RWTH AACHEN
RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN

COM SYS

Energy Efficient Neighbor Discovery for Podcast Sharing on Mobile Devices under Linux

Rheinisch-Westfälische Technische Hochschule Aachen
Informatik 4 ComSys

Diploma Thesis

Christoph Wollgarten

Advisors:

Dipl. Inf. Jó Ágila Bitsch Link
Prof. Dr. Klaus Wehrle
Prof. Dr. Bernhard Rumpe

Registration date: 16. Dec 2010
Submission Date: 16. June 2011

I hereby affirm that I composed this work independently and used no other than the specified sources and tools and that I marked all quotes as such.

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Aachen, den 16. Juni 2011

Abstract

We present LiND, an energy efficient neighbor discovery framework that enables Linux based systems to discover and connect to neighbors via IEEE 802.11, which are only available sporadically. Using quorum schemes, we schedule on and off times of the wireless transmitters, to guarantee mutual discovery with minimum power given a specific latency requirement. Neighbor discovery is a fundamental part of intermittently connected networks, such as disruption and delay tolerant networks and optimizing it, can lead to significant overall energy savings. Our real world tests show the discovery probability typically to be higher than 99% within the given latency requirement.

On top of this framework, we implemented a prototype podcast sharing application, based on Peer-to-Peer mechanisms, such as BitTorrent and magnet URIs. We further show the application's feasibility for university scenarios using the Reality Mining data set, allowing assumed course material to disseminate to absent course participants within 48 hours.

Zusammenfassung

Diese Arbeit präsentiert LiND, ein Framework zur energieeffizienten Nachbarerkennung, das Linux-basierten Systemen ermöglicht ihre Nachbarn via IEEE 802.11 zu entdecken und sich mit ihnen zu verbinden. Unter Verwendung von Quorum Schemata, werden die "an" und "aus" Zeiten so gewählt, dass eine gegenseitige Entdeckung mit minimalem Energieaufwand bei einer festen Latenzvorgabe ermöglicht wird. Diese Nachbarerkennung stellt einen fundamentalen Teil von sporadisch verbunden Netzwerken, wie Unterbrechungs- oder Verzögerungs-toleranten Netwerken, dar und die Optimierung selbiger kann zu signifikanten Energieeinsparungen führen. Unsere Tests zeigen eine Entdeckungswahrscheinlichkeit von über 99% bei einer gegebenen Latenz.

Auf diesem Framework aufbauend haben wir eine prototypische Podcastsharing Anwendung implementiert, welche auf Peer-to-Peer Mechanismen wie BitTorrent und Magnet URIs zurückgreift. Weiter zeigen wir die Güte des Ansatzes in einem Universitätsszenarium auf Basis des Reality Mining Datensatzes. So wird angenommenes Kursmaterial in diesem Szenario innerhalb von 48 Stunden auch den abwesenden Kursteilnehmern zugeführt.

Thank You

This work is dedicated to my beloved parents which always believe in me, without being pushy, somehow got me moving in the right direction. Thank you for your love and the support through my entire life.

I am heartily thankful to my supervisor, Jó Ágila Bitsch Link, whose extraordinary encouragement, guidance and support from the initial to the absolute final level enabled me to perform my thesis. Without you this thesis would not have been possible - Thank you Jó!

I want to express my appreciation to my girlfriend Tamara, without her love, support and her appreciation for my work, I would not have finished this thesis in time - I wish you a very happy birthday today, Tamara.

Furthermore, I want to thank my family and all my somehow involved friends for giving me generous support in various ways during my thesis period. Thank you Mika for proofreading my thesis. I'd like to show my gratitude to Ronny, for the helpful discussions during our coffee breaks. Thank you Nico for borrowing me hardware equipment. I would also like to thank the research team of COMSYS (I4) for warmly integration into several activities, I will keep you all in good memory. Last but not least, many thanks to Prof. Klaus Wehrle, for granting the opportunity to write my thesis at his chair.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Structure of the Thesis 2

2 Background **5**

2.1 Podcasting 5

2.2 Neighbor Discovery 6

2.2.1 Wireless Sensor Networks 6

2.2.2 Neighbor Discovery of Bluetooth 7

2.2.3 Neighbor Discovery of IEEE 802.11 7

2.3 Power Management in Ad Hoc Mode defined by IEEE 802.11 9

2.4 WMM/WME Power Save defined by IEEE 802.11e 10

2.5 The Linux RFkill Procedure 11

2.6 Quorum Schemes 12

2.6.1 Quorum Systems and Rotation Closure Property 12

2.6.2 Using Quorum Systems for Neighbor Discovery 13

2.6.3 Perfect Difference Sets 14

3 Related Work **15**

3.1 Enhancements of the IEEE 802.11 Power Management 15

3.1.1 Adaptive Power Saving Mechanism 15

3.1.2 Self-Tuning Power Management 16

3.1.3 Asynchronous Wakeup for Ad Hoc Networks 16

3.2 Wireless File Distribution 16

3.2.1 BlueTorrent 16

3.2.2 Penumbra . . . 17
3.3 Quorum Schemes Analysis and Evaluation . . . 18
3.3.1 Disco . . . 19
3.3.2 U-Connect . . . 19
3.3.3 Grid . . . 20
3.3.4 Torus . . . 20

4 Design **21**
4.1 Requirements . . . 21
4.1.1 Independence of Wireless Device Hardware . . . 21
4.1.2 Energy Efficient Neighbor Discovery . . . 22
4.1.3 Prevent Polling Requests . . . 24
4.1.4 Tag Based Podcast Request . . . 25
4.1.5 Podcast Transmission and Retransmission . . . 26
4.1.6 Scalability and Limitation of Simultaneous Transmissions . . . 28
4.1.7 Independence of Graphical User Interface . . . 29
4.1.8 Independence of Power Toggle and Basic Application . . . 29
4.2 General Issues . . . 30
4.2.1 Beacon Loss Handling . . . 30
4.2.2 Security, Privacy and Identity Protection . . . 30
4.2.3 Spam, Undesirable Content and Abuse . . . 31
4.3 Basic Modules . . . 31
4.3.1 LiND_pwr kernel module . . . 32
4.3.2 LiND daemon . . . 32
4.4 Slot Behaviour . . . 33
4.5 Communication Data Structures . . . 33
4.5.1 Beacon and Reply Beacon . . . 34
4.5.2 File List Request and Response . . . 35
4.5.3 File Request and Response . . . 35
4.5.4 Domain Socket and Control Interface . . . 36

5 Implementation **39**
5.1 Kernel Source Analyzing and Logging . . . 39
5.2 IEEE80211 Operations . . . 40
5.3 Generic mac80211 Module . . . 41
5.4 LiND_pwr Kernel Module . . . 42
5.5 LiND Daemon Application . . . 44
5.5.1 Runtime Arguments of LiND . . . 45
5.5.2 Daemon and Thread Organization . . . 45
5.5.3 Beacon Injection and Capturing . . . 46
5.5.4 Force Power on Mode . . . 47
5.5.5 Neighbor and Podcast Maps . . . 47
5.6 Porting to Android . . . 49

6 Evaluation **51**
6.1 Measurements and Analyses . . . 51
6.1.1 Setup and Circuit Diagram for Measurements . . . 51
6.1.2 Power Toggle Energy Saving . . . 52
6.1.3 Power Toggle Delay . . . 53
6.1.4 Power Toggle with Neighbor Discovery Algorithms . . . 54
6.1.5 Beacon and Reply Beacon Error Rate . . . 58
6.2 Simulation of a Podcast Sharing Scenario . . . 59
6.2.1 Reality Mining Dataset Analyses . . . 59
6.2.1.1 User Behavior . . . 59
6.2.1.2 User Groups and Tutors . . . 60
6.2.2 Podcast Dissemination . . . 61

7 Conclusion **65**
7.1 Summary . . . 65
7.2 Limitations . . . 66
7.3 Future Work . . . 66
7.4 An Outlook into Broader Applicability . . . 67

Bibliography **69**

List of Figures **72**

List of Tables **76**

1

Introduction

Mobile systems, like smartphones, are world-wide distributed powerful devices that manage people-centric information. Beside voice calls and SMS, these devices have to process a set of other tasks during a day. People use their smartphones to photograph, listen to internally stored music or internet streams, podcasting, watching videos, writing, sending and receiving emails, localize their position, playing games, reading ebooks and browsing the world-wide-web. A smartphone's battery drain depends on the user's usage. Power capacities of mobile devices are limited, at least with respect to the limitations of the build-in battery. Thus, every task on such devices has to be adjusted to strict follow low-energy usage. Focusing the sporadic neighbor discovery problem, a trade-off between power consumption and discovery latency is necessary. This is, were our framework LiND comes in.

1.1 Motivation

We currently experience an ongoing spread of mobile smartphones with built-in wireless devices, started in the recent past, see press release of Gartner Inc., section 7.4. That enables us to follow, that in a very near future, the probability to be in range of other wireless devices, does highly increase. For urban regions that means, walking through the streets, sitting in a café or waiting for a bus, as illustrated in Figure 1.1, there will always be a good chance to establish a sporadic Peer-to-Peer based communication between two wireless devices and share podcasts based on the owners interests. Sharing partners do not necessarily have to know each other and might meet in a physical wireless range only once in their lifetime. Our idea is not only limited to public urban places. In universities, a set of students take the same courses, thus, they meet each other, possibly, multiple times a day. Between wireless mobile devices, our intention is to perform sporadic Peer-to-Peer connections automatically, without the need of user interactions. The connection establishment necessarily requires a neighbor discovery process. Data exchange can possibly bases on predefined tags or topics that describe a user's interests.

Figure 1.1 Motivation: Students waiting at a bus stop. Reliable and energy efficient neighbor discovery provides a basis to establish a sporadic Peer-to-Peer based communication between wireless mobile devices, like smartphones. We present a corresponding applicable prototype framework to perform energy efficient neighbor discovery and automatic podcast sharing under Linux.

1.2 Structure of the Thesis

This thesis introduces and evaluates our LiND framework to perform energy efficient neighbor discovery using Linux based systems associated with a prototype public podcasts sharing application. This thesis is structured as follows: Chapter 2 gives background information. We explain what podcasts are, how neighbor discovery works and why this is an important research area. As example, we give a short description of wireless sensor networks. We introduce the neighbor discover procedure of Bluetooth and the corresponding discovery latency. We introduce the W-LAN specifications and describe how neighbor discovery works using an infrastructure and an ad hoc network topology. The power management features of ad hoc networks are described. We explain how the WMM/WME power save modes work and why they are not applicable to perform energy efficient discovery. The Linux RFkill procedure is introduced and we show that it is possible to switch the wireless transmitter of a WiFi device *off* to save energy. At the end of chapter 2 we give necessary theoretical knowledge of quorum schemes. In chapter 3 related work is discussed. First we describe already published solutions to improve the energy consumption in an ad hoc W-LAN networks, and why these are not compatible to generic ad hoc networks. We introduce two existing wireless file sharing applications, BlueTorrent and Penumbra, both are not designed with respect to energy efficiency. Quorum systems can be used to perform energy efficient neighbor discovery, this is shown at the end of chapter 3. Additionally, four different quorum algorithms are presented: Disco, U-Connect, Grid and Torus. The design of our proposed solution is described and illustrated in chapter 4. First we describe a list of desired requirements for our podcast sharing application and present corresponding specific solutions. After this we discuss general issues, like the beacon loss problem. Furthermore, we describe the basic modules of our LiND implementation and the necessary communication structures. Our implementation is described in chapter 5. Source code of centralized parts, for example of the necessary Linux kernel modifications,

are denoted. Further, we illustrate how parallel operations within our application are proceed. The evaluation of our application is characterized in chapter 6. Underlying energy measurements of our real world tests, and the respective evaluation results are presented. According to desired latency times, we evaluate our implementation with respect to the presented different neighbor discovery algorithms. Our results of a simulated file sharing scenario within a university, using the Reality Mining data set, is shown at the end of chapter 6. In the last chapter we summarize our work and denote future applications.

2

Background

This chapter provides background information on podcasts, the neighbor discovery problem itself and the problem of energy consumption with respect to neighbor discovery. Further we explain the basic facts of the neighbor discovery procedures of two popular protocols, Bluetooth and W-LAN. Thereby we unveil the general power management in W-LANs. Under Linux it is possible to power off all modern radio transmitters with the RFkill subsystem, that saves energy and we give a short introduction to that. Further, we explain what quorum systems are and how quorum algorithms can relate to the neighbor discovery energy challenge. A detailed analysis and evaluation of different quorum algorithms for neighbor discovery in wireless sensor networks is presented later in the chapter of our related work.

2.1 Podcasting

The internet connects people to each other and makes it possible for everyone to publish any kind of digitalized data. In developed countries, people are digitally linked through high-available digital networks. That allows publishing private information to the world and give people the option to receive other information than the classic media that tv and radio productions, or newspaper and magazine agencies, offers. One example for a centralized platform is YouTube. Podcast, composed of two short words: pod and cast, in combination it simply stands for portable on-demand broadcast. In technical terms, a podcast is a digital audio or video file containing encoded media informations. However, the content of a podcast file is not only limited to audio or video, any other file format is possible, for example pictures, articles or presentations. In general, a podcast is offered as a file download on the web provided by a podcaster. A podcaster is a person who publishes content on web-servers to subscribers which use a podcatcher software to receive the offered podcasts. Podcasting allows any person, connected to the publisher over a shared medium, to receive informations of his own interests. At present, the shared medium is the internet including different types of data storage possibilities. Our work offers

ways to extend the dissemination range of podcasts on devices that are not regularly connected to the internet.

2.2 Neighbor Discovery

In the area of dynamic wireless mobility management, the problem of neighbor discovery is a major challenge. It is the first step that must be completed before 2 or more self-organized wireless appliances have the chance to form a temporary network and establish a communication between each other. Particularly in an variable environment without predefined or regular rendezvous points when devices are in communication range, it is necessary to perform a reliable neighbor discovery scheme to make device detection and further data request and response possible.

There are two different types of performing neighbor discovery, probabilistic approaches and deterministic algorithms. In the case of randomized neighbor discovery algorithm, a device discovers devices in communication range within a given time with a, potentially, high probability. Each of the deterministic solutions make use of a previously defined schedule that is implemented as a deterministic algorithm. Without concerning radio frequency interferences, theoretically, a well designed deterministic algorithm guarantees to execute a neighbor discovery successfully within a specific bound of time.

To learn about the presence of other devices in range, each device has to listen to the wireless channel in the hope to capture beacons of other devices. Correspondingly, from time to time, each device has to send beacons to call attention to itself. As a logical step, the highest probability to capture foreign beacons can be obtained by listening for beacons to the shared wireless channel all the time. Listening to the wireless channel, however, consumes energy similar to the energy that is consumed while sending beacons or application data. Listening to the wireless channel at all time uses up a maximum of energy. Switching the wireless device to a low power level or off makes it impossible to send beacons and listen to the channel, that results in a higher discovery latency time. For that reason, it is necessary to handle a trade-off between energy consumption and discovery latency. Our goal is to invest a minimum of energy for the wireless local area network device and simultaneously to achieve acceptable discovery latency.

2.2.1 Wireless Sensor Networks

Wireless sensor nodes are small electronic devices equipped with different kinds of environmental sensors to gather informations of a nodes ambient environment, for example temperature, audio signals, chemical substances, vibrations, ambient light and electromagnetic radiation. Their task is to monitor their own surrounding area and, sometimes based on special events, establish communication links to neighbouring nodes so that the result is a network to transmit or store the information distributed in the network. Sometimes wireless sensor nodes have to be placed randomly, sometimes they are moving, depending on their application field. To build up a wireless network they have to perform some kind of neighbor discovery. The development of smaller hardware and the growing need of industrial, civil and military

applications of wireless sensor networks increase the need for achieving maximum efficiency for the whole node system. This includes the neighbor discovery problem. In the area of research, evolving and optimizing alternative, more efficient discovery protocols is necessary, particularly with respect to energy consumption, because wireless sensor network nodes are powered by batteries, and long service lifetime is, in most applications, an important requirement. However, in wireless sensor networks the wireless hardware is one of the highest energy consumers. This is also the case for wireless mobile devices equipped with a wireless local area transmitter. The wireless sensor networks prepare a basis to evolve energy efficient neighbor discovery protocols for asymmetric discovery purposes. Hence that mechanism can also be adapted to perform efficient wireless neighbor discovery on mobile devices like netbooks or smartphones.

2.2.2 Neighbor Discovery of Bluetooth

Bluetooth is a short-range communication technology developed by the Bluetooth Special Interest Group. Different versions are released which differ in steady improvements of the specified protocols in terms like communication data rates, discovery delay times, security, error correction and energy consumption. The latest version of 2010 is Bluetooth 4.0 [21]. This radio frequency technology works in the unlicensed industrial, scientific and medical (ISM) baseband of 2.4 GHz, as the wireless local area networks, and has a typical range of about 10 meters. The first versions achieved approx. 721 Kilobits per second - in theory. Later enhanced versions data rates of 2.1 Megabits per second, in the EDR (Enhanced Data Rate) mode, and 24 Megabits per second with an 802.11 AMP (Alternate Media Access) controller are theoretically possible. With an AMP controller, the discovery process is performed by the Bluetooth device and the data transmission by an IEEE 802.11 compatible device through the PAL (Protocol Adaptation Layer).

To perform discovery, a Bluetooth device enters the inquiry mode. In that mode it broadcasts beacons, called inquiry messages, while hopping through different predefined frequencies called hopping sequence. These inquiry messages contain information of classes of devices that are invited to connect, that is the inquiry access code. The inquiry access codes are based on a devices address. The GIAC (general inquiry access code) indicates all Bluetooth devices. The DIAC (dedicated inquiry access code) indicates that only devices of certain classes are invited to connect. To capture inquiry messages, a device has to periodically enter the inquiry scan mode. When a device receives an inquiry message it should answer with an inquiry response, if the inquiry access code matches. The inquiry response packet may transmit extended informations such as the supported services or the device name. However, an inquiry response packet is optional. Depending on the delay of the necessary frequency hopping mechanism, the overall discovery process is theoretically performed within a period of 10.24 seconds.

2.2.3 Neighbor Discovery of IEEE 802.11

Wireless communication via radio signals with a limited variety of radio frequencies must be well defined to provide each wireless unit controlled physically medium ac-

cess. The IEEE 802.11 standard [13] (ISO/IEC 8802-11, Wi-Fi Alliance) describes a set of interfaces, functions, procedures and physical signaling techniques to provide integrative wireless local area network (WLAN) communication supporting wireless hard- and software of different manufacturers. Considering the IEEE 802.11 specifications, a stationary, portable or hand-held wireless device that follows that requirements can mainly operate in 2 different modes: *infrastructure mode* and *ad hoc mode*. The use of different modes only affects the protocols and not the physical layer.

In infrastructure mode, at least, one device is defined as access point and all other devices are able to connect to that access point. The access point itself and all connected clients are called *basic service set* (BSS). The network is identified by the *basic service set identifier* (BSSID). The BSSID is a binary sequence with a length of 6 bytes, it is defined by the BSS access points MAC address. Devices that enter the wireless range will receive a beacon packet from the access point. The packet is sent on a predefined frequency and in short intervals of 100 ms. Each beacon includes the access points BSSID that identifies the network. After capturing such a beacon, a potential client has knowledge of the presence of the corresponding network and is capable to try to join it as a client. After joining the network a client may communicate with other clients using the access point as a physical gateway, direct inquiries between multiple clients without passing packets through the access point are not possible.

An ad hoc network describes a wireless network between two or more mobile devices that to not require a fixed infrastructure to exchange data. It is possible to build up ad hoc networks with mobile wireless devices like laptops, netbooks or smartphones equipped with a wireless local area radio module. In ad hoc mode, all participants have to organize the network decentralized. The set of all nodes that form the same ad hoc network is called *independent basic service set* (IBSS). There is no centralized access point that manages communication among the network, devices communicate by forming a peer-to-peer network. The absence of a centralized access point means that two communication partners must be in each others communication range to exchange data and each device has to send out beacons including a shared pre-assigned *service set identifier* (SSID). All devices of an ad hoc network must operate on the same agreed radio frequency, thus a fixed channel must also be predefined. First a device receives a beacon from another node of an ad hoc network, already existing of at least one device. It compares the SSID of that beacon frame to a pre-defined expected SSID. On success it performs a time synchronization process according to the beacons time value. This is necessary to send further beacons in coordinated time slots synchronized to the member of the existing ad hoc network. The Time Synchronization Function (TSF) works as follows: Each device, configured in an ad hoc mode, starts a timer (TSF clock) with time zero and increases that value constantly. Each device sends out beacons which include that time value. By receiving a foreign beacon from a member of the configured ad hoc network, a device must adopt it's own timer to the transmitted time value if the transmitted value is higher than the own one. By that, after short time, all devices adjust their timer to the same value. For more in-depth information on time synchronization see Time Synchronization Function (TSF) in [13]. Further, a new device adopts the beacon period out of the received beacon from the network it joins. In the end, the results are synchronized timers and a fixed beacon period among the ad hoc network

participants. This is a basic condition for any device to perform power management. We want to keep our main focus on a decentralized network topology, therefore, in this thesis, we explain and analyze most detail issues only for the ad hoc mode and not for the infrastructure mode.

2.3 Power Management in Ad Hoc Mode defined by IEEE 802.11

After describing the initial neighbor discovery procedure in IEEE 802.11 ad hoc mode in the previous section, we now focus on the defined power saving mode (PSM) of IEEE 802.11 as defined in [13]. Using wireless LAN hardware in mobile devices, it is advised to switch to a lower power level, as often as possible, to save energy in phases where no exchange of data is necessary, and perform a wake-up of the radio device, set it back to a higher power level, when communication is needed. The problem is a reliable prediction of the of time when the next data packets arrive. When the wireless hardware operates at a low power level, it is not possible to receive any beacon or data frames of potential communication partners. The IEEE 802.11 power management covers that issue.

The necessary time synchronization process in an ad hoc network is already described in 2.2.3. After that, each device is able to calculate the time periods when beacons have to be send and when beacons should be captured. With that knowledge, a device is able to switch to a lower power state outside that critical times, however, only if no further transmission, for example an exchange of application data packets, has to be performed. Emerging out of that, each device needs an indicator when to stay active for capturing additional wireless data frames. A listening phase at the beginning of each beacon period, called Announcement Traffic Indication Message (ATIM) window, allows each node to transmit a selective communication request to other nodes. ATIMs are not limited to single destination requests. Multicast requests are possible. By receiving such a request, a device sends an acknowledgement frame back to the source and stays active for the rest of the beacon period to receive data frames. After receiving the acknowledgement frame, the source of the ATIM request starts to deliver the buffered data packets to the awake receiver. After the beacon period, participants have to negotiate new transmissions by further ATIMs.

However, analyzed in [25] and illustrated in Figure 2.1, the quantitative results of the analytical study of the IEEE 802.11 PSM in the IBBS mode verify, that the specified power management, particularly for low workload, does not fulfill high energy efficiency:

> "Indicated by the analytical results, it keeps the device active around 50% of the time for 10% traffic load and 18% for traffic load around 1%. This is mainly due to the periodical nature of the IEEE 802.11 PSM. Similar results can be obtained for systems with multiple competing users. This justifies the need for adaptive power saving solutions that operate at finer control granularity." [25].

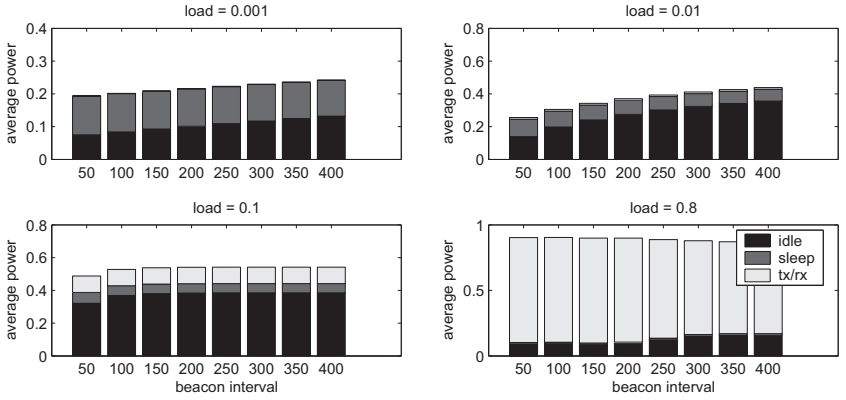


Figure 2.1 Analysis of the energy consumption of the 802.11 Power Saving Mode with respect to different beacon intervals grouped by different traffic loads. The analysis and the graph is shown in [25]

In general use cases, following our intention, to continuously abuse wireless LAN hardware in mobile devices for neighbor discovery all day long produces not heavy traffic load on the wireless channel. Most of the time, at runtime, the device would be in idle state and does not have to operate on high traffic load. Under this assumption, the specified IEEE 802.11 PSM is not energy efficient, but there exist approaches that can enhance that energy problem, listed below in the next chapter 3.1.

2.4 WMM/WME Power Save defined by IEEE 802.11e

The WiFi Multimedia (WMM) standard, also called Wireless Multimedia Extension (WME), introduced by the WiFi-Alliance, is based on the IEEE 802.11e specifications. The main challenge of that multimedia extensions is to adapt the packet flow of Voice over IP, audio streaming or video streaming, these are all time critical applications that need a constant data packet delivery time. Anyway, the data throughput is not guaranteed by that standard, that is physically not feasible. The data packets are classified by their desired maximum delay, which depends on the case of application. The standard defines four different categories for data packets, each category is represented by a transmission queue: voice, video, best effort and background. The mechanism can be compared to the quality of service (QoS) strategy. Clients of an access point are able to send a WMM-trigger frame to the access point. The access point then replies with the desired, buffered data stored in the corresponding queue. Thus, the sleeping and awake periods of a wireless device are not fully limited to the rules of the IEEE 802.11 PSM with fixed delays based on the predefined beacon interval periods. The WMM power save mechanism increases the efficiency of the data transmission and additionally is backward compatible with the IEEE 802.11 PSM. Optimization of energy consumption only takes place when transferring data through optimizing the traffic flow, thus it does not improve the neighbor discovery procedure of IEEE 802.11 in IBSS mode.

2.5 The Linux RFkill Procedure

The previously introduced power management mode shows, that a W-LAN device can operate on different energy consumption levels. At the lowest level, the transmitter is without power and this is not defined in the power management. In general, the manufactures of modern netbooks, tablets and smartphones, equipped with wireless hardware, enabled the possibility to turn off the internal wireless transmitter. That basic feature is necessary to switch off the wireless components without powering down the complete system. As an example, in airplanes, people might use their netbook to work, prepare their customer presentations or use their smartphone to listen to music or watch videos, but the radio device must be deactivated. Devices that transmit radio signals are not allowed in public airplanes. Therefore, modern wireless devices provide a hardware or software controlled switch to deactivate the wireless transmitter. Because of the airplane case, the state of a deactivated wireless transmitter is also called flight-mode. High power saving, locally regulated security aspects or the recognizability of the own device ids, transmitted in beacons or privacy concerns, may also be good reasons to deactivate the wireless components.

Radio Frequency Kill (RFkill) is a generic interface created for the wireless Linux drivers to deactivate wireless transmitters in the system, including wireless LAN, Bluetooth, GPS, radio fm, GSM and the 3G system. A deactivated transmitter, does not consume any power for radiation. RFkill also provides standardized hooks for the user space to listen to the responsible hardware button events, invoked by a user. The unpowered state of the transmitter is called blocked state. RFkill provides two types of radio blocks:

hard block:	radio block that cannot be influenced by the systems software
soft block:	radio block that is writable by the system software

The physical effect of both blocks, with respect to the wireless transmitter, is the same. The hard block just indicates that the device is deactivated by a hardware switch and cannot be unblocked by any software commands. Different to that, the soft block is invoked by the systems software, thus a software command is able to unblock the blocked device. However, for the soft block, there is no need to be readable for the system. As example, to easily switch off and on all radio devices from the console, required that at least one radio device and the corresponding drivers support RFkill:

```
# block and unblock using the sys filesystem:
for i in ‘find /sys -name "rf_kill"‘ ; do echo 1 > \$i ; done
for i in ‘find /sys -name "rf_kill"‘ ; do echo 0 > \$i ; done
# .. or with the RFkill command of the package "rfkill":
rfkill block all
rfkill unblock all
```

The RFkill architecture design consists of three components: the RFkill core, a RFkill input module and the hardware drivers itself. Linux kernel device drivers communicate with the RFkill through an API provided by the RFkill core. In general

every wireless driver implements the RFkill functionality. Deactivating a wireless LAN device, using the RFkill command, does not only deactivate the transmitter, first RFkill shuts down the logical interface to deconfigure the network based system operation definitions in /etc/network/interfaces. With respect to the deactivated network device, that will additionally break every established IP connection, remove every assigned IP address, remove every rule from the network routing table and deactivate the corresponding link layer connection. The other way, activating the transmitter with RFkill, additionally brings the logical interface up again. That will reset it under the purpose of a desired IP and netmask. First the link layer is activated, after that the IP address is assigned and the routing table is updated. With respect to this, we observed, that the RFkill block and unblock operations take some time. Each duration highly depends on the used wireless hardware and the implementation of the hardware drivers. For example one device may need 20 ms to be blocked and another device may need 120 ms.

2.6 Quorum Schemes

Quorum schemes are used for very different computer science applications. For instance to assist organization of data mirrors on RAID (Redundant Array of Independent Disks) systems, or for replication control and commit methods in distributed database systems, other mutual exclusion problems, neighbor discovery over a shared medium or for power saving purposes in ad hoc networks [14]. To introduce quorum schemes and the opportunity to apply such quorum systems for the neighbor discovery challenge, we first give some basic facts and mathematical definitions, already discussed in [14, 20].

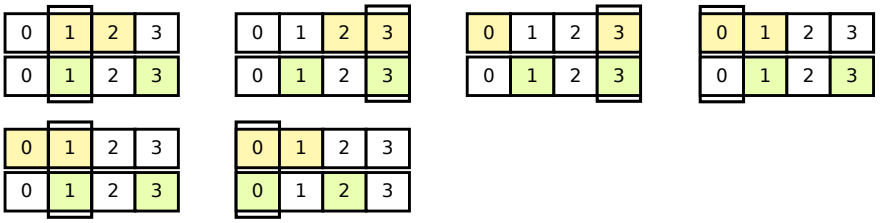
2.6.1 Quorum Systems and Rotation Closure Property

Under a given superset S, the set $Q = \{S_1, .., S_n\}$ is called a quorum system under S, if for any $S_i, S_j \in Q$ holds: $S_i, S_j \subseteq S$ and $S_i \neq S_j$ and $S_i \cap S_j \neq \emptyset$. Each $S_i \in Q$ is called a quorum of Q.

For a quorum system Q under the superset of $S = \{0, .., n-1\} \subset \mathbb{N}_0$, a positive number m and a quorum $S_i \in Q$, the *rotation function* is defined as follows:
$rotation(S_i, m) = \{(j + m)\ mod\ n \mid j \in S_i\ \}$.

A quorum system Q under the superset of $S = \{0, .., n-1\}$ is said to have the *rotation closure property* if for Q holds:
$\forall\ S_i, S_j \in Q,\ m \in \{0, .., n-1\} : S_i \cap rotation(S_j, m) \neq \emptyset$.

For example, the quorum system $Q = \{\{0, 1\}, \{0, 3\}\}$ under $\{0, 1, 2, 3\}$ does not have the rotation closure property, because $\{0, 1\} \cap rotation(\{0, 3\}, 3) = \emptyset$. The quorum system $Q' = \{\{0, 1\}, \{1, 3\}\}$ under $\{0, 1, 2, 3\}$ does have the rotation closure property. The following illustration represents $rotation(\{0, 1\}, m)$ in the first row and $rotation(\{1, 3\}, m)$ for $m = 0..3$ in the second one:

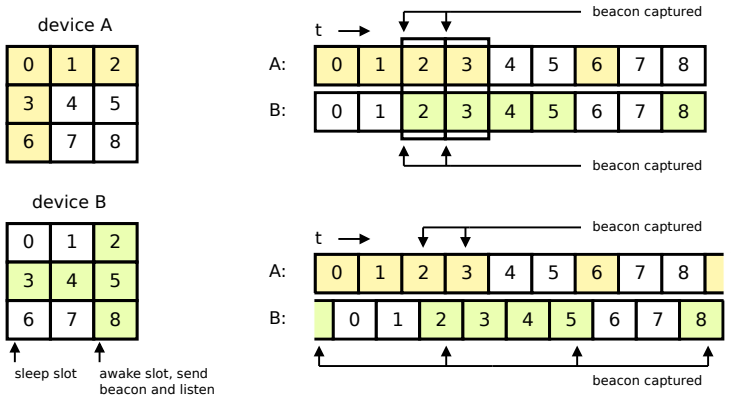


We would like to point out, that it is possible to construct quorum systems consisting of only one quorum. For example the system $Q'' = \{\{0, 1, 3\}\}$ under $\{0, 1, 2, 3\}$.

2.6.2 Using Quorum Systems for Neighbor Discovery

The idea of neighbor discovery between already synchronized devices is easy. Nodes wake up within predefined intervals and listen to beacons within that intervals. The idea to use quorum schemes for asynchronous neighbor discovery is not new, further details and a mathematical proof, that it works is given in [14]. Asynchronous neighbor discovery, without adjusting time values, can be much more energy efficient [20, 24] and does not need additional time synchronization functions. Within that section we want to pass the general idea how energy efficient neighbor discovery can be realized.

Assume the quorum system $Q = \{\{0, 1, 2, 3, 6\}\}$ under $\{0, .., 8\}$ is used for an asynchronous neighbor discovery process between two devices. We assign to device A the quorum itself and to device B a rotation with 2 of the quorum, that represents an initial clock drift between the two devices. Each element of a quorum represents an active slot, each other element of the superset, that does not exist in the quorum itself, indicates a sleeping slot. Both devices start the discovery process at different clock drifts, illustrated in the following figure. The clock drift doesn't have to be a multiple of the used slot length to work successfully, that is also already discussed in [14]. A device sends out a beacon at the beginning of an active slot and listens the rest of that active slot to capture foreign beacons.



Out of the assigned slot length and the minimum number of the overlapping slots, depending on the used quorum itself and its possible rotation combinations, it is, theoretically, possible to give a lower and an upper bound for the discovery latency time.

2.6.3 Perfect Difference Sets

As already published in [20], PDS (Perfect Difference Sets) can be used for neighbor discovery to achieve an optimal balance between power consumption and discover latency time. In simple words, a PDS is an optimized quorum, that means the quorum size is minimal in reference to the number of elements. As explained before, to perform asynchronous neighbor discovery, each element in a quorum represents an active slot. Each active slot consumes an amount of energy. For an asynchronous discovery process, we should keep the number of active slots as low as possible, but as high as necessary to perform discovery within a certain time period. Constructing PDS under $\mathbb{N}_n$, particular for higher n, is computational intensive. However, we want to give the mathematical definition of PDS:

A set $S \subset \mathbb{N}_n = \{0, .., n-1\}$ is called a PDS under $\mathbb{N}_n$ if $\forall m \neq 0 \in \mathbb{N}_n$, $\exists!$ $\{s_i, s_j\} \subset S : m = (s_i - s_j)\ mod\ n$. For the existence of a perfect difference set, it is a necessary condition that n is of the form k^2+k+1 with $k \in \mathbb{N}$, [12]. It is a sufficient condition, that k is a power of a prime number, for example $k = p^x$ with a prime number p and $x \in \mathbb{N}$, a PDS then can be created under $\mathbb{N}_n$ with $n = p^{2m} + p^m + 1, m \in \mathbb{N}$, [22, 12].

A PDS fulfills the requirements of a quorum system, that means non–empty intersection and the rotation closure property. Further, by applying the rotation function on a PDS, it is possible to build a quorum system with equal size of any quorum, [20].

3

Related Work

Work has already been done to enhance the IEEE 802.11 power management and further perform neighbor discovery using quorum systems. We introduce these work in the first part. Two applications, which already perform wireless file sharing, based on Peer-to-Peer mechanisms, are presented in the second part. The third part gives knowledge of four applicable quorum algorithms, based on quorum systems, which enable energy efficient neighbor discovery.

3.1 Enhancements of the IEEE 802.11 Power Management

This chapter gives a brief insight into already published solutions to enhance the power management of W-LANs. The solicited approaches require to change the highly established basic IEEE 802.11 protocol or modify the default settings of delay times. That isn't a bad thing, but incompatibility may rise between stations. With this thesis we want to provide an energy efficient solution for reliable neighbor discovery by preserving the original IEEE 802.11 protocol.

3.1.1 Adaptive Power Saving Mechanism

Sangheon Pack and Yanghee Choi published an adaptive power saving mechanism [19] for IEEE 802.11. Their solution, estimate the idle period length, evolving an *exponentially weighted moving average scheme*, that adjusts the devices wake-up interval adaptively on the basis of the packet flow of IP communication. For comparison only, the basic wake-up interval of the IEEE 802.11 specification simply is fixed and by default 100 ms. Further their solutions outperform the existing standard.

3.1.2 Self-Tuning Power Management

Another work of Anand, Manish and Nightingale, Edmund B. and Flinn, Jason introduced their *self-tuning power management* (STPM) [2] that reduces the power consumption on heavy loaded wireless networks by switching a wireless device from PSM only in a higher power mode, called Constantly Awake Mode (CAM), when higher performance is needed. That switching operation has a specific duration time depending on the wireless hardware and consumes power within the process. For STPM, modified user applications and an energy-aware operation system is necessary and it has to communicate with their developed STPM kernel module. The Linux kernel module works as centralized controller between user applications, the operating system and the wireless hardware. The applications forward their maximum acceptable delay tolerance of their communication streams to the STPM module, that calculates, out of the wireless hardware characteristics and the energy policies of the operating system, a decision which power mode should be used next under the criteria of low energy consumption. For example, if more power would be saved by switching the device from CAM the lower PSM, despite that the switching operating takes time and some energy, and it would not influence the applications data stream latency badly, then it will be done. That requires that each user application denotes a desired delay tolerance to the STPM module, that then chooses the best power mode with respect of the most critical delay time. Their solution is designed to handle more than two different power modes, that generally enables the support of an additional lower power mode than the PSM mode to achieve a higher power saving than PSM on low network traffic.

3.1.3 Asynchronous Wakeup for Ad Hoc Networks

Rong Zheng, Jennifer C. Hou and Lui Sha evolved an asynchronous neighbor discovery solution for wireless ad hoc networks [24]. They apply a block design problem in combinatorics onto the neighbor discovery problem. The result is a schedule that gives optimal wake-up and sleep durations under consideration of low energy consumption. In summary, the power saving effect of their asynchronous mechanism is denoted with up to 70% compared to the deactivated PSM. But their solution does not switch a wireless device to the lowest power saving mode, which is “off”.

3.2 Wireless File Distribution

There are already applications available to share media files, for example podcasts, over wireless connections between decentralized devices without the existence of a centralized coordinator or wireless access point. We propose two selected applications and cover commonalities and differences to our approach.

3.2.1 BlueTorrent

BlueTorrent [15] is a cooperative P2P file sharing application that uses the wireless Bluetooth technology [21] for neighbor discovery and further communication. Blue-

tooth is developed to easily share audio and video files with other nearby Bluetooth enabled handheld devices, for example cellphones, smartphones or PDAs (Personal Digital Assistant). Also in the area of bluetooth, neighbor discovery is a central challenge. Beside finding optimized settings for the initial bluetooth inquiry and the connection establishment, the proposed paper [15] gives detailed information about the underlying dissemination protocol, that resembles the BitTorrent approach. Before the transmission of a media file, it is divided into a set of pieces. BlueTorrent devices are able to exchange these pieces in an unregulated order. That increases the probability to successfully continue an interrupted transmission, however, related to another new device that contains the needed pieces of a media file. We pick up the fundamental idea of sharing media files automatically across multiple wireless devices. Instead using the wireless Bluetooth technique we evaluate in this thesis the prevailing advantages of the WiFi techniques, for example the significant higher data throughputs of several megabits per second, the extended wireless range of specified up to 300 meters and the better energy efficiency, especially under high throughputs [11]. Transmitting audio and video files results in high throughput. Low throughput dominates only in the discovery mode, when no file transmission is running, therefor we later introduce our power toggle. Further, nowadays, produced devices are equipped with WiFi devices. Their employed torrent-like approach would be a more advantageous additional feature to improve file dissemination. We should keep that in mind for future work, but it is not a basic condition for our first implementation to evaluate our power toggle and to show that it works effectively. However, without using file pieces, our implementation includes a transmission continuation of already partial received files across different sources.

3.2.2 Penumbra

I'd like to mention the penumbra project [3]. Penumbra is a broadcast-based wireless network software. The idea is to connect different WiFi enabled devices to a local network and manage the whole communication by injecting broadcast frames directly to the data link layer. The basic transport and network layer of the currently associated network, that includes TCP, used for reliable and ordered communication, and IP, used for addressing and routing, are circumvented. Two penumbra stations communicate with each other by a predefined packet structure through a PF_PACKET / SOCK_RAW socket or libpcap. The library libpcap allows to inject individual constructed packets directly to the data link layer. Each station adds, additionally to the basic wireless interface, a second logical wireless interface and sets it into the, so called, monitor mode. In monitor mode state, a wireless interface can not only be used to inject packets to the data link layer directly, further it also allows to listen to the wireless channel and capture data link layer packets that are not addressed to the own basic wireless interface. The generic mac80211 wireless stack for Linux gives the functionality to add multiple logical interfaces to one wireless hardware device and assign different operating modes among them. For example, one interface can be assigned to a wireless access point while another interface is set to the monitor mode without serious interference. The mac addresses of the mac header of every penumbra packet are set to the significant value 13:22:33:44:55:66, before it is injected to the layer and broadcasted. Other penumbra stations in wireless range are able to identify that penumbra packet with that significant header. The mac

header does not include sender or destination mac addresses, that preserved privacy. A reed-solomon forward error correction is implemented to protect the transmitted application data of a penumbra packet. On a higher level, penumbra appropriates an individual file sharing protocol that also supports SSL-encryption between stations. A file exchange takes place after an initial request packet, that describes the wanted files by category, mimetype and filename. Other stations which capture that request packet, perform a local media library search and response if a match occurs. After that the file transmission is invoked and all data is send out as broadcast packets.

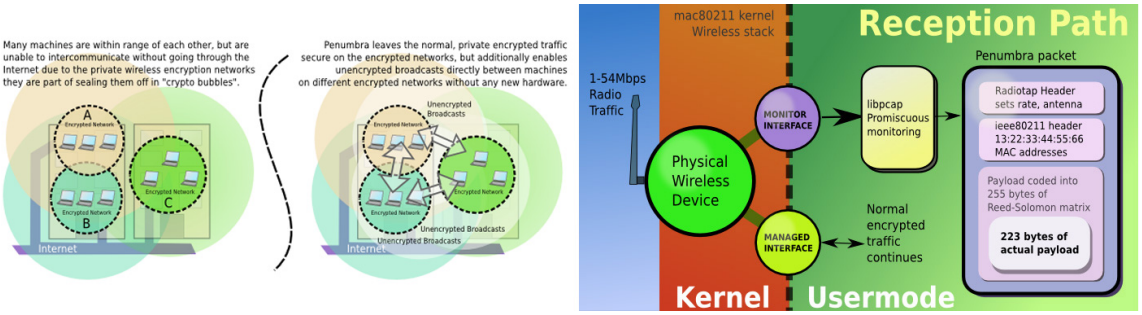


Figure 3.1 Penumbra wifi network illustration of the application, the interface management and how packet capturing is performed. Images are part of the website [3]

In this thesis, we take care of that basic idea to manage multiple logical interfaces simultaneously, supported by the generic mac80211 wireless stack for Linux, further we add our power toggle, later presented, to switch the wireless transmitter on and off in combination with quorum algorithms to perform energy efficient neighbor discovery. One device is set to a monitor device to operate on the beacon management, the other is set to ad hoc mode to manage peer-to-peer connections for reliable file transmissions. We do not need to take care about the error correction, as penumbra does by using reed-solomon, because in ad hoc mode we use the transmission control protocol (TCP). Encryption can be realized easily, for example SSL/TLS can be used on TCP/IP, but it is not really necessary in a network that is based on public content. For now, in this thesis, encryption isn't the challenge. However, our design includes a privacy protection, limited to randomly assigned mac and IP addresses. More technical details of our approach are given in a later chapter.

Incidentally, our implementation should demonstrate a practicable solution for our energy efficient power toggle implementation, hence, our proposed implementation does not include as many additional features like the herein proposed applications.

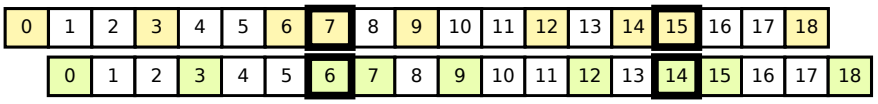
3.3 Quorum Schemes Analysis and Evaluation

Quorum schemes and the mathematical definitions of a quorum and a quorum system, are already introduced in the previous chapter. We also demonstrated their relation to perform asynchronous, reliable neighbor discovery. In that chapter, we introduce different neighbor discovery algorithm, based on quorum schemes, that we want to implement as basement of our neighbor discovery process of our application. Further, we compare the analytical results, of [20], for the different algorithms, with

focus on discovery latency and energy consumption. We know, each of the chosen algorithms fulfills the rotation closure property, see [20] for the proofs and further details. We shortly describe how they work and what conditions are necessary, for example their input parameters. Each algorithm is based on slots with a global slot duration time. Each node uses a slot counter which counts the slots to determine the state of the current slot.

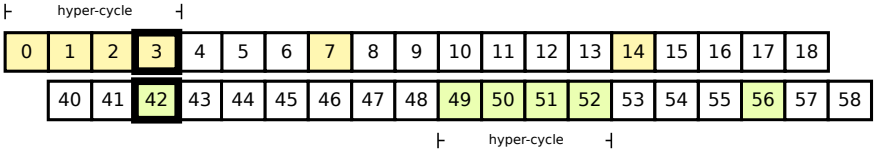
3.3.1 Disco

As presented in [8], by Dutta and Culler, the Disco algorithm is initialized by choosing a pair of two different prime numbers on each node. Each node performs an active slot at each multiple of the first and at each multiple of the second prime number. However, It is allowed to choose for both nodes the same pair of prime numbers (p_1, p_2). In that case, Dutta and Culler denoted, with high probability, the theoretical worst-case neighbor discovery latency with $p_1 * p_2 * slottime$. The discovery proof of Disco is based on the Chinese Remainder Theorem, which is satisfied if at least one pair of 2 primes of distinct nodes are relatively prime, [8]. We illustrate the Disco algorithm for two nodes which chose the same prime pair (3,7):



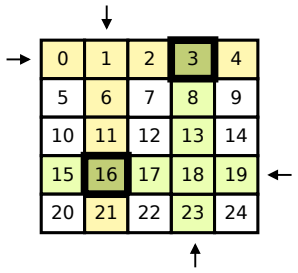
3.3.2 U-Connect

U-Connect, presented in [16], is also based on the Chinese Remainder Theorem. Each node choses a prime number, nodes are active at each multiple of that prime. In a special case, intersection between two nodes, choosing the same prime number, can not be guaranteed. Therefore each node performs a hyper-cycle at the square of the chosen prime, that means it sends out beacons for the half of the prime, rounded to the nearest higher integer, slots. Considering the Chinese Reminder Theorem, discovery between two nodes i and j are then guaranteed within $p_i * p_j * slottime$, and then that is the worst-case latency time. The hyper-cycle slots are not used as a listening phase, only beacons are injected. We note, that in our thesis, up to our later introduced implementation, we only differ two states. Because of that, we implement the hyper-cycle slots also as active slots, consequently, beacons can be captured within the hyper-cycle and the overall power consumption is higher as originally intended. The following illustration shows the U-Connect algorithm for two nodes, each with a chosen prime of 7:



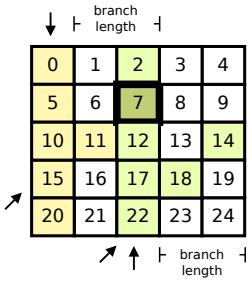
3.3.3 Grid

As the name indicates, the Grid algorithm is based on slots arranged as a rectangle. The idea was introduced by Maekawa [18], who originally worked on mutual exclusions. The constructed rectangle has a width of w and a height of h slots and is globally fixed for all nodes. One period starts at the upper left slot, walks through a row and continues with the first left slot of the next row. The period length of the algorithm logically then is $w * h$. Each node choses exactly one row and one column of the rectangle. Only each slot of the chosen row and the chosen column is used as an active slot. Different nodes, using the same rectangle width and height, have at least two intersections. The Grid algorithm gives optimal performance, considering power consumption and discovery latency, if the rectangle is a square with $w = h$. In general, the worst-case discovery latency is denoted with $w * h$.



3.3.4 Torus

The Torus algorithm [17] adopts the basic idea of the Grid algorithm, but improves the size of the quorum, that depends directly on the width and height of the chosen grid. The idea is to take a torus and separate its surface into multiple well ordered cells, such that the planar rolled out surface of the separated torus forms a grid of a distinct width and height, like the Grid algorithm. Each cell indicates one time slot and each node uses the same separated torus. As active slots, a node choses exactly one column of the planar grid, which denotes a closed ring on the torus. That alone does not guarantee an intersection, therefor, each node additionally denotes a branch of the ring with the length of half of the width of the planar grid, rounded to the nearest lower integer. That branch is arranged diagonal and docked somewhere directly to the left to the ring elements. Like the Grid algorithm, the Torus algorithm performs optimal if the width and height of the chosen grid forms a square. The worst-case discovery latency results in $w * h$, like the Grid algorithm does.



4

Design

In this chapter we first define the basic functional and technical requirements of our implementation. Further, we introduce solutions with respect to the requirements and for problematic cases, which our subject raises. For example, we want to stay indepent from specific vendors, implement an energy efficient neighbor discovery and prevent polling requests. Our podcast transmission has to support transmission continuation and limit simultaneous transmissions. Scalability is an important issue and we want to have independence from a graphical user interface. We response to the beacon loss problem, security and privacy aspects. We close this chapter by presenting the basic entities of our application, the slot behaviour and communication protocol examples.

4.1 Requirements

As a basic requirement, the exchange of podcasts between different devices should proceed automatically. The user herself should be able to add podcasts to a public library and set up individual tags. Tags are used to describe the users interests. A user should be able to add multiple unbounded tags. Every communication process should proceed automatically without human interaction. Further, it is advisable to design the implementation energy efficient. It is important to change the operation mode of the wireless hardware from a lower power mode to a high power mode only if it is necessary. Additionally, interaction between multiple devices should be scalable.

4.1.1 Independence of Wireless Device Hardware

As far as possible, we want to stay indepent from specific vendors and the model of the wireless hardware. Our intention is to make our implementation processable in generic Linux environments. However, our energy efficient neighbor discovery

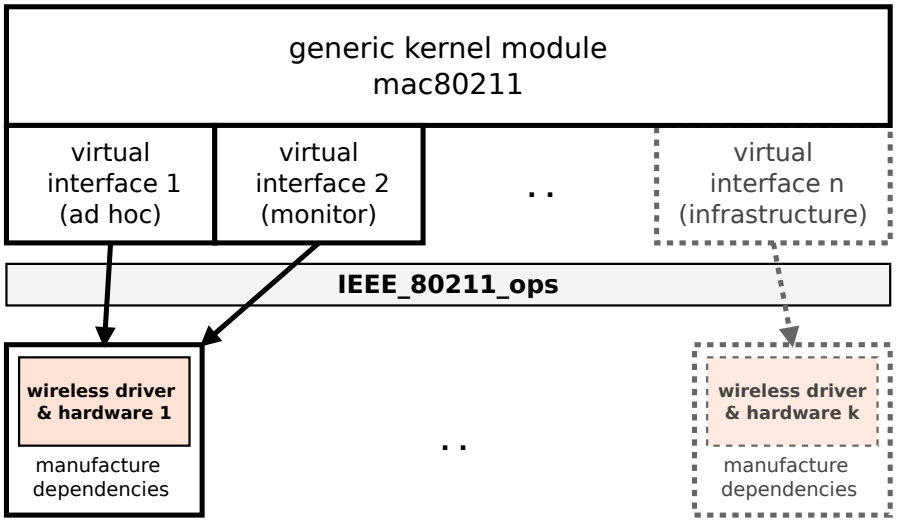


Figure 4.1 The generic mac80211 stack supports multiple virtual interfaces for a single wireless hardware device. Additionally it is possible to manage multiple hardware devices. We want to set up only 2 virtual interfaces on one single wireless hardware device. One interface is set to the ad hoc mode, the other to the monitor mode.

process makes it necessary to interact with the power control structures of the wireless hardware. These control structures are implemented in the individual hardware drivers.

Solution

We support the mac80211 generic Linux kernel module, which directly interacts with the individual wireless hardware driver, see Figure 4.1. The power control functions are located in the hardware drivers, but the mac80211 module communicates through predefined callback functions with the drivers. We call these generic functions *ieee80211_ops*, a documentation of that API and individual details are located here [4]. Additionally, this kernel module allows us to add multiple virtual network interfaces for a single wireless hardware device. These interfaces are able to operate on different modes simultaneously. It is possible to assign distinct ip addresses and routing tables. To set up an additional monitor interface, next to the already existing interface in ad hoc mode, we take advantage of the *airmon-ng* command from the aircrack-ng project, see [1]. We set the standard wireless interface into the ad hoc mode and assign a predefined BSSID. To exchange podcasts, this ad hoc interface is used to establish reliable TCP/IP connections to nearby devices.

4.1.2 Energy Efficient Neighbor Discovery

While energy efficient neighbor discovery is our main goal, we also want to achieve reliability. In the previous chapters we have introduced quorum systems and different algorithms which are based on quorum systems. We showed that such algorithms

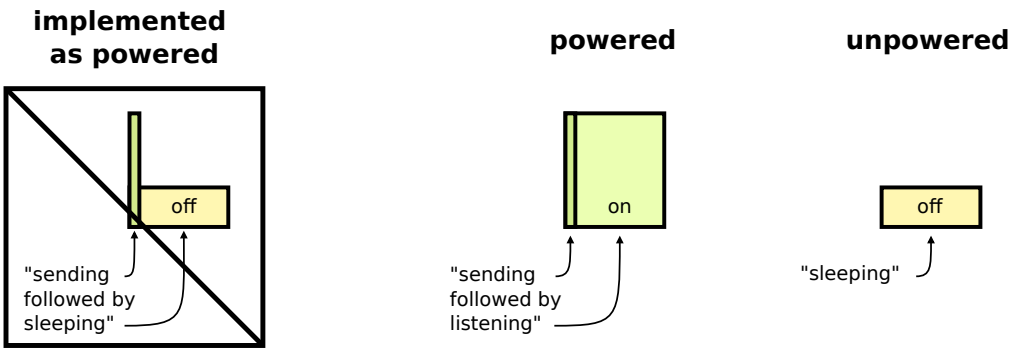


Figure 4.2 We implement two slots: powered and unpowered. Thus, because the "sending followed by listening" slot does not allow to receive a reply beacon frame, which possibly is send back with a delay within the rest of the slot time.

are feasible for energy efficient neighbor discovery, see section 2.6.2. Now we transfer this theoretical knowledge to a practical application by using a wireless local area (W-LAN) device as radio transmitter and receiver.

Solution

With respect to the original quorum algorithms we need to implement 3 basic slot states: "sending followed by listening", "sending followed by sleeping" and "sleeping". Our request process, described later in this subsection, is initiated by reply beacons. According to this approach a device must listen on the channel for reply beacons after a beacon was send out. Further, a switch of the wireless hardware transmitter, from a powered state to an unpowered, and the other way round, takes a specific period of time, described later in section 4.4. Thus, we decide to implement only 2 logical slot states: "sending followed by listening" and "sleeping", see Figure 4.3. In this thesis we named the two states *powered* and *unpowered*. To perform neighbor discovery, with respect to the quorum algorithm, a device has to send out beacons in each powered slot and listen to beacons during each unpowered slot. According to this, we add a virtual interface on each wireless device and set the operating mode to *monitor mode*. With an interface in monitor mode, it is possible to capture all wireless frames directly from the mac layer and inject individual frames to it. Each device injects a clearly recognisable beacon frame at the beginning of each powered slot. A beacon can be detected by filtering the BSSID address of the packet. We use a fixed private BSSID address for all injected packets which is AC:DE:48:88:88:88. Further technical information of the filter process are denoted in section 5.5.3. The beacon also includes the senders mac address. Other devices in wireless range, which performs a powered slot at the same time, are able to capture such beacons and determine the senders mac address out of the packet header. Each captured beacon results in a successful one sided discovery.

To save energy within the unpowered state, we make use of the RFkill functionality, see section 2.5, and switch off the wireless transmitter, but without an overall shut

down of the corresponding logical network interface to keep the settings of the additional ad hoc interface up. We analyzed how RFkill works and identified the main function which is responsible for the initial command to the wireless hardware to unpower the transmitter in the hardware drivers. Further we analyzed the interaction process with the mac80211 module. We located a related power control function in the source of the generic kernel module mac80211. These generic function falls back on the special hardware driver power control functions. By invoking this generic kernel space function directly, it is possible to switch off the wireless transmitter by not touching the settings of the assigned logical interfaces. This prevents a reset of assigned virtual interfaces, the respective IP addresses, the routing tables, and the operation settings. The logical network interface will still be available for applications. Already established TCP/IP connections will run into a standard protocol timeout and the power cut will not rise critical side effects on the upper network layers.

4.1.3 Prevent Polling Requests

We need to consider a mechanism that signals a device automatically when to connect to an already discovered device to synchronize possible new data from the respective other one. There are two possibilities that could arise which makes synchronization again necessary. A user adds a new podcast to the internal library or the device receives a new podcast from a third device in its range. Another device, that may be actually a longer period in range should be informed about such library changes. We want to prevent regularly polling of other nodes, because that consumes unnecessary energy and we need to find a reasonable period when to do this. We assume, the best way is to combine this functionality with the already used beacons. This neither requires an extra active slot nor an extra data frame. We want to perform the synchronization request event-based.

Solution

After discovery is performed and a device keeps the source mac address of another device in range, a device is able to send out a reply beacon frame to that discovered device, directly after the captured beacon. The structure of our beacon and the corresponding reply beacon is describes in section 4.5.1. A reply beacon signalizes a TCP/IP request and both involved devices, the sender and the receiver of a reply beacon, have to keep the wireless device powered. We call this state of a device *force_power_on* mode. The sender of a reply beacon tries to establish a TCP/IP connection after the reply beacon is send out, at the beginning of the force_power_on mode, to further initiate a file list request. After a file list request, and possible further file requests and corresponding possible file transmissions during the force_power_on mode, both devices leave the force_power_on mode. If the TCP/IP connection or transmission times out, according to a predefined timeout value, the force_power_on mode is also left. The change for both devices to the force_power_on mode, to perform a file list request, makes sense if the destination device is discovered the first time. But if that is done after each beacon, both nodes won't ever leave the force_power_on mode. After the first discovery of a device we

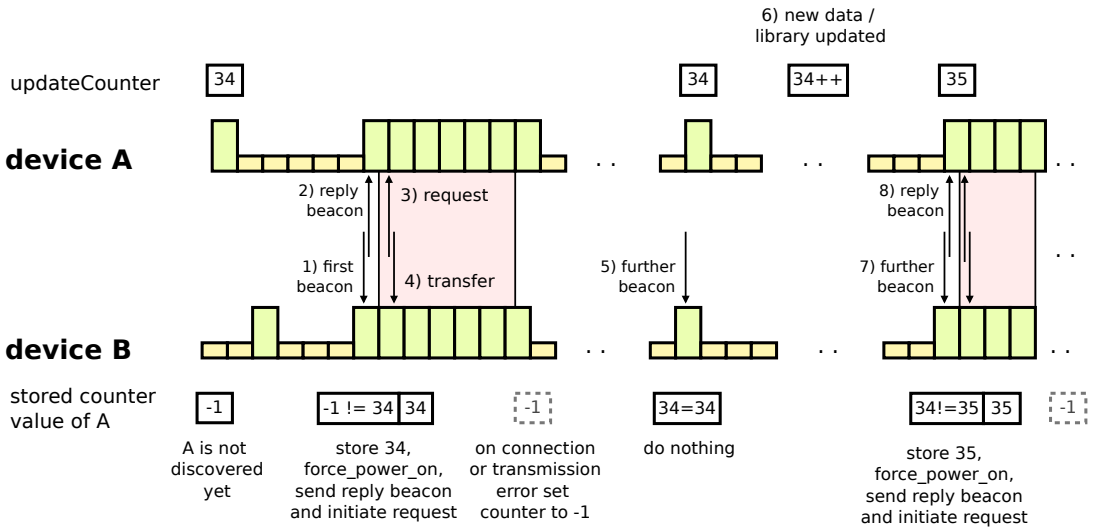


Figure 4.3 Illustration of how our updateCounter works. As example, this value of device A is set to 34. The updateCounter value is transmitted within each beacon. According to this value and the possibly already stored updateCounter value of the corresponding other device, it is possible to decide if a switch to the force_power_on mode, to perform further podcast transmissions, is necessary.

need a mechanism that indicates if it makes sense to send a beacon reply and switch both nodes into the force_power_on mode or not. Our solution is illustrated in Figure 4.3. Each beacon includes an integer value, called *updateCounter*, which serves as indicator. Every time a device receives new podcasts from other devices or adds new files to the internal podcast library, it increases its own indicator value and sends this new value within every further beacon. Nodes which receive beacons of other nodes keep the denoted indicator integer value of the discovered devices with respect to the senders mac address. Due to this locally stored indicator values it is possible to determine on a captured beacon if an already discovered neighbor has changed its library. Only if this is the case, the new updateCounter value is stored, a reply beacon is send out and both nodes switch to the force_power_on mode to communicate over TCP/IP. If the corresponding TCP/IP connection or transmission fails, the initiator of the transmission, the device which has sent out the request, sets the locally stored updateCounter value of the corresponding other device to -1. Thus, the initiator will perform another file list request when the next beacon of the corresponding other node is received. That makes it also possible to continue the before interrupted transmission, according to the already transmitted data.

4.1.4 Tag Based Podcast Request

The user should be able to add podcasts to a public library. A categorization and a short description of these podcasts should also be supported. Further a user has to define a list of keywords accordingly to his own interests. These keywords should be used to identify suitable podcasts stored on other devices.

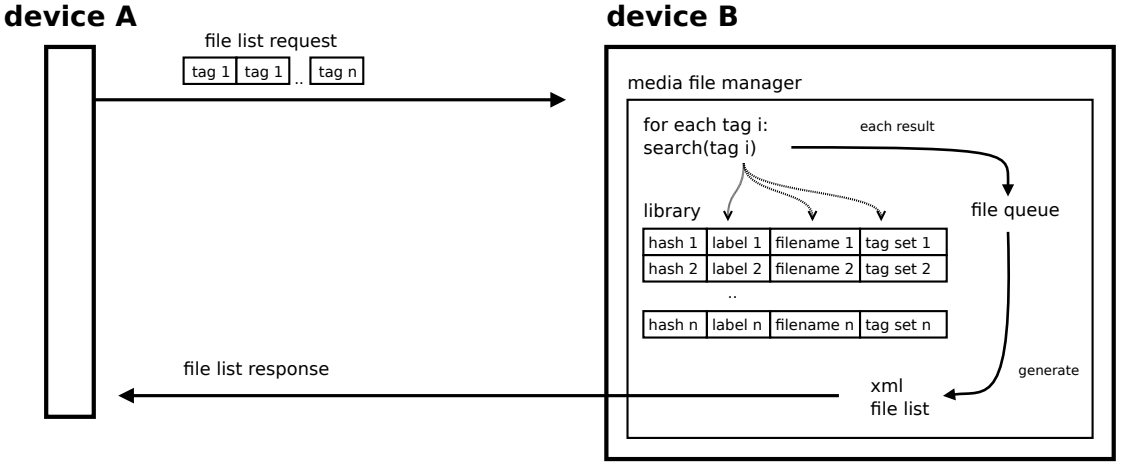


Figure 4.4 A file list request is performed using TCP/IP, by transmitting the user's previously defined tags, which describe the interests of the user. The corresponding other device search it's local podcast library, according these tags, and generates a request structured as XML document.

Solution

Technically, for our applications, a podcast is represented by a file. The user is able to add and remove media files of the internal library of our application. To describe and categorize a podcast file the user has the possibility to bind to each added file one podcast label and multiple tags. These additional informations are necessary to identify a podcast accordingly to a keyword based search. Other devices are able to release a file list request over TCP/IP, which is based on keywords. A file list request is initiated from other devices after a reply beacon is received and both concerned devices have switched to the force_power_on mode. But this will only be the case if before a beacon was send out, which includes an increased indicator value of the local library. A file list request transmits at least one keyword, a set of multiple keywords is also supported. After a file list request is received, a device scans the local file library for matches and returns every necessary information of each file back to the sender of the request, see Figure 4.4. Each file in this file list is identifiable by a primary ID. The primary ID should be fixed on distributed devices, hence, it can be calculated out of the binary content of a file by using a secure hash function. After receiving a file list a device searches for differences between this list and the local podcast library. If in the received file list verified podcasts are locally stored partially, due to a previous transmission interruption, or locally not located at all, a file request over TCP/IP is initiated. The internal library's file list keeps the following values for each managed file: *hash* (SHA1), *label*, *filename*, *tag set*, *filesize* and *bytes* (stored bytes). The structure of a file list request and response is presented later in section 4.5.2.

4.1.5 Podcast Transmission and Retransmission

Our implementation should enable the functionality to initiate a podcast file transmission by denoting a globally computable and distributed primary file identifier.

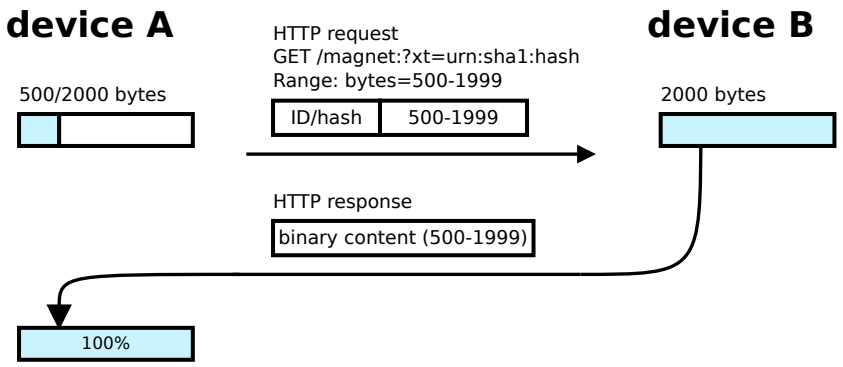


Figure 4.5 A file request is performed using HTTP over TCP/IP, by transmitting the desired file's hash and byte range. The file's secure hash value and the corresponding maximum amount of stored data on device B is already known from the previously transmitted file list.

We develop an application for devices in mobile environments. Thus, a connection to a centralized identification directory may not be possible. During a connection session, a file transmission could get interrupted as two communicating devices may move out of range. It could then be possible that these devices later discovery each other again, then the transmission should continue. As another possibility, a third device, which already stores the interrupted podcast file itself, could move closer, enter the wireless range and file transmission continuation may be possible.

Solution

As already described, a podcast file transmission is invoked within the force_power_on mode by denoting a primary file identifier, which is computed by applying a secure hash function on the content of a podcast file. The identifier, the stored file size and the exact original file size is transmitted within the content of a respond of a file list request. Corresponding structures of requests and responses are denoted in section 4.5.2 and section 4.5.3. The file list, the file request itself and the transmission of the binary file data is performed using a TCP/IP connection. Requests are initiated as HTTP file requests. According to the HTTP request header, a file request supports a range field to denote an individual requested part of a file in bytes. Hence, it is possible to control the file transmission according to still missing content of a file after a transmission interruption, illustrated in figure 4.5. The requested filename itself is denoted by it's secure hash value within the HTTP request header as a magnet-link URI, shown in section 4.5.3. The requested device does not necessarily have to store the full file's content already, to serve a corresponding incoming request. However, it is possible to request a file's content by a denoted range's upper limit value according to the corresponding *bytes* value transmitted within a previously received file list response.

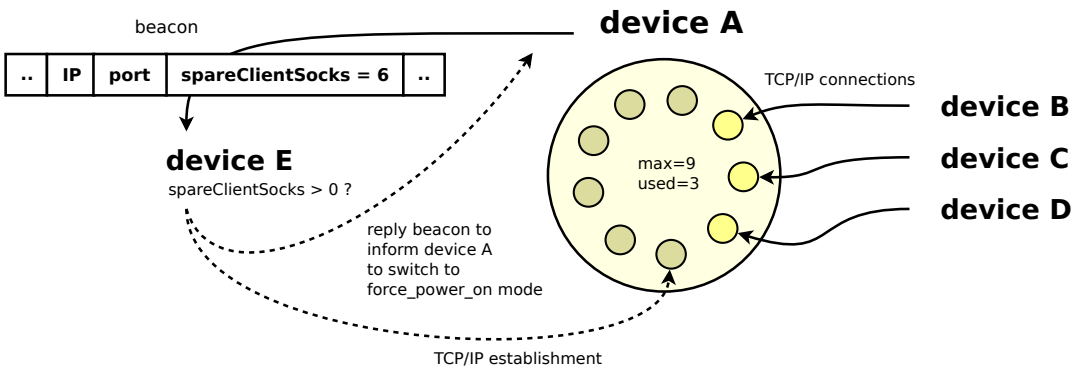


Figure 4.6 Illustration of *spareCLientSocks* usage. Device A is configured to serve a maximum number of 9 client connections. Three client connections are currently processed within threads. Accordingly, device A denotes six spare client sockets in it's beacon. Device E captures a beacon from A, switches itself to the force_power_on mode, sends a reply beacon to device A to inform it to switch to the force_power_on mode (eventually device A may already perform the force_power_on mode) and tries to establish a TCP/IP connection.

4.1.6 Scalability and Limitation of Simultaneous Transmissions

The number of simultaneous devices in range can not be predetermined. Our solution has to be scalable and it must be optimized to handle multiple possible cases due to the variable number of devices. Client requests should be served simultaneously, but limited to a maximum number of connections.

Solution

We integrate a neighbor manager that handles any necessary information of multiple devices in range. The information of devices that are discovered can be added to an internal dynamic neighbor list. Thus, the number of neighbors within that list are not previously limited to a specific value. Without capturing a beacon of an individual device, within a reasonable period of time, the device is automatically removed from the neighbor list. Outgoing transmission requests and incoming file transfers are limited to only one individual destination per time. Multiple simultaneous incoming request of other devices and outgoing file transmissions can be processed up to a predefined fixed limit. Each accepted client request is served within a new created thread. Any sent beacon includes an additional information if the respective device has spare capacities, an idle TCP/IP socket thread, or not. This value is calculated as the difference of a predefined maximum limit and the current established connections. We call this value *spareClientSocks*, according to our beacon structure denoted in section 4.5.1. Before a device sends an outgoing request to another device, initiated by a reply beacon after a received beacon, it checks this spareClientSocks value of the corresponding captured beacon frame. If the respective other device does not denote free capacities, the reply beacon will not

be send out, also no HTTP request. If this is the case and a device nevertheless tries to establish a TCP/IP connection, the request will not be served and the connection establishment will timeout at the initiator. However, the final decision to switch to the force_power_on mode and accept TCP/IP connections is ruled by any device itself.

4.1.7 Independence of Graphical User Interface

The user of a device should be able to interact with our application. During operation, the user should have access to the internal podcast library. It must be possible to add new podcasts to the public library to offer them to other devices, remove already existing ones and visualize a list of the currently fully or partially stored podcasts. Furthermore, a visualization of the internal neighbor list should be supported.

Solution

We implement an inter-process communication to keep the graphical user interface independent and interchangeable, as illustrated in Figure 4.7. Therefore we implement a Unix domain socket to make it possible to exchange data between our application daemon process and an independent graphical user interface process. Further we define an individual API (application interface) to support interaction with third party applications. The graphical user interface itself is not our task, but we give an example of a python code how the establish an inter-process communication and run commands by using our application's domain socket. The corresponding commands are denoted in sections 4.5.4.

4.1.8 Independence of Power Toggle and Basic Application

Our general power toggle function, used to switch a wireless device off and on without taking the logical assigned interfaces down and later up again, should be designed as a stand-alone module independently of our podcast sharing application. With a look ahead, it should be possible to control our switch functionality out of other applications as our neighbor discovery and podcast sharing implementation.

Solution

To realize our wireless hardware power toggle, we implement an independently loadable Linux kernel module. This module directly interacts with the mac80211 kernel module to switch the hardware off and on. Further, on load time, it registers a Linux character device which is accessible from the user-space. The character device is located in the /dev directory and we named it LiND_pwr. A user-space application is able to control the power toggle by writing two single commands to the character device.

4.2 General Issues

We want to face a set of general issues related to our topic. We describe the problem of losing beacons and present a solution. Security, privacy and identity protection is not a main topic within our thesis, nevertheless it is basically important, thus we want to design our implementation to protect identity. Also not a main topic is how to handle undesirable content with respect to podcasts, however we describe the existence of that issue.

4.2.1 Beacon Loss Handling

Our application injects beacon and beacon reply frames directly to the shared medium as broadcast packets. Due to this, an explicit acknowledgement packet will not be send out if another device receives a beacon. Our injection method does not support reliable confirmation. An injected frame could get lost on the wireless channel and our application has to be fault-tolerant with respect to such cases. We paid attention to this and set our application back to the neighbor discovery mode if a scheduled expected event doesn't rise up within a predefined timeout. For example, after sending out a reply beacon, the sender itself and the intended receiver both have to switch to the force_power_on mode and establish a TCP/IP connection. If the reply beacon gets lost or the intended receiver has moved out of range during a TCP/IP connection, the communication will be reseted after a timeout. A device only leaves the force_power_on mode, if no simultaneous third communication with another device is executed. However, considering a change of the status of the force_power_on mode, our application has to take care of multiple active connections. To counteract beacon and reply beacon loss and to keep our discovery process reliable, we inject three beacons successively at the beginning of each slot and respectively send three reply beacons if replying is proceed. We determined this value out of the beacon error rate analysis, evaluated in chapter 6 of this thesis.

4.2.2 Security, Privacy and Identity Protection

Since our application's intention is to provide podcasts without copyright protection to the public, for now, data protection is of minor interest to us. However, if privacy is required, it is a simple task to later add encryption support for the interests and file transmissions globally across all participants or in addition between any two endpoints. For a closed user group, the network itself can be secured by adding a standard encryption to the ad hoc interface. To realize endpoint-to-endpoint security for each TCP/IP connection, it is possible to implement a transport layer security protocol (TLS/SSL). But to provide authentication, individual keys for each possible communication partner have to be exchanged first. This is possible in a closed group, but our intention is to share files with, possible unknown, participants. Due to this key-exchange problem and the fact, that we want to share publicly accessible podcasts, we benevolently omit encryption and authentication. However, considering transmission errors, data integrity primarily is fulfilled by using the connection oriented TCP/IP protocol and further by comparing a file contents' hash value after transmission. It is possible to implement identity protection by assigning a random

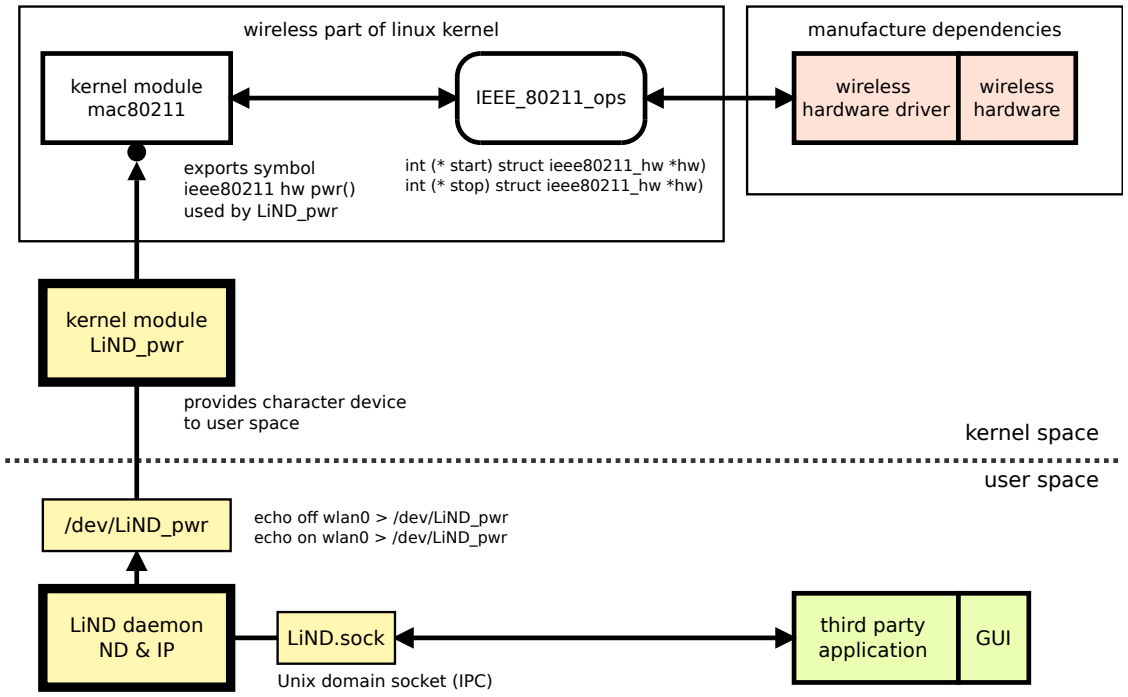


Figure 4.7 Interaction of the basic modules. Our LiND_pwr module uses exported symbols of the mac80211 kernel module and it offers a character device to user space applications. Our LiND daemon writes to the character device to initiate a power switch. Our LiND daemon offers a domain socket for third party applications, for example for a GUI.

mac address, a random IP address and a random port number used by the application's server socket, at each start time. It can be possible to change these settings during the runtime, if no neighbor is in range for a predefined period of time. Furthermore, we do not take care of any kind of potential security or flooding attacks, for example, with respect to our predefined maximum limit of client connections per device, it is easily possible to perform denial of service attacks.

4.2.3 Spam, Undesirable Content and Abuse

We know that the problem with spam, abuse and undesirable content can arise wherever public accessible information services are offered. With respect to our solution, we would like to mention that this might also be a problem. This is, however, no focus in this thesis, but may be content for future research.

4.3 Basic Modules

Our solution consists of two different parts. The first part is our LiND_pwr loadable kernel module, the other one is our LiND daemon that runs in user space. It is

necessary to add one function the the generic mac80211 kernel module, recompile and reload it into the kernel. We keep the already existing individual network drivers untouched. A third party application is necessary to interact with a user. All entities are illustrated in Figure 4.7.

4.3.1 LiND_pwr kernel module

The LiND_pwr module registers a character device /dev/LiND_pwr and processes commands that can be written to /dev/LiND_pwr. As described in the implementation, see section 5.4 , two different commands can be performed to control the power of a registered wireless device's radio transmitter, "on" and "off". We implement the LiND_pwr module to keep the number of necessary changes in the mac80211 module low. Due to that we added to the mac80211 module only one function that matches a denoted net device to an inner instance of the *ieee80211_sub_if_data* structure to further get access to the corresponding *ieee_80211_ops*, "start" and "stop". It is possible to use our LiND_pwr toggle in combination with other applications than our LiND daemon. Also it is possible to release the switch directly from the shell, the following example will switch the wireless transmitter of the registered network device "wlan0" off and on:

```
echo off wlan0 >/dev/LiND_pwr
echo on wlan0 >/dev/LiND_pwr
```

4.3.2 LiND daemon

Our LiND daemon processes the neighbor discovery algorithm and writes corresponding commands "on" and "off", due to the computed slot status, to the character device /dev/LiND_dev. The process offers a domain socket LiND.sock to accept a predefined list of commands from other applications and to send corresponding responses back, see Figure 4.7. It sets up the network interface settings, like the BSSID for the ad hoc interface, the common communication channel frequency, the mac and the IP address. It integrates a neighbor manager, a file manager and a socket manager. The neighbor manager handles a list of discovered neighbors in range. The file manager represents a set of functions to add, search and remove files. It manages an internal file list, the public accessible podcast library. The socket manager handles the procedures that are necessary to initiate a file list transfer or a file transfer. This manager sets up the TCP/IP server socket, accepts the client connections and organizes the corresponding threads. For a file list or a file request we use HTTP. A HTTP request also supports data range specific file requests. We use this option for file continuation requests. The socket manager also controls the times when to enter and when to leave the force_power_on operation mode. It is possible to handle within one LiND daemon process multiple virtual device nodes, each node instance then owns it's own managers and communications. We implemented that functionality for testing purposes. However, the final version of our implementation creates only one node instance per running daemon process. Additionally, our daemon supports a logging functionality. It is possible to write individual information to an external log file, we implemented that feature to debug our application.

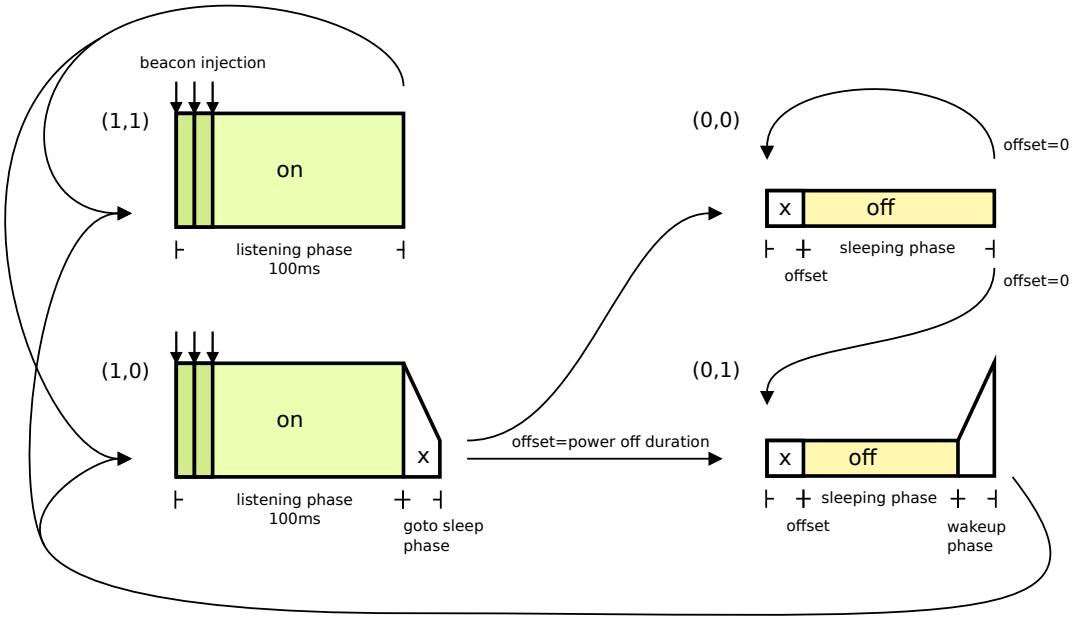


Figure 4.8 Slot behaviour with respect to the "wake up" and "goto sleep" duration, respectively δ_+ and δ_-. The power on command must be initiated δ_+ before the active slot begins. The power off command must be initiated directly after an active slot with the beginning of a sleep slot but overlays with the sleeping slot during δ_-.

4.4 Slot Behaviour

To perform asynchronous reliable neighbor discovery by using quorum systems, it is necessary to keep a fixed slot time δ_{slot} for each slot. The active duration must cover the full active slot period δ_{slot}. Our analysis of our LiND_pwr toggle delay, in chapter 6, shows, that our wireless device takes time δ_+ for one switch operation "on" and δ_- for one switch operation "off". According to that periods we have to adjust the sleeping time in two cases. If a sleeping slot follows after an active slot, the wireless transmitter has to be switched off. This takes the time δ_-. The adjusted sleep time then is $\delta_{slot} - \delta_-$, but only if the next slot is also an inactive slot. The other case, if the next slot after the inactive slot is an active slot again, we have to decrease the sleep period further by time δ_+. All possible cases are denoted in Figure 4.8, which illustrates our slot behaviour implementation.

4.5 Communication Data Structures

Communication between devices requires predefined data structures. Our beacon and our reply beacon frame is directly injected to the physical layer. Thus, we have to construct a frame that includes the necessary header structures like the *radio tap* header and the *ieee80211* header. Any following communication uses TCP/IP and HTTP.

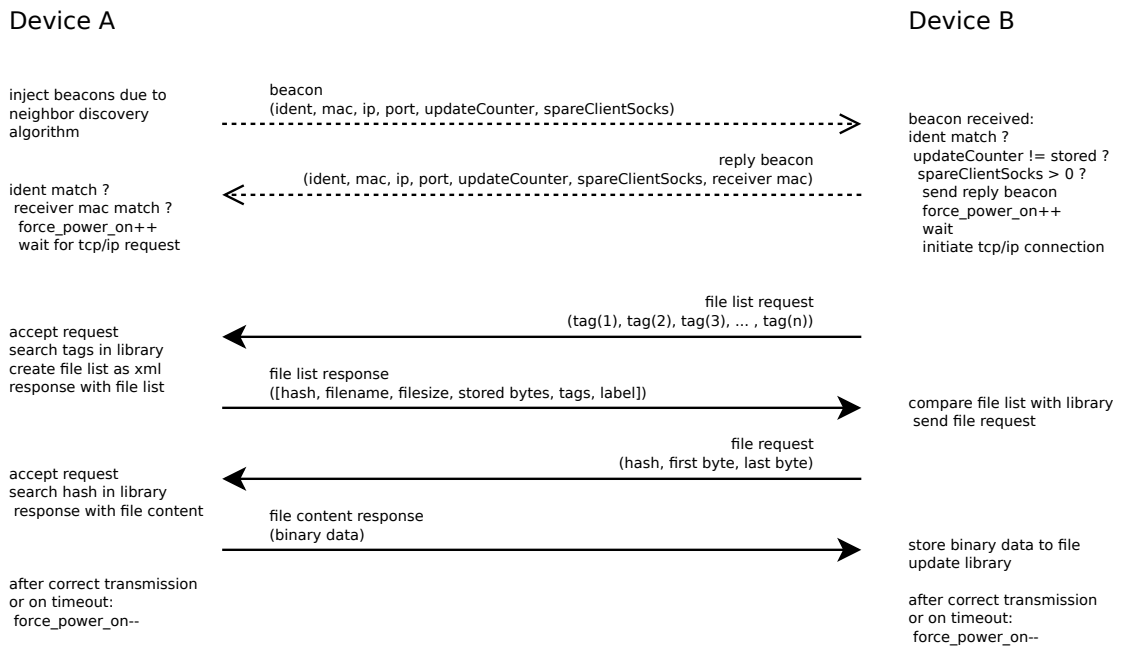


Figure 4.9 The communication between two devices is illustrated. A device injects beacons to perform neighbor discovery. A reply beacon initiates the force_power_on mode on both devices. Further the file list and file transmission is proceeded by using TCP/IP.

4.5.1 Beacon and Reply Beacon

Considering our neighbor discovery process,we send out beacon frames. The structure of a beacon frame is denoted in Table 4.1. We support different logical groups, a device does only process beacons after comparing the group string of the length identLen. This group string is placed at offset byte 0 in the payload field and defines the group membership. A reply beacon has the same structure but additionally includes the intended receiver mac address in the payload field, directly after the group string at offset byte identLen.

RadioTapHeader	18 Bytes	struct	version, data rate, frequency channel, antenna, ..
Ieee80211Header	24 Bytes	struct	destination mac, source mac, cell id, ..
LLCFrame	4 Bytes	struct	logical link control
identLen	1 Byte	uint8	group identification length (in payload)
packetType	1 Byte	uint8	packet type: 0=beacon, 1=reply beacon
IP	4 Bytes	uint32	assigned ad hoc ip address
port	2 Bytes	uint16	port where server socket is listening
updateCounter	2 Bytes	uint16	indicator for library updates
spareClientSocks	2 Bytes	uint16	free client sockets for TCP/IP communication
payload	x Bytes	uchar	individual chars and x00

Table 4.1 Structure of a beacon and a reply beacon frame.

4.5.2 File List Request and Response

A file list request is a generic http request that includes a magnet-link. We use the magnet-link scheme, because we reference files by content and not by location. Magnet-links support a set of parameter, for our file list request we implement the parameter ”kt“, which means ”keyword topic. A file list HTTP request has the following structure:

```
GET /magnet:?kt=tag1+tag2+tag3 HTTP/1.1
Host: ip:port
Accept */*
```

After a file list request is received, the internal library is searched for podcasts that match to the given tags. All matches are returned by the following XML response:

```
<?xml version="1.0"?>
<LiND_filelist>
 <file hash="" filename="" label="" tags="" filesize="" bytes="">hash</file>
 <file hash="" filename="" label="" tags="" filesize="" bytes="">hash</file>
 ..
</LiND_filelist>
```

The hash value for a file is computed over the original file content by using SHA1 (secure hash algorithm). The file size attribute denotes the original file size and the bytes attribute the partial stored bytes of the file.

4.5.3 File Request and Response

After receiving a file list response, a device compares the listed files, identified by the hash value, to the files which are currently stored in the local podcast library. If the file list contains files that are not already in the local library, a file request is directly initiated. If a locally stored partial file is denoted in the list, but the local bytes value is lower than the transmitted bytes attribute in the file list, a file requests and transmission is initiated, too. First one file requests is performed. After receiving a file list response, correspondingly necessary file transmissions are performed one after another. Thus, a single device does receive the binary contents of only one file at a time. A file list request has the following structure, the “xt” parameter stands for “exact topic”:

```
GET /magnet:?xt=urn:sha1:hash HTTP/1.1
Range: bytes=start-stop
Host: ip:port
Accept */*
```

If a file doesn’t exist locally, the start value is defined as 0 and the stop value is the bytes attribute value, denoted for the desired file in the file list, minus 1. If a file does partly exist locally, then the start value is the locally stored bytes value and

the stop value is also the transmitted bytes value minus 1. In this last case, the file request is only initiated if the start value is lower than the stop value, otherwise the respective other device does not have additional file content of this file.

After receiving a file request, a device searches the denoted hash value of the desired file in the local podcast library. If the file exists, then the desired binary content from byte start up to byte stop is transfered as binary response.

4.5.4 Domain Socket and Control Interface

To communicate to our LiND daemon from another application's process, it is necessary to connect to our LiND's domain socket LiND.sock. Requests and responses are structured using XML. It is possible to add new podcast files by denoting the filename, a label for the podcast and multiple tags. Further, it is possible to list the current podcast library and remove files by denoting the corresponding hash value. If a file is removed within a currently active file transfer, the transmission process will be canceled. Additionally the interface supports commands for a neighbor list request and for a list of the currently active file transmissions. It is possible to set new tags that describe the users interests. Within each request and response structure, which uses tags, tags are separated by commas. A request is delivered to our application by writing it to the LiND.sock domain socket. It is possible to write one command, or multiple, over multiple lines. The command first is parsed and processed by writing an additional empty line (*newline*) after the structured request. If the written XML structure is invalid, or the request does not conform the definition, it is not processed. Our domain socket's inter-process communication requests are defined as follows:

```
<LiND_filemanager command="add">
 <file filename="" label="" tags="">
 ..
</LiND_filemanager>

<LiND_filemanager command="list" />

<LiND_filemanager command="remove">
 <file hash="" />
 ..
</LiND_filemanager>

<LiND_neighbormanager command="list" />

<LiND_socketmanager command="list" />

<LiND_node command="tags" tags="" />
```

According to the stored podcasts within the library, the file list command's response is structured as follows:

```
<?xml version="1.0"?>
<LiND_filelist>
 <file hash="" filename="" label="" tags="" filesize="" bytes="" />
 ..
</LiND_filelist>
```

The *<LiND_neighbormanager command="list" />* response has the following structure:

```
<?xml version="1.0"?>
<LiND_neighborlist>
 <neighbor mac="" ip="" port="" spareClientSocks="" updateCounter="" />
 ..
</LiND_neighborlist>
```

A response of the *<LiND_socketmanager command="list" />* request, which lists the current processed file transmissions has the following structure:

```
<?xml version="1.0"?>
<LiND_socketlist>
 <socket file="hash" range_from="" range_to="" bytes="" direction="" />
 ..
</LiND_socketlist>
```

The attribute *bytes* denotes the number of currently already transmitted bytes of the denoted range and starts counting from zero before the first byte *range_from* is transmitted. The attribute *direction* is set to "in" or "out", according to the file transfer's direction.

5

Implementation

This chapter starts with a description of how we started our work. We describe how we analyzed the wireless device's behaviour and get familiar with the driver's and the kernel's source code. We show two callbacks of the IEEE80211 callback operations, which are necessary for our implementation, and describe why we use them. We present our patched version of the mac80211 kernel module and describe in detail how our patch works. Furthermore, we present the key features of our LiND_pwr kernel module and centralized parts of our LiND daemon application, for example how we organized threads and how we implemented the *force_power_on* mode. We finish that chapter by explaining why our implementation does not run on Android, as of now.

5.1 Kernel Source Analyzing and Logging

We invested work on researching the behaviour and structure of the hardware drivers and the mac80211 module. We used the underlying API documentations [5], explored parts of the public accessible Linux kernel source code and the wireless device drivers. Thereby we added log functions into suspected kernel functions and into functions of the device drivers. After every source code modification we had to compile the kernel again and to reboot the system to load the modified kernel. Our logging function wrote the kernel time and the name of the corresponding parent function. This was the function in which we have placed the log function in, to the kernel log file within the file system. We placed our log function at the beginning and at the end of suspected functions we wanted to observe. Additionally we activated the device drivers debugging functionality. As an example, the produced log data of the function *drv_start()* of the file *mac80211/driver-ops.h* is denoted while running the *rfkill unblock* command:

```
kernel: [25683.856450] === RUN: mac80211/driver-ops.h drv_start()
```

```
kernel: [25683.856457] === RUN: base.c ath5k_start() (no END)
kernel: [25683.856464] === RUN: base.c ath5k_init()
kernel: [25683.856470] === RUN: base.c ath5k_stop_locked()
kernel: [25683.856795] === RUN: phy.c ath5k_hw_phy_disable()
kernel: [25683.856801] === END: phy.c ath5k_hw_phy_disable()
kernel: [25683.856806] === END: base.c ath5k_stop_locked()
kernel: [25683.856812] === RUN: base.c ath5k_reset()
kernel: [25683.856824] === RUN: reset.c ath5k_hw_reset()
kernel: [25683.858775] === RUN: reset.c ath5k_hw_reg_write()
kernel: [25683.858782] === END: reset.c ath5k_hw_reg_write()
kernel: [25683.861258] === END: reset.c ath5k_hw_reset()
kernel: [25683.861424] === END: base.c ath5k_reset()
kernel: [25683.861430] === RUN: ath5k_rfkill_hw_start()
kernel: [25683.861436] === RUN: ath5k_rfkill_disable()
kernel: [25683.861444] === END: ath5k_rfkill_disable()
kernel: [25683.861449] === RUN: ath5k_rfkill_set_intr()
kernel: [25683.861460] === END: ath5k_rfkill_set_intr()
kernel: [25683.861487] === RUN: ath5k_rfkill_toggle()
kernel: [25683.861493] === RUN: ath5k_is_rfkill_set() (no END)
kernel: [25683.861501] === END: ath5k_rfkill_toggle()
kernel: [25683.861515] === END: ath5k_rfkill_hw_start()
kernel: [25683.862000] === END: base.c ath5k_init()
kernel: [25683.862006] === END: mac80211/driver-ops.h drv_start()
```

Our log results strongly support us in understanding how the kernel functions interact and how they are used with respect to their dependencies. These facts are not documented in APIs and a close look to the source is unavoidable. However, this is typical for most parts of the Linux kernel source. Finally we found out that we have to consider the functions *start()* and *stop()* of the structure *ieee80211_ops*, to control the power of a wireless device.

5.2 IEEE80211 Operations

The IEEE80211 operations are generic callbacks from the mac80211 module to the wireless hardware's device driver. The corresponding processes are implemented in the individual driver. Due to this, these callbacks are the last generic instances, considering different hardware devices. To switch the transmitter on and off, we used the following two predefined callbacks from the generic mac80211 module to the wireless hardware driver module of the *ieee80211_ops* structure:

```
struct ieee80211_ops {
  ..
  int (* start) (struct ieee80211_hw *hw);
  void (* stop) (struct ieee80211_hw *hw);
  ..
};
```

The API documentation, which can be found at [4], denotes, that these callback functions must be implemented by a wireless device driver. Therefore we can be

sure that those functions generically are supported by the wireless hardware. The start function turns the power of a denoted hardware device on and the stop function accordingly off. A pointer to the structure *ieee80211_hw*, which keeps the wireless hardware configuration, is used as parameter. That structure keeps the information which wireless device is affected. For example, *wiphy* points to the wireless hardware description and *priv* to the allocated drivers private area. *Ieee80211_hw* is structured as follows:

```
struct ieee80211_hw {
  struct ieee80211_conf conf;
  struct wiphy * wiphy;
  const char * rate_control_algorithm;
  void * priv;
  u32 flags;
  unsigned int extra_tx_headroom;
  int channel_change_time;
  int vif_data_size;
  int sta_data_size;
  int napi_weight;
  u16 queues;
  u16 max_listen_interval;
  s8 max_signal;
  u8 max_rates;
  u8 max_report_rates;
  u8 max_rate_tries;
  u8 max_rx_aggregation_subframes;
  u8 max_tx_aggregation_subframes;
};
```

5.3 Generic mac80211 Module

The first Linux wireless extensions were added to the Linux kernel in 1997. These extensions underlie a redesign and replacement process and today the supported

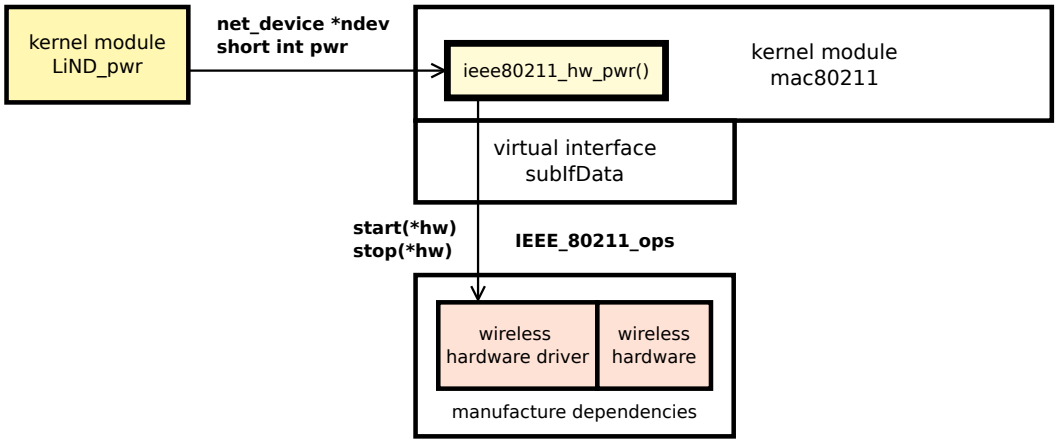


Figure 5.1 Our implemented function *ieee80211_hw_pwr()* in the mac80211 module. Interaction with LiND_pwr and *ieee80211_ops*.

generic Linux drivers are called Compat Wireless Drivers or Linux Device Drivers. The loadable kernel module mac80211 is part of these drivers. The presented callback functions are located in the kernel space, the mac80211 module controls these functions. These drivers are shipped with the generic Linux kernel. To work with the latest version we used the generic Compat Wireless subsystem source code from *http://wireless.kernel.org*, version *compat-wireless-2011-02-13*. We added one function to the source file *net/mac80211/main.c* of the generic mac80211 kernel module, named *ieee80211_hw_pwr()*. That function is additionally exported to allow accessing from our LiND_pwr module, see Figure 5.1. We compiled the patched source code against our kernel, we used Linux Ubuntu OS 10.10 and the kernel source of *linux-image-2.6.35-25-generic*. For successful compilation it was necessary to add our exported symbol function to the modules symbols list */lib/modules/2.6.35-25-generic/modules.symbols*:

```
alias symbol:ieee80211_hw_pwr mac80211
```

After compilation we unloaded the running mac80211 module and loaded our modified version into the kernel. The following code shows the function *ieee80211_hw_pwr()* we have implemented:

```
void ieee80211_hw_pwr(struct net_device *ndev, short int pwr) {
  struct ieee80211_sub_if_data *subIfData = IEEE80211_DEV_TO_SUB_IF(ndev);
  struct ieee80211_local *local = subIfData->local;
  if(pwr) local->ops->start(&local->hw);
  else local->ops->stop(&local->hw);
}
EXPORT_SYMBOL(ieee80211_hw_pwr);
```

The interaction of this functions is shown in Figure 5.1. The first parameter of our function *ieee80211_hw_pwr* is a pointer to the desired affected net device, the second parameter is a short integer value that represents the desired power state, 1 denoted "power on" and 0 "off". First we get the pointer **subIfData* to the data structure that represents the assigned virtual sub-interface. From this data structure we are able to get the pointer **local* to the structure *ieee80211_local*, which represents the corresponding wireless device. From this structure we are able to initiate the start and stop functions of the *ieee80211_ops* by accessing *local->ops*. The structure *ieee80211_hw* is a part of the *ieee80211_local* structure and represents the corresponding hardware. The memory address to that structure is then given by *&local->hw*. That is the necessary memory address we have to use as parameter for the *ieee80211_ops* start and stop functions, which we already introduced. The *EXPORT_SYMBOL()* method exports our implemented function globally to the kernel space. This makes it possible to use the function directly out of other kernel modules that are loaded into the same kernel, such as our LiND_pwr.

5.4 LiND_pwr Kernel Module

Our LiND_pwr kernel module is implemented in pure c code, because the Linux kernel itself is written in c code and we have to compile our module against the

kernel source. Our LiND_pwr module has a flat design structure and a constant memory assignment. Finally, experiences developers highly recommend to use pure c programming language for kernel and kernel module development. The source of LiND is organized in three c files, see Table 5.2. Additionally we add a makefile to build the module automatically against the kernel source.

LiND_pwr.c	definition of module init() and exit()
LiND_pwr_comm_dev.c	character device operations implementation
LiND_pwr_toggle_dev.c	device name processing and ieee80211_hw_pw() link

Table 5.1 Source files (#3) of our LiND_pwr kernel module.

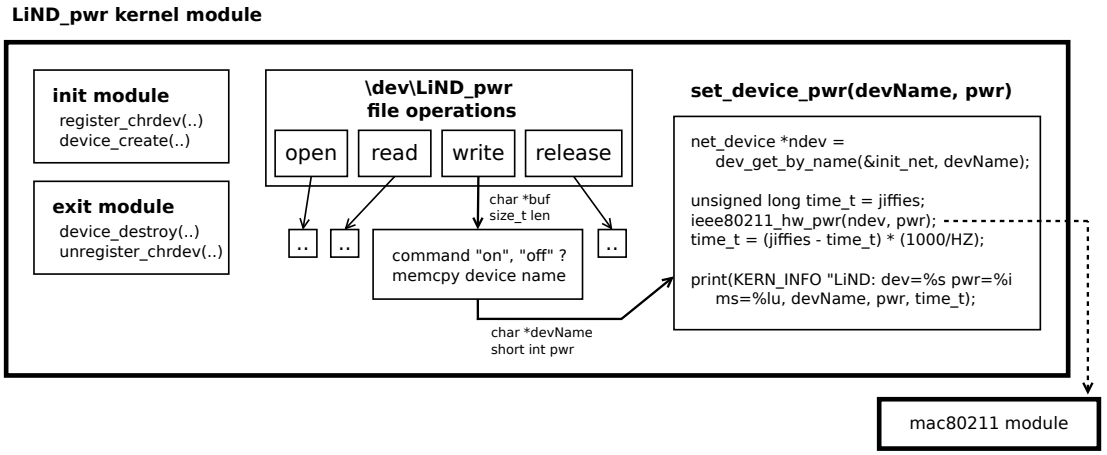


Figure 5.2 LiND_pwr module: basic operations init and exit on load and unload time, character device file operations and input processing. The function *ieee80211_hw_pwr* is implemented and exported in the mac80211 module, the dotted arrow at the right signalizes access to it.

The main processes within our LiND_pwr module are illustrated in Figure 5.2. On initialization, when the module is loaded into the kernel, it registers a character device. Therefore it is necessary to define a communication device name. We chose *LiND_pwr*. Four functions have to be implemented and linked to the communication device: *open*, *read*, *write* and *release*. These four file operations define the behaviour of the corresponding actions of the communication device. First we register our character device to the kernel by using the subordinate kernel function *register_chrdev()*, after that we create a corresponding inode structure for the file system by *device_create()*. When the module is unloaded we remove the corresponding inode structure from the file system by *device_destroy()* and unregister the device from the kernel by *unregister_chrdev()*. After initialization, the file /dev/LiND_pwr exists. If a user space application runs under a system's user that has the right to write to that file, the application can open the file, write bytes to it and eventually close it. Such a procedure affects the three linked functions *open*, *write* and *release*. We implemented the *write* function as follows: If bytes are written to the character device /dev/LiND_pwr, the function *write* within our module is called, and by two parameters, *const char *buf* and *size_t len*, the function gets the pointer to the buffer of the written bytes and the number of the corresponding byte length. Then we analyze these bytes with respect to the commands “on” and “off”. If a

command is identified, interpret the following bytes of the input as the name of a wireless net device. Then try to get a pointer to the corresponding device kernel structure *net_device* by using the Linux kernel function *dev_get_by_name()*. If the pointer can be found, pass this pointer and the command, which affects the power state, to the *ieee80211_hw_pwr(struct net_device *ndev, short int pwr)* exported from the mac80211 module. This module then initiates the corresponding ieee8022_ops callback function, *start()* or *stop()*, which are implemented in the corresponding hardware device driver. Figure 5.2 shows how, for debugging purposes, our implementation logs the corresponding duration in milliseconds of each switch operation to the standard kernel log output. For example, our character device can also be used directly from the command line:

```
echo off wlan0 >/dev/LiND_pwr
echo on wlan0 >/dev/LiND_pwr
```

5.5 LiND Daemon Application

Unlike the LiND_pwr kernel module, our LiND daemon is written in c++. We take advantage of the support of object-oriented programming and generally used the C++ Standard Template Library (STL) for designing complex containers. Threads are implemented by using the POSIX Threads Library (pthread). For our communication to the monitor device we used the Packet Capture Library (libpcap). We also used the OpenSSL Project Library, but only to compute Secure Hash Values (SHA). To perform HTTP requests, necessary for file list and file transfer initiation, we used the Curl Library (libcurl). To generate and parse file lists, which are structured as XML trees, we used the XML2 Library (libxml2). In this section, we provide an overview of our LiND daemon implementation and present central procedures and structures.

LiND.c	main, args, creates node instances
LiND_node.h/c	node class: controls, injection, capturing
LiND_neighbor_discovery_algorithm.h/c	quorum algorithms as node extensions
LiND_packet_header.c	radiotap-, IEEE80211-, LLCFrame-header
LiND_control_pwr.c	file operations, writes to /dev/LiND_pwr
LiND_domain_socket.c	creates and manages domain socket LiND.sock
LiND_functions.c	msleep, stringToLower, charToLower
LogFunctions.c	log functions for debugging
MediaFile.h/c	structure, methods and implementation
MediaFileManager.h/c	manages instances of class MediaFile
Neighbor.h/c	structure, methods and implementation
NeighborManager.h/c	manages instances of class Neighbor
keyMac.h	struct keyMac as key for neighbor map
ProtocolHeader.h/c	parse HTTP requests: byte range, magnet urn
MagnetLink.h/c	parse magnet urn: xt, kt
SocketManager.h/c	HTTP server, client sockets & threads, requests
WirelessDeviceOperations.h/c	setChannel, setMode, setEssid, setBitRate

Table 5.2 Source files (#27) of our LiND daemon.

Compilation of LiND:

```
c++ LiND.c -o LiND -lpthread -std=c++0x -lpcap -lssl -lxml2 -lcurl
                   -I/usr/include/curl -I/usr/include/libxml2
```

5.5.1 Runtime Arguments of LiND

The first process after running the application is to check the arguments. The first argument is a string that indicates the name of monitor device. The second argument is a string that indicates the name of the socket device. The third argument is a locally existing directory that defines the shared public directory. In this directory the received podcasts are stored. The fourth argument is a string of the tags, that should be used for the file list request, based on the users interest. Multiple tags are connected by commas, for example "tag1,tag2,tag3". During runtime, the user should later have the possibility to change these tags by using the domain socket interface from a GUI. The fifth parameter is an integer, that indicates the neighbor discovery algorithm: 1=*disco*, 2=*u-connect*, 3=*grid*, 4=*torus*. The sixth and the seventh argument are also integer values that denote the initialization parameters for the chosen neighbor discovery algorithm. For *disco* these arguments have to be 2 prime values, for *u-connect* only one prime value is necessary, for *grid* and *torus* these values denote the width and the height of the underlying rectangle. The advantage of setting these values as arguments makes it possible to run our evaluation process for each algorithm and each chosen initialization values, see section 6.1.4, from a prepared shell script.

After LiND is compiled and, according to the denoted arguments, correctly executed, one LiND_node instance is created. A mac address, which is used for the sender address field of each injected beacon frame, is randomly generated. The first 3 bytes of this mac address are fixed, according to the allowed private address range: 0xAC 0xDE 0x48. The last 3 bytes are assigned randomly on node creation. Furthermore, for TCP/IP connections, one IP address and one port number is randomly generated. The first octet of the IP address is fixed: 10. The following three octets are randomly assigned. A port, where the LiND daemon is listening on, is also randomly assigned.

5.5.2 Daemon and Thread Organization

After daemonizing, the running LiND daemon process has to handle multiple user threads. Figure 5.3 gives an overview of the different threads and their dependencies. The class *LiND_node_ND* extends the *LiND_node*. Here *ND* stands for the neighbor discovery algorithm, the exact classes are *LiND_node_Disco*, *LiND_node_UConnect*, *LiND_node_Grid* and *LiND_node_Torus*. For each neighbor discovery algorithm, the implementation of the function *run_loop_ND()* is specific within each of these classes. The corresponding individual initialization parameters for each algorithm are assigned by individual constructor methods of these four classes. For the chosen neighbor discovery algorithm, the computation of the respectively next slot behaviour is done within the thread *NDThread*, see Figure 4.8. It is necessary to compute the exact sleeping phase period for every slot according to the four different slot states. The capturing of packets is done by listening to the assigned monitor device of a LiND_node within the thread *captureThread*, see Figure 5.3.

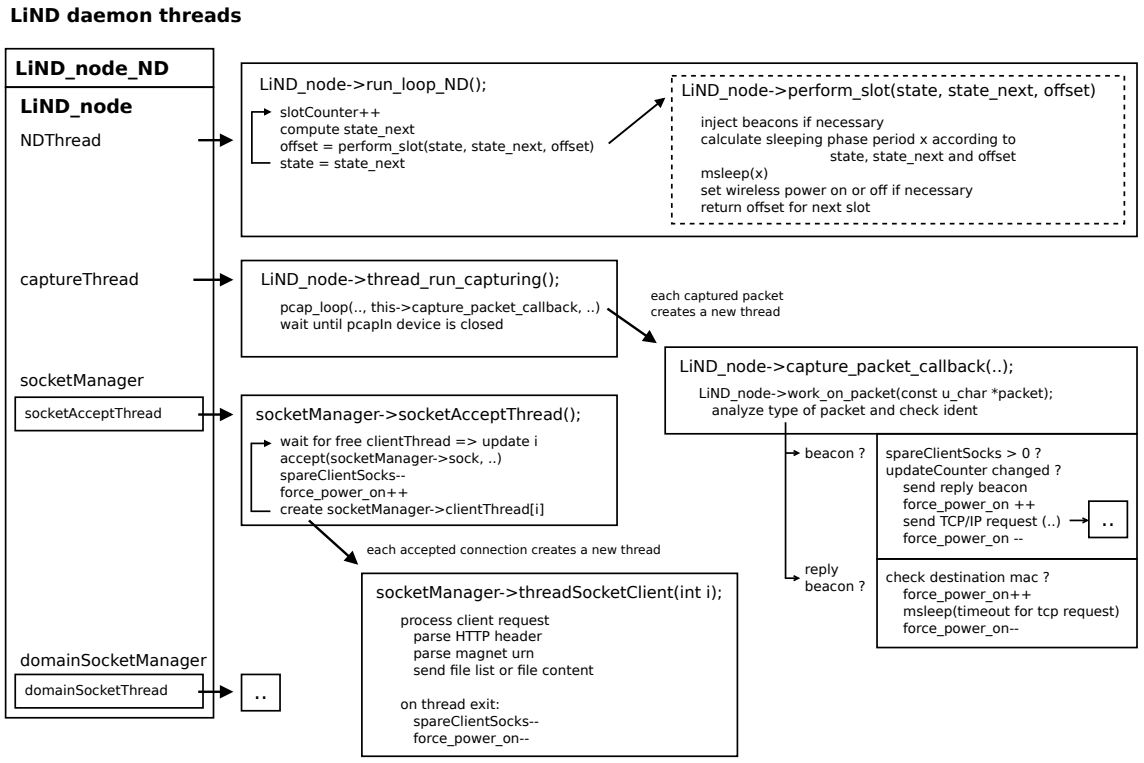


Figure 5.3 LiND daemon: User threads and their dependencies.

5.5.3 Beacon Injection and Capturing

We used the pcap library to inject packets to the monitor device and capture packets from it. To inject packets, we create a pcap device *pcapOut*. The parameters *outBuf* and *bufLen* denote the data of the packet we want to inject. The function *pcap_inject* returns the length of bytes that are injected:

```
len = pcap_inject(pcapOut, outBuf, bufLen);
```

As shown in Figure 5.3, the function *pcap_loop()* creates for each captured packet a new thread and gives a pointer to the packet's data and the byte length of the packet as an integer value by parameters to the thread. We want to ensure that only desired LiND_packets are passed to our application, otherwise every unwanted W-LAN packet would create a new thread and possibly overload the system. By using pcap, it is possible to compile a pcap filter from a filter string and copy the compiled filter to the wireless device. The monitor device then will only pass captured packets that match to the assigned filter. For each device we use the same fixed BSSID AC:DE:48:88:88:88 of the private mac address range. The private mac address range is defined by the first 3 bytes of a mac address from AC:DE:48 to AE:DE:48. The following code compiles and sets a filter to our pcap input device *pcapIn* according to our used BSSID:

```
struct bpf_program pcap_filter;
```

```
if (pcap_compile(pcapIn, &pcap_filter,
    "ether[16]=0xac && ether[17]=0xde && ether[18]=0x48 &&
     ether[19]=0x88 && ether[20]=0x88 && ether[21]=0x88", 0, 0) == -1)
 { if(debug==1) LOG("pcapIn: pcap compile filter error"); }
 else { if (pcap_setfilter(pcapIn, &pcap_filter) == -1)  {
         if(debug==1) LOG("pcapIn: pcap set filter error");
      }
 }
```

5.5.4 Force Power on Mode

A switch to the *force_power_on* mode can be initiated from different threads within our LiND daemon, see Figure 5.4. If a reply beacon is send out, a switch to the *force_power_on* mode is necessary to keep the wireless device in the active state. After sending the reply beacon our application tries to establish a TCP/IP connection. If a device captures a reply beacon, it also has to switch to the *force_power_on* mode to give the corresponding other device a chance to establish the TCP/IP connection. If a HTTP request is successfully processed, or interrupted, for whatever reasons, a device has to leave the *force_power_on*, but only if no other thread needs the *force_power_on* mode. For example, two simultaneous transmissions are running using different threads. If one of them is finished or interrupted, then the other transmissions has to block the *force_power_on* mode. We solved this problem by implementing a counter. If a method has to switch to the *force_power_on* mode, then it increases the counter. If the method does not need the *force_power_on* mode it decreases the counter. If the counter is zero, then the wireless device is powered exactly according to the computed slots of the neighbor discovery algorithm. If the counter has a value greater than zero, then the neighbor discovery slot computation is deferred and each slot is an active slot, and the wireless device remains powered.

5.5.5 Neighbor and Podcast Maps

The STL provides template classes, for example *list*, *map*, *hash_set*, *hash_map* and *queue*. That makes it more comfortable for us to implement such structures, for example the neighbor manager's neighbor map or the file manager's file map. Both have to reference multiple objects by a primary key value for each object, the mapped value for a key then is the structured data of an object. Elements in such a map are sorted from the lowest up to the highest key value, internally. The sort algorithm follows a strict weak ordering, which is a binary relation between keys. By using this map it is possible to access its elements by their key efficiently. We use the computed SHA1 values of files, which have a length of 20 bytes (160bits), for the key value for our file map. By knowing a hash value already, a search within our *fileMap* has a complexity of $O(log(n))$. Our neighbor map uses mac addresses of the discovered devices for the primary key. This key has a length of 6 bytes and is transmitted within each beacon frame.

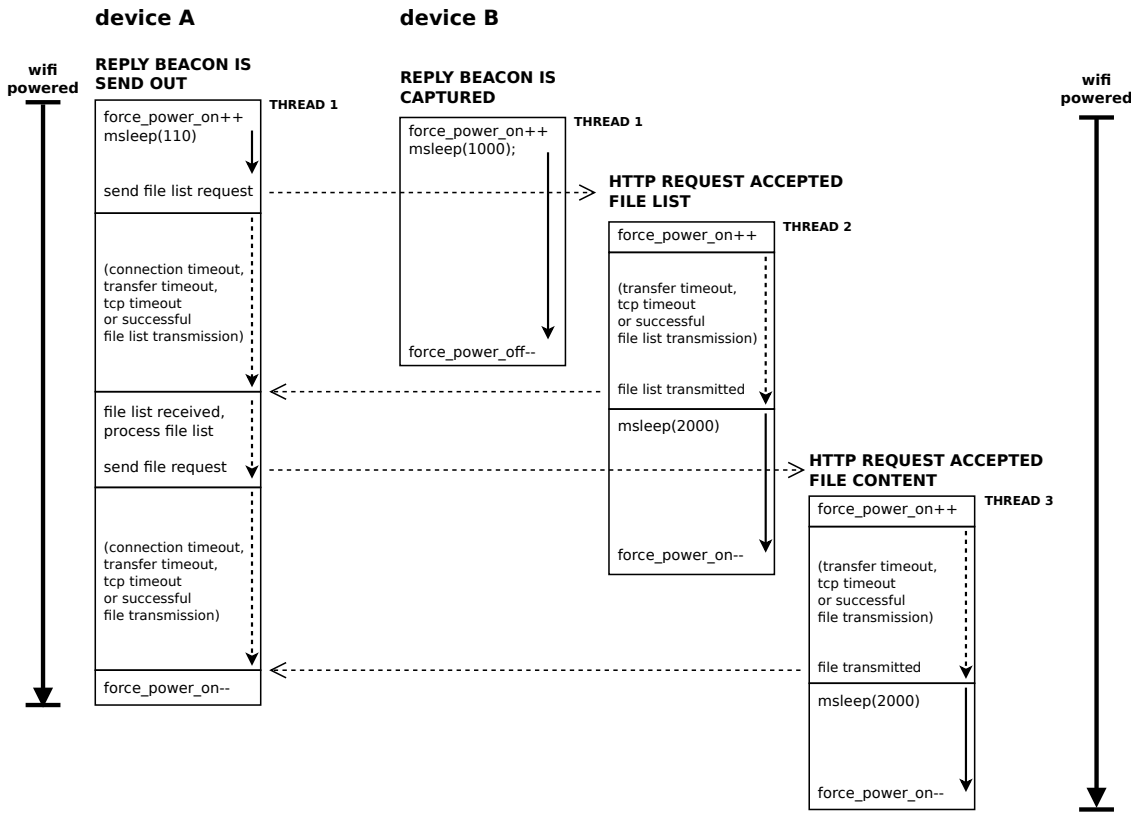


Figure 5.4 Illustration of the *force_power_on* counter on two communicating devices during processing of a reply beacon, a file list request and a file request. The delay msleep(2000) at the end of each accepted HTTP request gives the opportunity to the respective other node to initiate a further TCP/IP request directly after a closed connection.

5.6 Porting to Android

We experimented with our LiND_pwr module on an Android based smartphone. We had access to an already rooted Android HTC Google Nexus One. We used the CyanogenMod firmware, see [7]. It was possible to compile a modified kernel, patched with our LiND_pwr kernel module. Further we get the kernel running on the smartphone and we were able to load our LiND_module. However the main problem was, that the generic stack mac80211 did not work with the internal wireless device, a Broadcom “bmc4329“. The kernel configuration shows us two options ”Broadcom 43xx wireless support (mac80211 stack)“ and ”Broadcom 43xx-legacy wireless support (mac80211 stack)“, both were denoted as ”EXPERIMENTAL“ features. We assume that this may change soon, the wireless kernel developers currently try to port the generic stack to the Android system, see [6].

6

Evaluation

The evaluation chapter mainly consists of 2 selfcontained parts. In the first part we evaluate the effectiveness of our power toggle through measurements of energy consumption over time. Out of our measurements we get the device specific delay times of switching our wireless device off and on. We calculate the power consumption for different asynchronous neighbor discovery algorithms over a fixed time. We analyze the power consumption and the average discovery latency time of each discovery algorithm.

In addition we simulate a podcast sharing scenario based on daily social behavior of students and researchers as the second part of our evaluation. First we analyze the MIT Reality Mining Project dataset [9] in detail. By categorizing some of the occurring nodes as afflicted with high presence in groups, we assumed that a set of nodes present tutors. A such defined tutor is used as source for a simulated file diffusion in succession to irregularly connected nodes.

6.1 Measurements and Analyses

Within this chapter we analyze the power consumption of a wireless radio transmitter, by real current measurements. We observe different power consumption levels according to the different operation modes of our implemented power toggle. Further we denote the specific toggle delay period of the used wireless device. We evaluate the power efficiency of our framework using different quorum algorithms with respect to different desired worst-case discovery latencies.

6.1.1 Setup and Circuit Diagram for Measurements

To find out how efficient our power toggle behaves, with respect to energy consumption and delay times, we used an oscilloscope combined with a wireless USB device.

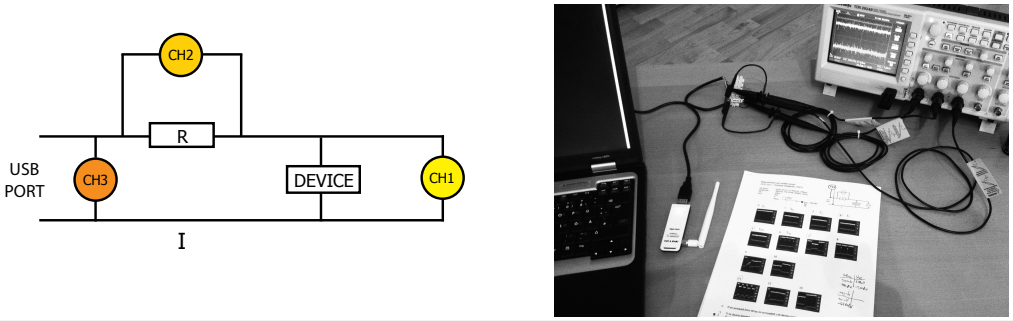


Figure 6.1 Circuit diagram of our experiment setup that we used to record different voltages V_{CH1}, V_{CH2} and V_{CH3} for a later calculation of the power consumption of our wireless usb device. Oscilloscope type: Tektronix, TDS 2024B (200MHz, 2GS/s). R=0,108 Ω

The oscilloscope, a Tektronix TDS 2024B (200MHz, 2GS/s), is connected to the conductor track of a modified USB cable. Therefore we uncovered the electrical conductor tracks of an USB cable at the middle of the length of it. One end of this cable is connected to a Laptop's USB host port, running our modified kernel module under Linux Ubuntu OS, the other is connected to a ZyDAS ZD1211 (TL-WN422G, 54Mbps) wireless USB device [23] with an external antenna. See Figure 6.1 for schematic circuit diagram and a picture of our setup.

By connecting a linear resistor between the USB cable's main power track, in series to the USB device, we were later able to calculate the power consumption of the wireless device in watt. Therefore the recording of two different voltages, V_{CH1} and V_{CH2} were necessary. V_{CH1} gives us the voltage of the USB device and V_{CH2} the voltage of the resistor over time. V_{CH3} was principally used to directly observe variations of the main voltage on the oscilloscope display, we did not use that voltage for our further calculations. Our digital oscilloscope was equipped with an USB host port. Through that benefit we were able to store the different voltages and times, of each test execution, as csv/text formated files on a connected USB flash drive.

6.1.2 Power Toggle Energy Saving

According to Figure 6.1 we calculate the overall power consumption of our wireless device by the following equation:

$$\begin{aligned} P_{dev} &= V_{CH1} * I \\ &= \frac{V_{CH1} * V_{CH2}}{R} \text{ [W]} \end{aligned}$$

Each of the diagrams of Figure 6.3 and Figure 6.4 contains 2 tinted areas. The area between the x-axis and the dotted line at 182 milliwatt represents the wireless USB device base operating power P_{base}, see Figure 6.2. We measured that value before the operation system and the wireless drivers where loaded. To do that, we entered the system's BIOS setup. The power level denoted as P_{off}, which is 187

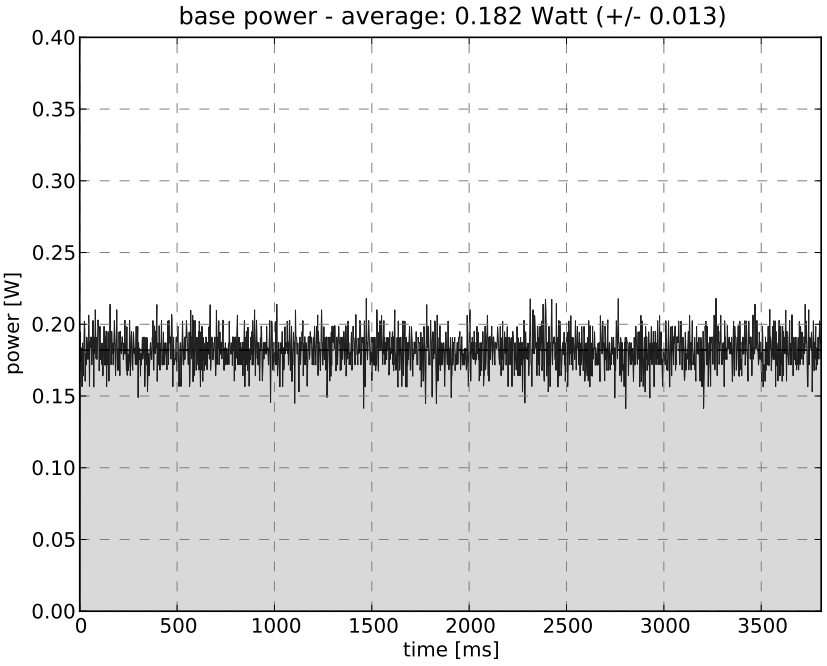
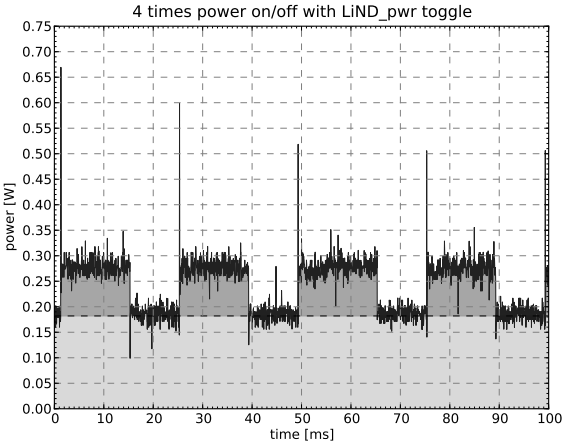


Figure 6.2 We measured the power level value before the operation system and the wireless drivers were loaded. Therefor, we entered the system's BIOS setup. Determination of $P_{base} = 0.182$ W (+/-) 0.013.

milliwatt, is the average consumption after the system was loaded and LiND_pwr module has switched the wireless hardware off, see Figure 6.3. As we can see, P_{off} is 5 milliwatt greater than P_{base}. We assume that difference is caused in a base computation activity of the wireless device. Our power toggle only affects the wireless transmitter and amplification unit. The above darker area illustrates the power that is additionally used by the wireless hardware including the radio control and amplifier. These parts must be powered to receive and transmit radio signals. After the wireless device is switched on, the average power consumption of our device is $P_{on} = 277$ milliwatt. Note, that the ad hoc mode also operates at that power level, see Figure 6.4. Out of that the average power consumption in ad hoc mode over a time δ_{adhoc} can be calculated through $P_{on} * \delta_{adhoc}$. Summarized, using the LiND_pwr toggle in combination with a neighbor discovery algorithm, it is possible to denote the saved energy over a specific period in Watt as $(P_{on} - P_{off}) * \delta_{off}$, where δ_{off} represents the duration of time that the device is switched off in that period.

6.1.3 Power Toggle Delay

We analyzed the delay time of our LiND_pwr toggle by switching the wireless device off and on over a period of time. We logged that process and determined the average duration for both cases. To make sure that the hardware reacts according to our code, we analyzed the power consumption behavior simultaneously, see Figure 6.3. Summarized, it takes 12 ms to switch the wireless hardware off or on. That is acceptable for asynchronous neighbor discovery algorithms with short slot duration time of approximate 100 ms. It can be assumed that this value strongly depends on the wireless hardware and the implementation of the hardware drivers.



i	P^i_{off}	(+/-)	P^i_{on}	(+/-)
1	0.1872 W	0.0137	0.2780 W	0.0151
2	0.1856 W	0.0134	0.2772 W	0.0160
3	0.1872 W	0.0118	0.2777 W	0.0153
4	0.1873 W	0.0123	0.2751 W	0.0174
∅	**0.187 W**	0.0007	**0.277 W**	0.0011

This table gives the average power levels of the "powered" and "unpowered" periods, illustrated in the left diagram.

Figure 6.3 Without deceleration, we switched the wireless hardware on and off. Out of that we get the specific reaction rate time of our LiND_pwr toggle. Simultaneously, we observed the power consumption and determined P_{off} and P_{on}.

6.1.4 Power Toggle with Neighbor Discovery Algorithms

To evaluate how efficient our neighbor discovery implementation works, we tested our implementation on 2 laptops, each was running Linux Ubuntu with a generic kernel (linux-image-2.6.35-25-generic). We made use of the compat wireless drivers (compat-wireless-2011-02-13) including our modification of the *net/mac80211/main.c*. The modification effects the LiND_pwr connectivity, available by the added kernel symbol *ieee80211_hw_pwr()*. The LiND_pwr module was loaded into the kernel on each laptop. Further, we equipped each with a wireless usb device (ZyDAS ZD1211). Airmon-ng of the aircrack-ng package was used on to create a monitor device to inject beacons. The wireless devices were deployed indoor and within eyeshot at a distance of 5 meters.

We analyzed 4 different algorithms, that can be used to achieve energy efficient neighbor discovery [20]: Disco, U-Connect, Grid, and Torus. To compare our later results across the different algorithm, we first determine a set of desirable worst-case discovery latencies, given in the first column 'limit' of table 6.1. Accordingly, for each algorithm, we choose parameters which fulfill that theoretical limitation. For the algorithms Disco and U-Connect we chose different prime numbers for each node, for Grid and Torus we chose parameters that result in a square area. The calculation of the exact theoretical worst-case discovery latency, in seconds, for each algorithm and setting, is specified in the column 'th.wc' (theoretical worst-case). Each algorithm is based on time slots, active slots and sleeping slots [20]. The length of the slots are fixed, we adopt a constant length of 100 ms. 100 milliseconds is the default beacon period of W-LANs. We logged every power toggle process of LiND_pwr (switch on/off), every injected and every captured beacon with respect to the runtime in milliseconds. All data was stored in a separate log file for each algorithm and parameter setting. A python script analyzed these log files, computed the duration of the time that the transmitter was powered and the average discovery latency for each setting, summarized and compared in Figure 6.3. Our desired dis-

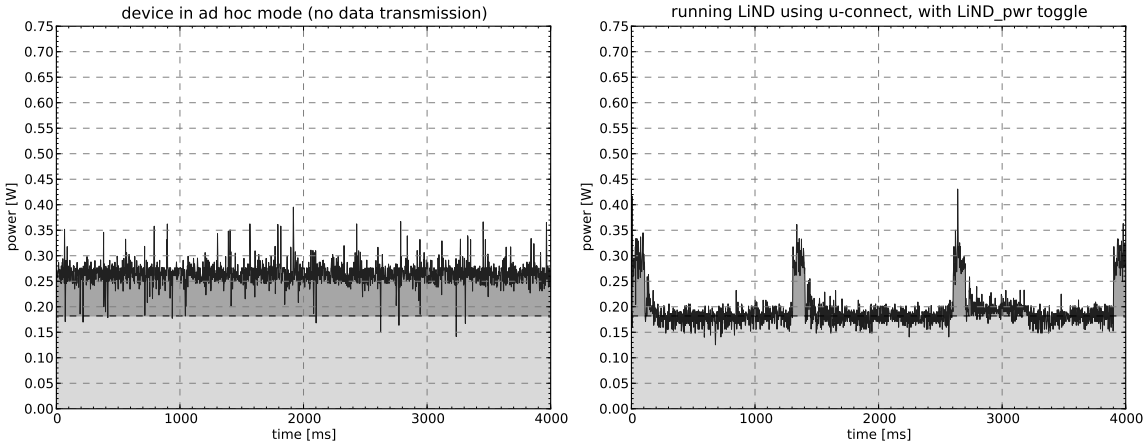


Figure 6.4 The left diagram shows the overall power consumption of the wireless device running in ad hoc mode, the right one shows the power consumption of the ad hoc mode in combination with LiND_pwr using the u-connect neighbor discovery algorithm.

covery limit in seconds and the corresponding settings for the 4 different algorithms, Disco, U-Connect, Grid and Torus:

	desired	Disco		U-Connect		Grid		Torus	
id	limit	$p_1 * p_2$	th.wc	$p_1 * p_2$	th.wc	$w * h$	th.wc	$w * h$	th.wc
A	10 s	(9,11)	9.9s	(9,11)	9.9s	(10,10)	10.0s	(10,10)	10.0s
B	20 s	(11,17)	18.7s	(11,17)	18.7s	(14,14)	19.6s	(14,14)	19.6s
C	40 s	(17,23)	39.1s	(17,23)	39.1s	(20,20)	40.0s	(20,20)	40.0s
D	60 s	(19,31)	58.9s	(19,31)	58.9s	(24,24)	57.6s	(24,24)	57.6s
E	120 s	(31,37)	114.7s	(31,37)	114.7s	(34,34)	115.6s	(34,34)	115.6s

Table 6.1 Initial settings for each quorum algorithm of our experiments due to five different desired maximum discovery time limits.

Table 6.2 shows the average discovery latency, in the column 'avg', measured and computed out of the log files for each of the algorithm. Compared to the initial settings table 6.1, we see, that the average discovery latency is always lower than our desired limit, and, moreover, lower than the theoretical worst case latency, nearly always lower than the half of it. As assumption, with respect to discovery latency, our implementation meets the requirements for a practical application.

Table 6.3 shows the measured and out of the log files calculated duration that the wireless transmitter is powered within the discovery process, denoted as δ_{on}. Values are shown in seconds and in percent with respect to the overall duration of the experiment, which was one hour for each setting:

Considering the values of both tables, for example for U-Connect and the corresponding row B, we have evaluated: Using the U-Connect algorithm on two nodes with chosen prime numbers 11 and 17, the discovery process between two devices sharing a room takes on average 6.15 seconds, the wireless transmitter is 12.25% of the whole discovery application time unpowered. By comparison, our external USB wireless

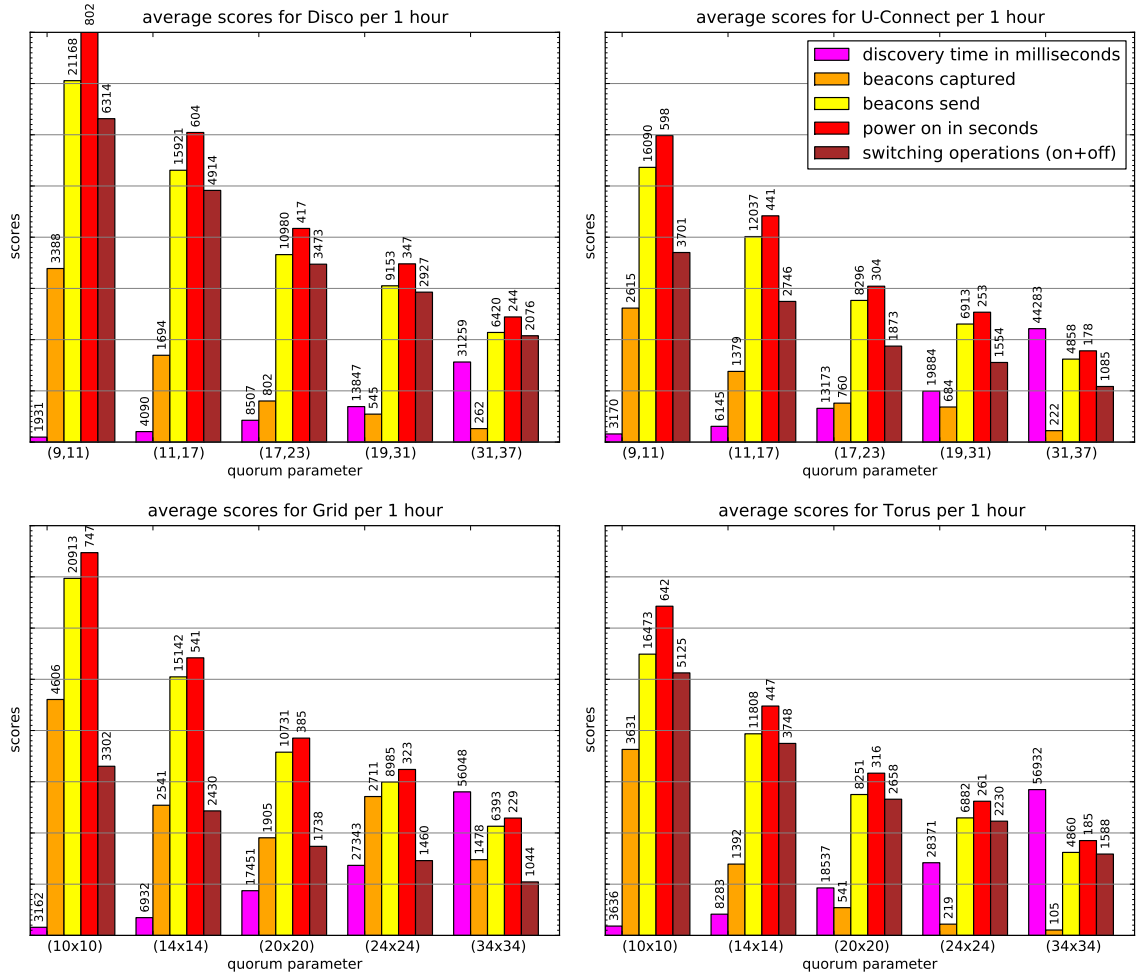


Figure 6.5 Evaluation of our neighbor discovery implementation using 4 different algorithms: Disco, U-Connect, Grid, and Torus. We analyzed the discovery time in seconds, the amount of send and captured beacons, the time that the wireless device is powered and we count how often the power was switched. We count 'from on to off' and 'from off back to on' as one switching operation.

id	Disco avg	U-Connect avg	Grid avg	Torus avg
A	1.93s	3.17s	3.16s	3.64s
B	4.09s	6.15s	6.93s	8.28s
C	8.51s	13.17s	17.45s	18.54s
D	13.85s	19.88s	27.34s	28.37s
E	31.26s	44.28s	56.05s	56.93s

Table 6.2 Discovery time results of our implementation due to the different denoted algorithms and settings listed in Table 6.1. The values are also illustrated in Figure 6.5. Each implemented algorithm fulfills our expectation due to the theoretical worst case value.

	Disco		U-Connect		Grid		Torus	
id	δ_{on}	%	δ_{on}	%	δ_{on}	%	δ_{on}	%
A	802s	22.28%	598s	16.61%	747s	20.08%	642s	17.83%
B	604s	16.78%	441s	12.25%	541s	15.03%	447s	12.42%
C	417s	11.58%	304s	8.44%	385s	10.69%	316s	8.78%
D	347s	9.64%	253s	7.03%	323s	8.97%	261s	7.25%
E	244s	6.78%	178s	4.94%	229s	6.36%	185s	5.14%

Table 6.3 Time in seconds that the wireless device is powered during a runtime period of 1 hour for each algorithm and setting. Based on the measured time values, with respect to the runtime length, we denote the powered time additionally in percent.

transmitter, operating in general ad hoc mode without any modification, would consume within one hour the energy $E_{adhoc} = P_{on} * 3600s = 0.277W * 3600s = 997.2J$, and with our LiND_pwr power toggle and the U-Connect settings of our example $E_{UC,B} = P_{on} * \delta_{on} + P_{off} * \delta_{off} = 0.277W * 441s + 0.187W * (3600 - 441)s = 712.9J$.

We assume, by using an internal wireless adapter, the base power P_{base} is lower than that of our USB device which has additionally to power the USB hardware. Out of that, as speculation, the resulting quotient that denotes the energy relation between the generic ad hoc mode and our energy efficient solution, would be better than that of our experiment, which is calculated as follows for U-Connect and settings B: $E_{UC,B}/E_{adhoc} = 712.9J/997.2J = 71.5\%$. For the chosen example, by using our LiND_pwr toggle implementation, the consumed overall energy is 71.5% of that using the generic ad hoc mode only. However, by ignoring the base power level P_{base}, with assumption that this base power is consumed in every operation case anyway, we are able to compare the generic ad hoc mode and our implementation, using the same example, by the following calculation:

$P'_{on} = P_{on} - P_{base} = 0.277W - 0.182W = 0.095W$
$P'_{off} = P_{off} - P_{base} = 0.187W - 0.182W = 0.005W$
$E'_{adhoc} = P'_{on} * 3600s = 342J$
$E'_{UC,B} = P'_{on} * \delta_{on} + P'_{off} * \delta_{off} = 0.095W * 441s + 0.005W * (3600 - 441)s = 57.69J$
$E'_{UC,B}/E'_{adhoc} = 57.69J/342J = 16.87\% \approx 17\%$

Without considering the base power P_{base}, our LiND_pwr implementation, by using U-Connect and settings B, makes the wireless transmitter to consume only 17% of the overall energy, comparing to the generic ad hoc mode.

6.1.5 Beacon and Reply Beacon Error Rate

A device sends out beacons to call attention to itself. After receiving a beacon, a device checks if it is worth to establish a TCP/IP connection to synchronize new data. Should this be the case, the device goes into the "force power on" mode, waits a random period of 5 ms up to 15 ms and sends out one reply beacon. That reply packet could get lost on the channel, due to interferences or different wireless range of different nodes. For each implemented neighbor discovery algorithm we analyzed the reply beacon error rate in order to determine the basic error rate of reply beacons and to evaluate the correctness of our implementation. We used the same experimental settings as we have used to evaluate the discovery latency, two wireless devices were deployed indoor with a distance of 5 meters and they had direct eye contact. We counted the sent and the captured reply beacons on each device. Out of that we calculated the reply beacon error rate in percent. We stopped each measurement, after each of the devices has send out at least 2000 or more reply packets. The following table shows our results for 10 discrete runs:

			send from		capt. from		error rate in %		
run	algorithm	para.	dev0	dev1	dev0	dev1	dev0	dev1	∅
1	Disco	(3,5)	4991	5041	4359	4824	12.66%	4.30%	8.48%
2	Disco	(3,5)	3271	2232	2979	2160	8.92%	3.23%	6.08%
3	U-Connect	(3,5)	2279	2248	2157	2130	5.35%	5.25%	5.30%
4	U-Connect	(3,5)	2598	2619	2466	2534	5.08%	3.13%	4.11%
5	Grid	(3,3)	3054	3086	2611	3066	14.51%	0.65%	7.58%
6	Grid	(5,5)	4653	2178	4238	2045	8.92%	6.11%	7.52%
7	Grid	(5,5)	2038	3817	1942	3153	4.71%	17.40%	11.06%
8	Torus	(3,3)	2342	2013	1984	1983	15.29%	1.49%	8.39%
9	Torus	(5,5)	2207	5872	2151	5387	2.54%	8.26%	5.34%
10	Torus	(5,5)	3753	2129	3153	2076	15.98%	2.49%	9.22%

Table 6.4 Evaluation of the reply beacon error rate of two devices in range. A single reply beacon was send out within a random period of 5 ms up to 15 ms after a beacon frame was received. The average error rate is 7.31% with a standard deviation of 2.11%.

We observed an average reply beacon error rate of 7.31% with a standard deviation of 2.11%. After knowledge of that observation we adopt our implementation and send out 3 reply beacons successively with a short random delay before each repetitive frame. That should result in an effective overall average error rate for a reply message lower than 0.01%, under the assumption that the errors are statistical independent errors, based on single random and not on burst interferences. Further we adopt this knowledge for our beacon injection procedure and send out 3 beacons successively at the beginning of each powered slots, with a delay of 2 ms after each beacon. The presented tests and the corresponding evaluation of our neighbor discovery

performance, in chapter 6.1.4, already underly this improvement. Summarized, to keep high reliability of our neighbor discovery process, our framework sends out three beacons at the beginning of each powered slot. If a neighbor wants to establish a connection to the sender, it sends out three reply beacons, but this only once for a received beacon of a sequence of maximal three beacons of the corresponding sender. Thus, the average overall probability for a successful discovery procedure related with an eventually initiated reply beacon is higher than 99%.

6.2 Simulation of a Podcast Sharing Scenario

In that part of my thesis we analyze, by simulation, that a podcast dissemination within a group of people, equipped with wireless devices, performs properly. First we analyze logged wireless traces of the MIT Reality Mining Set [9] to get knowledge of fixed times in a week of high neighbor discovery activity. Hence, we assume that people have group meetings at that times. We calculate the group of people, whose devices logged a general high distributed neighbor discovery activity in terms of quantity of different connection partners. We define these people as tutors, choose them as source and simulate the file dissemination over time.

6.2.1 Reality Mining Dataset Analyses

Our simulation is based on the results of the MIT Reality Mining project [9] from the MIT Media Lab. The project is one of the largest in the area of tracking user mobility. Over a period of 9 month, in the course of the 2004/2005 academic year, 100 Bluetooth enabled smart phones were given to researchers and students. All phones were running a modified software (based on the Nokia Symbian Series 60) that logged several mobility data. They captured communication, proximity, location and activity information of each device. The result is a dataset that reflects over 350,000 hours, continuously over all devices. This responds to nearly 40 years of logged data on human mobility behavior at the MIT. We downloaded the source traces from crawdad.org [9] and used them for all further simulation.

Concerning the MIT Reality Mining Project, each device was scanning for nearby devices every 5 minutes. This limitation was made, because of the high energy consumption of Bluetooth scans in association to the daily necessary battery lifetime of a smart phone. We analyzed the logged Bluetooth connections to simulate a podcast dissemination in a group. That results give us a lower bound of dissemination in a group when using WiFi devices. WiFi devices have a ten times higher range and higher transfer speeds than what the Bluetooth technology offers today. Further, depending on the chosen quorum, our solution performs a fast reliable neighbor discovery in a matter of seconds.

6.2.1.1 User Behavior

To figure out general daytimes, when people have meetings, we divide 8 month of the traces duration into hours. For each hour we calculate the sum of new discoveries

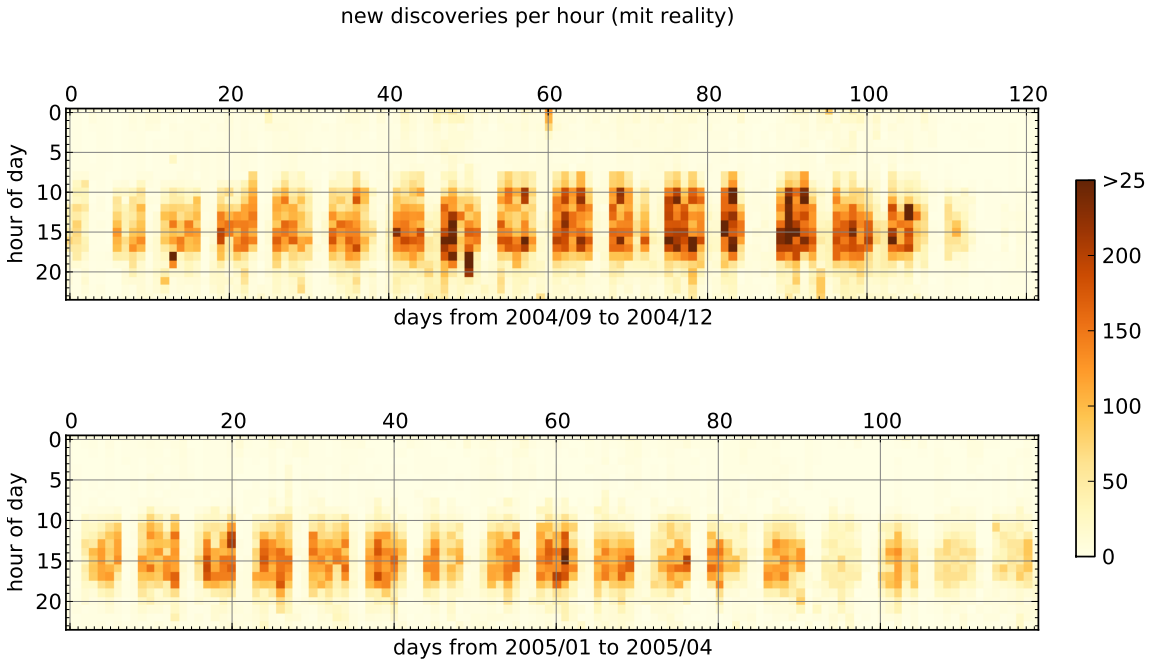


Figure 6.6 Sum of new discoveries of the 103 relevant devices per hour between September 2004 and end of April 2005 (8 month) of MIT Reality Mining project traces.

of all devices. We assume that a significant number of people must come together when that sum is high, Figure 6.6 shows a visualization of the results. As expected, we principally observe that the number of contacts is high during the main working hours from 8 am until 6 pm. Furthermore there are less logged Bluetooth connections on weekends. Evidently we are able to revise that people don't go to work on thanksgiving, that is celebrated, in the United States, on the fourth Thursday of November. To prevent our later simulations from the influence of Christmas and New Year, we restrict our further analysis on 14 continuous weeks in a range from September to December.

6.2.1.2 User Groups and Tutors

Out of the traces descriptions, we know, that the subjects, that logged the traces, are comprised of to 2 separate sections. The first section contains first-year business school students, the second one is composed of individuals working together in the same building at the MIT [10].

From now onwards we call a wireless device a node. If one node discovers another one, the two nodes are close to each other and we express that by linking both nodes through an edge. All edges are undirected. The betweenness of two nodes is defined a) by the number of new discoveries and b) by the time they stay in range. To compute how the nodes of the MIT traces are grouped concerning their mobility behavior, we weighted each edge between two nodes according the two different betweenness definitions. Figure 6.7 shows for both cases the top 10% of

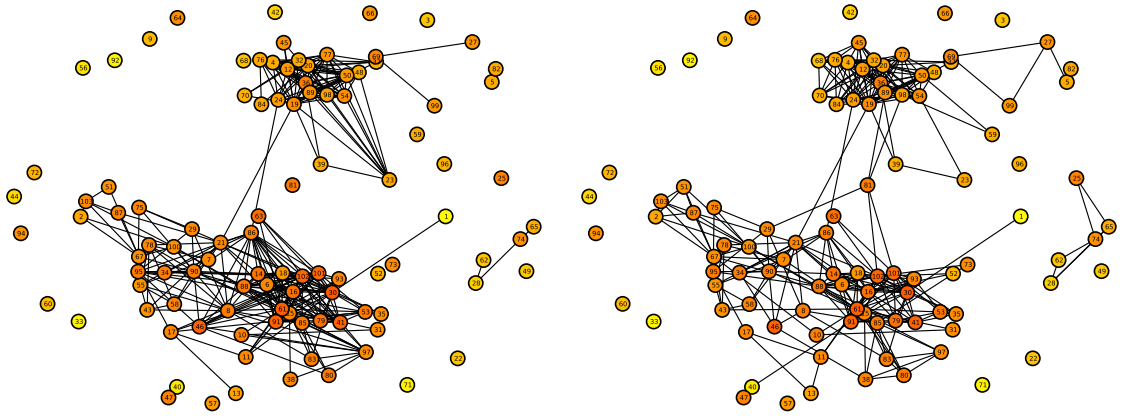


Figure 6.7 Top 10% of weighted edges between the 103 relevant devices of MIT Reality Mining project traces. Edges are weighted by number of new connections (left) and by connection time in minutes (right) in a time frame of 14 weeks, from 2004-09-07 to 2004-12-16. The more darker a node the more different nodes are discovered within the time frame.

that weighted edges. Both graphs nearly have the same main edges. In other words, nodes that meet each other more often also have a higher average meeting duration. Out of that we are able to suggest that these very active nodes define a group related to their routinely movements.

We colored all nodes depending on how much overall distinct edges a node is linked to within the time frame of the analysis. Hence we are able to set a basis to select nodes assumed as tutors for our podcast dissemination simulation.

6.2.2 Podcast Dissemination

We want to simulate the effectiveness of podcast dissemination based on real world mobility traces. We discuss the following interesting question: How many devices of students and researchers could be reached if a tutor's device, or a similar person's device endued with high physical presence towards other devices, would be used as dissemination source in a meeting.

Like our implementation in chapter 5, communication in our simulation over all devices conforms, on a lower level, a one-hop communication paradigm. One device shares podcasts only with directly connected devices. In one's own way, each user of a device generates movements, and transports actually on their device stored podcasts from one physical location to another. The combination of multiple one-to-one communications over time lets the podcast dissemination operate like a delay tolerant multi-hop network does.

In section 6.2.1.2 we described how to find good candidates for our tutors, for a starting point of our podcast dissemination. Arising from section 6.2.1.1 we define an assumed meeting at each Tuesday morning at 10 am. The simulation works as follows: The assumed tutor node starts offering a podcast file every Tuesday morning at 10 am. At that time, this node is the only one that has a copy of the file. Every

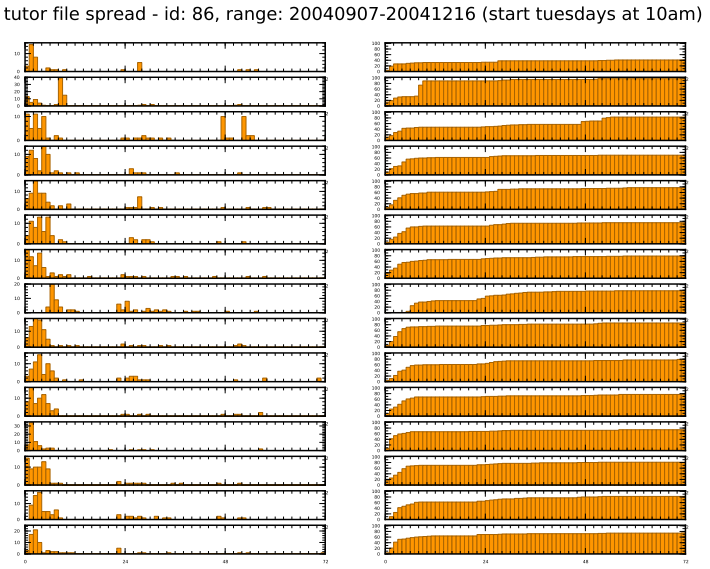


Figure 6.8 File spread within the first 3 days after a tutor starts offering a file on each Tuesday (from 2004-09-07 to 2004-12-16) at 10 pm on an assumed meeting. The figure shows the dissemination over all devices for each Tuesday in detail.

time the tutor node discovers another node, with respect to the logged traces, the file is replicated to the discovered node. Such new file owners are able to offer the file to nodes that they discover, and so on. We simulate the file dissemination on 15 Tuesdays, always from 10 am on with a simulated duration of 72 hours (3 days).

Figure 6.8 shows the dissemination in detail for a fixed tutor for each Tuesday.

We observe that dissemination, in our scenario, is performed at working hours and pauses over night, that is not unusual. Over 50% of all nodes received the diffused podcast file within the first 6 hours, after 24 hours 65% and after 48 hours over 70% of all nodes are reached. Figure 6.9 illustrates that results. We visualize the average file dissemination and the number of reached nodes over time in Figure 6.9.

As illustrated in figure 6.10, with respect to the corresponding underlying connections of figure 6.7, we observe that the file dissemination performs according to the previously selected edges. This result confirms the corollary assumption that the possibility of a file transmission, between longer and more often connected nodes, is higher than between other ones. Thus, a file transmission between course participants, which meet regularly during a day, is feasible.

As a note, the resolution of the Reality Mining Dataset is denoted with 5 minutes between two discovery executions and the underlying wireless technology is Bluetooth which has a typical range of 10 meters. Our LiND framework uses a W-LAN device with an enhanced wireless range and a neighbor discovery latency lower than 10 seconds, see table 6.2. According to our solution, the presented university simulation results can be interpreted as indicating a lower bound.

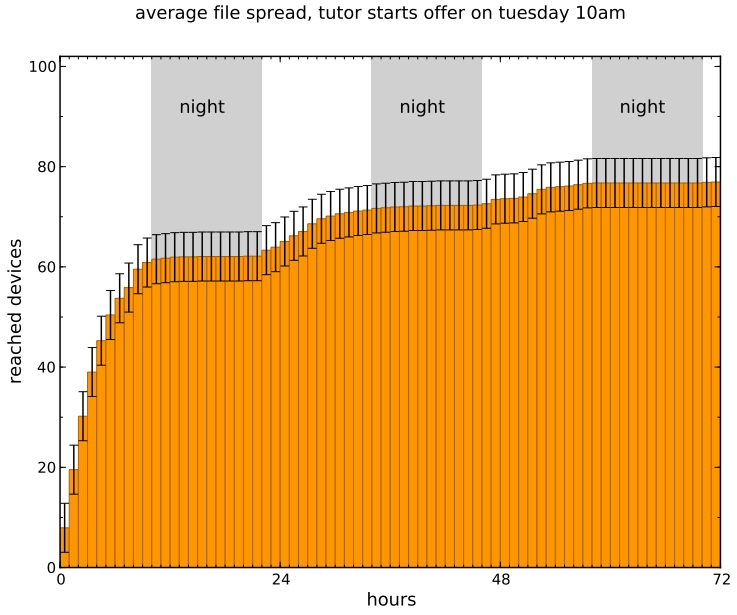


Figure 6.9 File spread within the first 3 days after a tutor starts offering a file on each Tuesday (from 2004-09-07 to 2004-12-16) at 10 pm on an assumed meeting. The figure shows us the mean result calculated over all Tuesdays.

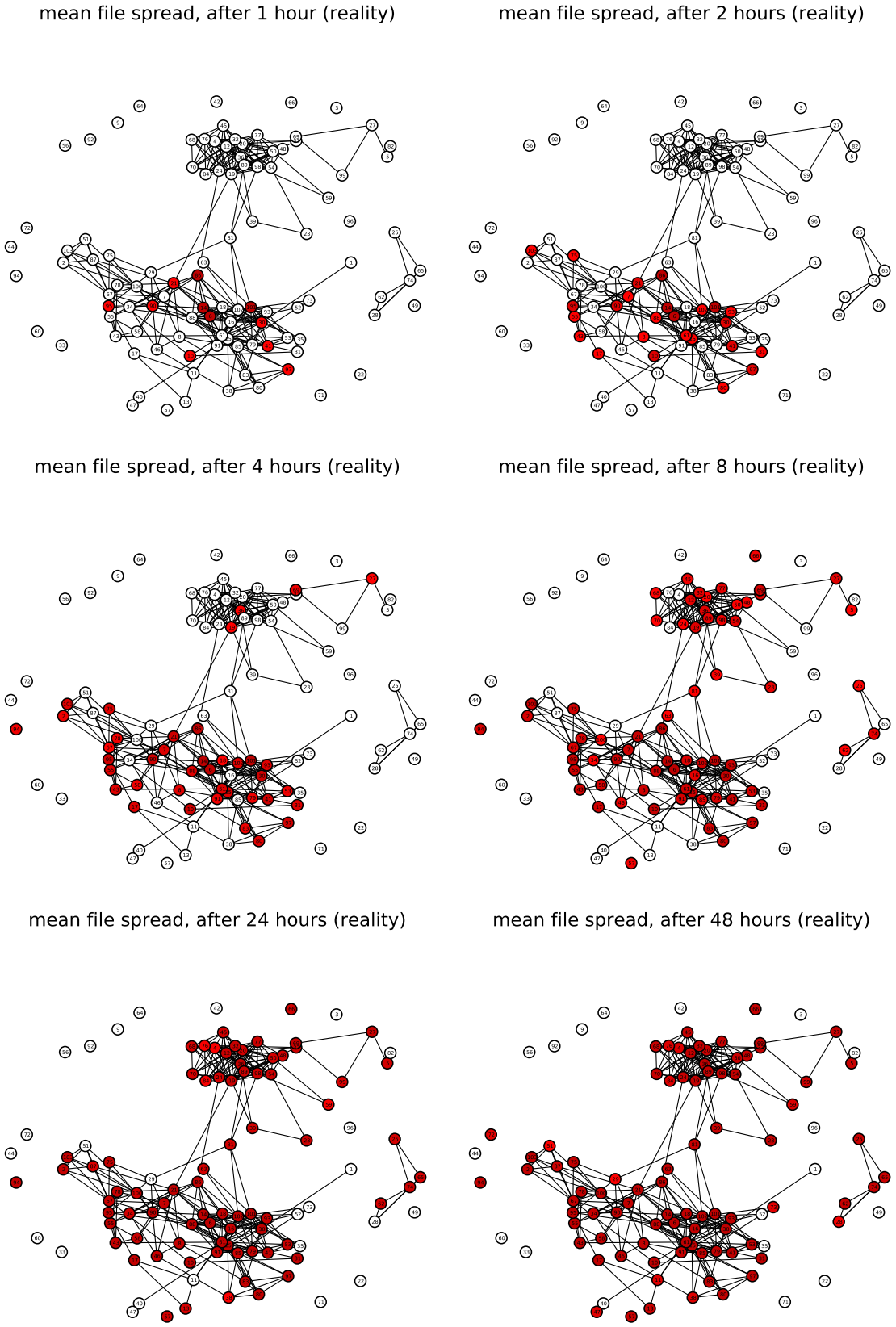


Figure 6.10 The mean file spread within the periods of 1h, 2h, 4h, 8h, 24h and 48h combined with the connection graph of Figure 6.7 (right) is illustrated. Dark tinted nodes of a figure are reached more to the beginning of a denoted period and lighter nodes are more to the end. White nodes are reached above a denoted period, or never. Mean calculation over all Tuesdays is used.

7

Conclusion

A trade-off between power consumption and discovery latency is unavoidable. Achieving a minimum of discovery latency means to invest energy. Saving energy results in higher latencies. With respect to our intention, conceptualize and implement an energy efficient neighbor discovery framework to enable podcast sharing for generic Linux based mobile devices, we assume a worst-case discovery latency with a maximum of 10 seconds, as we have also evaluated, as sufficient. However, our presented prototype framework LiND is not limited to share podcasts, our essential energy efficient neighbor discovery procedure can be adopted on other application scenarios. The worst-case discovery latency can be enhanced or reduced, according to various areas of application.

7.1 Summary

We present LiND, an energy efficient neighbor discovery framework for generic Linux based systems. On and off times of IEEE 802.11 based wireless devices are scheduled by quorum schemes and performed by using our LiND_pwr kernel module. We show that our implementation fulfills the theoretical assumptions according to different quorum algorithms and different theoretical worst-case latency times. Thus, our solution enables asynchronous wireless neighbor discovery between sporadically available devices. For example, the results of our real world experiments show, that by a given worst-case discovery latency of 10 seconds, discovery can be achieved within an average latency time of 3.2 seconds, while the radio transmitter is powered only 17% of the time. Furthermore, the probability of a successful two sided-discovery within the worst-case latency is higher than 99%. Additionally, we present our prototype podcast sharing application. This performs in close collaboration with the neighbor discovery procedure, to keep, most of the time, the radio transmitter in the energy efficient discovery mode. Only if the possibility, to receive new podcasts, is high, the radio device is powered continuously to reliably perform podcast transmissions. We simulate a university scenario using the Reality Mining Dataset, which is based

on realistic sporadic device discoveries. Our results show that a sharing scenario is applicable and course participants receive subscribed course material within 48 hours. As a note, the finest resolution of the Reality Mining Dataset is denoted with 5 minutes between two discovery executions and the underlying wireless technology is Bluetooth. Our developed and evaluated LiND framework performs neighbor discovery successfully within seconds and uses the enhanced range of the IEEE 802.11 technology. Thus, with respect to the Reality Mining Dataset, our framework will significantly outperform the simulated file dissemination of the presented university sharing scenario.

7.2 Limitations

Our presented framework is limited to Linux based systems only, using the IEEE 802.11 specified open source mac80211 Linux wireless stack. A patch of the corresponding generic kernel module is necessary. One single method has to be added and exported as kernel symbol. However, while performing energy efficient neighbor discovery with our framework, Linux systems equipped with only one WiFi hardware transmitter are able to add a third virtual interface for infrastructure mode. Simultaneous operation is possible by using a common fixed frequency for the infrastructure mode and the ad hoc mode. However, we cannot toggle the power of the transmitter continuously, as this would be detrimental to the transmissions between the device and its base station. For the neighbor discovery process it is necessary to operate on a predefined fixed wireless channel. As wireless terminals in infrastructure mode adapt to different channels, following commands by the connected access point, we cannot guarantee that the device stays on the predefined discovery channel. It might therefore be necessary to further adjust the neighbor discovery process to scan on different channels.

7.3 Future Work

Our goal was to develop an energy efficient neighbor discovery solution based on generic Linux operating systems. The Android platform is a Linux operating system for mobile devices - smartphones and tablets. Each Android based mobile device equipped with a W-LAN device must have the functionality to deactivate the wireless transmitter, because any mobile Android device must support the “flight-mode”. We also invested work to run our solution on an Android based smartphone - an HTC Google Nexus One. While our LiND application is not the problem, our LiND_pwr kernel module depends on the underlying mac80211 wireless stack and we didn’t have access to a smartphone, which already supports the generic mac80211 wireless stack, as of now. However, this feature is already integrated within the Android kernel, denoted as an “experimental” feature, at least in the modified CyanogenMod firmware, see [7].

7.4 An Outlook into Broader Applicability

> "Egham, UK, April 7, 2011 - Worldwide smartphone sales will reach 468 million units in 2011, a 57.7 percent increase from 2010, according to Gartner Inc. By the end of 2011, Android will move to become the most popular operating system (OS) worldwide and will build on its strength to account for 49 percent of the smartphone market by 2012."
>
> *Press release from Gartner Inc. - a world leading information technology research and advisory company.*

To the present day, to the best of our knowledge, there exists no popular, generic and permanently activated energy efficient neighbor discovery solution implemented on Android based smartphones using the WiFi device, nor on other smartphone operating systems like iOS, Windows Mobile or BlackBerry.

Enhancement to the energy efficiency of smartphones is a current widespread research topic. According to the future oriented ongoing research and development of new technologies, within the area of mobile energy storage devices, new beneficial smartphone hardware components and features may be integrated. It may be possible that future smartphones are equipped with two or more independent WiFi transmitters and antennas, using multiple-input and multiple-output (MIMO) technology, as already used for high speed wireless communications. That enables the possibility to continuously use one single transmitter to connect to access points, using the infrastructure mode. And another one to only perform ubiquitous coordinated energy efficient neighbor discovery and neighbor communication, like presented with this thesis. Our preliminary framework, and our results, provide Linux based system developers a very new basement to build useful applications in the area of sporadic Peer-To-Peer wireless mobile device communication, building upon energy efficient neighbor discovery.

Bibliography

[1] Aircrack-ng Community. Aircrack-ng. http://aircrack-ng.org, accessed 2011-06-13.

[2] Anand, M., Nightingale, E. B., and Flinn, J. Self-tuning wireless network power management. In *Proceedings of the 9th annual international conference on Mobile computing and networking* (New York, NY, USA, 2003), MobiCom '03, ACM, pp. 176–189. http://doi.acm.org/10.1145/938985.939004.

[3] Andy Green. Penumbra, Broadcast-based Wireless network. http://penumbra.warmcat.com/, accessed 2011-06-13.

[4] Berg, J. Linux Wireless, Basic hardware handling, struct ieee80211_ops. http://wireless.kernel.org/80211books/API-struct-ieee80211-ops.html, accessed 2011-06-13.

[5] Berg, J. mac80211 Linux API Documentation. http://linuxwireless.org/en/developers/Documentation/mac80211, accessed 2011-06-13.

[6] Berg, J. Support for cfg80211 / mac80211 Linux 802.11 drivers on Android. http://wireless.kernel.org/en/developers/Documentation/Android, accessed 2011-06-13.

[7] CynogenMod Community. CynogenMod, Android Community Rom based on Gingerbread. http://www.cyanogenmod.com/, accessed 2011-06-13.

[8] Dutta, P., and Culler, D. Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In *Proceedings of the 6th ACM conference on Embedded network sensor systems* (New York, NY, USA, 2008), SenSys '08, ACM, pp. 71–84. http://doi.acm.org/10.1145/1460412.1460420.

[9] Eagle, N., and Pentland, A. S. CRAWDAD data set mit/reality (v. 2005-07-01). Downloaded from http://crawdad.cs.dartmouth.edu/mit/reality, 2005. http://crawdad.cs.dartmouth.edu/mit/reality.

[10] Eagle, N., Pentland, A. S., and Lazer, D. Inferring friendship network structure by using mobile phone data. *Proceedings of the National Academy of Sciences 106*, 36 (Aug. 2009), 15274–15278. http://dx.doi.org/10.1073/pnas.0900282106.

[11] Friedman, R., Kogan, A., and Krivolapov, Y. On power and throughput tradeoffs of wifi and bluetooth in smartphones. In *Proc. of the 30th IEEE International Conference on Computer Communications INFOCOM 2011* (2011).

[12] Halberstam, H., and Laxton, R. R. Perfect difference sets. In *Proceedings of the Glasgow Mathematical Association* (1964), pp. 177–184. http://journals.cambridge.org/action/displayFulltext?type=1&fid=5229280&jid=GMJ&volumeId=6&issueId=04&aid=5229272&bodyId=&membershipNumber=&societyETOCSession=.

[13] IEEE. IEEE Standard for Information technology, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, 2007. http://standards.ieee.org/getieee802/download/802.11-2007.pdf.

[14] Jiang, J.-R., Tseng, Y.-C., Hsu, C.-S., and Lai, T.-H. Quorum-based asynchronous power-saving protocols for ieee 802.11 ad hoc networks. *Mob. Netw. Appl. 10* (February 2005), 169–181. http://dx.doi.org/10.1145/1046430.1046443.

[15] Jung, S., Lee, U., Chang, A., Cho, D.-K., and Gerla, M. Bluetorrent: Cooperative content sharing for bluetooth users. In *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 47–56. http://portal.acm.org/citation.cfm?id=1263542.1263697.

[16] Kandhalu, A., Lakshmanan, K., and Rajkumar, R. R. U-connect: a low-latency energy-efficient asynchronous neighbor discovery protocol. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks* (New York, NY, USA, 2010), IPSN '10, ACM, pp. 350–361.

[17] Lang, S., and Mao, L. A torus quorum protocol for distributed mutual exclusion. In *Proc. of the 10th Int'l Conf. on Parallel and Distributed Computing and Systems* (1998), Citeseer, pp. 635–638.

[18] Maekawa, M. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comput. Syst. 3* (May 1985), 145–159. http://doi.acm.org/10.1145/214438.214445.

[19] Pack, S., and Choi, Y. An adaptive power saving mechanism in ieee 802.11 wireless ip networks.

[20] Schupp, S. Using quorum schemes for neighbor discovery in wireless sensor networking. Bachelor Thesis, RWTH Aachen, 2010.

[21] SIG, B. Bluetooth Technical Core Specification Version 4.0, 2010. http://www.bluetooth.org/Technical/Specifications/adopted.htm.

[22] Singer, J. A theorem in finite projective geometry and some applications to number theory. *Trans. Amer. Math. Soc. 43* (1938), 377–385.

[23] TP-LINK Technologies Co., L. Specifications of TL-WN422G downloaded from http://www.tp-link.com/products/, 2010. http://www.tp-link.com/products/productDetails.asp?class=wlan&content=spe&pmodel=TL-WN422G.

[24] Zheng, R., Hou, J. C., and Sha, L. Asynchronous wakeup for ad hoc networks. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing* (New York, NY, USA, 2003), MobiHoc '03, ACM, pp. 35–45. http://doi.acm.org/10.1145/778415.778420.

[25] Zheng, R., Hou, J. C., and Sha, L. Performance analysis of the ieee 802.11 power saving mode. In *CNDS 2004, San Diego, USA* (01 2004). http://www2.cs.uh.edu/~rzheng/docs/cnds04.pdf.

List of Figures

1.1 Motivation: Students waiting at a bus stop. Reliable and energy efficient neighbor discovery provides a basis to establish a sporadic Peer-to-Peer based communication between wireless mobile devices, like smartphones. We present a corresponding applicable prototype framework to perform energy efficient neighbor discovery and automatic podcast sharing under Linux. 2

2.1 Analysis of the energy consumption of the 802.11 Power Saving Mode with respect to different beacon intervals grouped by different traffic loads. The analysis and the graph is shown in [25] 10

3.1 Penumbra wifi network illustration of the application, the interface management and how packet capturing is performed. Images are part of the website [3] 18

4.1 The generic mac80211 stack supports multiple virtual interfaces for a single wireless hardware device. Additionally it is possible to manage multiple hardware devices. We want to set up only 2 virtual interfaces on one single wireless hardware device. One interface is set to the ad hoc mode, the other to the monitor mode. 22

4.2 We implement two slots: powered and unpowered. Thus, because the "sending followed by listening" slot does not allow to receive a reply beacon frame, which possibly is send back with a delay within the rest of the slot time. 23

4.3 Illustration of how our updateCounter works. As example, this value of device A is set to 34. The updateCounter value is transmitted within each beacon. According to this value and the possibly already stored updateCounter value of the corresponding other device, it is possible to decide if a switch to the force_power_on mode, to perform further podcast transmissions, is necessary. 25

4.4 A file list request is performed using TCP/IP, by transmitting the user's previously defined tags, which describe the interests of the user. The corresponding other device search it's local podcast library, according these tags, and generates a request structured as XML document. 26

4.5 A file request is performed using HTTP over TCP/IP, by transmitting the desired file's hash and byte range. The file's secure hash value and the corresponding maximum amount of stored data on device B is already known from the previously transmitted file list. 27

4.6 Illustration of *spareCLientSocks* usage. Device A is configured to serve a maximum number of 9 client connections. Three client connections are currently processed within threads. Accordingly, device A denotes six spare client sockets in it's beacon. Device E captures a beacon from A, switches itself to the force_power_on mode, sends a reply beacon to device A to inform it to switch to the force_power_on mode (eventually device A may already perform the force_power_on mode) and tries to establish a TCP/IP connection. 28

4.7 Interaction of the basic modules. Our LiND_pwr module uses exported symbols of the mac80211 kernel module and it offers a character device to user space applications. Our LiND daemon writes to the character device to initiate a power switch. Our LiND daemon offers a domain socket for third party applications, for example for a GUI. 31

4.8 Slot behaviour with respect to the "wake up" and "goto sleep" duration, respectively δ_+ and δ_-. The power on command must be initiated δ_+ before the active slot begins. The power off command must be initiated directly after an active slot with the beginning of a sleep slot but overlays with the sleeping slot during δ_-. 33

4.9 The communication between two devices is illustrated. A device injects beacons to perform neighbor discovery. A reply beacon initiates the force_power_on mode on both devices. Further the file list and file transmission is proceeded by using TCP/IP. 34

5.1 Our implemented function *ieee80211_hw_pwr()* in the mac80211 module. Interaction with LiND_pwr and *ieee80211_ops*. 41

5.2 LiND_pwr module: basic operations init and exit on load and unload time, character device file operations and input processing. The function *ieee80211_hw_pwr* is implemented and exported in the mac80211 module, the dotted arrow at the right signalizes access to it. 43

5.3 LiND daemon: User threads and their dependencies. 46

5.4 Illustration of the *force_power_on* counter on two communicating devices during processing of a reply beacon, a file list request and a file request. The delay msleep(2000) at the end of each accepted HTTP request gives the opportunity to the respective other node to initiate a further TCP/IP request directly after a closed connection. 48

6.1 Circuit diagram of our experiment setup that we used to record different voltages V_{CH1}, V_{CH2} and V_{CH3} for a later calculation of the power consumption of our wireless usb device. Oscilloscope type: Tektronix, TDS 2024B (200MHz, 2GS/s). R=0,108 Ω 52

6.2 We measured the power level value before the operation system and the wireless drivers were loaded. Therefor, we entered the system's BIOS setup. Determination of P_{base} = 0.182 W (+/-) 0.013. 53

6.3 Without deceleration, we switched the wireless hardware on and off. Out of that we get the specific reaction rate time of our LiND_pwr toggle. Simultaneously, we observed the power consumption and determined P_{off} and P_{on}. 54

6.4 The left diagram shows the overall power consumption of the wireless device running in ad hoc mode, the right one shows the power consumption of the ad hoc mode in combination with LiND_pwr using the u-connect neighbor discovery algorithm. 55

6.5 Evaluation of our neighbor discovery implementation using 4 different algorithms: Disco, U-Connect, Grid, and Torus. We analyzed the discovery time in seconds, the amount of send and captured beacons, the time that the wireless device is powered and we count how often the power was switched. We count 'from on to off' and 'from off back to on' as one switching operation. 56

6.6 Sum of new discoveries of the 103 relevant devices per hour between September 2004 and end of April 2005 (8 month) of MIT Reality Mining project traces. 60

6.7 Top 10% of weighted edges between the 103 relevant devices of MIT Reality Mining project traces. Edges are weighted by number of new connections (left) and by connection time in minutes (right) in a time frame of 14 weeks, from 2004-09-07 to 2004-12-16. The more darker a node the more different nodes are discovered within the time frame. 61

6.8 File spread within the first 3 days after a tutor starts offering a file on each Tuesday (from 2004-09-07 to 2004-12-16) at 10 pm on an assumed meeting. The figure shows the dissemination over all devices for each Tuesday in detail. 62

6.9 File spread within the first 3 days after a tutor starts offering a file on each Tuesday (from 2004-09-07 to 2004-12-16) at 10 pm on an assumed meeting. The figure shows us the mean result calculated over all Tuesdays. 63

6.10 The mean file spread within the periods of 1h, 2h, 4h, 8h, 24h and 48h combined with the connection graph of Figure 6.7 (right) is illustrated. Dark tinted nodes of a figure are reached more to the beginning of a denoted period and lighter nodes are more to the end. White nodes are reached above a denoted period, or never. Mean calculation over all Tuesdays is used. 64

List of Tables

4.1 Structure of a beacon and a reply beacon frame. 34

5.1 Source files (#3) of our LiND_pwr kernel module. 43

5.2 Source files (#27) of our LiND daemon. 44

6.1 Initial settings for each quorum algorithm of our experiments due to five different desired maximum discovery time limits. 55

6.2 Discovery time results of our implementation due to the different denoted algorithms and settings listed in Table 6.1. The values are also illustrated in Figure 6.5. Each implemented algorithm fulfills our expectation due to the theoretical worst case value. 57

6.3 Time in seconds that the wireless device is powered during a runtime period of 1 hour for each algorithm and setting. Based on the measured time values, with respect to the runtime length, we denote the powered time additionally in percent. 57

6.4 Evaluation of the reply beacon error rate of two devices in range. A single reply beacon was send out within a random period of 5 ms up to 15 ms after a beacon frame was received. The average error rate is 7.31% with a standard deviation of 2.11%. 58