be run just after the messages from the slave arrive. This would solve the delay in this situation, but introduce it when the application and the dutycycle of the master overlap. Furthermore, if the messages from the slave are delayed or lost, the gossip protocol is not run, which can lead to problems. Implementing this would require more changes to the existing software, because work is done outside of the dutycycle. Because it would give little improvement and would be hard to implement, we chose to run the application in the dutycycle of the master.

6.1.2 Tunnel delay

The tunnel that connects the gateway nodes introduces a time delay. If this delay is constant, messages are simply delayed. However, the variance in this delay (*jitter*) can cause messages to skip a dutycycle and arrive one or several dutycycles later, as shown in fig. 6.2.

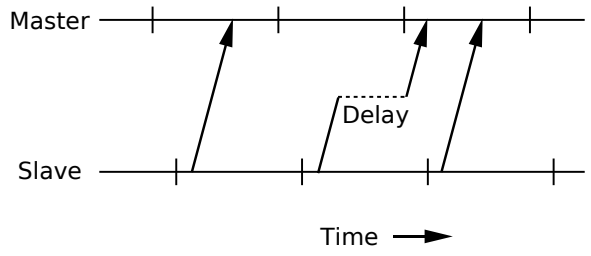

Figure 6.2: A message is delayed one dutycycle due to jitter.

6.1.3 Dutycycle length

This also happens when dutycycles in one network are slightly longer or shorter than in another network. Because of clock drift and synchronization, not all networks have exactly the same dutycycle length. This means that the two networks do not have the same number of dutycycles in a certain period of time. Because of this, it may happen that zero or two messages are received within one dutycycle.

6.2 Implications of delays

6.2.1 Handling delayed message sets

The master and slave transfer messages to each other. We will call a group of messages that are transferred in one dutycycle a *message set*. When a message set is delayed long enough, it will not be received in the current dutycycle. However, it is desirable for the master to run the gossip protocol and send a message. After all, the application may depend on it that it is run each dutycycle. If the master does not send a message, nodes may think it has failed and initiate an error-recovering mechanism. If it does send out a message, it may still take into account messages received from the slaves neighbours, but from earlier dutycycles. Even if nothing is received from the slave, the master will still run the application as expected. However, this means that the input for the gossip protocol will only be the messages received from the neighbours of the master.

When a message set from the previous dutycycle is delayed, it will be received within the current dutycycle. This means it is possible that multiple message sets are received within one dutycycle. One possibility is to put all message sets in a FIFO queue and use one every dutycycle. However, in that case it is possible for the queue to become bigger and bigger. That means that increasingly older messages are used. Furthermore, on the resource-limited nodes there is not enough memory to store many messages. Therefore, we store only one message set. This is the last one we received, in order to use the most recent data available.

6.2.2 Synchronous protocols

In a normal network without gateways, the data that a node sends at dutycycle k is based on the data received in dutycycle $k-1$. In other words, the process of receiving, processing and sending of messages takes one dutycycle. However, because of the tunnel delay this may not be possible within a gateway. After all, it may take more than one dutycycle to transfer information from the slave to the master, run the gossip protocol and transfer it back. Therefore, the gossip protocol should be asynchronous. This means it does not assume that the output of the gossip protocol is based on data received a fixed number of dutycycles before.

Consider for example a synchronous protocol that counts dutycycles. One node will start with a value, say 0. Each dutycycle, it will increase this value by one and broadcast it. Each other node that does not yet have a value will adopt the value it hears first from one of its neighbours. This protocol converges to a state where each node has the same value for its counter and the counter is increased by one each dutycycle.

However, when there is a gateway somewhere in the network where it takes longer than one dutycycle to transfer something, an old value will arrive at the other side. This way, the value is not the same in every node anymore.

6.2.3 Effect on traversal time

Although there are many ways the gateway can introduce a delay of several dutycycles, the implications on a whole network are small. In simulations done on a 100 node network, the delay introduced by the gateway had a negligible effect on the propagation of data throughout the whole network. Fig. 6.3 shows the average number of dutycycles it takes for a data item to arrive at 50% or 90% of the nodes. The x-axis indicates the delay of the gateway, in dutycycles. For the 90% line, a delay in the gateway has 2.1 times the influence on the whole network. This means that when the gateway has a delay of 2 dutycycles, the data items reach 90% of the nodes 4.2 dutycycles later, on average. This is only 3.3% of the total time it takes to reach 90% of the nodes.

In this simulation, we measured the time it took for a data item to reach all the nodes in the network. Altough we took the average of 300 of such measurements, the variation is relatively high: the standard deviation of this average is 7.4%. The delay a gateway introduces is lower than this, which is acceptible.

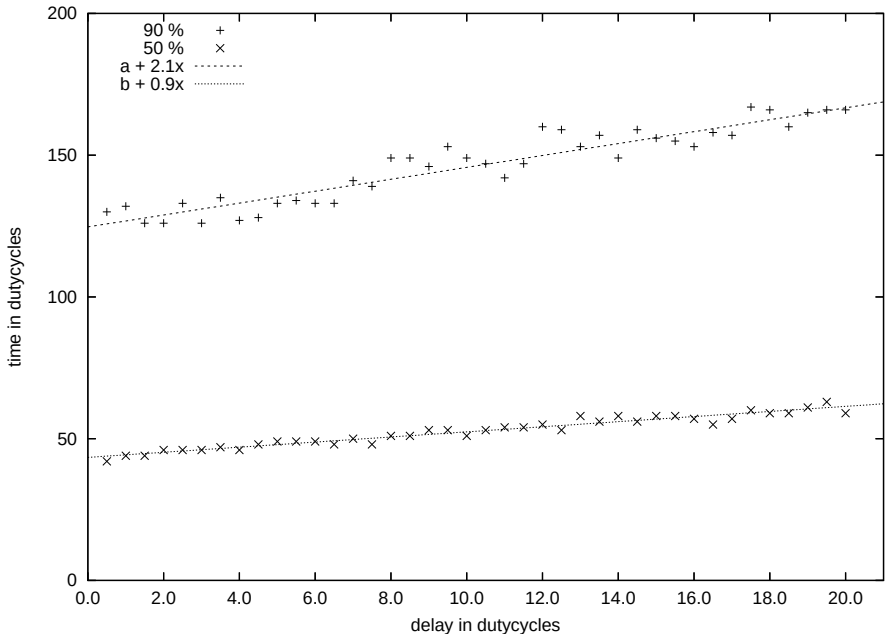

Figure 6.3: Influence of delay in the gateway on the whole network.

Chapter 7

Configuration

When connecting two networks, the placement and number of gateway nodes can be varied. Increasing the number of gateways gives better connectivity between the two networks, reducing the time it takes for a message to travel from one network to the other. Placing the gateway nodes more in the center of the networks results in shorter distances, again increasing connectivity.

7.1 Number of gateways

Shown in fig. 7.1 are the results of a simulation where the number of gateways varies. This simulation is done with two networks of 50 nodes each, connected with a number of gateways. The networks are arranged like a grid, most nodes having four neighbours. In section 8.1, we further describe the simulation environment. The number of gateways is on the x-axis, the time it takes for messages to transfer to 50% or 90% of the nodes is on the y-axis. The horizontal lines show the situation in a normal network, which is the aim for our configuration. As can be seen, adding more gateways has little effect at a certain point.

Fig. 7.2 shows the same simulation, but here most nodes have eight neighbours. As can be seen compared to the situation with four neighbours, messages travel faster over the network. This can be expected, because the diameter of the network is smaller. Fig. 7.2 also shows a smoother line than fig. 7.1: adding a gateway has less an effect when each node has eight neighbours than when the nodes have four.

When the gateway nodes have more neighbours, they receive more messages. Because the gossip protocol will pick one out of these messages, the chance of passing a specific message is reduced. Let's assume that every gateway node receives a certain message. It may put this message in its cache and it may send out this message in the next dutycycle. Lets say that a gateway sends out this message with chance c, which varies depending on the number of neighbours, the cache size and the gossip protocol. The chance that *any* gateway passes the message through in the current dutycycle is $1 - (1 - c)^n$, where n is the number of gateways.

This explains why the line in fig. 7.1, becomes flat relatively quickly. Since $1 - (1 - c)^n$ approaches 1 pretty fast, the benefit of adding an extra gateway

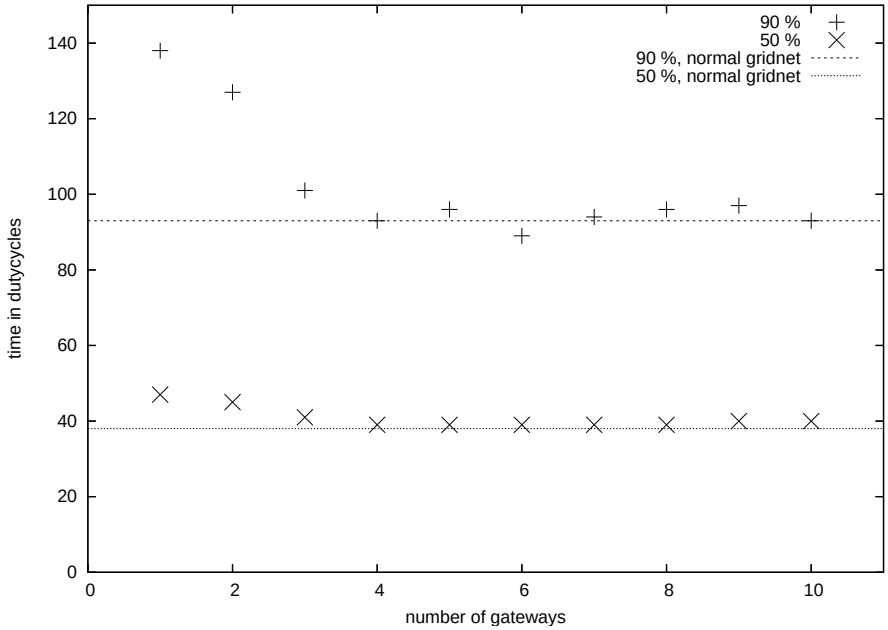

Figure 7.1: Influence of the number of gateways on traversal time (4 neighbours).

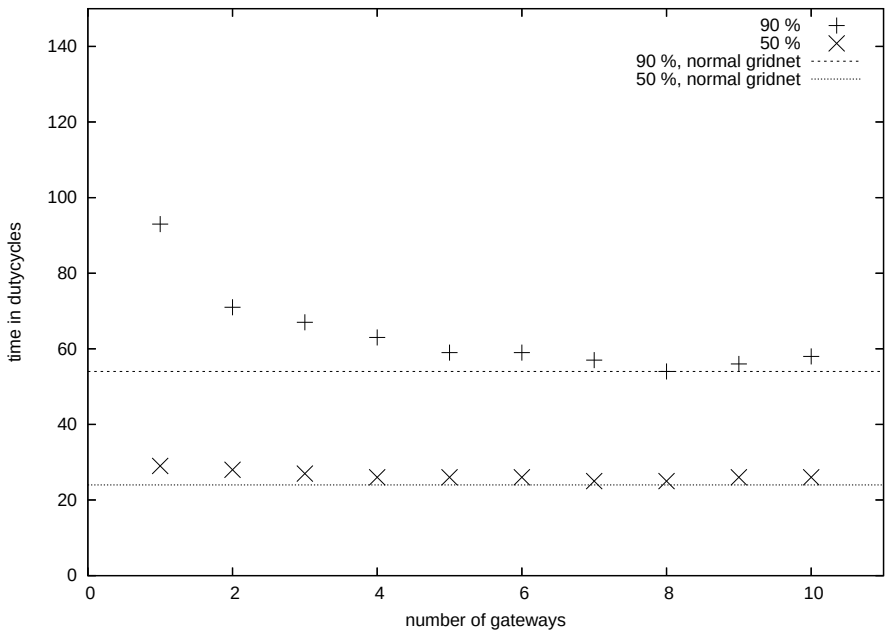

Figure 7.2: Influence of the number of gateways on traversal time (8 neighbours).

becomes small. However, in fig. 7.2, c is much smaller because every gateway has twice as many neighbours, so our message has to compete with many more messages. Because c is smaller, $1 - (1 - c)^n$ approaches 1 slower and the graph decreases slower.

The time it takes for a message to spread is not the only criteria to determine the number of gateways between two networks. Graph connectivity can also be taken into account: if the two networks are so well connected that any n nodes can be removed without breaking up the network, it can be attractive to use $n + 1$ gateways to preserve this property on the whole network.

7.2 Placement of gateways

Besides the number of gateways, the place of the gateway also has influence on connectivity. This is simply because the placement has influence of the diameter of the network. By placing the gateway nodes at specific points, the connectivity becomes better. This can reduce the number of gateways needed to connect two networks together.

Fig. 7.3 shows the results of a simulation where the gateway is placed in three different positions. The positions are given in fig. 7.4. The three upper lines show the spreading of a message in the network where the source is located. The three lower lines show the spreading in the other network. In the situation where the diameter of the network was the greatest, the line starts rising later because it takes longer for the message to travel to the gateway node. After that, the message also propagates slower.

It is expected that a network with a larger diameter has slower propagation. What fig. 7.3 also shows is that there is an adverse effect on the network containing the source. While the total network shows faster propagation with a lower diameter, the network containing the source actually spreads the message slower. While we are measuring the propagation of one particular message, other messages also travel through the network. Because propagation is faster, these messages are also propagated faster to the source network. There, they interfere the spreading of the message we are measuring. This also means that inserting a gateway in a network can slow this network down.

7.3 Connecting multiple networks

Thus far, we have looked at the characteristics of two connected networks. However, it is desirable to connect more networks together. There are two ways to do this, given the gateways described earlier. The first is to use multiple gateways. Since the connections are transitive, pairs of networks can be connected to make one network out of several. For example, network A, B and C can be connected together by connecting A to B and B to C. Fig. 7.5 show the results of a simulation where a network of 100 nodes is split in a number of networks. These networks are pairwise connected with gateways. As can be seen, increasing the number of networks gives a slower propagation of messages. This can be expected: cutting a network in two and connecting both networks with a gateway gives slower propagation.

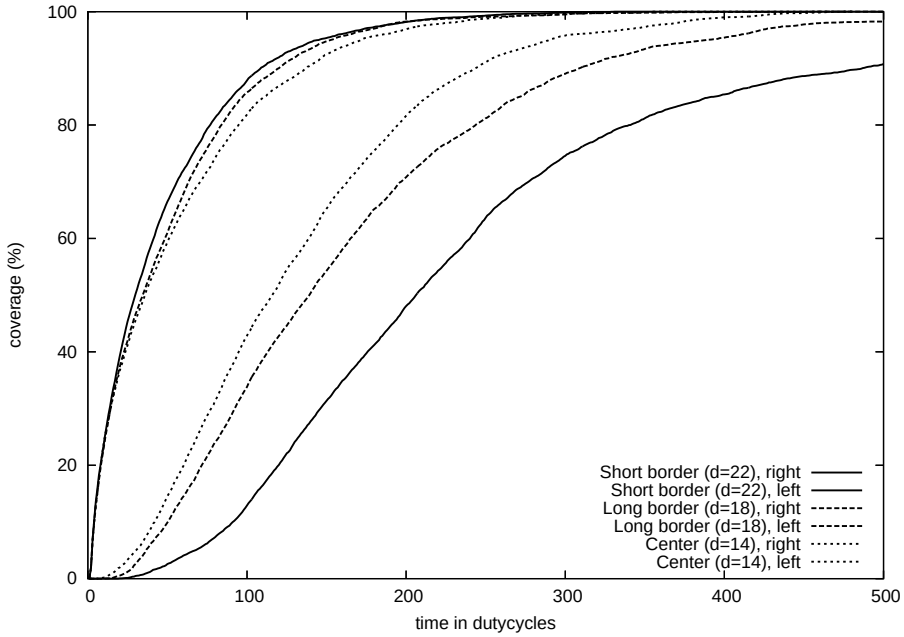

Figure 7.3: Influence of the placement of the gateway on traversal time. Variable d indicates the diameter of the whole network.

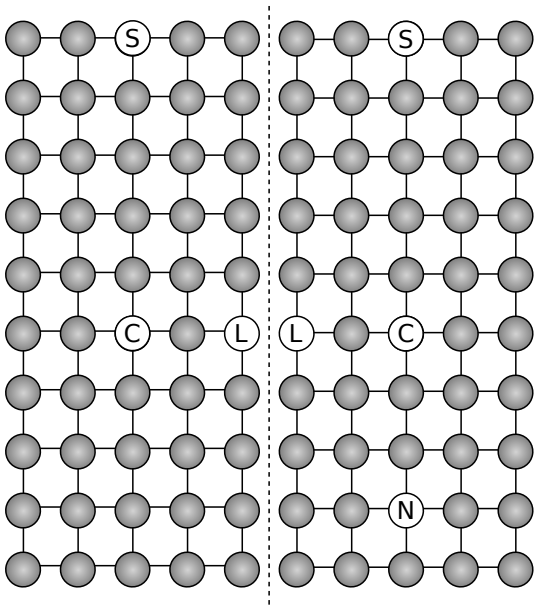

Figure 7.4: Situation for the experiment where the place of the gateway nodes changes. S, L and C correspond to the short border, long border and center in fig. 7.3, respectively. N indicates the source node.

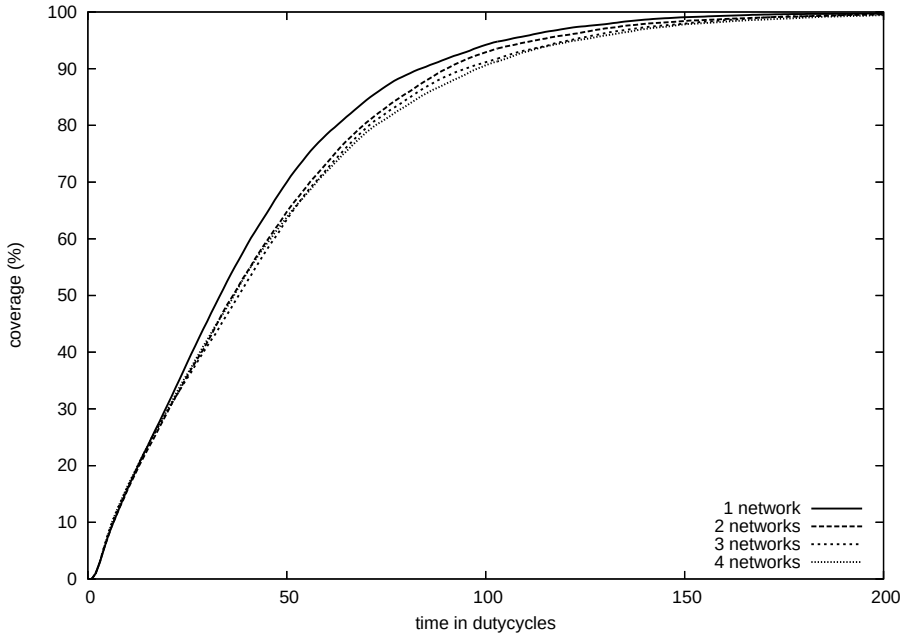

Figure 7.5: Traversal times in a varying number of networks, totally consisting of 100 nodes. Top line shows one network, bottom line four networks.

The second way to connect multiple networks is to couple several slave gateway nodes to one master. This way, multiple gateway nodes emulate one node, effectively connecting the networks. This will probably give worse performance than using pairs of gateway nodes, because messages from all networks compete for one output message. However, less gateway nodes are needed. The number of networks that can be connected in this way is somewhat limited, because the master has limited resources and needs to process messages from multiple networks. Although this method could be implemented with relatively few changes, we did not investigate this any further.

Chapter 8

Evaluation

8.1 Simulation environment

In order to evaluate our design, we did several simulations. Although simulation software already exists for simulating network protocols [Jel06], we chose to implement our own simulator to gain more insight and control over the simulations.

In a real network, all nodes communicate with each other simultaneously and then process the received messages all at the same time. Since this is difficult to implement, we chose to let the nodes do communication and processing in a serial way. As shown in pseudocode in fig. 8.1, first all output buffers are copied to the neighbours' input buffers. After that, each node does the processing on the messages. If the messages from the current dutycycle are used as input for the gossip protocol, the output is transmitted in the next dutycycle. This resembles the current behaviour in MyriaNed.

The simulation program uses the MyriaNed gossip protocol as explained in section 3.4: every dutycycle, a node will broadcast a random cache entry. Then, if it receives new items, it will put a random new item in its cache.

Unless specified otherwise, the simulations are of two networks, each having 50 nodes. These nodes are connected with each other in a grid, thus most have four neighbours. This situation is shown in fig. 8.2.

```
for each node in network {
  for each neighbour of node {
    neighbour.receiveBuffer += node.sendBuffer
  }
}
for each node in network {
  node.sendBuffer = []
  node.runGossip()
  node.receiveBuffer = []
}
```

Figure 8.1: Main loop of simulation program in pseudocode.

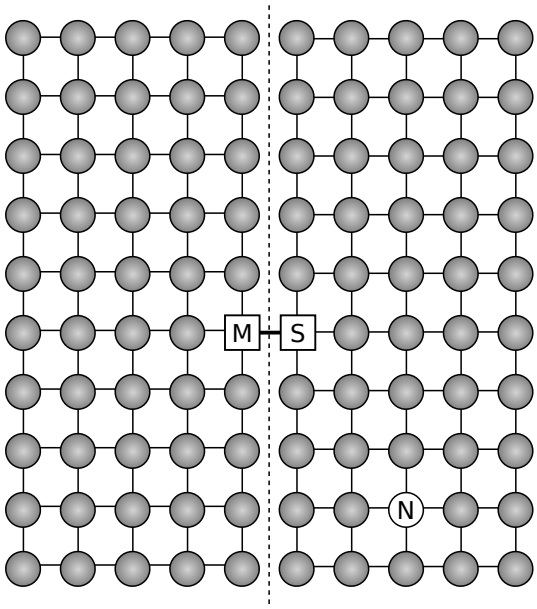

Figure 8.2: Simulation situation, where M, S and N correspond to the master, slave and source node, respectively.

A simulation consists of 300 runs. Each run, a data item is continuously sent out by a source node. All other nodes run the MyriaNed gossip protocol. The caches of all nodes are filled with random data, but during the run none of the nodes other than the source node inserts new data into the network. Only when the item has reached all nodes, a new data item is chosen and the next run starts. During the run, the number of nodes that have received this data item and the number of elapsed dutycycles are logged. These give an indication on how fast the data spreads and these are plotted in the graphs.

The gossip protocol sends out a random message, which means that the time it takes to send out a particular message varies pretty much. Different runs give very different results. To compensate for this, we did 300 runs and plotted the average of them in the graph. However, the standard deviation of this average is still 7.4% in some cases.

8.2 Network size

Fig. 8.3 shows the results of an experiment where we varied the size of one of the two networks. In this experiment, the size of the left network ranges between 5 and 125 nodes, with the gateway node in the middle of the right edge. The size of the right network remains constant at 50 nodes. On the x-axis is the total number of nodes in both networks. The y-axis shows the time it takes for messages to reach 50% or 90% of the nodes.

As can be expected, messages take longer to spread in a large network, because they have more hops to make. A linear relation can be expected, because the diameter of the network increases linearly with the network size. It

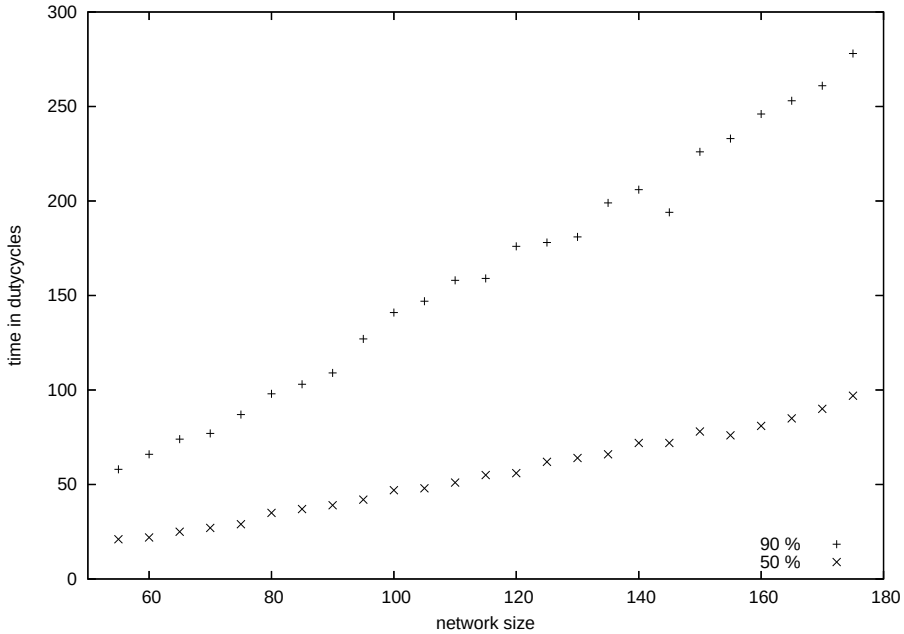

Figure 8.3: Influence of the network size on traversal time.

is not expected that the gateway introduces an extra delay when the network is larger or when the size of the two networks differ, because it does not know what the size of either network is: its job of passing messages between the networks stay the same. If the gateway would introduce an extra delay depending on the size of the network, the graph would show a curve because this delay comes on top of the linear delay. As the figure shows a linear relation, the gateways do not seem to have a negative impact on the traversal speed when the network size increases.

8.3 Dutycycle length

As shown in fig. 8.4, varying the length of a dutycycle of one network has influence on the spreading speed. In this experiment we varied the length of the dutycycles of one network. To avoid influencing our measurements, we measured time in dutycycles of the other network, which were not varied. The graph shows a rising line. This makes sense: if the dutycycles takes longer, there are less dutycycles in a specific period and thus messages travel slower. Since only one of the networks has a varied dutycycle, the graph also shows the results of a situation in which the two networks have a different dutycycle length. As the graph shows a linear relation which can be well explained by the length of the dutycycles alone, the difference between the two networks does not have a significant influence on the traversal time.

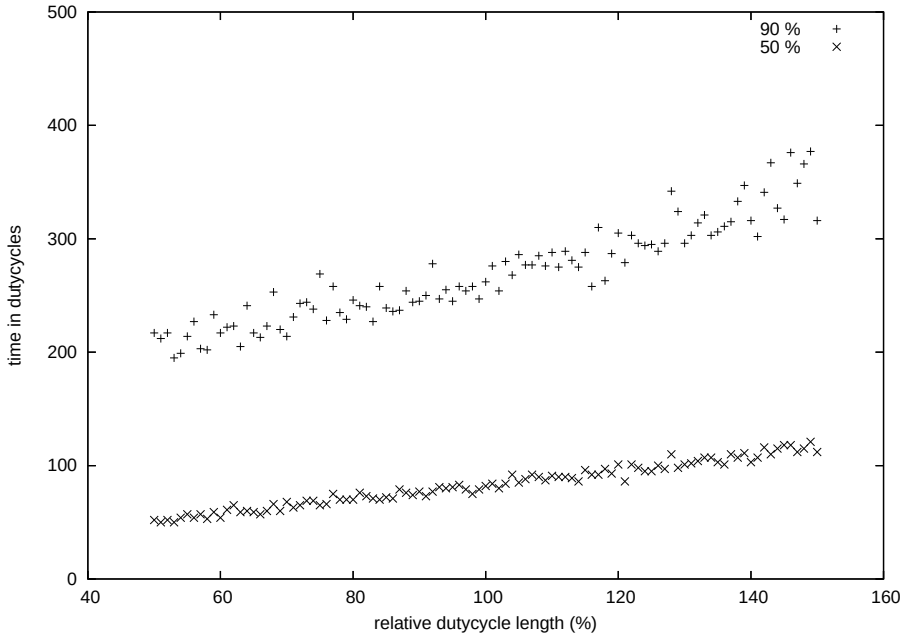

Figure 8.4: Influence of the length of the dutycycle of one network.

8.4 Cache size

A simulation where the cache size varies is shown in fig. 8.5. The graph shows a rising line, except for a cache size of one. It makes sense that increasing the cache size gives a linear decrease in speed in the network. Every dutycycle, a node chooses a random message from its cache. If the cache is small, the message is sent often and propagation is fast. If the cache is larger, the gossip protocol has more messages to choose from and it will propagate one specific message slower.

For comparison, the graph also shows the results for a situation with one network, without gateways. This situation also shows a linear relation between the cache size and the propagation speed. This means that the gateway does not introduce any additional delays when the cache size is increased.

The results of a cache size of one give results which do not correspond with the linear relation. Although we did not found a reason for this, it can be conceived that messages are more often overwritten. With a large cache size, a new message is put somewhere in the cache. The possibility that a particular message in the cache is overwritten is fairly low. However, with a cache size of one the contents of the cache are always overwritten. One message may easily disturb the propagation of another message, by repeatedly overwriting it in the cache. This may cause a slower propagation with a cache size of one.

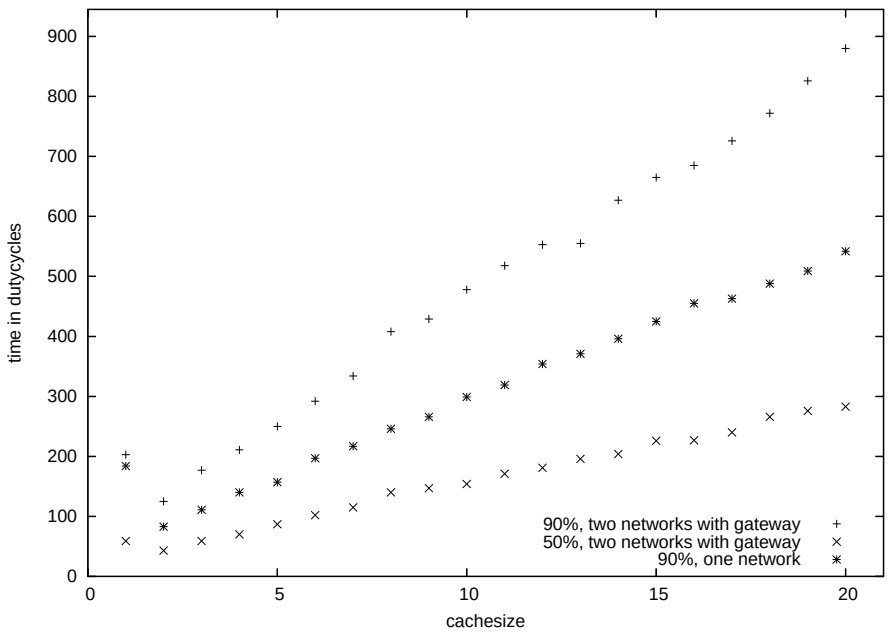

Figure 8.5: Influence of the size of the cache.

Chapter 9

Conclusion

We have proposed a system to connect multiple wireless sensor networks with each other using gateways. These gateways consist of two gateway nodes which can make a long-range connection with each other. The system, unlike related work, does not aim to access the two networks transparently but to connect them, so that both networks appear as one big network.

The two gateway nodes together emulate one node. This makes the connection transparent, which means it is not distinguishable from a part of the network. We have shown through simulations that the gateway behaves well under many circumstances, including with varying network size, cache size and dutycycle length.

The gateway may introduce a delay and therefore does not work with synchronous protocols. That is, protocols that depend on messages to transfer within one dutycycle. Other than that, our simulations have shown the delay to be negligible in a reasonably sized network.

The main application for the system is to connect wireless networks running gossip protocols, so that experiments can be done on the behaviour of such networks.

In future work, we will implement the system described in this paper in order to obtain experience on its performance and behaviour. Having done that, we believe that we have delivered a practical way to connect two or more wireless networks running gossip protocols with each other.

Bibliography

[Hac06] *Agimone: Middleware Support for Seamless Integration of Sensor and IP Networks*, Gregory Hackmann, Chien-Liang Fok, Gruia-Catalin Roman and Chenyang Lu. International Conference on Distributed Computing in Sensor Systems, 2006.

[Dai04] *Unifying Micro Sensor Networks with the Internet via Overlay Networking*, Hui Dai, Richard Han. 29th Annual IEEE International Conference on Local Computer Networks, p. 571 – 572, 2004.

[Hu03] *Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks*, Yih-Chun Hu, Adrian Perrig, David B. Johnson. 22nd Annual Joint Conference of the IEEE Computer and Communications Societies, p. 1976 – 1986, 2003.

[Wan04] *Visualization of Wormholes in Sensor Networks*, Weichao Wang, Bharat Bhargava. Proceedings of the 3rd ACM workshop on Wireless security, p. 51 – 60, 2004.

[Dun04] *Connecting Wireless Sensornets with TCP/IP Networks*, Adam Dunkels, Juan Alonso, Thiemo Voigt, Hartmut Ritter, Jochen Schiller. Proceedings of the Second International Conference on Wired/Wireless Internet Communications, 2004.

[May06] *IP-enabled wireless sensor networks and their integration into the internet*, Karl Mayer and Wolfgang Fritsche. Proceedings of the first international conference on Integrated internet ad hoc and sensor networks, 2006.

[Shn04] *Hourglass: An Infrastructure for Connecting Sensor Networks and Applications*, Jeff Shneidman, Peter Pietzuch, Jonathan Ledlie, Mema Roussopoulos, Margo Seltzer, Matt Welsh. Technical Report TR-21-04, Harvard University, EECS, 2004.

[Abe06] *Global Sensor Networks*, Karl Aberer, Manfred Hauswirth, Ali Salehi. Technical Report LSIR-2006-001, Ecole Polytechnique Fédérale de Lausanne, 2006.

[Lei06] *VIP Bridge: Integrating Several Sensor Networks into One Virtual Sensor Networks*, Shu Lei, Hui Xu, Wu Xiaoling, Zhang Lin, Jinsung Cho, Sungyoung Lee. International Conference on Internet Surveillance and Protection, 2006.

[Pit87] *On Spreading A Rumor*, Boris Pittel. SIAM Journal on Applied Mathematics, volume 47, issue 1, 1987.

[Dem87] *Epidemic Algorithms For Replicated Database Maintenance*, Alan Demers et. al. Proceedings of the sixth annual ACM Symposium on Principles of distributed computing, 1987.

[Vou03] *A Robust and Scalable Peer-to-Peer Gossiping Protocol*, Spyros Voulgaris, Márk Jelasity, Maarten van Steen. Agents and Peer-to-Peer Computing, Second International Workshop, p. 47 – 58, 2003.

[Haa02] *Gossip-Based Ad Hoc Routing*, Zygmunt Haas, Joseph Halpern, Li Li. IEEE/ACM Transactions on Networking, volume 14, issue 3, 2002.

[Jel02] *Large-Scale Newscast Computing on the Internet*, Márk Jelasity and Maarten van Steen, Technical Report IR-503.02, Vrije Universiteit Amsterdam, 2002.

[Osi96] *Open Systems Interconnection – Basic Reference Model*, ISO standard 7498-1. 1996.

[Dun03] *Full TCP/IP for 8-Bit Architectures*, Adam Dunkels. Proceedings of the First ACM/Usenix International Conference on Mobile Systems, Applications and Services, 2003.

[Zúñ03] *Integrating Future Large-scale Wireless Sensor Networks with the Internet*, Marco Zúñiga Z., Bhaskar Krishnamachari. USC Computer Science Technical Report CS 03, 2003.

[Jel06] *PeerSim*, Márk Jelasity, Gian Paolo Jesi, Alberto Montresor, Spyros Voulgaris. URL: `http://peersim.sourceforge.net`.