Network Coding in Bluetooth Networks

UNIVERSITY OF BERGEN

Masters Thesis

by

Roger Stenvoll

October 1, 2009

Supervisor:
Prof. Ø. Ytrehus

University of Bergen
Faculty of Mathematics
Institute of Information Technology

Abstract

This thesis discusses the possibility to apply network coding to a Bluetooth piconet. A protocol is proposed. This protocol is based on using deterministic linear network coding. The proposed alphabet size is binary, and the encoding equation is a trivial parity check code. The encoding scales easily by the number of source nodes in the network, and does not require exchange of coding equations. Encoding and decoding are performed using bitwise XOR of the packets, and do not require any pre-computed look-up table, nor a great amount of dedicated memory to store intermediate packets. Finally, the encoding and decoding processes are not computational hard.

Network coding applied as the proposed protocol is only beneficial to the master node and the communication from the master node to the slave nodes. Furthermore, it does not give any protection against errors, and information exchanged in the network will be available to all the slave nodes in the network.

A theoretical study of the proposed algorithm shows a gain in throughput, and reduced power consumption. These features are appreciated by computational and power challenged nodes. This efficiency is maximized when there are few source nodes in the network, and large frame sizes (DH5). The theoretical study is verified by a simulator designed to this purpose.

Preface

This thesis is the final work of my master study at University of Bergen. During the work I have had the possibility to pursue several of the subjects studied along the path to complete my master degree. It is with great pleasure I experience how all loose ends finally tie together in such a major work as the thesis is. In particular my insight in coding and information theory has improved a lot during this work.

Network coding was discussed the first time in 2000. Working with such a recent subject has been interesting. I have been able to follow some of the research and experience how it has evolved. Having the possibility to tie the research to an existing technology, has given me useful and broad insight of both ad hoc networking, Bluetooth, and not at least network coding.

As a full time employee and having a family with three wonderful girls and a great wife, it has been a challenge balancing the work, my family and the study. I am therefore grateful for all support from my family. Without their support, I would never have been able to finalize the work. Furthermore, I am gifted having colleagues and a manager supporting me, and opening up my schedule to enable me focusing on the master thesis in the final phase. In particular I would like to thank my good colleague Tom Kjetil Landgraf who, in addition to supporting me, also read through the thesis and gave me useful tips.

Finally I would like to thank my supervisor, Professor Øyvind Ytrehus, for giving me the possibility to work with this exciting field of research. His support has been priceless. His enthusiasm about coding theory in general and network coding in particular has given me a wish to study this subject further.

Roger Stenvoll

Bergen, September 2009

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent improvements have enabled wireless communication at a much greater extent than what was imaginable few years ago. 150 years ago, nobody knew about the good old wired telephone. At the beginning of the eighties, huge and heavy cell phones were a curiosity. And at the beginning of the nineties, only a handful people knew about the Internet. Today 'everybody' owns a small cell phone, and surf the web and check their e-mail from wherever place in the house or the world.

The improvements in technology have enabled what we know as the information age. Increased demand for communication (speech and data) is pushing the boundary every day.

As a majority of the communication up to the seventies was tied to fixed lines, an increasingly part of the communication networks nowadays are connecting to a cell based communication network. This is e.g. cell phones, satellite communication and wireless networks (WLANs). These types of network require a centralized and fixed installation, connected to an infrastructure.

Based on the demand for communication, even in geographical areas with less developed communication infrastructure, the need for 'something else' has risen. A solution that seems to solve some of these problems is ad hoc network. Ad hoc network, is as the name indicate, a network created when needed. (In fact, ad hoc is Latin for: *for this purpose.*) Ad hoc network is decentralized, and all nodes in the network are peers. Different protocols to enable routing in such networks have been proposed. Selecting routing protocol is based on properties of the network regarding, among other things, the number of nodes, mobility, timeliness, throughput, and power consumption.

Although the development of the technology has increased the available bandwidth, the demand increases as well. Systems to utilize the available bandwidth efficiently are of great importance. Network coding is one such approach to utilize the network more efficiently. Network coding was first introduced in 2000, and extensive research has been done in this area since then.

1.1 Problem

Bluetooth is a widely spread technology, and a number of different devices are shipped with this support. This makes Bluetooth a candidate to be used in ad hoc networks. This thesis will explore the possibilities to apply network coding in Bluetooth networks, and proposes a protocol on how this can be done. This thesis will perform a theoretical study on application of network coding in Bluetooth piconets.

1.2 Organization of the report

Chapters 2 to 4 give a theoretical framework for the discussion later in the thesis. Chapter 2 discusses ad hoc networks in general terms, and gives an overview of differences and problems in an ad hoc network as opposed to a centralized network. An introduction to Bluetooth is given in Chapter 3. And finally, network coding is explained in Chapter 4.

A discussion on how to possibly apply network coding in a Bluetooth network is given in Chapter 5. The protocol proposed in this chapter is simulated, and the description of the simulator and the results are given in Chapter 6. The thesis is concluded, and future work is outlined, in Chapter 7.

Detailed results from the simulation and the source code of the simulator are available in the appendix.

Chapter 2

Ad Hoc Networks

Ad hoc mobile networks are usually temporary networks [6, Chapter 2.6]. I.e. the networks are established when the communication nodes are within radio range and there exists a demand for communication. As a consequence of this, there is no centralized component initiating the session. Typically all nodes are peers. The networks are usually temporary as the nodes may be added or removed, either because they have moved out of radio range or the need for communication has vanished.

Furthermore ad hoc networks may operate stand alone or be connected to another network such as the Internet. In this case, this is done through a gateway. That is, one or several nodes of the ad hoc network acting as gateways between the networks. As opposed to cabled and wireless cell networks there is no central router in ad hoc networks. Instead traffic is routed between nodes, as the nodes agree to relay traffic on behalf of the source node. Due to this fact, ad hoc networks pose new challenges such as routing and medium access. The routing protocol plays a fundamental role in an ad hoc network, as it will be difficult to offer services to the other nodes without it. Additionally, without a proper medium access control protocol (MAC), ad hoc networks will suffer from degradation of services due to collision and interference between nodes. Several protocols have been proposed to address these issues.

Two types of ad hoc network are defined: mobile ad hoc networks (MANET) and wireless sensor networks (WSNs)[6]. MANETs and WSNs have different applications and characterizations, and are treated individually.

2.0.1 Mobile Ad Hoc Networks

Mobile ad hoc networks (MANET) are mobile hosts communicating with each other using wireless links based on the peer-to-peer paradigm [6, Chapter 2.6.1]. As a mobile host acts as a router and at the same time is free to move independently, it is likely that the network topology changes unpredictably. Thus a MANET has to be self-configuring and autonomous for any two or more nodes forming the network. This ideology applies to all layers in the communication stack from data link layers up the application layer. As a consequence, use of distributed applications is preferred.

Typical mobile nodes could be Laptops, PDAs, cell phones and other electronic devices containing a radio device. However, efforts have been made in the military to apply this technology to establish communication within vehicles, vessels, airplanes and even soldiers. (E.g the Norwegian Modular Arctic Network Soldier (NORMANS) [14], Subnet Relay [65] and UIDM [12]). Image a scenario where soldiers carrying a mobile node are able to establish communication to the headquarter using a network of nodes relaying traffic through other soldiers, vehicles and aircrafts. This network could enable a communications system providing IP connectivity (to be used to send voice, images, movies and tactical situation report) back to the HQ. Figure 2.1 illustrates an example of such a network. The line indicates the communication lines. The figure illustrates how traffic relays between nodes in the network, and that there may exist several paths through the network. The frigate hosts a gateway between the ad hoc network and the fixed satellite communication network, enabling communication between the two nets.

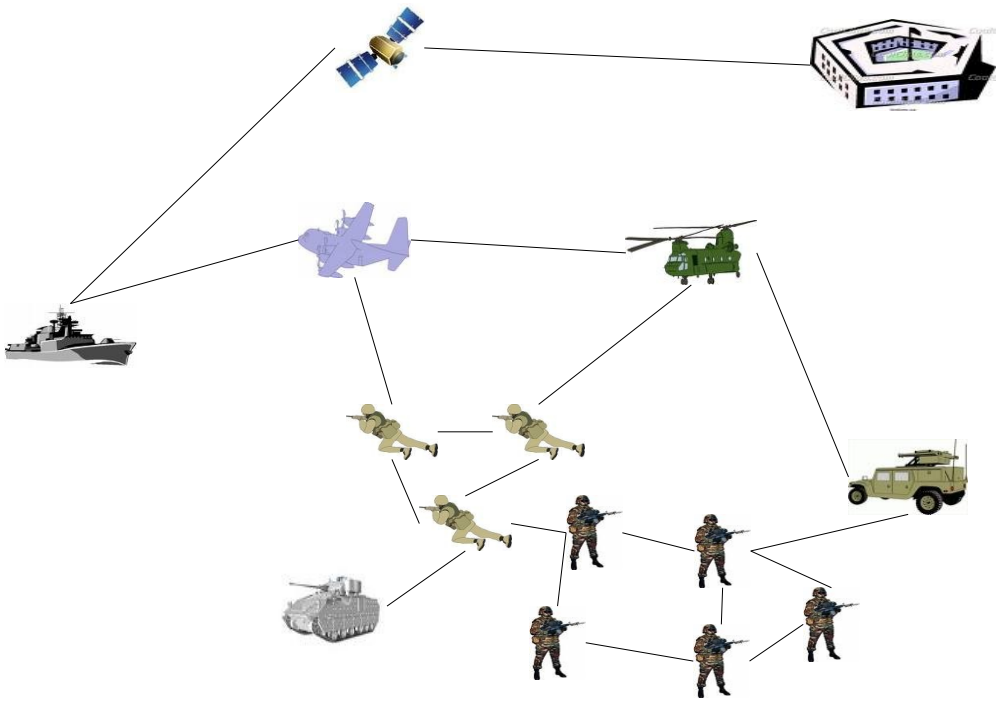


Figure 2.1: An example of an ad hoc network

A similar use of MANET could be applied to scenes where there are no established communications structures, or the communication structure has been destroyed as a consequence of a natural disaster.

Common to both of these scenarios, is a MANET operating in an environment where there is no established or available cell network like GSM with 3G capacity, WLAN hot spot, WiMax, or any satellite systems. MANETs are able to extend the geographical

coverage of the network beyond the radio range of the devices or even line-of-sight (LOS).

2.0.2 Wireless Sensor Networks

A wireless sensor network is a network consisting of low-cost sensors able to communicate wirelessly with each other. Typically, they have limited computing capability and memory, and operate with limited battery power [6]. Distributed sensing tasks are the main purpose of such networks, e.g. temperature, magnetic field, or gases. Furthermore, often all nodes are involved in the same sensing task. Thus, wireless sensor networks differ in characteristics from MANET. To provide high sensing resolution, wireless sensor networks are potentially formed by hundreds and thousands of sensors. To conserve power, sensors may enter a dormant state during periods of low activity. Even though the sensors are stationary, the networks still have a dynamic topology as a result of this behavior. Replacement or recharging of batteries after deployment is often difficult. Energy conservative design is therefore important in these networks.

2.1 MAC protocols

The performance of a network is directly dependent of the *medium access control* (MAC) protocol [6]. As ad hoc networks are wireless, they share the same medium. MAC controls the sharing of the medium, and therefore the performance of the network. Ad hoc networks (both MANETs and WSNs) are distributed, and also the MAC is distributed. As opposed to fixed network like cellular system (GSM) and satellite system which have a centralized MAC. I.e. a central node controls the fair access, quality of service (QoS) and other decision made by the MAC.

Common MAC techniques are *frequency division multiple access* FDMA, *time division multiple access* TDMA, spread spectrum multiple access, and *carrier sense multiple access* CSMA.

FDMA assigns individual channels to each user. This is like the well known FM radio broadcasting, where each radio channel has assigned a channel.

TDMA uses a channel divided in time slots. TDMA assign time slots to each user. The time slots are grouped into frames, and repeated cyclically. Thus, a user channel is repeated in every frame. To obtain duplex communication half of the time slots are used in each direction. This is called *time division duplex* TDD. Alternatively, the forward and return traffic are transmitted on different frequency. This is called *frequency division duplex* FDD. E.g. the GSM mobiletelephone system uses a combination of TDMA and FDD. Different channels are used for both sending and receiving, and different channels are used for traffic and control signal [57].

There are two common form of spread spectrum multiple accesses techniques, *frequency hopped multiple access* (FHMA) and direct sequence multiple accesses. The last method is also called *code division multiple access* CDMA. In FHMA data are transmitted in burst, each burst is transmitted on different channel.In CDMA the data is coded onto a large bandwidth signal. To decode the data the receiver needs to know the code

used to encode the signal. The *global positioning system* GPS uses CDMA [55].

CSMA is a protocol to avoid collisions of packets by first listening to the channel for another nodes carrier. There is no central coordination of who is transmitting. If the channel is unavailable, the node will back off for a random period. After the back off period, the node then listens to the channel again. CSMA is usually combined with either *collision detect* CD or *collision avoidance* CA. CSMA/CD implies a protocol how to act upon detection of a collision, and are used in the Ethernet (IEEE 802.3). CA is primarily used in half duplex systems which are not able to detect collision (listen) when transmitting. Instead small handshake packets are sent prior to the data and establish a 'channel' before the data is transmitted. CSMA/CA is used in wireless LANs (IEEE 802.11).

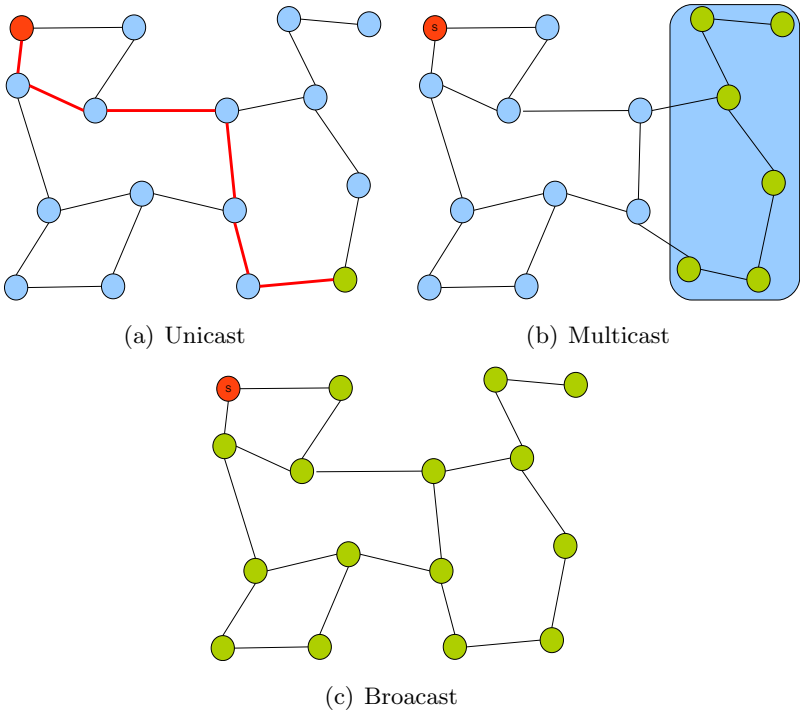
2.2 Routing

The key problem in an ad hoc network is to know which nodes are available and possible to communicate with, and how to route the traffic between the two nodes through the network. Due to the dynamic nature of ad hoc networks, they are facing different routing problems than traditional cabled networks. That is, nodes appear/disappear from the network, and nodes move around changing the topology frequently. To challenge the router problems there have been established two major approaches: *Proactive* and *reactive* routing algorithms. The proactive routing algorithms try to keep an updated routing table to all nodes at all time. The reactive routing algorithms try to establish a route from the source to the destination at the time the need for communication appears, i.e. at the time the data are generated. The proactive routing algorithms are able to pass the traffic instantly, whereas the reactive routing algorithms have to find the route before the traffic can be routed. On the other hand the proactive routing algorithms are power consuming even when there is no traffic, as the routing protocol traffic will be running all the time.

The different routing protocols are classified into three different classes in [6], unicast routing, broadcast routing and multicast routing, as shown in Figure 2.2.

2.2.1 Unicast routing

Unicast routing (Figure 2.2(a)) is routing between to single nodes in a network, one-to-one. These protocols make it possible for any two nodes in the network to communicate. Different routing protocols address the issue of finding the shortest path through the network differently. The unicast routing protocols can be further classified by two different approaches, flat and hierarchical routing protocols. The flat routing protocols give each node the same function and responsibilities in the network. Hierarchical routing protocols, on the other hand, organize the nodes as a tree of clusters. Example of flat routing protocols are Dynamic Source Routing (DSR) [32], Ad hoc On-demand Distance Vector (AODV) [51], Temporary Ordered Routing Algorithm (TORA) [48] [49]. Thus examples of hierarchical routing protocols are Zone Routing Protocol (ZRP) [25] and



(a) Unicast (b) Multicast

(c) Broacast

Figure 2.2: Routing in ac hoc network. Red node is source, green nodes are sink and blue nodes other nodes

Hierarchical State Routing (HSR) [50].

2.2.2 Multicast routing

Multicast routing (Figure 2.2(b)) is routing from one source to multiple nodes, one-to-many. Multicast routing is important for application communicating to a group of nodes based on geographically or functional criteria. E.g. conferences, multi player games, information to users within a specific area, or communication dedicated to a group of users like doctors, nurses and patients. [6] divide multicast protocols into five types: tree-based, mesh-based, backbone-based, stateless multicast and application layer multicast. Examples of multicast routing protocols are: Core-Assisted Mesh Protocol [20], and a Maximum-Residual Multicast Protocol for Large-Scale Mobile Ad Hoc Networks (MRMP) [30].

2.2.3 Broadcast routing

Broadcast routing (Figure 2.2(c)) is routing from one source to all nodes in a network, one-to-all. Broadcast routing is often a key part of unicast and multicast protocols [6]. The simplest form of routing is flooding. I.e. all nodes relay received traffic to all its connected neighbors. Such flooding could cause what is called a *broadcast storm* [43]. This is a situation where each node receives the same broadcast multiple times, because a node who received the broadcast decides to rebroadcast it to all its neighbor nodes even though they have received it before. In a wireless network this could cause collision and degradation of the network. In addition valuable battery power could be wasted on unnecessary broadcasts. Examples of broadcast routing protocols are: *scalable broadcast algorithm* (SBA) and *ad hoc broadcast protocol* (AHBP). A comparison of some of the different broadcast routing protocols have been done by Williams and Camp [70].

2.3 Security

Basic principles of information security are defined as confidentiality, integrity and availability [61].

Confidentiality is preventing disclosure of information to unauthorized system entities.

Integrity is preventing modification or destruction of information without authorization.

Availability for a system to serve its purpose it has to be available as designed (accessible and usable) whenever users request it.

A key to establish the security principles is ability to authenticate nodes in the network, and encrypt the information transferred.

2.3.1 Security threats to ad hoc networks

Nodes in an ad hoc network relay data on behalf of other nodes in order to establish a route from the source to the sink. A consequence of this kind of routing, as opposed to a cabled fixed network, is the revealing of information to nodes along the path. Thus ad hoc networks need to maintain confidentiality differently. Furthermore, intermediate nodes are able to modify the data before passing it to the next node. Finally, if an intermediate node stops passing traffic, the route is broken. And the availability of the network lost. Traditional cabled network often include a *public key infrastructure* (PKI) to verify legitimacy of components (e.g. hosts, servers, printers and routers) [67], [1]. The lack of a server infrastructure is a challenge to impose a system preventing spoofing. It is possible to use signed certificates and public key to identify nodes, however this require either distribution and storage of certificates or a distributed *certification authority* (CA). A 'bad' node can easily perform passive and active attacks against an ad hoc network unless security countermeasure is made. RFC4593 gives a description of generic threats to routing protocols [8].

2.3.2 Security countermeasures in ad hoc networks

Recent research on security in ad hoc networks has proposed countermeasures in three different categories: secure routing, trust and key management and service availability protection [6].

Secure routing

Several protocols for secure routing in ad hoc networks have been proposed. As opposed to previously mentioned routing protocols these protocols does not assume all nodes behave properly according to the routing protocols, and that no malicious nodes exist in the network. Both symmetric and asymmetric schemes have been proposed.

Trust and key management

As an ad hoc network is distributed, an implementation of a PKI is not feasible. Both partially and fully distributed certificate authorities have been proposed. In principle, the proposed schemes involve a distribution of the certification authority between the nodes. A user is trusted if k different neighbor nodes verify the authenticity of that user.

Service availability protection

A problem in ad hoc network is nodes behaving selfish. That is, they use services provided by other nodes, but do not provide any services to the community. [6] discuss two approaches to solve this problem, reputation-based and monitoring. The reputation based approach is based on some sort of reputation, like credit, for incentive to cooperation. The monitoring approach is based on neighbors watching misbehaving nodes and isolates them by sharing this information with other nodes.

2.4 Challenges and direction in ad hoc networks

According to [6, Chapter 3] the research direction of ad hoc network challenges is to implement quality of service (QoS) support, fairness, power management, and smart antennas. As ad hoc networks have limited shared bandwidth, ability to prioritize some nodes or application over other is important. Fairness is another functionality required to balance the shared wireless channel. The lifetime of battery operated nodes in an ad hoc network is improved using power management algorithms. One approach is to shut down the receiver, during idle periods. The challenge is to decide where to buffer packets to idle nodes, and when or how to turn a node on. Particularly wireless sensor networks are depending on power management, as it is sometimes is not feasible to replace batteries.

Chapter 3

Bluetooth

3.1 Introduction to Bluetooth

Bluetooth is a short range wireless technology intended to replace cables connecting portable and/or fixed devices [23]. The Bluetooth specification is maintained by the Bluetooth Special Interest Group (SIG). The Bluetooth SIG is a privately held not-for-profit trade association and has more than 10.000 members, among them Ericsson, IBM, Intel, Microsoft, Motorola and Nokia. Bluetooth has become a global standard for short-range wireless personal connectivity, and is included in a number of electronic devices like cell phones, PDA's, Laptops, game consoles, cars, headsets, and even toys. As of May 2008 there were between 1.5 and 2 billion Bluetooth devices, and every week 13 million new units are shipped out [23]. This makes Bluetooth a interesting technology to be used in mobile ad hoc networks.

3.2 Bluetooth architecture

The Bluetooth system is divided into four layers, Radio-, Baseband-, Link Manager- and L2CAP layer. The three lowest layers are often grouped together and referred to as the Bluetooth controller. *Host Controller Interface* (HCI) is the interface between the Bluetooth host and the Bluetooth controller, see Figure 3.1.

3.2.1 Radio layer

The radio layer is responsible for transmitting and receiving packet of information on the physical channel. Bluetooth operates in the *Industrial Scientific Medical* (ISM) band (2.4GHz), and uses 79 channels spaced 1MHz apart from each other [60]. To avoid interference and fading a frequency hopping scheme is used. Two types of modulation are defined, *frequency shift keying* (FSK) and *phase shift keying* (PSK). FSK is used by the mandatory *basic data rate* (BDR) mode (1Mbps), whereas the PSK modulation is used by *enhanced data rate* (EDR) (2 and 3Mbps). The EDR is available from Bluetooth version 2.0. Bluetooth uses a fixed symbol rate of 1Ms/s for all modulation schemes. For

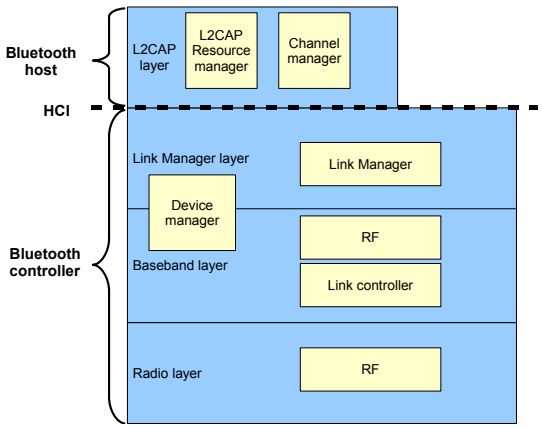


Figure 3.1: Bluetooth Architecture

full duplex transmission support, a time division duplex (TDD) is used in both modes. In a TDD scheme data are transmitted in one direction at a time, and transmission alternating between the two directions. There are 1600 time slots per second, giving each time slot a 625μS duration. The Bluetooth specification defines three classes of transmitters based on output power as listed in table 3.1.

Class	Power [mW]	max range [m]
1	100	100
2	2.5	10
3	1	1

Table 3.1: Bluetooth classes

3.2.2 Baseband layer

The baseband layer is responsible for medium access control and physical layer procedures between Bluetooth devices. A physical channel is defined by a pseudo-random frequency hopping sequence, slot timing and an access code. The hop rate is 1600 hops per seconds, i.e. one hop per time slot. The hopping sequence is determined by the Bluetooth device address (BT_ADDR, se Chapter 3.2.2) of the master device. Different modes use different hopping sequence. Furthermore, the phase of the sequence is determined by the Bluetooth clock. A transmission of data is called a packet. Each data packet may be 1, 3 or 5 slots. During multi slot packets, the radio remains on the same frequency until the entire packet has been transmitted. The Bluetooth standard support coded and uncoded transmission, called DM and DH accordingly. Table 3.2 list the maximum packet size for the different modes.

Type	Max Payload	FEC
DM1	17	2/3
DM3	121	2/3
DM5	224	2/3
DH1	27	No
DH3	183	No
DH5	339	No

Table 3.2: Bluetooth ACL packet sizes

Bluetooth addressing

All Bluetooth devices have a unique 48 bit address called BT_ADDR. Addressing in a piconet is done using a 3 bit address assigned by the master in the range of 1-7. This address is called LT_ADDR. The LT_ADDR is used by the master node polling the slaves. In the reply from the slave, the sender adds the LT_ADDR in the header field. The special LT_ADDR=0 is used by the master in an *active slave broadcast*. According to the Bluetooth specification [22] a slave node only process packet addressed with its own LT_ADDR or address 0.

Bluetooth packets

Figure 3.2 shows the structure of a Bluetooth packet. Some packet types used during the pairing process does not include a header, and in such cases the trailer in the access code is removed. The access code is in these cases 68 bit long. This is not the case for master - slave communication, where the access code is 72 bits and a header is included. I.e. packet overhead is 126 bit. The header field contains 18 bit of information. However, the header is encoded using 1/3 FEC. The FEC is a simple three times repetition code, thus each bit is repeated three times. A short description of the different field:

- Acces Code Every packets start with an access code containing preamble, sync word and trailer
- Header contains the link control information and consist of LT_ADDR, Type, Flow, ARQN, SEQN and HEC
- Payload is the data carried by the link. The size range from 0-2754 bits
- Preamble is a fixed zero-one pattern
- Sync Word is constructed on the basis of packet type and data to encode
- Trailer is a fixed zero-one pattern
- LT_ADDR is a three bit logical transport address
- Type is a 4 bit type code

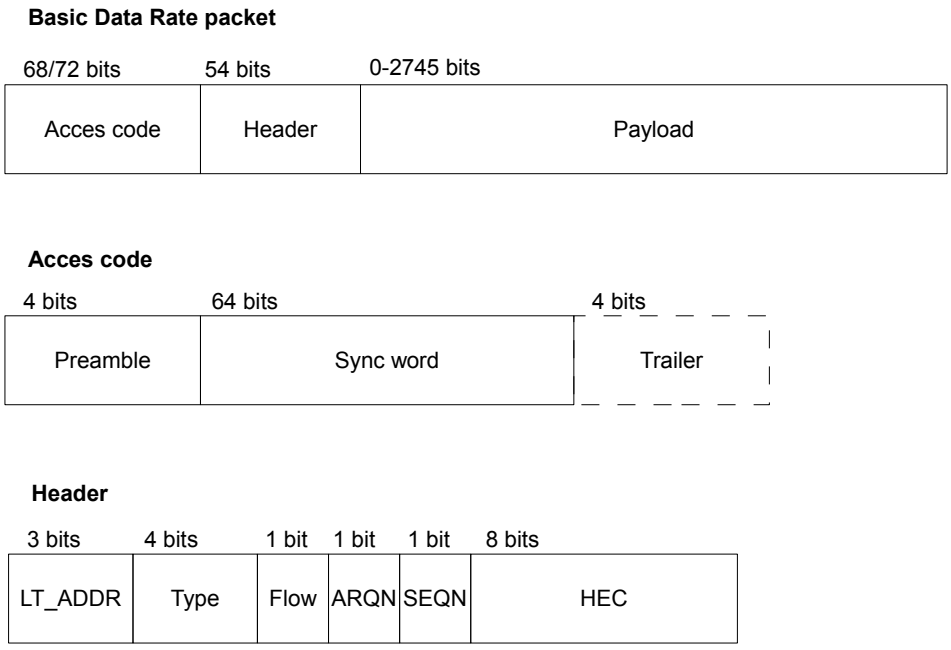


Figure 3.2: Bluetooth packets

- Flow is used for flow controll
- ARQN is acknowledge indication
- SEQN is sequence numbering
- HEC is header error check

Link Controller

The link controller is encoding and decoding Bluetooth packets. The Bluetooth packets consist of three fields access code, header and payload. There are three types of access codes, channel access code (CAC), device access code (DAC) and inquiry access codes (IAC).

Baseband Resource Manager

The baseband resource manager is responsible for all access to the radio medium.

3.2.3 Link Manager Layer

The link layer is responsible of establishing, modifying and tear-down of logical links

3.2.4 Logical Link and Adaption Protocol (L2CAP) layer

The L2CAP layer is responsible for managing the order and submission of PDU fragments to the baseband. Furthermore, it performs some relative scheduling between channels to ensure that L2CAP channels with QoS requirements are not denied access to the physical channel as a result of an exhausted Bluetooth controller.

3.3 Bluetooth protocols

Figure 3.3 shows a block diagram of the Bluetooth protocol stack. Only protocols essential for this thesis will be explained here.

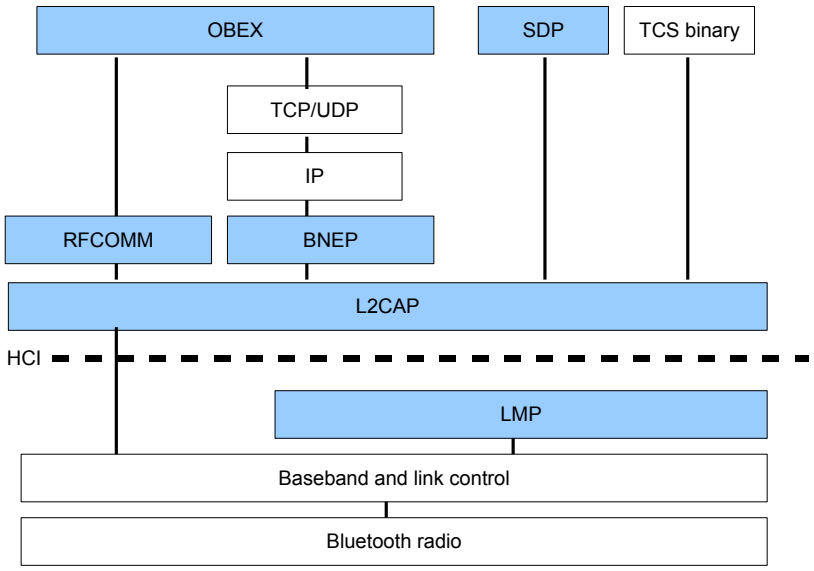


Figure 3.3: Bluetooth protocol stack

3.3.1 Bluetooth protocols overview

Link manager protocol

The *Link Manager Protocol* (LMP) is responsible for link setup and link configuration between the Bluetooth devices. The LMP is also responsible for managing and negotiating the baseband packet size. Finally the LMP manages the security aspects, like authentication and encryption.

L2CAP

The main task of the *Logical Link Control and Adaption Protocol* (L2CAP) is multiplexing the different logical connections made by higher levels. L2CAP also perform packet segmentation and reassembly.

RFCOMM

The RFCOMM protocol provides emulation of serial ports over L2CAP. Higher level protocols that uses a serial interface are supported by the RFCOMM.

BNEP

The *Bluetooth Network Encapsulation Protocol* (BNEP) provides transport services for network protocols over Bluetooth such as IPv4 and IPv6.

OBEX

Object exchange(OBEX) is a transfer protocol that defines data objects and a communication protocol two devices can use to exchange those objects.

SDP

The *Service Discovery Protocol* SDP provides a protocol for application to query services of connected Bluetooth devices. A Bluetooth device query available services after discovered a new device.

3.4 Bluetooth profiles

A Bluetooth profile defines standard ways of combining selected protocols and protocol features. These profiles enable a particular usage between different implementations of Bluetooth. Each implementation may choose to support one ore more profiles. There are however four "'basic"' profiles [66], *Generic Access Profile* (GAP), *Serial Port Profile* (SPP), *Service Discovery Profile* (SDAP) and *Generic Object Exchange Profile* (GOEP).

Generic Access Profile

The GAP provides the basis of all other profiles. Generic procedures to establish connection between two devices are defined.

Serial Port Profile

The SPP used the RFCOMM protocol and defines how to set up virtual serial port communications.

Service Discovery Application Profile

The SDAP describes how a Bluetooth device should use SDP to discover services on a remote Bluetooth device.

Generic Object Exchange Profile

The GOEP defines how to use OBEX to exchange objects between two Bluetooth devices.

3.5 Intra unit administration

A connection has to be made before two Bluetooth devices are able to communicate. The process of connecting two devices happens in two step, device discovering and paging. Figure 3.4 illustrate the process of establishing connection.

3.5.1 Establishing a Bluetooth connection

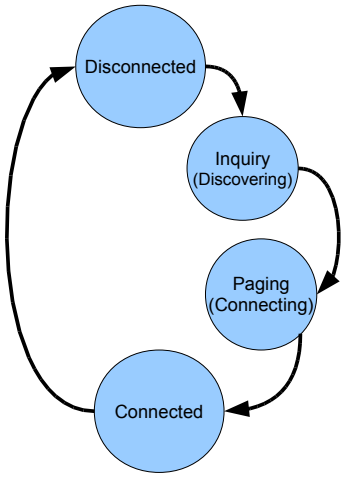


Figure 3.4: Bluetooth connection procedure

Inquiry - Device discovering

The device discovering procedure is asymmetrical. Bluetooth devices searching for other Bluetooth devices are in an inquiring mode. The Bluetooth devices accepting connection are in a discoverable mode. During the inquiry scan all discoverable devices will reply to a special inquiry request with an inquiry response. If a device trying to connect to another devices knows the address of the device, this step may be skipped. Such

devices will always respond to connecting requests. These devices are known as *cached* or *pre-known devices*.

Paging - Connecting

After a successful device discovering the Bluetooth devices may perform paging as the final stage of connecting the devices. Also the paging procedure is asymmetrical. One of the devices perform the paging (connection) and the other devices enables page scanning (connectable). When the paging procedure is completed, the Bluetooth devices are connected. During the connection process two channels are crated ACL-C and ACL-L. The ACL-C connects the LMP control protocol, and the ACL-U connects the L2CAP channel, see Figure 3.5. Devices are free to establish other channels. Either of the devices can delete the channel, and disconnect the connection. The devices then return back to disconnected mode.

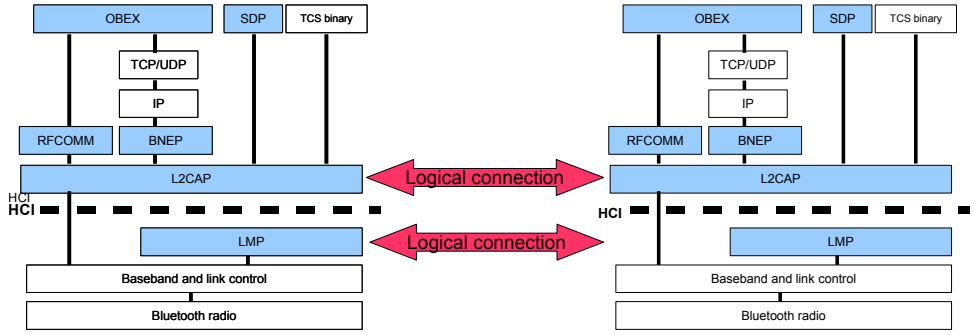


Figure 3.5: Bluetooth nodes connected

3.6 Bluetooth Security

3.6.1 Bluetooth security mechanism

Bluetooth employs three modes of security for Bluetooth access between two devices. These are:

Security mode 1 Non-secure

Security mode 2 Service level enforced security
This is two devices may establish a non-secure ACL link. Security procedure, like authentication, authorization and optional encryption, is enforced when a L2CAP channel is established.

Security mode 3 Link level enforced security
This implies establishing of security mechanism before a connection is established.

Pairing

Bluetooth devices may depending on the security setting establish a relationship before link is established, this is known as *pairing*. After a successful pairing of two devices, they devices are said to be *bonded*. During the pairing process an initialization key is established. A passkey (PIN) is used to calculate the initialization. This initialisation is used to calculate the link key and other keys used to authenticate other devices or encrypt the communication. Devices that have performed the pairing process, are *bonded*. Either of the bonded devices can delete the link key, and implicitly remove the bonding.

A main threat against Bluetooth security is the pairing phase. An eavesdropper recording the pairing phase could record the challenge and reply between the devices. By calculating the initial key of all potentially passkeys, and then calculate the link key the eavesdropper can match the possible values with the response, and such find the link key. Consequently all further communication can be decrypted by the eavesdropper. A Bluetooth Security White Paper [21] recommends using long passkey and only pair devices in a secure environment. The computing complexity of generating the initialization key based on the passkey is not large. Using long passkey will increase the work load for an eavesdropper of finding the matching link key. Avoiding pairing of Bluetooth devices in public spaces eliminate the possibility of an eavesdropper to find the link key. The white paper gives recommendations on how to implement security in different application over Bluetooth.

3.7 Bluetooth and Java

There are three Java platforms: Java *Enterprise Edition* (EE), *Standard Edition* (SE) and *Micro Edition* (ME). The Java EE is aimed at servers and enterprise computers, SE is aimed at desktop computer and finally ME is aimed at consumer and embedded devices such as PDAs, mobile phones, entertainment devices, and navigation systems. Support for Bluetooth was first added in the Java Micro Edition because it was thougth that the initial set of devices using Java language capabilities over Bluetooth would be in this space [66]. The *Java APIs for Bluetooth Application* (JABWT) is defined in the JSR-82[1].

Support of Bluetooth in Java SE is defined in JSR-197. JSR-197 defines API for Bluetooth Application based on the JSR-82. Java Standard Edition that includes JSR-197 may support the Java API for developing Bluetooth applications. However, this support is not common today. There are both commercial and open source Java SE implementations of JSR-82. An open source implementation is BlueCove [59].

[1] JAVA API are defined through the *Java Community Process* (JCP), and each new API is developed as a *Java Specification Request* (JSR)

3.8 Piconet

Two or more Bluetooth devices sharing the same physical channel form a piconet. The unit initiating the connection becomes the master, and the other node(s) becomes the slave(s). In a piconet there can be up to 7 active and 255 parked slaves. A parked slave is not able to communicate. If a piconet is holding 7 active slaves, it must park one of the active slaves before one of the parked slaves can become active. In a piconet the Bluetooth device address and clock of the master node is used to determine the physical channel. As different piconets will have different masters, each piconet will use different (and unique) hopping sequences. This makes it possible for different piconets to co-exist within radio range without interfering each other. However, occasionally multiple piconets may use the same frequency during the same time slot; collision will occur and cause packet loss. As this is happening infrequently, possible consequences are prevented by using FEC.

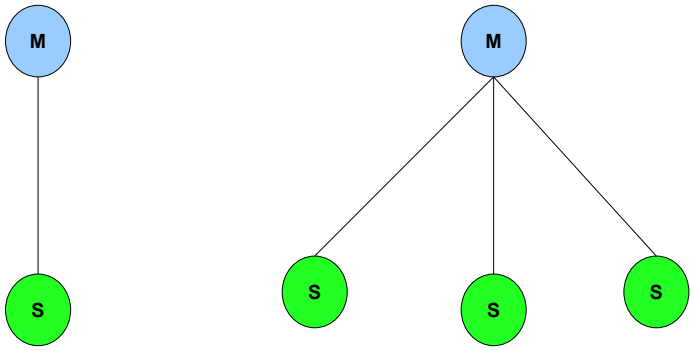


Figure 3.6: Left: Single Master/Slave, Right: Piconet

3.9 Scatternet

As described in chapter 3.1 the purpose of the Bluetooth technology is to replace cable in a point-to-point communication. However, the Bluetooth technology offers a method to interconnect multiple devices using a piconet. As a piconet has maximum one master and up to seven slaves, a network of more than 8 participants is formed by interconnecting multiple piconets. This is called a *scatternet*. Each piconet has a master and multiple slaves. Interconnection is done by gateway nodes. A gateway node is either a node acting as slave in two piconets, or a master node acting as master in one piconet and a slave(s)

in the other one(s). A Bluetooth device is only able to act as master in one piconet at the same time. The gateway nodes are not able to listen to several piconets at the same time. Time sharing is used to receive and transmit in the different networks. During the time a gateway node is sending or receiving in a network, it is not able to listen to traffic in the other ones. The load balance of the gateway nodes is of importance to the design of scatternet.

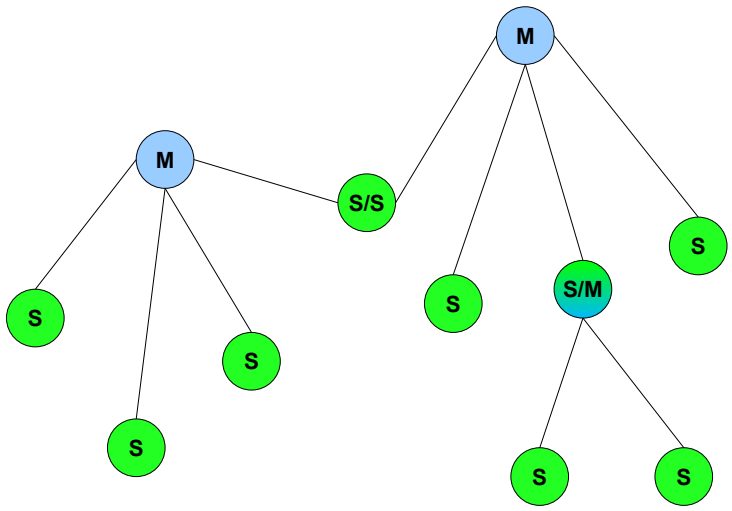


Figure 3.7: A Scatternet M=Master, S=Slave, S/S= Slave bridge, M/S=Master bridge

A *scatternet formation algorithm* is responsible of establishing the scatternet. Addition to the scatternet formation algorithm a routing algorithm has to be implemented in the network. The different routing algorithms are designed to work in a specific environment incorporating techniques to emphasis design goals as discussed in Chapter 2.2.

3.10 Scatternet formation algorithms

Several algorithms for scatternet have been proposed. Some algorithms are limited to a situation where all nodes are within radio range (single hop), whereas other do not have this constraint (multi hop). A comparison of the different formation algorithms is basically based on following criteria [6]:

- Duration until the scatternet is formed is low
- Average number of piconet is low
- Average piconet size is small

- The scatternet is fully connected
- The algorithm has a low message and time complexity
- The delay for topology maintenance is small
- A high aggregate throughput and a small average packet transfer delay between any pair of nodes

None of the proposed scatternet formation algorithms is fulfilling all of the above requirements. [4] compare some of the proposed algorithms, and evaluate the performance of three of the protocols (Blutrees [74], BlueStars [52] and New Formation protocol [38]) using simulation. Slightly different criteria than the ones above have been emphasized in the comparison. Recent papers propose other algorithms like the *Overlaid Bluetooth Piconet and Temporary Scatternet* [34] and *Enhanced AODV* [3]. The first one proposes an interconnection method without scatternet formation, instead interconnecting stand alone piconet. A performance comparison of OBP and Bluetooth Scatternet [9] is made in [33], and conclude a resiliency to mobility and higher throughput. Low power consumption is emphasized as an important feature in the latter one (EAODV).

A classification based on the formation algorithms limitations to be within radio range of all nodes or not, can be done as in table 3.3. In practical MANETs and WSNs

Single hop	Multi hop
Bluetooth Topology Construction Protocol (BTCP)	Bluetrees [74]
Scatternet Formation Algorithm [9]	Bluenet [69]
Overlaid Bluetooth Piconets [34]	Tree Scatternet Formation Algorithm (TSF) [63]
Temporary Scatternets [34]	BlueStars [52]
	New Formation protocol [38]
	Enhanced AODV [3]

Table 3.3: Classification of Scatternet Formation Algorithms

the geographical prevalence is likely to be bigger than the radio range of each single node, i.e. 10m radius. Thus scatternet formation algorithms in the left column of table 3.3 are primarily of theoretical interest.

3.11 Scatternet security

A scatternet challenges the same security issues as a Bluetooth piconet. Liong and Barnaghi propose in a paper [41] an approach to use asymmetric encryption to exchange session key. The session keys are used to symmetrical encrypt data. This approach eliminates eavesdropping and man-in-the-middle attack. However, the paper does not

address how to block 'bad' node joining the scatternet. After successfully joining the scatternet, it is possible to perform denial of service attack from inside the network.

Chapter 4

Network Coding

A model where a network is presented as a directed graph is adopted from [7], where the communication links are edges and the network components (e.g. routers, switches, data sources and sinks) are nodes or vertices, as shown in Figure 4.1.

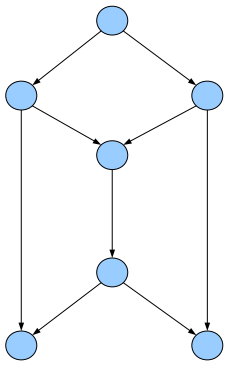


Figure 4.1: Network presented as a graph

4.1 The Max-Flow-Min-Cut Theorem

The max-flow-min-cut theorem is defined as [10]:

> The maximal flow value from s to t is equal to the minimum of the capacities of cuts separating s from t.

Where s is the source, and t is the sink in a directed graph.

A cut in the graph is a set of edges such that, when removed, will disconnect the source from the sink. The sum of the capacities of the edges in the cut is called *the value of a cut*. A *minimum cut* is a cut with minimum value. There could be several different minimum cuts in the same graph. Figure 4.2 shows example of different cut for

the same graph. The number next to the edges is the capacity of the edge. The values of the cut for 4.2(a), 4.2(b), 4.2(c), 4.2(d) are 5, 11, 18 and 15 accordingly. The cut 4.2(a) is therefore the minimum cut. And based on the max-flow-min-cut theorem, the maximum flow from s to t in the graph is 5.

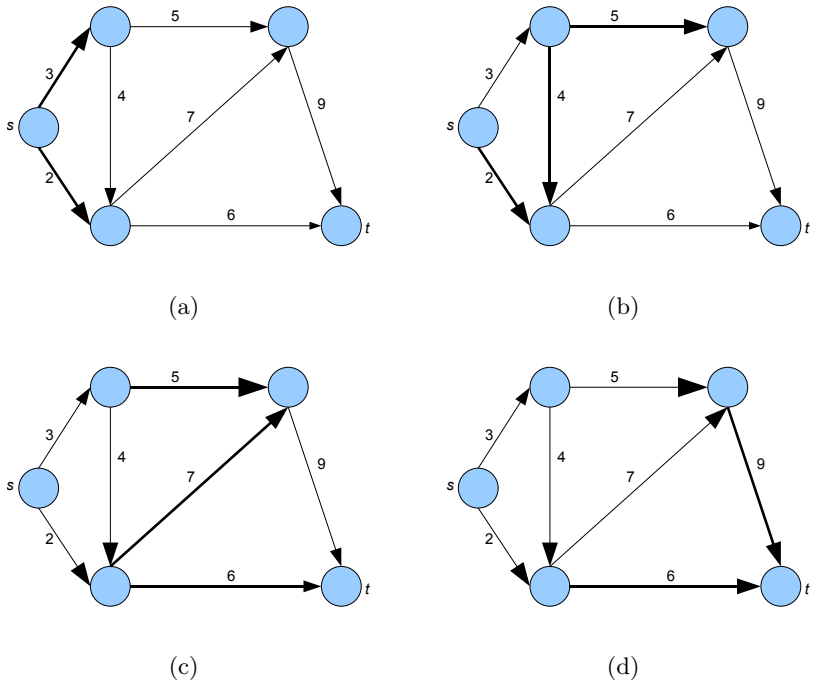


Figure 4.2: Cuts for a graph

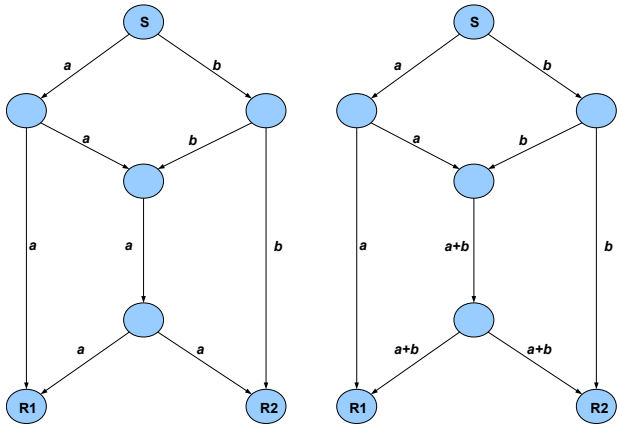
The Ford-Fulkerson algorithm [16] can be used to find the maximum flow path. Note that the maximum flow assumes a single source. The max-flow-min-cut theorem will be applied in the following as a reference for the discussion.

4.2 Introduction to network coding

Intermediate network components, like routers, have traditionally not performed any processing of packets. Rather routers stored and forwarded essentially unmodified packets. Network coding, on the other hand, allows intermediate nodes to perform coding in the network [73]. In network coding, an intermediate node is typically forwarding a (linear) combination of the received incoming packets [7]. Network coding is considered to be discussed the first time in a seminal paper in 2000 by Ahlswede et al [2]. The benefits of network coding are to be able to improve throughput, robustness, complexity and security [26]. The benefits are further discussed in Chapter 4.4. Web pages dedicated to network coding containing useful information about this field can be found at [37] and [46].

Consider a network with one source node and two receiving nodes, as outlined in Figure 4.3(a). This network is commonly referred to as the butterfly network. The source transmits two packets (a and b) per time unit on the outgoing links. All links have a capacity of transmitting one message per time unit. According to the max-flow-min-cut theorem, the capacity from the source to the sink nodes e.g. node R_1 and R_2 is two packets per time unit. However, the max-flow-min-cut theorem is under the condition of single unicast, not multicast. Using traditional multicast and store-and-forwarding routing, it is not possible to achieve the max-flow for both nodes at the same time. Instead, one node is able to receive one packet and the other two packets. E.g. node R_1 only receives message a, and R_2 receives both message a and b.

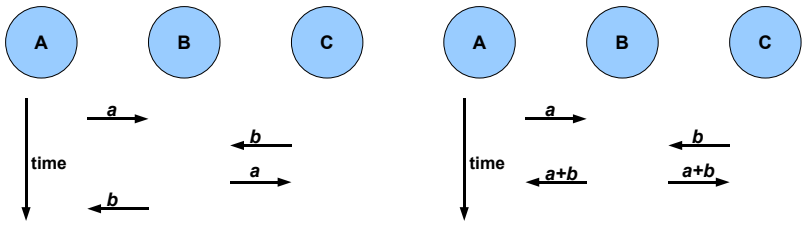
If we instead apply network coding, and the nodes are allowed to combine the network traffic, both nodes can receive both messages as shown in Figure 4.3(b) and the max-flow-min-cut rate is achieved for both of the receiving nodes at the same time. Node R_1 calculates b by combining the two messages a and $a+b$. Node R_2 calculates a by combining the two messages b and $a+b$.



(a) Without network coding (b) With network coding

Figure 4.3: A butterfly network

A wireless network is another application of network coding that increases throughput. Figure 4.4(a) illustrate how traffic between two wireless nodes A and C are routed through node B. Node A send packet a to node B, and node C send packet b to node B. Then node B send packet a to node C, and finally packet b to node A. This could be done using network coding as illustrated in Figure 4.4(b). Node B receives packet a and b from node A and B respectively, combines them in one packet $a+b$ and sends one single packet to both node A and B. Node A and B are able to solve the equation and get the other message. This example is also discussed by Shannon in 1961 [58] in what he called a two-way channel.



(a) Without network coding (b) With network coding

Figure 4.4: Network coding in a wireless network

4.3 How encoding is performed

Intermediate nodes receiving more than one packet combine the packets and forward this combination to the next node. The combination of the packets is performed over some finite field $\mathbb{F}_q$ [19]. The coefficients used to perform this operation constitute of what is called a *local encoding vector* $c^{\ell}(e)$ for edge e see Section 4.3.1. Throughout a network this operation may be performed several times.

By tracking the local encoding vectors recursively, the *global encoding vector* is found [73]. The global encoding vector is necessary for the sink to decode the received packets.

Given a sufficient field size q, use of linear network coding is sufficient. Furthermore, when the local encoding equations are linear, the global encoding equations also become linear.

The process of determining the encoding equations can be approached in two different ways. These are known as *deterministic linear encoding* (Section 4.3.4) and *random linear encoding* (Section 4.3.5). For all known efficient algorithms for deterministic network coding it is necessary to know the network topology before assigning local encoding equations [7]. As the knowledge of the network is centrally available, this approach is considered a *centralized* algorithm. The random encoding approach, on the other hand is considered a *distributed* algorithm as no central knowledge of the topology of the network is required.

An overview of how network coding is performed is given in [17]. Assume that an original packets consists of L bits. In case of different length packet, the shorter ones are padded with trailing 0s such that all packets are of equal length. s consecutive bits of a packet is treated as a symbol over the field $\mathbb{F}_{2^s}$. The original packet is therefore considered consisting of L/s symbols.

4.3.1 Encoding

Given n original packets $M^1, ..., M^n$ generated in the network by one or multiple sources. The encoding is performed by combining the original packets and the coefficients $g_1, ..., g_n$. The vector of coefficients is called an encoding vector. The combined packet

X is equal to

$$X = \sum_{i=1}^{n} g_i M^i$$

This is known as the local encoding equation. The summation is performed for every symbol position. The kth symbol is therefor $X_k = \sum_{i=1}^{n} g_i M_k^i$, where M_k^i is the kth symbol of M^i. If the symbol length is 1, i.e. the field $\mathbb{F}_2$, the combination is simple bitwise *xor*.

A node that has received and stored a set of encoded packet may generate a new encoded packet X' by combining the stored packets and its local encoding vector h_j, $X' = \sum_{j=1}^{m} h_j X^j$, where X^j is a stored encoded packet. This operation may be performed at several nodes in the network. Thus encoding can be performed recursively to already encoded packets.

4.3.2 Decoding

Received packets are decoded by solving the system $X^j = \sum_{i=1}^{n} g_i^j M^i$, where X^j and g^j are the set of the m received encoded packets and its corresponding coefficients, recursively. The receiver needs $m \leq n$ packets to be able to recover all data. The equations can be solved using Gaussian elimination. The received packets are stored in a decoding matrix, and by using Gaussian elimination the matrix is transformed to triangular matrix. If the received packet is not innovative, i.e. the packet is a linear combination of previously stored packets; it is reduced to a row of 0s by the Gaussian elimination. An innovative packet increases the rank of the matrix. A solution is found at latest when n linear independently packets have been received.

Practical implementation of network coding operates on a limited size decoding matrix. It is possible to balance this during design of a deterministic network code, but is more difficult for random network coding. A possible solution is to group packets into *generations*, so that only packets of the same generation are combined.

Network coding does not require decoding to be performed throughout the network, only at the receiving nodes.

4.3.3 Global encoding equation

It is possible to determine the *global encoding equation*, when all the local linear encoding equations are defined. The global encoding equation is found by combining all the local encoding equations influenced to the packets on the path from the source to the sink.

4.3.4 Deterministic linear coding

Deterministic linear encoding requires knowledge of the entire network to determine and design the encoding equations. Thus this approach requires preprocessing before encoding can be performed. However, as long as the topology is stable it is not necessary to re-design the encoding. There are several approaches on how to assign encoding equations [19], and special considerations have to be made if the network contains cycles

[5]. All known approaches includes a centralized algorithm, however it is not proven non-existence of distributed approaches.

After succesfully design of the encoding equations, these are distributed and assigned in the network. Once assigned, the encoding equations are used during all successive transmissions. As all local and global equations are known beforehand, overhead is not necessary. The centralized requirement of the deterministic linear encoding and requirement for re-design of encoding equations whenever changes in the network occurs, makes this approach poorly suited for ad hoc networks that change topology frequently.

It is previously stated that the field $\mathbb{F}_q$ has to be of sufficient size. It has not been determined up till now what is sufficient. A basic upper bound on the requirement for q is that $q \geq r$ [56] always works, where r is the number of sinks. Also if $r^*(e)$ is the number of sinks that uses the edge e, then $q \geq max\ r^*(e)$, $e \in \varepsilon$ [5]. Later work [18] proved existence of a particular network of two unit rate sources h, and gave an accurate maximum bound of $q \geq \lfloor \sqrt{2r - 7/4} + 1/2 \rfloor$. In [7] an opportunistic approach is proposed, where one start in $\mathbb{F}_2$ until no more edges can be encoded over that field while maintaining the full rank invariant, i.e. the encoding equations are linear independent and it is possible to decode the combination. At that point the encoding is shifted to an extensive field. The work proves the possibilities to work under a smaller field size than the previous mentioned bound.

4.3.5 Random linear network coding

When the network topology is known, it is possibly to use a deterministic algorithm to find network encoding equations. However, when the topology is not known, use of *random linear network coding* is required [7]. The algorithm is called random because each encoding node chooses its own encoding coefficients randomly. The coefficients are chosen uniformly distributed from a sufficiently large field. The process of constructing and finding network encoding coefficients is more flexible using random coding; however a much larger base field is usually needed [73].

As opposed to deterministic network coding, where the local and global encoding equations are distributed from a central node, each node in a random network coding network choose its own encoding equations. The global encoding equations are added to the header of the packet. Each node performing combinations of packets, also calculate the new global encoding equation g_i' by adding its local encoding equation h with the previous global encoding equation g_i. This is done by using straightforward algebra [17]

$$g_i' = \sum_{j=1}^{m} h_j g_i^j$$

The new global encoding equation forwarded along with the combined packet. This operation is repeated at each node that combines packets.

A receiver is able to restore the original K packets after receiving K linear independent packet forming a set of K linear equations. Ho et al. [28] showed that with

uniformly distributed network coding coefficient in a finite field $\mathbb{F}_Q$, all the sinks can decode the source with a probability at least $(1 - d/Q)^\eta$ for $Q > d$, where d is the number of sinks and η is the number of links with randomized coefficients.

4.4 Network coding benefits and challenges

4.4.1 Benefits of network coding

Throughput

It is shown previously in this chapter different application of network coding increasing throughput in a network. In fact there are cases where network coding give a solution to a problem not solved using traditional routing. One of them is achieving the max-flow-min-cut boundary to each of the nodes simultaneously in the butterfly network. A paper by Chekuri et.al. [11] describes the benefits over routing using network coding. [64] propose a routing algorithms of network coding on multicast.

Robustness

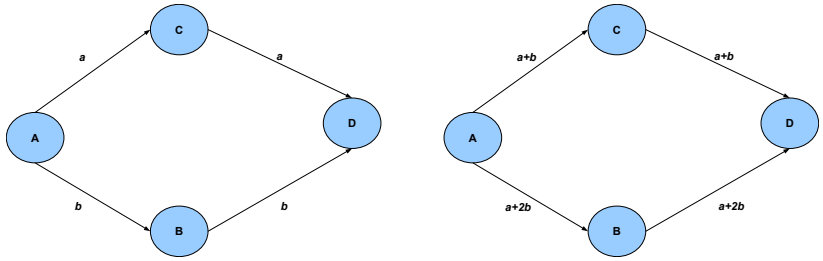
When (random) network coding is applied to wireless networks, the network becomes more robust against packet losses. Such packet losses can arise from link failure, buffer overflow and collisions of packet on the medium in a shared medium topology. Traditional, erasure coding is applied at the sender side and decoding at the receiver side. If instead erasure coding was applied on all links, the robustness would increase at the cost of the delay in the network. However, network coding gives this benefit without requiring decoding on every link, and hence without adding further delay in the network. This example is described in [17].

Complexity

Finding optimal routing is, in some cases, a complex task. Finding optimum routing using network coding is linear optimization [26]. In fact network coding can enable communication that is not achievable using traditional routing. The work by Ho et al. discusses network coding versus routing in paper [27].

Security

Network coding is able to add some security to the network. However, it raises some security concerns as well (see Chapter 4.4.2). Due to the nature of the linear combination of packets, tampering can be discovered [17]. This can be illustrated by an example: Consider a network of four nodes as shown in Figure 4.5. The source node A transmits two messages to node D through the intermediate node B and C. All links have unit capacity. The traditional routing way of doing this is shown in Figure 4.5(a). An eavesdropper listening to one of the paths AB-BD or AC-CD is able to get this information. If instead network coding is applied, it is possible to transfer the same amount of data



(a) Without network coding (b) With network coding

Figure 4.5: Security benefit using network coding

from A to D 4.5(b). An eavesdropper listening to either of the paths is not able to deduct either of the messages a or b. The receiver D is able to combine the two packets ($a+b$ and $a+2b$) to get both messages a and b. A requirement for this to hold is redundancy in the information transmitted. If not, network coding enables an eavesdropper to recover both data streams at a single location. [19] outlines a protocol to protect against Byzantine attacks using network coding.

4.4.2 Challenges using network coding

Even though network coding will give benefits, it also raises some challenges. Some of them are discussed in [19], and these are:

Complexity

The nodes in a network deploying network coding are required to store their own information and received packets. Thus network coding requires additional memory. In addition, the nodes need to perform operations over finite fields, and solve a system of linear equations to decode data. This requires additional computations. Both memory and computational power may be a challenge in ad hoc networks of small battery operated mobile nodes. [19] discusses a trade off between complexity and performance, where complexity refers to number of coding points as well as the size of the coding alphabet. Challenged node with limited resources may consider these trade-offs.

Security

From a security point of view, intermediate nodes should not modify data. In network coding intermediate nodes performs operations on the data. It is therefore necessary to deploy systems to maintain authenticity of the data transferred, despite use of network coding.

Integration of existing infrastructure

An open question is how to integrate network coding in existing network architecture. It is interesting, from an integrator's point of view, to benefit from network coding without major replacements of components. A Microsoft research project called *Avalanche* [53] applies network coding to perform fast file distribution over P2P in a system called *MS Content distribution*. The basic idea of the content distribution is to use network coding as opposed to swarming. In a swarming system peers download different pieces of information from multiple nodes. A problem in such a system is optimal scheduling of pieces to nodes, and problem distributing *rare* packets. Nodes in the network coding content distribution system receiving a piece of information, will produce a linear combination of the received piece and the block it already holds and pass this block to its peer. Included in the block is also the encoding equation. This encoding ensure that any piece of uploaded by any peer can be of use to any other peer. As soon as a peer has sufficient linear independent combinations, it is able to decode that part of the information and rebuild the original file. An analysis of the Avalanche application is made by Yeung [72].

4.5 Network coding applications

After the first initial paper discussing network coding [2] a number of papers have been produced. The published papers both discuss network coding framework and how to apply network coding in practical contexts. In addition to the previous mentioned (Chapter 4.4.2) Avalanche project by Microsoft Research [53] a simulation study on network coding parameters in P2P content distribution system is done Zeng et al. [75] and a P2P file sharing based on network coding by Yang and Yang [71]. Sundarajan et.al. discusses in a paper [62] how to incorporate network coding into TCP. Park et al. [47] discuss the performance of network coding in ad hoc networks. The paper confirms the established analytic results that network coding achieves higher throughput and lower power consumptions than conventional multicast.

In a paper by Li and Kong[40], use of network coding in WiMAX is explored. They observe an improved throughput using network coding on MAC layer compared to traditional *Hybrid Automatic Repeat reQuest* (HARQ). In a paper by Katti et al. [35] studies use of network coding in wireless mesh networks. Simulation and practical implementation shows an increased network throughput using a proposed architecture called *COPE*. A mesh network is an ad hoc network, where the nodes relay traffic on behalf of other nodes, extending the range of the network (Figure 4.6).

The COPE architecture adds a coding shim between the IP and MAC layer, and identifies coding opportunities. As opposed to traditional mesh network, the COPE nodes use broadcast extensively and record all overheard packets for a time period T (default $T = 0.5s$). In addition, all nodes broadcast a *reception report* periodically containing information about which packets each node know about. This enables COPE to opportunistic decide the optimal combination of packets in the next transmission. See the example in Figure 4.7 from [35]. The output queue of node B contains the

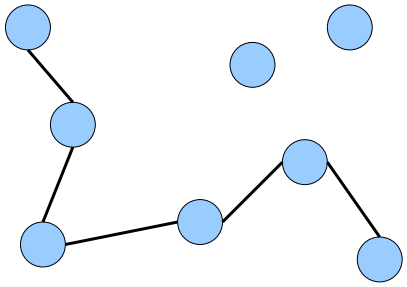


Figure 4.6: A mesh network extending the range by relaying

packets P1, P2, P3 and P4, and node B has to decide the best optimal combination of these packets such that most number of nodes is able to decode. As the example shows, the best combination would be to send P1+P3+P4, as this option enables all nodes to decode and increase their packet pool with one packet. The paper shows theoretical coding gains from 1.33 up to 2 depending on the network topology.

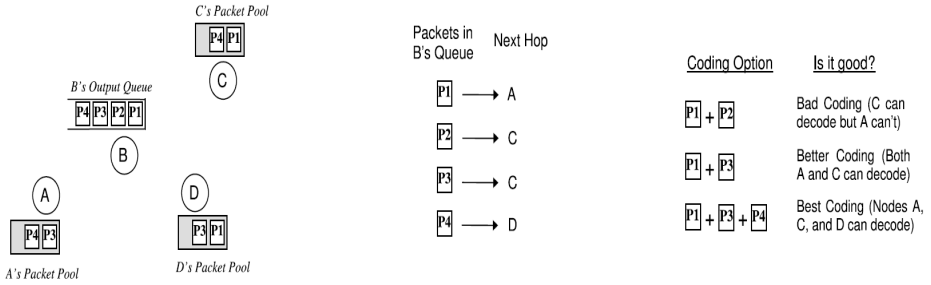


Figure 4.7: An example of the COPE coding

A framework for network coding in challenged wireless network (FRANC) [15] is a framework to deploy network coding in ad hoc networks. FRANC is implemented in Java, and applies random network coding.

Chapter 5

Network coding in a Bluetooth network

The author of this work is not aware of any effort made on applying network coding in Bluetooth networks. However, Bluetooth is widely used today [23] and the devices are cheap and have a low power consumption. Use of Bluetooth in a MANET as well as WSNs is therefore interesting.

The overall aggregated bandwidth in a piconet is limited to 768kbps (Basic Data Rate). Using TDD, this bandwidth is shared between the nodes in a piconet based on the fairness algorithm applied [6]. Thus the average bandwidth per node is reduced by the number of nodes transmitting in a piconet. If applying network coding to the piconet makes the communication more efficient, the required number of transmitted packets will decrease. A reduced number of transmitted packets will reduce the radiated power from the node. Furthermore, reduced radiated power will reduce the energy requirement, and increase the lifetime of the battery while maintaining the same bandwidth available. As Bluetooth is primarily used by small battery operated devices, an improved energy budget is welcome.

5.1 Theoretical analysis

Assume a piconet consisting of a master node and two slave nodes as shown in figure 5.1. The two slave nodes communicate with each other full duplex at equal data rate (symmetrical).

According to the TDD scheme of Bluetooth, the master polls the first slave. The slave responds and sends its packet destined for node 2. The master polls the next node, and passes data from the first node to the second node. The second node responds, and passes its data to the master. This cycle repeats itself until the communication terminates, this protocol is illustrated in Figure 5.2(a). Assume again the same network topology as previously. Instead of following the previous protocol, the first two polls from the master does not transfer any data but rather send simple poll packets with no payload. The master stores the received data from node 1 and node 2 in a buffer. It

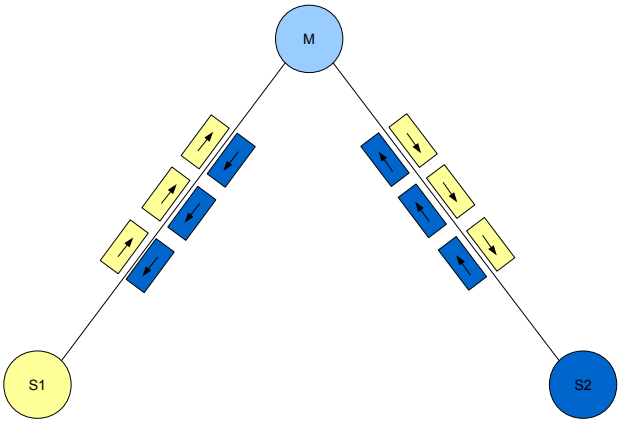
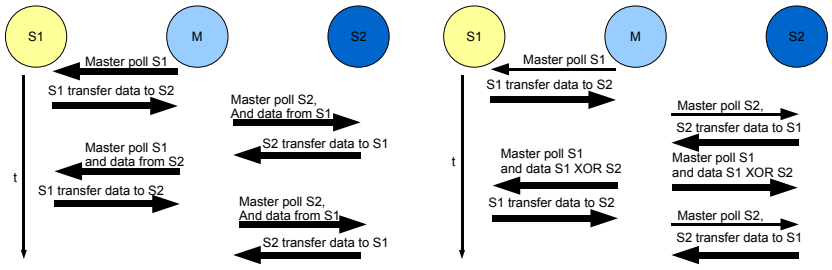


Figure 5.1: A simple piconet

then performs combination of the two packets, and sends the encoded packet to both nodes as illustrated in Figure 5.2(b). This is possible either by using the reserved *Active Slave Broadcast* address (as carried out in an Emergency Data Delivery System [39]) or forcing the receivers to operate in promiscuous mode. If possible in the hardware implementation, the latter is preferred due to polling scheme issues. If broadcast address is used, a separate poll packet has to be transmitted. Recall from Chapter 3.2.2, that a poll packet is 126 bit long, and the power budget will be increased accordingly.



(a) Without network coding (b) With network coding

Figure 5.2: A two slave and one master Bluetooth network

5.1.1 Throughout improvement in a piconet by using network coding

Is it possible to improve the throughput in a piconet using network coding? This section will discuss how this could possible be done, and a protocol for *deterministic network coding in Bluetooth piconet* (DNCBP) is proposed. The reason for choosing deterministic network coding over random network coding will be addressed in Chapter 5.2. Given

a network of a master node and four slave nodes. Between the four nodes traffic flows bi-directional and symmetric in pair as shown in Figure 5.3. Recall from Chapter 3 the

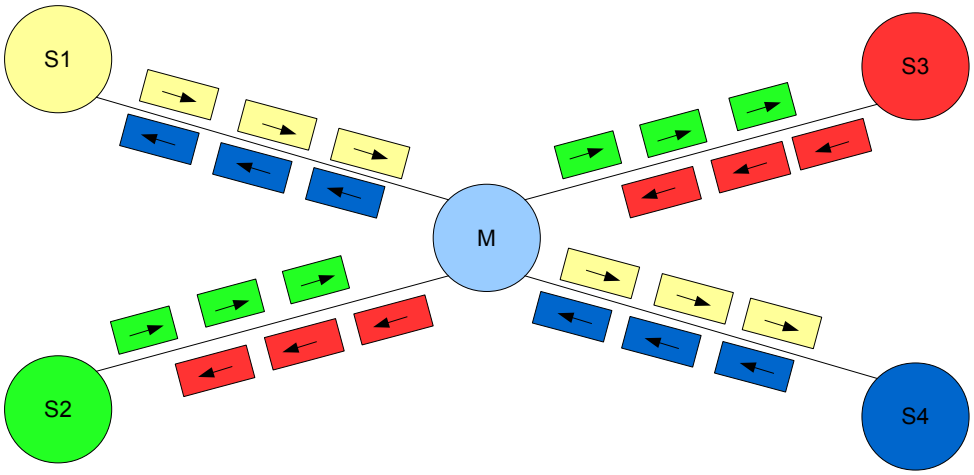


Figure 5.3: A piconet with four nodes and two bi-directional symmetric communication streams

Bluetooth frame structure, at every even numbered time slot the master polls a slave. The next time slot data from the polled slave node is transmitted. To explain how to apply network coding, a Bluetooth piconet could be viewed as a directed graph as in Figure 5.4. On the right hand side of the figure is the transmitting part of the slave nodes (the sources), and the receiving part of the slave nodes (the sinks) is on the left hand side. The master node is in the middle. Figure 5.5 indicates a possible distribution and use of time slot in a TDD system. The overall capacity C is divided by the users. Transmission of one packet from one slave to another slave, forwarded by the master is requiring two time slots; one from the source slave to the master, and another time slot from the master to the sink slave. Thus, with n slaves the average capacity per node C_n, is given by

$$C_n = C/2n$$

By applying broadcasted network coding on the communication from the master node to all the slave nodes, the master is able to communicate to all slaves in each of its time slots at the same time. A representation of this mode of operation is illustrated as a directed graph in Figure 5.6. On the right hand side the transmitting part of the slave

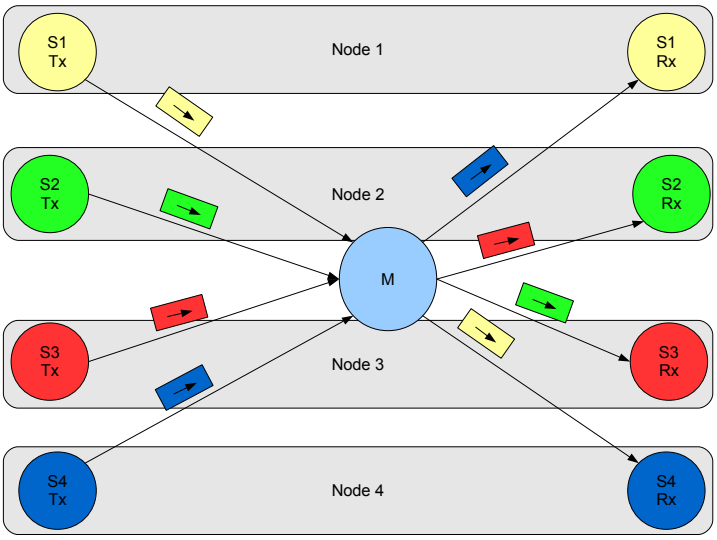


Figure 5.4: A piconet as a directed graph

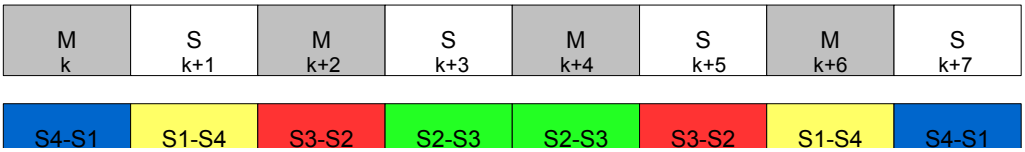


Figure 5.5: TDMA frame structure

nodes (sources), and the receiving part of the slave nodes are combined in a sink to the left. The edge between the master node and the sink on the left hand side, represent the broadcast from the master node to the slave nodes.

This will reduce the required bandwidth from the master; ergo it is possible to allocate more bandwidth to the slaves. In a Bluetooth network this is done by using DH3/ DH5 or DM3/DM5 in an asynchronous mode. In this mode the nodes are allowed to use three or five consecutive time slots as opposed to one. The receiving node requires $m-d$ linear independent packets to decode the message. Where m is the number of receivers and d is the number of known messages. (In the previous described butterfly network m is 2, and d is 1. The number of packets required is therefore 1.) The Bluetooth nodes know only its own information. Thus d is 1, and the required number of packets is $m-1$. Having four receiving nodes, the receivers require three linear independent packets to decode. Therefore, in this case the master is required to broadcasts three packets with the same length as the incoming packets.

Due to the TDD scheme the master nodes have four periods available. By using a synchronous mode, all frames are one time slot long, equally shared among all slaves and

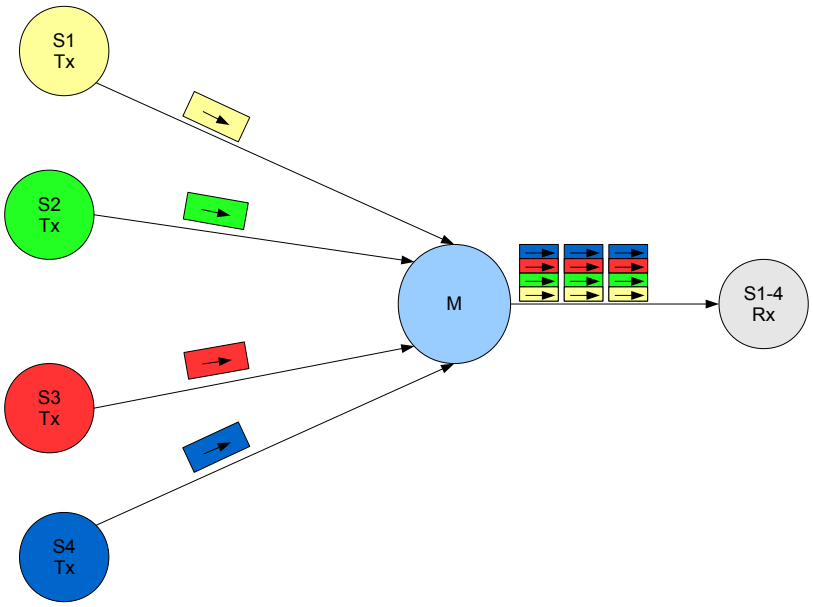


Figure 5.6: A piconet applying network coding illustrated as a directed graph

the master. Therefore, there will be no throughput improvement using network coding in synchronous mode as the master is required by the TDD protocol to transmit in all four time slots, even though the network coding only requires three.

By increasing the size of the packets a message can consist of three or five time slots. The empty frame from the master is set to a length of one time slot, and contains only the poll packet. It is possible to, if required, add network coding overhead (e.g. administration such as number of nodes, mapping of nodes to BD_ADDR and advertizing a new generation) in this packet. A possible distribution of the TDD scheme could be as shown in Figure 5.7. The master node saves transmission of 2 or 4 time slots depending on using DH3 or DH5 accordingly.

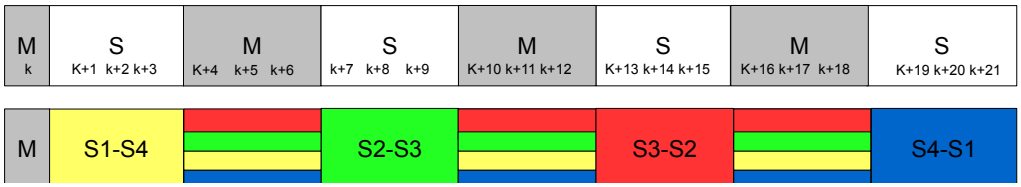


Figure 5.7: TDMA frame structure in a piconet with network coding

In the given example, each unidirectional communication link requires three time slots to transmit the date from slaves to master, and three times three time slots to transmit from the master to the receiving slaves. In total 22 time slots including one

poll packet. In general the required number of time slots to transmit a DH3 packet from all source nodes are $3n + 3(n-1) + 1 = 6n - 2$.

Theorem 1. *The average throughput gain, G, using DNCBP-DH3 is* $G = \frac{3n}{3n-1}$, *where n is number of transmitting nodes.*

Proof. The aggregated throughput in a Bluetooth network is found by

$$\begin{aligned} \text{throughput} &= \frac{\text{bytes transfered}}{\text{time used in seconds}} \\ \text{throughput} &= \frac{\text{bytes transfered}}{\text{number of timeslots used} \cdot \text{time per timeslot}} \end{aligned}$$

If the throughput in two different scenarios is to be compared, and the amount of data transferred is equal in the two scenarios. I.e. bytes $\text{transferred}_\text{A}$ = bytes $\text{transferred}_\text{B}$. The aggregated throughput gain is found by

$$\begin{aligned} \text{gain} &= \frac{\frac{\text{bytes transferred}_\text{A}}{\text{number of timeslots used}_\text{A} \cdot \text{time per timeslot}}}{\frac{\text{bytes transferred}_\text{B}}{\text{number of timeslots used}_\text{B} \cdot \text{time per timeslot}}} \\ \text{gain} &= \frac{\text{number of timeslots used}_\text{B}}{\text{number of timeslots used}_\text{A}} \end{aligned}$$

Based on this it is possible to find the aggregated throughput gain by counting the number of time slots used in the two scenarios to transfer a given amount of data.

The aggregated throughput gain G node using DNCBP-DH3 is:

$$G = \frac{2 \cdot 3n}{6n - 2} = \frac{3n}{3n - 1}$$

□

Theorem 2. *The average throughput gain, G, using DNCBP-DH5 is* $G = \frac{5n}{5n-2}$, *where n is number of transmitting nodes.*

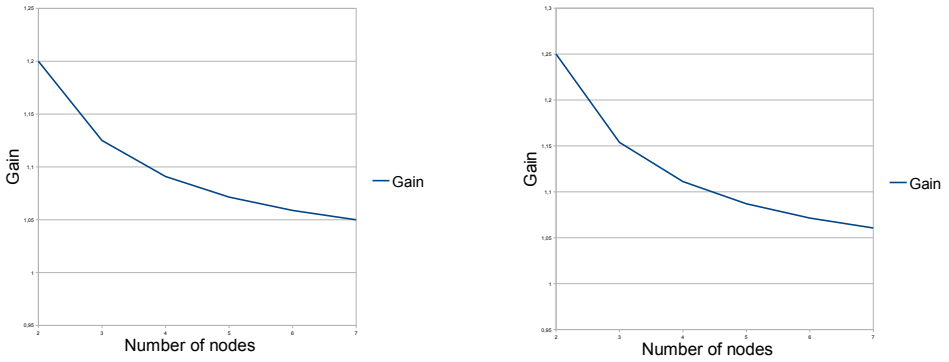
Proof. By following the same argument as in theorem 1 the number of time slots required to transmit a DH5 packet from all source nodes are $5n + 5(n-1) + 1 = 10n - 4$. And the number of time slots required to transfer the same amount of data is $2 \cdot 5n$. The gain G in average capacity per node using network coding and DH5 packet is:

$$G = \frac{2 \cdot 5n}{10n - 4} = \frac{5n}{5n - 2}$$

□

Figure 5.8, 5.8(a), and 5.8(b) shows the gain in a piconet using network coding with DH3 and DH5 accordingly. The result is also presented in Table 5.1.

In a piconet with 6 slave nodes transmitting data, the gain using DNCBP-DH3 is 5.9% and 7.1% using DNCBP-DH5. Thus the throughput in a Bluetooth piconet is improved by using network coding.



(a) 3DH Async mode

(b) 5DH Async mode

Figure 5.8: Gain by using network coding Bluetooth

	Number of transmitting slave nodes					
	2	3	4	5	6	7
DH3	1,200	1,125	1,091	1,071	1,059	1,050
DH5	1,250	1,154	1,111	1,087	1,071	1,061

Table 5.1: The gain using deterministic network coding in a Bluetooth piconet

5.1.2 Reduced energy consumption using network coding

Previously it has been shown how the amount of data transferred from the master node is reduced, still maintaining the same amount of information. The time a Bluetooth device is transmitting is dependent of the amount of data. Consequently, a reduced amount of data will reduce the time a master is transmitting. Reduced transmitting time will reduce the energy consumption of a Bluetooth node. As shown in Figure 5.7 the slave nodes do not reduce their amount of data, and therefore will not reduce their power consumption.

Theorem 3. *The master node reduces its energy consumption by a factor of* $\frac{n-1}{n} + \frac{126}{p^2 \cdot 625}$, *where* $p = 1, 3, 5$ *using DNCBP.*

Proof. From Figure 5.5 it is easily seen that the master node transmits n packets, with a length of three time slots (asume 3DH). Thus, the time the master transmitts in each cycle is $3n \cdot \frac{1}{1600}$, where $\frac{1}{1600}$ is the time of one timeslot. When using DNCBP-3DH it is seen from Figure 5.7 that the master transmits $3(n-1) +$ poll packets. The poll packet is 126 bit long. One time slot is 625 bit long, thus the poll packet is about 1/5 of a time slot. When operating 3DH and 5DH the poll packet is much smaller (1/15 and 1/25 of a packet accordingly), and could be neglected. The power required by the master to

transmit the same information using DNCBP as opposed to without network coding is

$$\frac{p(n-1)+\frac{126}{p\cdot 625}}{pn} = \frac{n-1}{n} + \frac{126}{p^2 \cdot 625}$$

By neglecting the poll packet, the power reduction is not a function of the packet length. I.e. it is not dependent to which of DH3 or DH5 packet to use. However, the effect of neglecting the poll packet increases by choosing a smaller packet size. □

5.1.3 Throughput improvement in a scatternet using network coding

Is it possible to improve the throughput in a scatternet using network coding? This section will discuss this question, based on the discussion on piconet.

Assume four bidirectional unicast in a scatternet as shown in Figure 5.9. In this

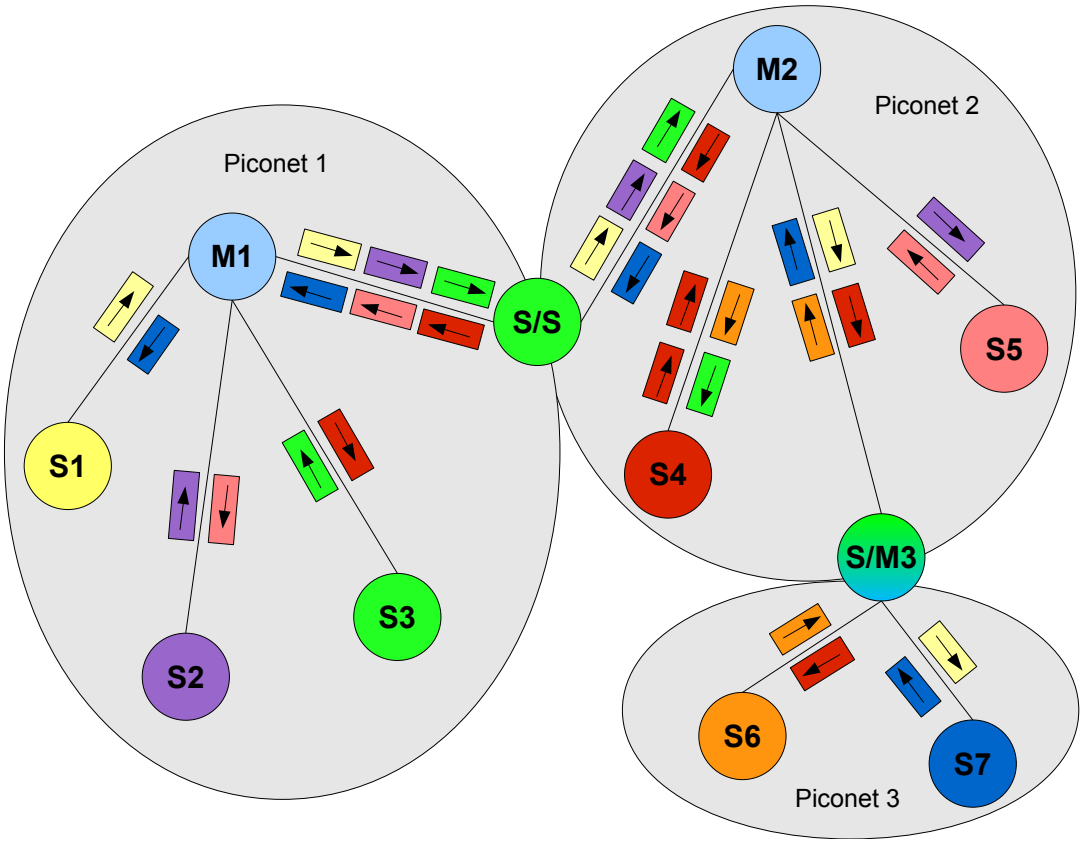


Figure 5.9: A scatternet with four bidirectional unicast

situation the gateway node between piconet 1 and piconet 2 is a slave-slave node. As a slave is not able to communicate directly to any other slave node, the gateway has to forward all the traffic between the piconets through the master nodes on both piconets. Thus, it is possible to benefit from applying network coding on the link to a slave-slave

node. The Figure 5.9 includes also a situation of a master-slave gateway. The links between the master-slave gateway and the M2 node transfer traffic to and from both S6 and S7. In piconet 2 the M3 is a slave, and it is possible to apply network coding on the traffic from M2 to the slave (and M3) in the piconet. The traffic from the M3 node to S6 and S7 yields the same argument as Chapter 5.1.1. Recall from Chapter 3.9 that a gateway node is not able listen to both piconets at the same time. If it is possible to make the communication from the master to the gateway node more efficient, the throughput between the piconets is increased. However, a protocol on how the gateway node balancing between the piconets ensuring reception of all packets necessary has to be established.

5.2 Implementation of network coding in a Bluetooth network

Based on the discussion above network coding could be implemented in a Bluetooth network under the following conditions:

- From the master node to the slaves in a piconet
- From the master node to the slave gateway node in a scatternet

Additionally, a possible use of network coding is from one or multiple nodes in a scatternet broadcast or multicasting messages. As the scatternet is highly mobile, it is likely that random coding is the preferred choice.

5.2.1 Network coding approach

As described previously there are two different approaches for network coding, namely deterministic and random linear encoding. Deterministic linear encoding is preferred when the network topology is known [73]. The random linear encoding is preferred when the network topology is not known or the network is frequently changing. That is, nodes may join or leave the network or the routing is changed over time. The latter situation is the case in an ad hoc network. By its nature a mobile ad hoc network changes the topology frequently. Thus, it would be inefficient using a deterministic code. A central node would have to generate the coding coefficient every time the topology changes. In addition, the node responsible for generation of the coefficient could be detached from whole or major parts of the network. This encourages the use of random network coding.

Despite a Bluetooth piconet is ad hoc by nature, the proposed use of network coding is applied on link level. The master is the central node knowing the topology in the network, in this case the piconet. This encourages the use of deterministic linear encoding. Furthermore it is possible to determine and use static coding coefficient, because the topology is always a star with a maximum of seven slaves.

A drawback of using deterministic network coding is the lack of erroneous protection of the data. If an error occurs, the receiver is not able to receive all packets in a generation. As a result, it is not able to decode all or some of the packets. Random network

coding would add complexity, and require more computational power and memory the nodes. This encourages the use of deterministic codes if possible.

However, applying network coding on broadcast traffic from one or multiple sources in a scatternet will require random linear encoding as:

1. There is no central node holding information about the topology
2. Nodes leave and join the network
3. Routing is changed as the nodes moves

The rest of this work will focus on the piconet situation.

As slave nodes in a piconet are not able to receive traffic directly from other slave nodes, there will be no gain using network coding on communication from the slaves to the master as this is a unidirectional link. Furthermore, a two way communication between the master and each of the slaves, does not give any gain using network coding due to the Bluetooth design limitation. See Figure 5.10. However, when a slave is source and another slave is a sink, network coding will give better performance. Assuming a configuration as Figure 5.3 consisting of eight unidirectional or four bidirectional communication streams. The maximum number of bidirectional communication links between any two slaves in a piconet is $\sum_{i=1}^{n-1} i$, where n is the number of slaves. If $n = 7$ i.e. the piconet is full, the maximum number of bidirectional communications between any two slave is 21.

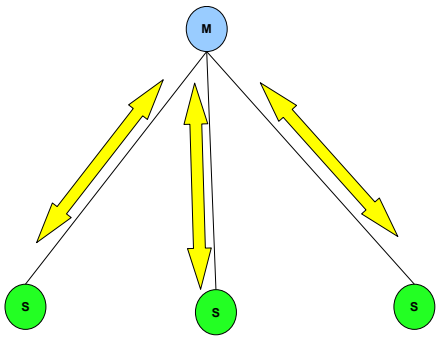


Figure 5.10: Two ways Master-slave communication

Cycles

The piconet can not contain cycles as the proposed protocol include only network coding on a single link.

Alphabet size

Using a binary alphabet is preferred due to a simpler implementation. $A \oplus B$ in the binary field, is simple bitwise XOR.

Theorem 4. *A binary field is sufficient to perform encoding by the DNCBP protocol.*

Proof. To be able to decode the received set of packets, and get the original information, require linear independent equations. In coding theory, this is equal to a Hamming distance of minimum 1. The master node sends $n-1$ combination of packets from n nodes. One of the packets is the packet transmitted by the nodes itself, thus one of the packets does not add any new information to the decoder. This is equal to removing one of the columns in the generator matrix. To ensure the possibility to decode the received set of packets, even when part of the information is known. The distance has to be increased by at least one. A minimum distance of 2 is therefore required, i.e. $d \geq 2$.

We would like to find a $[n, k, d]$ code, where n is code length, k is number of codeword and d is the distance of the code. The nature of the proposed algorithm gives $k = n-1$, for all n and k. A *maximum distance separable* code (MDS) is defined as [36] as a code where

$$d = n - k + 1$$

Substituting $k = n - 1$ in the definition, gives

$$d = n - k + 1 = n - (n-1) + 1 = n - n + 2 = 2$$

Vermani [68] states that the only binary MDS code is the trivial codes. These are the $[n, 1, n]$, $[n, n-1, 2]$ and $[n, n, 1]$ codes. We are therefore looking for a $[n, n-1, 2]$ code, which is a parity-check code with one parity check bit. And in the example of $n = 4$, the code is a $[4, 3, 2]$ code. Such code exists in the finite field $\mathbb{F}_2$, and one such code is the parity check code, shown here by its transposed generator matrix:

$$\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{matrix}$$

The encoding is possible to realize in the binary field using a trivial MDS code - the parity check code, and still keep full rank. Thus $GF(2)$ is sufficient alphabet size. □

Encoding

The encoding required in the proposed DNCBP protocol can be performed as listed below. Where the left most matrix is the situation with to slave nodes communicating bidirectional. This situation is a simple XOR of the two packets. The next matrix is used to encode in the situation where three nodes are communicating. Obviously it is not possible to communicate in pair with three nodes. Thus this situation requires one of the nodes to broadcast the same message to both of the other nodes. The third matrix is the situation where four nodes are communicating bidirectional pair wise.

	m_1
a	1
b	1

	m_1	m_2
a	1	0
b	0	1
c	1	1

	m_1	m_2	m_3
a	1	0	0
b	0	1	0
c	0	0	1
d	1	1	1

The matrix can be extended up to the general form

$$\begin{matrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \\ 1 & 1 & \dots & 1 \end{matrix}$$

where the rows are the original messages, and the columns are the combined packets

Packet lengths

Based on the discussion previously the length of the packets should be either 3 or 5 time slots to increase the throughput gain. The Bluetooth asynchronous modes support forward error correction (FEQ). If FEQ is used, the packets are called DM. The proposed protocol DNCBP does not add any erroneous correction. If required, DM could be used. Use of DM would decrease the efficiency, as the payload size decreases.

Choosing DH5 packets will increase the gain slightly over DH3 as shown in Figure 5.8. However, this will give a packet size of 339 bytes. For some applications this could be to long, and shorter messages like DH3 with a packet size of 183 is preferred. For the following the DH5 mode of operation will be used.

When encoding data, the combined data have to be of equal length. The shorter packets are padded with 0s to become as long as the longest packets. The padding is proposed done by the master during the encoding and not by the slave before transmitting. E.g. a slave transmitting a packet of 100 bytes, would only transmit the header and the 100 bytes. If the slave did pad the data up to 339 bytes, the power consumption would triple for that packet. Therefore the proposed protocol is to have the master pad data as necessary. This is illustrated in Figure 5.11.

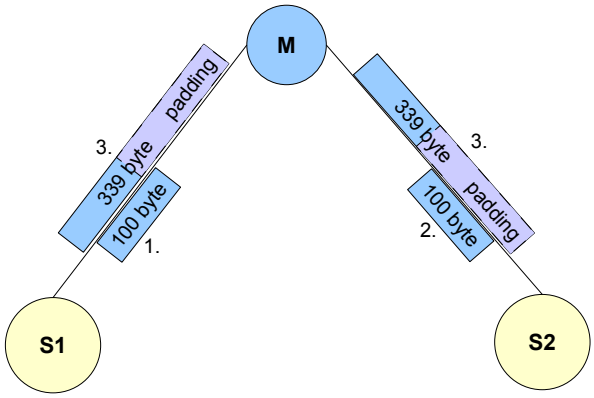


Figure 5.11: Padding of packets at master node

Security considerations

The proposed application of network coding in a Bluetooth network will enable all nodes to read all traffic in the piconet. In a piconet without network coding a slave is only receiving traffic destined by it self. However, there are recent presentation e.g. at the Shmoocon 2009 [24] and interview at the Revision3 Internet Television [54], showing that sniffing of Bluetooth network is not only reserved for expensive commercial sniffing tools. An article by Max Moser [42] gives an overview on how to turn a Bluetooth dongle into a sniffer.

Based on this security measures should be in place to maintain confidentiality between nodes in a piconet both when using network coding and not if confidentiality is important. There are applications (e.g. P2P file sharing and information board) where the information is supposed to be shared between the nodes in the network. However, in this case integrity of the messages could be an issue.

5.3 Possible application

The proposed protocol could be used to improve performance in respect of throughput and power reduction for a pair of nodes communicating. However, as all nodes are able to read information destined to the other nodes it is possible to use the same protocols in other application as well. This could be e.g. message board, and P2P file sharing.

Chapter 6

Simulations

There are several tools to simulate Mobile Ad Hoc Network, both commercial available and as open source. Some of them are compared in a paper by Hogie and Bouvry [29]. As the paper points out the simulations of MANET addresses other problems than wired network simulator. A popular network simulator is the NS2 [45]. NS2 is an open source network simulator. By default NS2 does not support the Bluetooth protocol. However, there has been developed an add on to NS2 at the University of Cincinnati called UCBT [44]. The simulator implements the Bluetooth stack, and different routing and scatternet formation algorithms. However, the NS2 simulator does not copy the payload between entities, but rather describe the size and type of the payload content in the header [31]. UCBT is not supported, and works only with an older version of NS2 (version 2.29). The benefit of adding network coding support to UCBT/NS2 is to take advantage of the propagation model, mobility model, scatternet formation algorithms and routing protocols included by NS2. However, these features are not critical for this project. NS2 could be used to simulate the benefits of using network coding instead of routing and/or scatternet formation algorithms.

SlimSim [13] is a simulator for network coding. However, this implementation does not add the MAC layer. It is not found trivial to extend SlimSim to support Bluetooth.

The proposed application of network coding in Bluetooth network is based on communication in an established piconet. Thus the time to establish the piconet (i.e. the joining procedure) will not impact the performance of the network coding. Furthermore, the scatternet and routing feature is not of consideration in this work. A nice feature of UCBT is the possibility to log the power consumption. However, after carefully reading the source code the power consumption is calculated by aggregating the transmit time. A similar approach to calculate power consumption will be done in this work as well. Based on this, a simulator has been created to compare a piconet running the proposed network coding against a traditional piconet. The simulator will assume an established piconet.

6.1 Description of the simulator

The simulator is written in Java. The rational for using Java as language as opposed to e.g. C++ is the possibility to reuse the code in other similar projects. A graphical user interface has not been developed to this simulator.

Every node (master and slave) in the network are running as a thread initiated by the main program. The nodes share the same channel, and read/write data to and from the channel in a TDD as in Bluetooth networks. The master node is responsible of manage the TDD scheme. At the initial phase of simulator each slave node instantiates a transmit buffer, and fills it with data. When all nodes have transmitted their data and the master node is done transmitting the encoded data, the simulator terminates. Figure 6.1 give an overview of the simulator. The work flow of the master is shown in

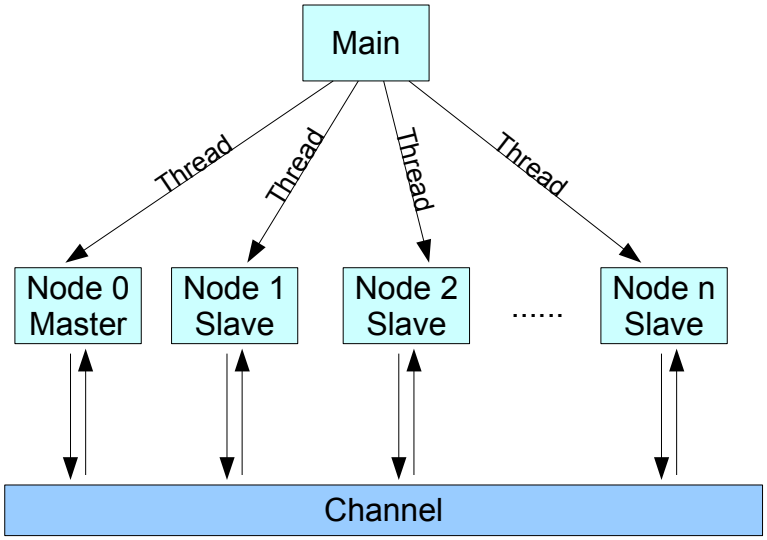


Figure 6.1: Overview of the simulator

Figure 6.2. The master node will initiate the TDD scheme by polling the first node. After receiving reply from the slave, the master will add the packet to the encoder. At the same time as the master poll the next node in TDD scheme, it will transmit the next encoded packet. The encoder buffers the packet from the n slaves, and whenever all n packets are received. The encoder produces *n-1* packets. One cycle of the TDD scheme is called a generation. Note there will be a delay of one generation between the slave nodes sending packets, and the master node sending the encoded packets. No packets are transmitted by the master during the first generation.

The slaves, on the other hand, have a work flow as shown in Figure 6.3. The slave continuously receives packets from the master until it is polled. The decoder is updated with new packets as they are received. When the slave node is polled, the slave sends

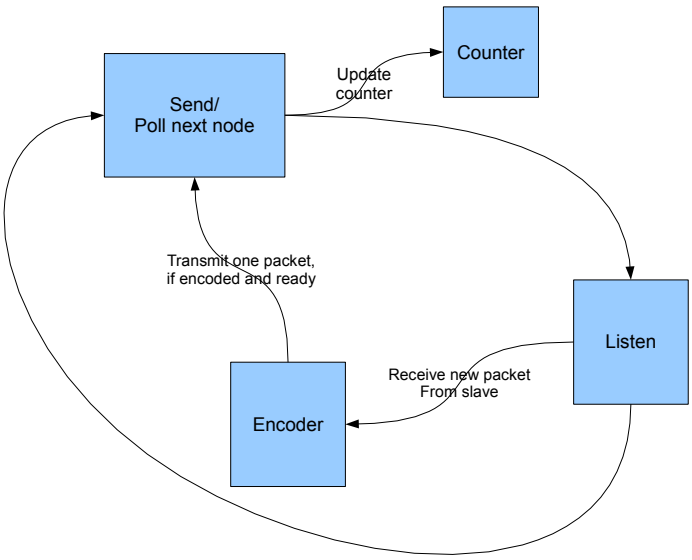


Figure 6.2: The work flow of the master node

one packet from the transmit buffer. At the same time a slave transmits its packet; the slave updates the decoder with its own packet.

The nodes, channel, and routing table are defined in the main part of the program. A reference to the channel and routing is passed to the node when instantiated. During the instantiation, the node creates encoder, decoder, txBuffer and RxBuffer objects. Only the master node creates the encoder object, and only the slave nodes create the decoder objects. Figure 6.4 shows the dependencies of the objects. The dotted line is the reference to the object created by the main method and passed to the node object as a part of the instantiation.

6.1.1 Encoding and decoding

The coding coefficient proposed in section 5.2.1 is used in the simulator. The implementation of the encoding and decoding give the possibility to change the number of nodes without modifying the code with a new encoding or decoding matrix. In fact the encoder and decoder scale automatically to the number of nodes. However, it should be noted that the simulator does not support adding or removing nodes during run time. Once the configuration is done, the network configuration is fixed throughout the simulation. The pseudocode for the encoder:

```
* n = number of transmitting nodes
* p = packet length
* buffer is a n x p array containing the n received packets
```

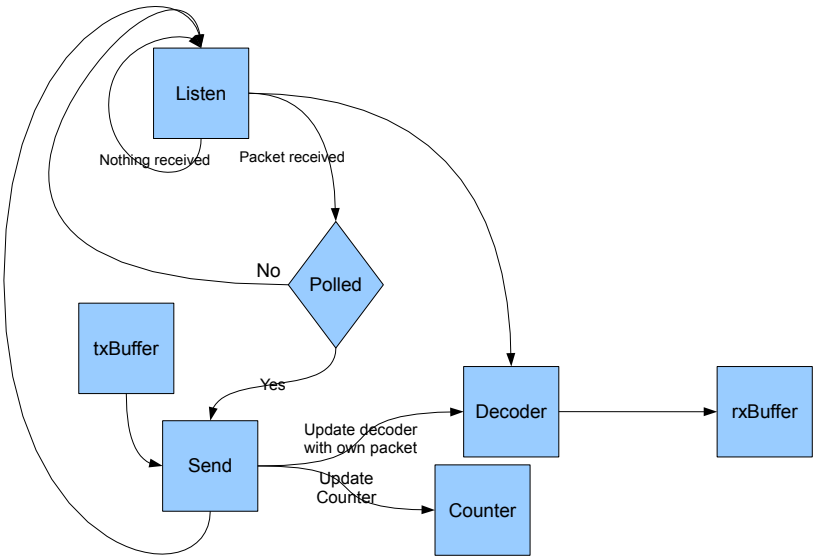


Figure 6.3: The work flow of the slave nodes

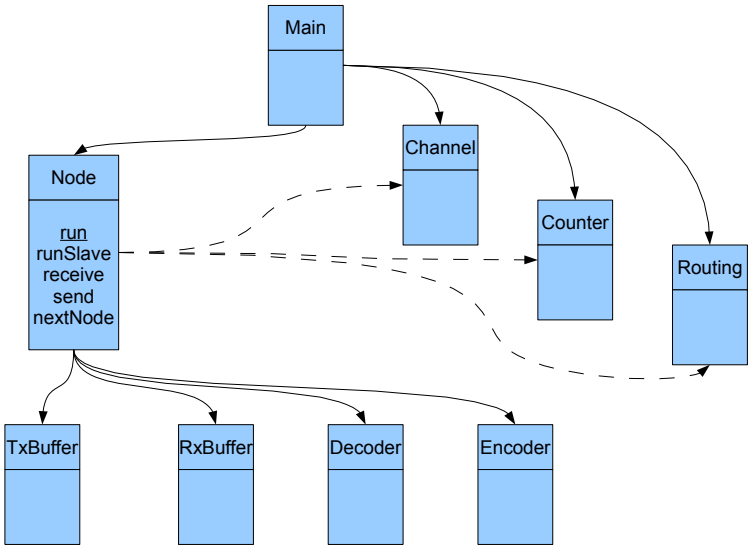


Figure 6.4: Object oriented view of the simulator

```
* (Assumes vector[0] is the first array element, as in java)
*
for(i=0; i < n-1; i++){
    encoded[i]=buffer[i] xor buffer[n-1] //XOR the i'th and the last packet
}
```

As the pseudocode shows, the encoding is very simple and does not require much computational power. Simple XOR of packets is supported by the majority of programming languages including machine code. This is a major advantage of the proposed code; it runs easily on challenged nodes. As the nodes in MANET and WSN may be challenged regarding computational power and battery lifetime.

The pseudocode for the corresponding decoder:

```
* n = number of transmitting nodes
* p = packet length
* id=own node id number
* buffer is a n x p array containing the n received packets
* decoded is a n x p array containing the n decoded packets
* (Assumes vector[0] is the first element, as in java)

copy own packet to position id in decoded array
if(not last node)
  decoded[n]=decoded[id] XOR buffer[id]
for(i=n-1; i>0; i--){
  decoded[i]=decoded[n] XOR buffer[i]
}
```

Also the decoding of the packets does not require much computational power, as shown in the pseudocode. However, this is at cost of error correcting feature provided by random linear coding.

6.1.2 Simulator implementation issues

Designing a simulator raises several issues not seen in real implementation. These have to be solved in a matter such that the simulation is as close up to the real world as possible, while overcoming the problems. The simulator made to prove the work of this report is designed to isolate the problem discussed. There is therefore some design choices made to not deal with real time problems.

The simulator is using threads to create the different nodes. This makes the nodes running independently of each other. Still they are reading and writing to the same channel. Having several objects running as thread and accessing the same channel object is a challenge. Java has a built in synchronization feature, preventing different objects accessing the same time. The synchronization method uses a queue to give access to the object, and let the objects access the same object by turn. However, every node listening to the channel for a new packet is accesing the channel in a loop until a new

packet arrives. To reduce the stress on the synchronization, the node is delayed for a while if there is not a new packet. This improved the run time of the simulator a lot.

The simulator is not running in real time. Instead the time slots are counted to calculate the time. A time slot counter keeps track of the current time slot.

The network coding does not add any error correcting feature, and of such the piconet applying network coding is assumed to have equal behavior regarding lost packets as a Bluetooth piconet not applying network coding. A noiseless channel is therefore assumed, and the simulator does not perform any bit error check nor count the number of bit errors.

To prove possibilities of using network coding in piconet the Bluetooth specific header information such as synchronization and packet type is not of interest to this work. In the simulator the header is not implemented completely. Only header information of interest to this work is available. However, the full header length is used when calculating throughput and power consumption.

The simulator is using *int* to store a byte. In Java an int is 32 bit signed data type. Thus the simulator is consuming more memory than what would be required using a language supporting *byte*.

Time slot counter

During the transmission the given time slot of all activity is printed. The duration of each Bluetooth time slot is 625μS (se Chapter 3). It is therefore possible to calculate the time to run a simulation by multiplying the number of time slots required for all nodes to transmit all data in the buffer by 625μS.

Estimated achievable throughput

Based on the time required to exchange all the data in the buffers, it is possible to estimate the average throughput per node. This is done by summarize the amount of data in the buffers, and divide it by number of time slots used. This number is estimated for the three different packet sizes with and without network coding using the formula

$$Average\ throughput = \frac{n \cdot (tb \cdot 8)}{ts \cdot \frac{1}{1600}}$$

where n is the number of transmitting nodes, tb is the transmit buffer in bytes, and ts is the number of timeslot required.

Each packet transmitted between two slave nodes is transmitted twice; First from the source slave to the master, and then from the master to the sink slave. The *aggregated throughput* of the network is therefore found by multiplying the *average throughput* by 2.

Counting transmitted bits

Every time a node (master and slaves) transmits, it updates the counter with the number of bits transmitted. This is done by multiplying number of bytes by 8, and adding the

overhead (126 bits per transmission). The empty poll and reply packets from the master and slave nodes accordingly, are added as a single 126 bit transmission.

Estimating power consumption

Based on the number of bits transmitted, the power consumption is estimated. The estimation assumes power class 2 Bluetooth devices, with a maximum transmitting power off 2.5mW. The [22] specifies the power level at the antenna connector. The efficiency of the implemented transmitter will affect the true power consumption. This is dependent on the transmitter of the Bluetooth device, and is of such not possible to take into account. The efficiency of the transmitter is equal for both with and without network coding, and is linear. The error of the estimated power is equal for both cases, and is therefore not considered in the estimation.

Given a *basic data rate* (BDR) mode, the gross data rate at the air is 1Mps [22]. The gross data rate includes both header and payload. The power consumption per bit is

$$\frac{2.5 \cdot 10^{-}3W}{1 \cdot 10^{6}bps} = 2.5 \cdot 10^{-}9 = 2.5pW/bit$$

The total number of bits transmitted is multiplied by 2.5pW, and the result is presented by the end of the simulation.

6.2 Results

Three different network scenarios have been simulated, two slave nodes and a master node, four slave nodes and a master node, and six slave nodes and a master node. The networks have been simulated with and without network coding using DH1, DH3, and DH5 mode. All simulations have been run with 10, 100 and 62037 byte transmit buffers at each slave node. The two smaller buffer sizes is to identify effects when there are not much data, less than a frame size and padding occurs. The biggest transmit buffer is a size in which there are no padding necessary for DH1, DH3 or DH5 frames. The size of the larger buffer is modest, and the effect of overhead and empty poll packets in the beginning of the simulations is minimized. Finally, a buffer size of {27, 183, 339} is initiated, when using DH1, DH3, and DH5 accordingly to find the break point where padding does not affect the results. If the size of the transmit buffer results in a bad utilized timeslot (i.e. not fully loaded), the calculated throughput will be reduced. To avoid of this effect, the size of the transmit buffer is carefully chosen. Finally, selected simulation using DM packets have been performed to compare throughput between the two modes of operations (DM and DH). The findings are represented in this section. Appendix A list all relevant results collected during simulation.

6.2.1 Achievable

The proposed protocol for deploying network coding in a piconet is achievable. It is only proven in a simulator, and should be tested in a real world testing as well. However,

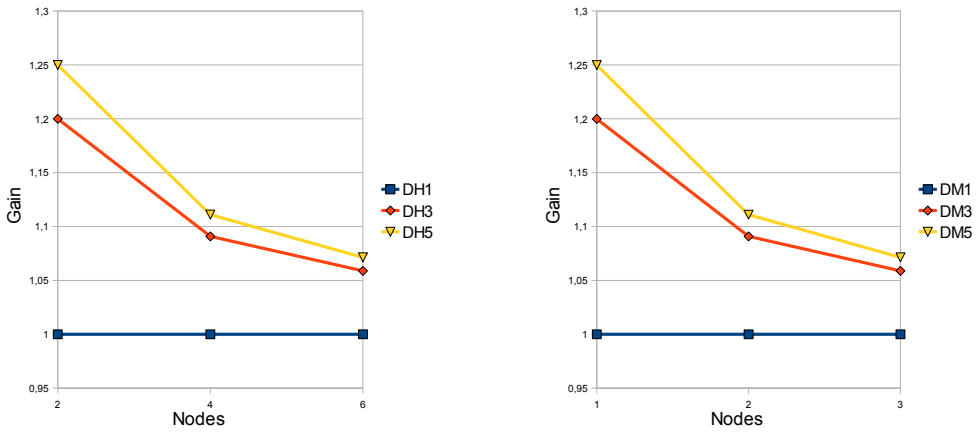
this work proves possibility of adopting the proposed protocol in a Bluetooth network. Furthermore, the implementation in a simulator also verifies the possibility of using non-complex code to apply deterministic network coding. This major finding is particular of interest to challenged nodes.

6.2.2 Throughput

For each simulation scenarios, the average throughput is calculated using the formula presented previously in this chapter. The throughput gain is calculated as

$$\text{gain} = \frac{\text{throughput with network coding}}{\text{throughput without network coding}}$$

The average throughputs numbers used to calculate the gain are from the simulations with the largest transmit buffer (62.037 bytes). The result is presented in Fgure 6.5. As shown in the figure, the best result is achieved in a small network (few transmitting nodes) with large packets (DH5 or DM5 packets). However, there are about 6 and 7% gain using DH3/DM3 and DH5/DM5 packets accordingly, in a network of six transmitting nodes. This is according to the theory presented in Chapter 5
Using DH1/DM1 packets will not affect the throughput in a Bluetooth network at all.



(a) (b)

Figure 6.5: Throughput gain using network coding - master node

The throughput is only achieved on the link from the master node to the slaves. Network coding does not affect the throughput on the link from the slave nodes to the master.

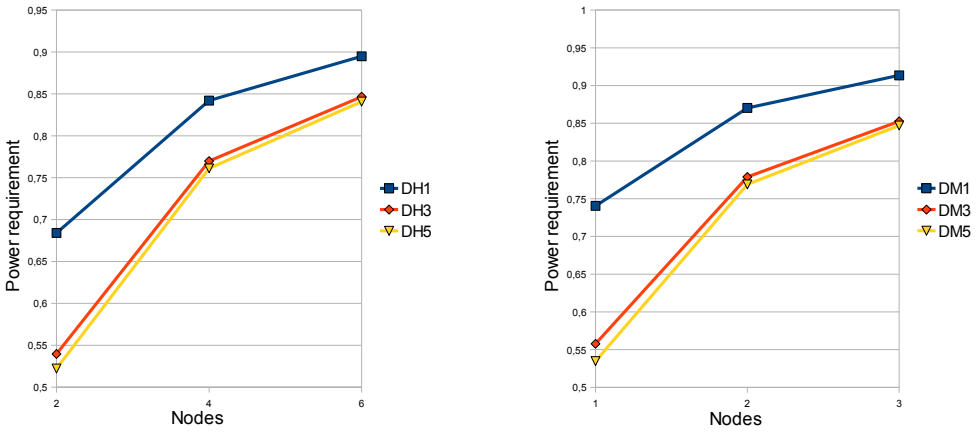
6.2.3 Power consumption

The power consumption is calculated, using the method described earlier in this chapter, for all simulation scenarios. The power requirement reduction is calculated as

$$\text{power requirement reduction} = \frac{\text{power requirement with network coding}}{\text{power requirement without network coding}}$$

There are no differences in power consumption between slave nodes applying network coding and those who do not apply network coding. The reduction in power consumption is only considered for master nodes in the following.

Figure 6.6 shows the master node's power requirement reduction using network coding. As shown in the figure, the most reduction is achieved in a small network (two transmitting nodes) and large packet size (DH5 or DM5 packets). As opposed to throughput benefits, there is a reduction in power requirement using network coding also for small packets (DH1/DM1). Even in a network with six nodes using DH1/DM1 packet, the reduction is 0.89 for the master node. I.e. the power requirement is 89% of the requirement without using network coding. Or the power requirement is 12% higher when network coding is not applied. However, *when the packet transmitted is smaller than the frame size, there is an increase in power requirement using network coding.* Increased power consumption for packet smaller than the frame size is a consequence of the padding of the data.



(a) (b)

Figure 6.6: Power requirement reduction using network coding

6.2.4 DM versus DH packets

The DM packets offer a protection against noise by applying FEC. This is done at the cost of smaller packets, and therefore reduced bandwith in the network. Table 6.1 shows the average throughput per node in the different configurations for a piconet with 6 slave nodes communicating bi-directional and symmetrical. As Figure 6.5 shows, the gain in throughput is equal for DM and DH packets using network coding.

	DH1	DM1	DH3	DM3	DH5	DM5
DNCBP	28786	18128	68840	45519	77478	50984
Non-DNCBP	28786	18128	65019	42992	72313	47588

Table 6.1: Average throughput per node in a piconet with 6 slave nodes

Chapter 7

Conclusions

This thesis is discussing a possible application of network coding in Bluetooth network. Due to the *time division duplex* (TDD) used in Bluetooth network there are several constrains to take into account, compared to other wireless network protocols like wireless LAN (WLAN). One of them is that the slave node is not able to communicate directly with other slave nodes, but rather is required to communicate through a master node.

A protocol for applying network coding in Bluetooth piconet has been proposed. The proposed protocol is using deterministic network coding to combine packets from the slaves at the master node. The master node forwards the combined packets to all slaves. It is required to transmit $n-1$ packets, where n is the number of source nodes. An encoding equation using a MDS code in the binary field is proposed. Encoding in the binary field makes the operation computationally easy. Furthermore, the proposed equation scales easily to the maximum number of nodes in a piconet. The encoding and decoding is performed using simple bitwise XOR of the packets, without need to store neither encoding matrices, nor look-up tables. This is an advantage particular for nodes with limited computational power and memory.

The proposed protocol does not give any benefits to the slave nodes, and the communication link between the slave nodes and the master node. It does only affect the power consumption of the master node, and the communication link from the master node to the slave nodes. Furthermore, the proposed protocol does not give any security benefits in respect to a Bluetooth network not applying network coding. As in Bluetooth network without network coding, countermeasures to ensure confidentiality should be applied if confidentiality is an issue. The Bluetooth network applying the proposed network coding protocol does not benefit from any noise protection of the data. The proposed network coding does not add any redundancy to the data, and a packet loss due e.g. interference causes the receiver to not be able to decode the original message.

A theoretical study proves a gain in throughput in a Bluetooth network (using DH3) equal to $G = \frac{3n}{3n-1}$, where n is the number of source nodes in the piconet. As a master in

a Bluetooth piconet applying network coding does transmits for a short time, compared to a Bluetooth network without network coding. The power consumption of the master node is reduced. Because the slave does not benefit from using network coding, the theoretical study emphasis the behavior and benefits for the master node.

A simulator has been created to study the behavior of a Bluetooth piconet, both when using network coding and in "'normal"' operation. The simulation has been performed using different modes of operation (DH1, DM1, DH3, DM3, DH5, and DM5), different number of source nodes, and with different sizes of the transmit buffer.

The simulation has proven the possibility to improve performance of a Bluetooth piconet using network coding. There is an improvement in the throughput and power consumption of the master node. The slave nodes will transmit the same amount of data, and require the same amount of power. However, packing the data on the air increases the available bandwidth. This increase is shared between the nodes (master and slaves) in the piconet. As such, the slave nodes will benefit from the effects on the master nodes using network coding.

7.1 Future work

The proposed protocol has not been tested in a real implementation. There are some challenges regarding how a slave node reads all packets from the master node. Depending on the Bluetooth controller, this could be approached differently. However, an alternative approach using *active slave broadcast* should be investigated. The framework of the simulator should be extended to study the impact of a noisy channel. Is it a possibility to utilize the throughput gain to improve the networks resilience against packet losses? E.g. using DH5 packets and network coding which is robust against packet losses, as opposed to DM5. This could potentially improve the throughput more than the proposed DNCBP protocol.

A possible use of network coding on multicast traffic in a scatternet is also proposed (but not in detail) in this thesis. A theoretical study on how this can be done, and a simulation to verify the protocol is of interest. As opposed to use deterministic network coding, it is most likely necessary to use random network coding because of the possibility to maintain knowledge of the entire network could be difficult, unless it is possible to find a distributed algorithm, which assigns the encoding coefficient e.g. at the piconet level. A possible implementation in a network is to use the FRANC [15] implementation on devices with a built in Bluetooth device e.g. a Lego mindstorm robot. A scatternet formation algorithm is required. However, due to limitation in the Bluetooth protocol and the implementation of Bluetooth in the Lego mindstorm robot, the OBP [34] is an interesting protocol to use in this application.

Bibliography

[1] C. Adams, P. Sylvester, M. Zolotarev, and R. Zuccherato. Internet x.509 public key infrastructure, data validation and certification server protocols. http://www.ietf.org/rfc/rfc3029.txt?number=3029, 2001.

[2] R. Ahlswede, N. Cai, S-Y.r. Li, and R.W. Yeung. Network information flow. *IEEE Transaction on Information Theory, vol. 46, no. 4*, 2000.

[3] Omar Al-Jarrah and Omar Megdadi. Enhanced aodv routing protocol for bluetooth scatternet. *Comput. Electr. Eng.*, 35(1):197–208, 2009.

[4] Mohammed Shafkat Amin and Fahima Amin Bhuyan. An empirical study of bluetooth scatternet formation protocol. *Proceedings of the 2006 IEEE International Conference on Mechatronics and Automation*, 2006.

[5] Ángela.I Barbero and Øvind Ytrehus. Cycle-logical treatment for "cyclopathic" networks. *IEEE Transaction on Information Theory*, 52:2795–2804, 2006.

[6] Boukerche Azzedine. *Handbook of Algorithms for Wireless Networking and Mobile Computing*. Chapman and Hall, 2006.

[7] Ángela.I Barbero and Øvind Ytrehus. Introduction to network coding for acyclic and cyclic networks. *World Scientific Review*, 2008.

[8] A. Barbir, S. Murphy, and Y. Yang. Generic threats to routing protocols. http://www.ietf.org/rfc/rfc4593.txt, 2006.

[9] Stefano Basagni and Chiara Petrioli. A scatternet formation protocol for ad hoc networks of bluetooth devices. *IEEE Vehicular Technology Conference*, 2002.

[10] Bela Bollobas. *Modern Graph Theory*. Springer, 1979.

[11] Chandra Chekuri, Christina Fragouli, and Emina Soljanin. On average througput and alphabet size in network coding. *IEEE Transaction on Information Theory*, 2005.

[12] Innovative Concepts. Uidm. http://www.innocon.com/uidm.asp, 2009.

[13] Aaron Drew. Slimsim - the wireless network coding network simulator. http://cs.anu.edu.au/ aaron/sim.php, 2005.

[14] Norwegian Defence Research Establishment. Research and projects. http://www.mil.no/felles/ffi/start/FFI-prosjekter/, 2008.

[15] Alaeddine El Fawal, Kave Salamatian, David Cavin, Yoav Sasson, and Jean-Yves Le Boudec. A framework for network coding in challanged wireless network. http://infoscience.epfl.ch/record/88682?ln=en, 2006.

[16] L. R Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics 8*, page 399404, 1956.

[17] Christina Fragouli, Jean-Yves Le Boudec, and Jörg Widmer. Network coding: an instant primer. *SIGCOMM Comput. Commun. Rev.*, 36(1):63–68, 2006.

[18] Christina Fragouli and Emina Soljanin. Information flow decomposition for network coding. *IEEE Transactions on Information Theory*, pages 829–848, 2006.

[19] Christina Fragouli and Emina Soljanin. Network coding fundamentals. *Found. Trends Netw.*, 2(1):1–133, 2007.

[20] J.J. Garcia-Luna-Aceves and Ewerton L. Madruga. A multicast routing protocol for ad-hoc networks. pages 784–792, 1999.

[21] Christian Gehrmann. Bluetooth security white paper. 2002.

[22] Bluetooth Special Interest Group. Specification of the bluetooth system 2.0 + edr. URLhttp://www.bluetooth.com, 2004.

[23] Bluetooth Special Interest Group. The official bluetooth technology info site. http://www.bluetooth.com, 2008.

[24] The Shmoo Group. The shcmoocon 2009 conference. http://www.shmoocon.org/2009/presentations.html, 2009.

[25] Zygmunt J. Haas, Marc R. Pearlman, and Prince Samar. The zone routing protocol (zrp) for ad hoc networks. http://tools.ietf.org/html/draft-ietf-manet-zone-zrp-04.txt, 2002.

[26] T. Ho and D.S. Lund. *Network Coding - An Introduction*. Cambridge University Press, 2008.

[27] Tracey Ho, Ralf Koetter, Muriel Mdard, David R. Karger, and Michelle Effros. The benefits of coding over routing in a randomized setting. In *In Proceedings of 2003 IEEE International Symposium on Information Theory*, 2003.

[28] Tracey Ho, Muriel Mdard, Ralf Koetter, David R. Karger, Michelle Effros, Jun Shi, and Ben Leong. Toward a random operation of networks. *IEEE Transactions on Information Theory*, 2004:1–8, 2004.

[29] Luc Hogie and Pascal Bouvry. An overview of manets simulation. *Electronic Notes in Theoretical Computer Science 150*, 2006.

[30] P. Hsiu and T. Kuo. A maximum-residual multicast protocol for large-scale mobile ad hoc networks. *Mobile Computing, IEEE Transactions on*, 2009.

[31] Teerawat Issariyakul and Ekram Hossain. *Introduction to Network Simulator NS2*. Springer, 2009.

[32] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, 1996.

[33] Sewook Jung, Alexander Chang, and Mario Gerla. Performance comparison of overlaid bluetooth piconets (obp) and bluetooth scatternet. In *Wireless Communications and Networking Conference*, pages 505–510, 2006.

[34] Sewook Jung, Alexander Chang, and Mario Gerla. New bluetooth interconnection methods: Overlay bluetooth piconets (obp) and temporary scatternets (ts). *Computer communications*, 30(10), 2007.

[35] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft. Xors in the air: practical wireless network coding. *IEEE/ACM Trans. Netw.*, 16(3):497–510, 2008.

[36] Torleiv Klove. *Codes for Error Detection*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2007.

[37] Ralf Koetter. The network coding home page. http://www.ifp.illinois.edu/k̃oetter/NWC/index.html, 2009.

[38] C. Law, A.K. Mehta, and K.-Y. Siu. A new bluetooth scatternet formation protocol. *Kluwer Academic Publishers journal on Mobile Networks and Applications*, 2003.

[39] Seung-Hoon Lee, Sewook Jung, Alexander Chang, Dae-Ki Cho, and Mario Gerla. Bluetooth 2.1 based emergency data delivery system in healthnet. In *IEEE Wireless Communication and Networking Conference*, 2008.

[40] Jin Jin Baochun Li and Teagon Kong. Is random network coding helpful in wimax. *INFOCOM 2008*, 2008.

[41] Sanif Sentosa Liong and Payam M. Barnaghi. Bluetooth network security: A new approach to secure scatternet formation. *IEEE TENCON 2005*, 2005.

[42] Max Moser. Busting the bluetooth myth getting raw access. http://www.remote-exploit.org/research/busting_bluetooth_myth.pdf.

[43] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 151–162, New York, NY, USA, 1999. ACM.

[44] University of Cincinatti. Ucbt - bluetooth extension for ns2 at the university of cincinnati. http://www.cs.uc.edu/ cdmc/ucbt/, 2008.

[45] University of Southern California Information Sciences Institute. The network simulator - ns-2. http://www.isi.edu/nsnam/ns/, 2008.

[46] Munich University of Technology Institute of Communications Engineering. Network coding bibliography. https://hermes.lnt.e-technik.tu-muenchen.de/DokuWiki/doku.php?id=network_coding: bibliography_for_network_coding, 2009.

[47] J.-S. Park, D.S. Lum, F. Soldo, M. Gerla, and M. Medard. Performance of network coding in ad hoc networks. *Military Communications Conference*, pages 1–6, 2006.

[48] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. pages 1405–1413, 1997.

[49] Vincent D. Park and M. Scott Corson. Temporally-ordered routing algorithm (tora) version 1. http://tools.ietf.org/html/draft-ietf-manet-tora-spec-04, 2001.

[50] Guangyu Pei, Mario Gerla, Xiaoyan Hong, and Ching chuan Chiang. Wireless hierarchical routing protocol with group mobility (whirl). In *In Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC*, pages 1538–1542, 1999.

[51] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc on-demand distance vector routing. In *IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1999.

[52] C. Petrioli, S. Basagni, and I. Chlamtac. Configuring bluestars: Multihop scatternet formation for bluetooth networks. *IEEE Transaction on computers, special issue on Wireless Internet*, 2002.

[53] Microsoft Research. Avalanche: File swarming with network coding.

[54] Revision3. Hak5, episode 426, shmoocon 2009. http://revision3.com/hak5/Shmoocon, 2009.

[55] Dennis Roddy. *Satellite Communications.* McGraw-Hill, 2006.

[56] Peter Sanders, Sebastian Egner, and Ludo Tolhuizen. Polynomial time algorithms for network information flow. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 286–294. ACM, 2003.

[57] Jochen Schiller. *Mobile Communications.* Pearson Education Limited, 2003.

[58] Claude E. Shannon. Two-way communication channels. *Proceedings Fourth Berkeley Symposium on Math. Statist. and Prob*, pages 611–644, 1961.

[59] Vlad Skarzhevskyy and Michael Lifshits. Bluecove jsr-82 project. http://www.bluecove.org/, 2009.

[60] William Stallings. *Wireless Communications and Networks.* Prentice-Hall, 2002.

[61] Mark Stamp. *Information Security : Principles and Practice.* John Wiley & sons, 2006.

[62] Jay Kumar Sundararajan, Devavrat Shah, Muriel Medard, Michael Mitzenmacher, and Joo Barros. Network coding meets tcp. *CoRR*, abs/0809.5022, 2008. informal publication.

[63] Godfrey Tan and Allen Miu. An efficient scatternet formation algorithm for dynamic. In *IASTED Communications and Computer Networks (CCN)*, 2002.

[64] Shaoguo Tao, Wenbo Qiao, Zongkai Yang, and Wenqing Cheng. Routing algorithm of network coding on multicast. In *CISW '07: Proceedings of the 2007 International Conference on Computational Intelligence and Security Workshops*, pages 354–357. IEEE Computer Society, 2007.

[65] Lars Erling Bråten et al. Subnet relay; a mobile wireless ad-hoc network. *FFI-report 2007/00901 ISBN:978-82-464-1130-9*, 2007.

[66] Timothy J. Thompson, Paul J. Kline, and C Bala Kumar. *Bluetooth Application Programming with the JAVA APIs.* Morgan Kaufman, 2008.

[67] International Telecommunication Unit. Recommandation x.509 - information technology - open systems interconnection - the directory public-key and attribute certificate frameworks. http://www.itu.int/itudoc/itu-t/aap/sg17aap/history/x509/x509.html, 2005.

[68] L. R. Vermani. *Elements of algebraic coding theory.* CRC Press, 1996.

[69] Zhifang Wang, R.J. Thomas, and Z. Haas. Bluenet - a new scatternet formation scheme. 2002.

[70] Brad Williams and Tracy Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 194–205, New York, NY, USA, 2002. ACM.

[71] Min Yang and Yuanyuan Yang. Peer-to-peer file sharing based on network coding. In *ICDCS '08: Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems*, pages 168–175. IEEE Computer Society, 2008.

[72] Raymond W. Yeung. Avalanche: A network coding analysis. *Commun. Inf. Syst*, 7(4):353–358, 2007.

[73] R.W. Yeung, S-Y.R. Li, N. Cai, and Z. Zhang. *Network Coding Theory.* now Publishers, 2006.

[74] Gerely V. Zaruba, Stefano Basagni, and Imrich Chlamtac. Bluetrees - scatternet formation to enable bluetooth-based ad hoc networks. *IEEE International Conference on Communications (ICC2001)*, 2001.

[75] Hui Zeng, Jiaqing Huang, Shaoguo Tao, and Wenqing Cheng. A simulation study on network coding parameters in p2p content distribution system. *Communications and Networking in China*, pages 197–201, 2008.

Appendix A

Results

This appendix list simulation results.

Transmit buffer	Transmitted bits		Power consumption		Number of timeslots	Average Throughput
	Master	Slave	Master	Slave		
10	538	332	1345	830	5	25600
27	810	468	2025	1170	5	69120
100	2734	1430	6835	3575	17	75294
62037	1571814	785970	3929535	1964925	9193	86378

Table A.1: Results DH1 and 2 transmitting slaves without network coding

Transmit buffer	Transmitted bits		Power consumption		Number of timeslots	Average Throughput
	Master	Slave	Master	Slave		
10	594	332	1485	830	5	25600
27	594	468	1485	1170	5	69120
100	1998	1430	4995	3575	17	75294
62037	1075158	785970	2687895	1964925	9193	86378

Table A.2: Results DH1 and 2 transmitting slaves with network coding

Transmit buffer	Throughput gain	Power reduction
10	1	1,1
27	1	0,73
100	1	0,73
62037	1	0,68

Table A.3: Gain using network coding, DH1 and 2 transmitting slaves

Transmit buffer	Transmitted bits		Power consumption		Number of timeslots	Average Throughput
	Master	Slave	Master	Slave		
10	664	332	1660	830	13	9846
100	1978	1052	4945	2630	13	98462
183	3306	1716	8265	4290	13	180185
62037	1078146	539136	2695365	1347840	4069	195152

Table A.4: Results DH3 and 2 transmitting slaves without network coding

Transmit buffer	Transmitted bits		Power consumption		Number of timeslots	Average Throughput
	Master	Slave	Master	Slave		
10	1842	332	4605	830	11	11636
100	1842	1052	4605	2630	11	116364
183	1842	1716	4605	4290	11	212945
62037	581850	539136	1454625	1347840	3391	234171

Table A.5: Results DH3 and 2 transmitting slaves with network coding

Transmit buffer	Throughput gain	Power reduction
10	1,18	2,77
100	1,18	0,93
183	1,18	0,56
62037	1,2	0,54

Table A.6: Gain using network coding, DH3 and 2 transmitting slaves

Transmit buffer	Transmitted bits		Power consumption		Number of timeslots	Average Throughput
	Master	Slave	Master	Slave		
10	1 978	458	4 945	1 145	21	6095
100	1 978	1 052	4 945	2 630	21	60952
339	5 802	2 964	14 505	7 410	21	206629
62 037	1 038 834	519 480	2 597 085	1 298 700	3 661	216901

Table A.7: Results DH5 and 2 transmitting slaves without network coding

Transmit buffer	Transmitted bits		Power consumption		Number of timeslots	Average Throughput
	Master	Slave	Master	Slave		
10	3 090	332	7 725	830	17	7529
100	3 090	1 052	7 725	2 630	17	75294
339	3 090	2 964	7 725	7 410	17	255247
62 037	542 538	519 480	1 356 345	1 298 700	2 929	271107

Table A.8: Results DH5 and 2 transmitting slaves with network coding

Transmit buffer	Throughput gain	Power reduction
10	1,24	1,56
100	1,24	1,56
339	1,24	0,53
62 037	1,25	0,52

Table A.9: Gain using network coding, DH5 and 2 transmitting slaves

Transmit buffer	Transmitted bits		Power consumption		Number of timeslots	Average Throughput
	Master	Slave	Master	Slave		
10	1 202	332	3 005	830	45	2844
100	4 082	1 052	10 205	2 630	45	28444
339	11 730	2 964	29 325	7 410	45	96427
62 037	2 077 794	519 480	5 194 485	1 298 700	7 325	108406

Table A.10: Results DH5 and 4 transmitting slaves without network coding

Transmit buffer	Transmitted bits		Power consumption		Number of timeslots	Average Throughput
	Master	Slave	Master	Slave		
10	9 018	332	22 545	830	41	3122
100	9 018	1 052	2 2545	2 630	41	31220
339	9 018	2 964	2 2545	7 410	41	105834
62 037	1 581 498	519 480	3 953 745	1 298 700	6 593	120442

Table A.11: Results DH5 and 4 transmitting slaves with network coding

Transmit buffer	Throughput gain	Power reduction
10	1,24	1,56
100	1,24	1,56
339	1,24	0,53
62 037	1,25	0,52

Table A.12: Gain using network coding, DH5 and 4 transmitting slaves

Transmit buffer	Transmitted bits		Power consumption		Number of timeslots	Average Throughput
	Master	Slave	Master	Slave		
10	1 866	332	4 665	830	69	1855
100	6 186	1 052	15 465	2 630	69	18551
339	17 658	2 964	44 145	7 410	69	62887
62 037	3 116 754	519 480	7 791 885	1 298 700	10 989	72261
558 333	28 045 746	4 674 312	70 114 365	11 685 780	98 829	72313
1 000 000	50 230 830	8 371 826	125 577 075	20 929 565	177 009	72313

Table A.13: Results DH5 and 6 transmitting slaves without network coding

Transmit buffer	Transmitted bits		Power consumption		Number of timeslots	Average Throughput
	Master	Slave	Master	Slave		
10	14 946	332	37 365	830	65	1969
100	14 946	1 052	37 365	2 630	65	19692
339	14 946	2 964	37 365	7 410	65	66757
62 037	2 620 458	519 480	6 551 145	1 298 700	10 257	77418
558 333	23 579 082	4 674 312	58 947 705	11 685 780	92 241	77478
1 000 000	42 232 830	8 371 826	105 582 075	20 929 565	165 209	77478

Table A.14: Results DH5 and 6 transmitting slaves with network coding

Transmit buffer	Throughput gain	Power reduction
10	1,06	8,01
100	1,06	2,42
339	1,06	0,85
62 037	1,07	0,84
558 333	1,07	0,84
1 000 000	1,07	0,84

Table A.15: Gain using network coding, DH5 and 6 transmitting slaves

Appendix B

main.java

```
package ncbtsimulator;

/**
 * This package is used to simulate a Bluetooth piconet.
 * It simulate a reference piconet with a master node and up to
     seven slave
 * nodes.
 * The slave nodes communicate bi-directional pairwise. Tha
    master node only
 * route traffic between the slave nodes.
 *
 * @author Roger Stenvoll
 */
public class Main {

    public boolean NC=false;

    /**
     * The main method is used to set up the simulation
     * Default values, n=2, network coding = true, DH5,
        buffSize=
     * @param args n {true, false} {DH1, DH3, DH5, DM1, DM3.
        DM5} buffSize
     *
     */
    public static void main(String[] args) {
        int n=3;
        boolean nc=false;
        int packetSize=339;
        int buffSize=62389;
```

```
if(args.length==0){
    System.out.println("Usage: ncbtsimulator n nc
       packetType bufferSize");
    System.out.println("Example: ncbtsiumaltor 3 true
       DH3 1000");
    System.out.println("Default values:");
    System.out.println("n=2");
    System.out.println("nc=true");
    System.out.println("packetType=DH5");
    System.out.println("bufferSize=62389");
}

if(args.length>=1){
    n=Integer.parseInt(args[0]);
    System.out.print("\tNumber of nodes " + n);
}
if(args.length>=2){
    if(args[1].compareToIgnoreCase("true")==0){
        nc=true;
    }
    else{
        nc=false;
    }
    System.out.print("\tNetwork coding " + nc);
}
if(args.length >= 3){
    if(args[2].compareToIgnoreCase("DH1")==0){
        packetSize=27;
    }
    else if(args[2].compareToIgnoreCase("DH3")==0){
        packetSize=183;
    }
    else if(args[2].compareToIgnoreCase("DH5")==0){
        packetSize=339;
    }
    else if(args[2].compareToIgnoreCase("DM1")==0){
        packetSize=17;
    }
    else if(args[2].compareToIgnoreCase("DM3")==0){
        packetSize=121;
    }
    else if(args[2].compareToIgnoreCase("DM5")==0){
```

```
            packetSize=224;
        }
        else{
            packetSize=339;
        }
        System.out.print("\tPacket Size " + packetSize);
    }
    if(args.length>=4){
        buffSize=Integer.parseInt(args[3]);
        System.out.print("\tBuffer Size " + buffSize);
    }



    Thread [] nodes = new Thread[n];


    // Create channel
    channel ch = new channel();
    counter count = new counter(n);

    //Create the router object
    routing route = new routing(n);


    //Establish nodes as threads
    for(int i=0; i<n; i++){
        nodes[i]= new Thread(new node(i, n, nc, packetSize,
            buffSize, ch,route, count));
    }

    //Setting up the routing
    //When using network coding the routing describe which
       slave nodes communicat in pair.
    for(int i=1; i<n; i++){
        route.addRoute(i, n-i);
    }

    //Initiate simulations
    for(int i=n-1; i>-1; i--){
        nodes[i].start();
    }
```

```
        }
}
```

Appendix C

node.java

```
package ncbtsimulator;

/**
 * Public class node
 * A simulation of a Bluetooth node. Depending upon the
    construction, the node
 * become slave or master node.
 *
 * @author Roger Stenvoll
 */
public class node implements Runnable{
    boolean run=true;
    boolean stop=false;
    boolean MASTER;
    boolean NC;
    boolean DEBUG=false;    //if true, give more debugging
       information
    int MAXTS = 200000;
    int tsRec=0;
    int id;
    int frameLength; //Change the size of the payload
    int ts; //Timeslot #
    int slaves; //Number of slave nodes in the network
    int lastNode;
    int buffLength;
    int[][] rxBuf;
    int[] msg;
    int currentGen;
    int genPolled;
```

```
int[] doneMx;
txBuffer tx;
rxBuffer rx;
channel ch;
routing route;
encoder enc;
decoder dec;
counter cnt;

/**
 * Constructor for a node. The master nodeID have to be 0.
 * The routing rt paramater is a routing object, containing
    the routing
 * information. I.e. how the master node should route
    traffic between the
 * slaves. It is also used by the slaves to determin which
    pair of slaves
 * communicat.
 * @param nodeId The node id
 * @param nodes Number of nodes in the network
 * @param nc Running network coding if true
 * @param packetSize The size of each packet
 * @param buffSize The lenght of the transmitt buffer
 * @param ch the channel object to read/write data
 * @param rt the router object to get routing information
 * @param count the counter object to store the bits
    transmitted
 */
    public node(int nodeId, int nodes, boolean nc, int
       packetSize,
            int buffSize, channel ch, routing rt, counter
               count){
    this.id=nodeId;
    this.slaves=nodes-1;
    this.NC=nc;
    this.frameLength=packetSize;
    this.buffLength=buffSize;
    this.ch=ch;
    this.ts=0;
    this.route=rt;
    this.cnt=count;
    rx= new rxBuffer(this.buffLength);
    doneMx = new int[nodes];
```

```
        for(int i=0; i < doneMx.length; i++){
            doneMx[i]=1;
        }
        if(nodeId==0){
            this.MASTER=true;
            rxBuf = new int[slaves+1][this.frameLength];
            enc = new encoder(this.slaves,this.frameLength);
        }
        else{
            this.MASTER=false;
            tx= new txBuffer(this.buffLength);
            tx.initiateBuffer(this.id);
            dec = new decoder(this.id, this.slaves, this.
                frameLength);
        }
    }


    /**
     * This method is used to write data to the channel.
     * Before calling this method the msg array is assumed to
        be initialized
     * with proper header and payload.
     */
    private void send(){


        ch.write(msg);
        this.cnt.add(this.id, msg[4]);
        if(this.DEBUG){
            System.out.println("Tx ts "+ msg[0]+ ", Node " +
                msg[1] +" transmit " +
                 msg[4] + " bytes to " + msg[2] + " poll gen " +
                    msg[3]);
        }


    }
    /**
     * A method to receive data from the channels. The receive
        method writes
     * data to the msg[] array.
```

```
 * The method writes data to a two dimensional array if the
     node is the
 * master.
 */
private void receive(){
   /*if(!MASTER)
        System.out.println(this.ts + this.tsRec); //used
           during debug*/
    while((this.ts>=this.tsRec)){
        msg = ch.read(); //Read from the channel
        if(msg[0]==-1){
            this.run=false;
            if(this.DEBUG){
                System.out.println("Node " + this.id + "
                   breaks");
            }
            break;
        }
        try
        {
        Thread.sleep(1);     //Otherwise all threads try to
           access the same
                             //channel continously causing
                                run time problem
        }
        catch (Exception e){
            System.out.print("Exception caught: ");
            System.out.println(e);
        }
        tsRec=msg[0];  //Update received timeslot to the
           current received.
    }

    //The slave nodes copy the received frame into their
       buffer.
    if(((msg[2]==id)||(NC&&msg[1]==0))&&!MASTER){
        int[] tmp=new int[msg[4]];
        for(int i=0; i<tmp.length; i++){
            tmp[i]=msg[i+5];
         }

         this.genPolled=msg[3];
```

```
            if(NC&&(msg[4]>0)){  //running simulator with
                network coding

                 dec.addPacket(tmp, (this.genPolled -1));
            }
            else{                //Simulator whitout network
                coding
            rx.writeNext(tmp);
            if(this.DEBUG){
                System.out.println("Rx ts " + msg[0] +", Node "
                    + id +
                     " receive from node: " + msg[1] + " to
                         node: "+ msg[2]);
            }
            //System.out.println("slave receive node "+ id);
            }
        }
        //But the master node should keep an two-dimensional
            array
        //
        if((msg[2]==0) && MASTER&&(msg[4]>0)){
            //int[] tmp=new int[msg.length - 5]; TODO remove
                line
            if(this.DEBUG){
                System.out.println("Rx ts " + msg[0] +", Node "
                    + id +
                    " receive from node: " + msg[1] + " to node
                        : "+ msg[2]);
            }

            if(!NC){
                for(int i=0; i<msg[4]; i++){
                      rxBuf[(msg[1])][i]=msg[i+5];
                }
                rxBuf[0][msg[1]]=msg[4];
            }
            else{
            int[] pkt=new int[msg[4]];
            System.arraycopy(msg, 5, pkt, 0, pkt.length);
            enc.addPacket(pkt, msg[1], msg[3]);
            }

        }
```

```
    if(msg[2]==0 && MASTER && (msg[4]==0)){
        doneMx[msg[1]]=0;
        if(isAllEmpty(doneMx)){
            this.stop=true;
        }
    }
    ts=tsRec;  //Set the current timeslot counter to the
       received time slot
               //counter. Such that the receiver is ready


}
/**
 *Method to initiate the simulations.
 *
 */
public void run(){
    if(MASTER){
        lastNode=slaves; //Initiate the counter of the last
            polled node
        currentGen=-1;   //Initiate the counter for the
           generation
        while((ts<MAXTS)&& run){
            if(ts!=0){  //Skip initial timeslot
                ts=increaseTs(msg[4]);  //Calculate next ts
                    based on previuos message length
            }
            else{
                ts++;
            }
            msg=new int[5+this.frameLength];
            msg[0]=ts;
            msg[1]=id;
            msg[2]=nextNode();
            if(msg[2]==1){
                currentGen++;
            }
            int src=route.getSrc(msg[2]);
            msg[3]=currentGen;
            msg[4]=this.rxBuf[0].length;
            //int[] payload= new int[msg[4]];
            //payload = tx.getNext(frameLength);
            if(!this.NC){
```

```
                msg[4]=this.rxBuf[0][src];
                for(int i=0; i<msg[4]; i++){
                    msg[i+5]=this.rxBuf[src][i];
                }
                if(msg[4]==0 && this.stop){
                    msg[0]=-1;
                    this.run=false;
                }
            }
            else{
                int[] temp= null;
                try{
                temp = enc.getNext(currentGen-1);
                }
                catch(Exception e){          //First gen
                    this fails!
                    // System.out.println("Exception " + e);
                }
                if(temp==null){
                    msg[4]=0;
                    if(this.stop){
                        msg[0]=-1;
                        this.run=false;
                    }
                }

                else{
                    System.arraycopy(temp, 0, msg, 5, temp.
                        length);
                    msg[4]=temp.length;
                }
            }
            this.rxBuf[0][src]=0; //reset the buffer6
            send();

        try
        {
        Thread.sleep(5);    //Otherwise all threads try to
            access the same
                            //channel continously causing
                                run time problem
        }
        catch (Exception e){
```

```
            System.out.print("Exception caught: ");
            System.out.println(e);
        }

            receive();
        }

        try
        {
        Thread.sleep(500);     //Make sure all threads are
           done

        }
        catch (Exception e){
            System.out.print("Exception caught: ");
            System.out.println(e);
        }

        System.out.println("Number of timeslots: " + this.
           ts);

        for(int i=0; i <= this.slaves; i++){
            System.out.println("The node " + i + " has
               transmitted " +
                    cnt.printBits(i) + " bits. Using " +
                    this.cnt.printPower(i) + " pW");
        }

    }
    else{
        runSlave();
    }

}
/**
 * Method to run the simulation by slave node. The run()
    method calls this
 * if the node is a slave.
 */
private void runSlave(){
   //System.out.println(ts + " " + id); //Used during debug
    while((ts<MAXTS)&& run){
        receive();
```

```
            if(msg[0]>=ts && msg[2]==id){
                int receipent=msg[1]; //This should allways be
                    the master node!
                ts=increaseTs(msg[4]);  //Calculate next ts
                    based on previuos message length
                msg[0]=ts;
                msg[1]=id;
                msg[2]=receipent;
                msg[3]=this.genPolled;

                if(tx.isEmpty()){
                    msg[4]=0;
                }
                else{
                    if(tx.isLeft()<this.frameLength){
                        msg[4]=tx.isLeft();
                    }
                    else{
                        msg[4]=this.frameLength;
                    }
                    int[] payload= tx.getNext(msg[4]);
                    for(int i=0; i<msg[4]; i++){
                        msg[i+5]=payload[i];
                    }
                    if(this.NC){
                        dec.addOwnPacket(payload, this.
                            genPolled);
                    }
                }
                send();
            }

        }
        if(this.NC&&this.DEBUG){
            dec.printReceived(this.id); //prints out the
                received message
        }
    }
    /**
    * The master node call this method to get the id of the
        next node to be
    * polled.
```

```
     * @return next node to be polled
     */
    private int nextNode(){
        int next=lastNode%slaves+1;
        lastNode=next;
        return next;
    }
    /**
     * Increse the number of timeslots used based on the
        payload size.
     * Take into account both DH and DM packets.
     * @param prevMsg Messages size of the previous received
        message
     * @return The next timeslot to be used
     */
    private int increaseTs(int prevMsg){
        int tmp;
        if(this.frameLength > 200){
            tmp=5;
        }
        else if(this.frameLength > 50){
            tmp =3;
        }
        else{
            tmp=1;
        }
       if(prevMsg > 0){
            return this.ts +tmp;
        }
        else{
           return (this.ts+1);
        }

    }

    private boolean isAllEmpty(int[] mx){
        int tmp=0;
        for(int i=1; i<mx.length; i++){
            tmp = tmp + mx[i];
        }
        if(tmp == 0){
            return true;
        }
```

```
            else{
                return false;
            }
        }


}
```

Appendix D

encoder.java

```
package ncbtsimulator;

/**
 * This class is to network encode packets in the Bluetooth
    simulator.
 * @author Roger Stenvoll
 */
public class encoder {
    int MEM=4; //Number of generetaions the encoder will keep
       in a memory
    int nodes; //Nubmber of nodes in the network
    int packetLength;
    int[][][][] buffer; //Keeps the packets to be encoded.
    int[] bufferStatus; //Keeps track on number of packet in
       the buffer
    int[][][][] encoded;   //Resulting buffer with MEM number of
       generations
    int[] next =new int[MEM]; //Keep track of next packet to
       be sent.


    /**
     * Constructor for the encoder object.
     * Based upon the parameters given, the constructor will
        create a buffer.
     * The buffer will keep a memory of 3 generations. One
        generation is defined
     * by one packet from each slave in the same superfram TDD
        cycle.
```

```
     *
     * @param nrNodes Number of nodes to be encoded. If the
        master only relay
     * traffic from the slaves, number of active slaves should
        be used
     * @param pl The length of the packets to be encoded.
     */
    public encoder(int nrNodes, int pl){
        this.nodes=nrNodes;
        this.packetLength = pl;
        this.buffer= new int[MEM][nodes][this.packetLength];
        this.bufferStatus = new int[MEM];
        for(int i=0; i<this.bufferStatus.length; i++){
                this.bufferStatus[i]=0;
        }
        encoded = new int [MEM][this.nodes-1][this.packetLength
           ];

    }

    /**
     * Method used by master to add received packet from the
        slaves
     * to the encoder.
     * @param packet The packet to be added
     * @param fromNode The node packet was received from
     * @param generation Generation received.
     * the packet should be added to.
     */
    public void addPacket(int[] packet, int fromNode, int
       generation){
        int gen=generation%MEM;
        if(packet.length < this.packetLength){
            int[] tmp = new int[this.packetLength];
            System.arraycopy(packet, 0, tmp, 0, packet.length);
            for(int i=packet.length-1; i < this.packetLength; i
               ++){
                tmp[i]=0;                           //add 0's add
                   the end to fill up
            }
            this.buffer[gen][fromNode-1]=tmp;
        }
        else{
```

```
            this.buffer[gen][fromNode-1]=packet;
        }
        this.bufferStatus[gen]=this.bufferStatus[gen]+1;
        if(this.isFull(gen)){
            this.encode(gen);
        }
    }

    /**
     * Method used by encoder to determin if all packet of a
        given generation
     * is received.
     * @param gen The generation
     * @return Return true or false
     */
    private boolean isFull(int gen){
        if(this.bufferStatus[gen]<this.nodes){
            return false;
        }
        else{
            return true;
        }
    }
    /**
     * Method to perform the network encoding.
     * @param gen The generation which the encoder should
        encode
     */
    private void encode(int gen){
        for(int i=0; i < (this.nodes-1); i++){
            int[] tmp = new int[this.packetLength];
            for(int j=0; j < this.buffer[gen][0].length; j++){
                tmp[j]=this.buffer[gen][i][j]^this.buffer[gen][
                    nodes-1][j];
            }
            this.encoded[gen][i]=tmp;
        }
    }

    /**
     * Method to retrive the next packet to be transmitted from
         the master
     * to the slaves
```

```
     * @param generation The generation of the next
       transmission
     * @return Returns an array of the packet to be transmittet
     */
    public int[] getNext(int generation){
        int gen=generation%MEM;
        if(isFull(gen)){
            int n=next[gen];
            next[gen]=(n+1)%(this.nodes-1);
            if(n==this.nodes-2){              //All packets sent
                this.bufferStatus[gen]=0;    //Reset buffer
                    rcounter
            }
            return this.encoded[gen][n];
        }
        else{
            return null;
        }
    }
}
```

Appendix E

decoder.java

```
package ncbtsimulator;

/**
 * Class to perform decoding.
 * This class is instantiatet from the slave nodes.
 * The decoder stores received packets until full rank is
    achieved.
 * Then the decoding is performed.
 * @author Roger Stenvoll
 */
public class decoder {
    int MEM=4;  //Number of generetaions the encoder will keep
       in a memory
    int nodes;  //Nubmber of nodes in the network
    int packetLength;
    int[][][] decBuffer; // buffer used to decode
    int [][] result;        //The resulting array
    int pos;
    int selfID;
    int[] count;

    /**
     * Constructor for the decoder object
     * @param id The id of the node "owning" the object
     * @param nrNodes Number of transmitting nodes in the
        network
     * @param pl The packet length
     */
```

```
public decoder(int id, int nrNodes, int pl){
    this.selfID=id;
    this.nodes = nrNodes;
    this.packetLength = pl;
    this.decBuffer = new int[this.MEM][nrNodes][this.
       packetLength];
    this.count= new int[MEM];
    for(int i=0; i < this.count.length; i++){
        this.count[i]=0;
    }
    this.result = new int[nodes][this.packetLength];
}
/**
 * A method to add own packet to the decoder
 * @param packet own packet
 * @param generation the generation of the packet added
 */
public void addOwnPacket(int[] packet, int generation){
    int gen=generation%MEM;
    if(packet.length<this.packetLength){
        int[] tmp = new int[this.packetLength];
        System.arraycopy(packet, 0, tmp, 0, packet.length);
        for(int i=packet.length-1; i < this.packetLength; i
           ++){
            tmp[i]=0;                //add 0's add the end
               to fill up
        }
        this.decBuffer[gen][0]=tmp;
    }
    else{
        this.decBuffer[gen][0]=packet;    //Alwasy add own
           packet in the first pos.
    }
}

/**
 * A method to add a received packet to the decoder
 * @param packet The received packet
 * @param generation The generation of the received packet
 */
public void addPacket(int[] packet, int generation){
    int gen=generation%MEM;
    this.count[gen]++;
```

```
        this.decBuffer[gen][this.count[gen]]=packet;
        if(isFullRank(gen)){
            decode(gen);
        }
    }

    /**
     * A method to initiate the decoding
     * @param gen
     */
    private void decode(int gen){
                this.count[gen]=0;
        this.result[this.selfID-1]=decBuffer[gen][0];      //
           Copy own data to pos
        if(this.selfID!=nodes){          //If this is not the
           last node, find the last nodes message
            for(int i=0; i< this.packetLength; i++){
                this.result[this.nodes-1][i]=
                        (this.decBuffer[gen][0][i])^
                        (this.decBuffer[gen][this.selfID][i]);
            }
        }
        for(int i=(this.nodes-1); i>0; i--){     //loop through
           all messages
            if(i!=this.selfID){                  //Skip own
               position, known
                for(int j=0; j< (this.decBuffer[gen][nodes-1]).
                   length; j++){
                    this.result[i-1][j]=(this.decBuffer[gen][i
                       ][j])^
                            this.result[nodes-1][j];
                }
            }
        }

    }

    /**
     * A private method to find if all packets are received
     * @param gen The generation to be checked
     * @return true or false is returned
     */
    private boolean isFullRank(int gen){
```

```
        if(this.count[gen]==(this.nodes-1)){
            return true;
        }
        else{
            return false;
        }
    }

    /**
     * A method to print the decoded packet
     * @param node the id of the source node. I.e. the
        originator of the message
     * printed
     */
    public void printReceived(int node){
        try
        {
        Thread.sleep(100*node);      //Otherwise prints out at
           the same time
                                     //causing mixed printout
                                        ....
        }
        catch (Exception e){
            System.out.print("Exception caught: ");
            System.out.println(e);
        }

        System.out.print("\nNode " + this.selfID +
                " received from node: " + node + ": ");
        for(int i=0; i < (this.result[node-1]).length; i++){
                System.out.print(this.result[node-1][i]);
        }
    }
}
```

Appendix F

channel.java

```
package ncbtsimulator;

/**
 * Simulation of the channel.
 * The first element of the array is the header.
 * TS Timeslot number, to avvoid the receiver to "receive" the
    same message
 * several times
 * TO 3 bits
 * FROM 3 bits
 * GEN The generation of the packets
 * PAYLOAD LENGTH values are {1,3,5} timeslots
 * @author Roger Stenvoll
 */
public class channel {
private int[] msg = new int[350];
//The channel synchronized to make sure to multi-threading read
   /write problems
//to the same object.
public synchronized void channel(){
    msg[0]=0;
}

/**
 * A method to read from the channel
 * @return int[] containing the next message
 */
public  int[] read(){
    return msg;
```

```
}
/**
 * A method to write a message to the channel
 * @param m (int[]) the next message
 */
public void write(int[] m){
    this.msg=m;
}
}
```

Appendix G

txBuffer.java

```
package ncbtsimulator;

/**
 *
 * @author Roger
 */
public class txBuffer {
    int[] txBuf;
    int j;
    public txBuffer(int len){
        txBuf = new int[len];
        j=0;
    }
    /*
     * Method to initiate the transmit buffer with some data
     *
     */
    public void initiateBuffer(int id){
        for(int i=0; i< txBuf.length; i++){
            this.txBuf[i]=id;  //TODO Change to a random number
                later??
        }
    }
    public int[] getNext(int nr){
        int[] ret = new int[nr];
        for(int i=0; i < ret.length; i++){
            ret[i]=txBuf[i];
        }
```

```
        j=j+nr;
        return ret;
    }
    public boolean isEmpty(){
        if(j>=txBuf.length){
            return true;
        }
        else{
            return false;
        }
    }
    public int isLeft(){
        return txBuf.length - j;
    }

}
```

Appendix H

rxBuffer.java

```
package ncbtsimulator;

/**
 * A object holding the receive buffer of the nodes in the
    simulator
 *
 * @author Roger Stenvoll
 */
public class rxBuffer {
    int[] rxBuf;
    int j;
    public rxBuffer(int len){
        rxBuf= new int[len];
        j=0;
    }
    /**
     * Method to add a receveid packet to the receive buffer
     * @param next the next packet
     */
    public void writeNext(int[] next){
        //If the rx buffer is to short, increase it!
        if((j+next.length)>rxBuf.length){

            int[] tmp = new int[j+next.length];
            for(int i=0; i< rxBuf.length; i++){
                tmp[i]=rxBuf[i];
            }
            rxBuf=tmp;
            }
```

```
        for(int i=0; i<next.length; i++){
            rxBuf[i+j]=next[i];
        }
        j=j+next.length;
    }

}
```

Appendix I

routing.java

```
package ncbtsimulator;

/**
 * An object holding the routing table.
 * If a pair of nodes are communication bi-directional, two
 * entries in the routing table are inserted.
 * @author Roger Stenvoll
 */
public class routing {
    int[] route;
    public routing(int nr){
        route = new int[nr];
    }
    /**
     * A method to add an entry to the routing table
     * @param src The source node
     * @param dst The destination node
     */
    public void addRoute(int src, int dst){
        route[dst]=src;
    }
    /**
     * A method for reading from the routing table
     * @param dst the destination
     * @return the source
     */
    public int getSrc(int dst){
        return route[dst];
    }
```

```
}
```

Appendix J

counter.java

```
package ncbtsimulator;

/**
 * Class to store the number of bits transmitted from the
    different nodes
 * Based on this calculate power consumption
 * @author Roger Stenvoll
 */
public class counter {
    int[] ctr;

    /**
     * Constructor of the counter class. Creates an array
        storing
     * the information about number of bits transmitted per
        node.
     * @param nodes Number of nodes (master + slaves) in the
        network
     */
    public counter(int nodes){
        ctr = new int[nodes];
    }

    /**
     * Method to add bits to the counter
     * @param node The node id
     * @param payloadSize The size of the payload transmitted
     */
    public void add(int node, int payloadSize){
```

```
        int tmp;
        if(payloadSize == 0){    //poll packet
            tmp=126;
        }
        else{                    //Data packet
            tmp=payloadSize*8 + 126;
        }
        this.ctr[node]=this.ctr[node]+tmp;  //increment the
           counter of the node
    }

    /**
     * Node to get the number of bits stored in the array
     * @param node The node id
     * @return returns the number of bits transmitted by the
        @node
     */
    public int printBits(int node){
        return this.ctr[node];
    }

    /**
     * Method to get the accumulated power consumption of the
        @node
     * @param node The node id
     * @return The power consumption
     */
    public int printPower(int node){
        double tmp;
        int temp;
        tmp=this.ctr[node]*2.5;
        temp = (int) tmp;
        return temp;
    }

}
```