UAVNET: A PROTOTYPE OF A HIGHLY ADAPTIVE AND MOBILE WIRELESS MESH NETWORK USING UNMANNED AERIAL VEHICLES (UAVS)

Master Thesis
of the Faculty of Science
of the University of Bern

written by

Simon Morgenthaler
2012

Supervisors:
Professor Dr. Torsten Braun
Dr. Thomas Staub
Institute of Computer Science and Applied Mathematics

Abstract

Wireless Mesh Networks (WMNs) are growing in attention and are becoming an important technology to interconnect computer systems and networks in a cheap and flexible way. Many different application scenarios use this new technology, such as monitoring and surveillance systems, the interconnection of different networks and buildings or the fast setup of a temporary communication infrastructure in areas without network coverage. Especially in emergency or disaster recovery scenarios, the immediate deployment of a working communication infrastructure is crucial for saving lives. Unfortunately, after happening of such natural disasters, impassable terrain makes it hard or even impossible to deploy a ground-based WMN.

For this reason, we developed *UAVNet*, a framework focusing on the autonomous deployment of a flying WMN, using small quadrocopter Unmanned Aerial Vehicles (UAVs). Every UAV carries a lightweight wireless mesh node, which is directly connected to the flight electronics of the UAV using a serial interface. The flying wireless mesh nodes are automatically interconnected to each other and are building an IEEE 802.11s WMN. Every wireless mesh node works as an Access Point (AP) and provides access for regular IEEE 802.11g wireless devices, such as notebooks, smartphones, and tablets.

UAVNet includes a concept and a prototype implementation of an autonomously deployable flying WMN. The current prototype is capable of autonomously interconnect two communication peers by setting up an airborne relay, consisting of one or several flying wireless mesh nodes. Furthermore, *UAVNet* includes ideas and concepts to extend the functionality of the prototype. To control the UAVs and to manage the network, the software running on every wireless mesh node follows a decentralised approach. The UAV swarm can, therefore, adapt to current communication needs.

Additionally, an iPad or iPhone, with a customized software running on it, can be used to simplify the configuration, deployment, and monitoring of a *UAVNet*. The complete configuration and automatic deployment process can be executed by using a user-friendly Graphical User Interface (GUI). The deployed network with all of its participants can be displayed and monitored on an interactive map.

We have proven the feasibility of an autonomously deployable, flying WMN using UAVs. The working prototype implementation is not yet able to cover autonomously a defined region, but it includes the basic functionality to setup, deploy and monitor such a network. It is already capable of interconnecting multiple client devices by setting up an airborne relay.

Our evaluations have shown that *UAVNet* can optimize the network on respect of its performance. They have proven that the performance of a flying network is much higher than a ground-based approach, due to the better network coverage.

Acknowledgements

I would like to thank everybody who supported me with ideas, help, encouragement, and motivation during my study and the course of my Master thesis. First, I would like to express my gratitude to Prof. Dr. Torsten Braun, head of the Communication and Distributed Systems group, for giving me the possibility to do this interesting and challenging project in his research group.

Special thanks also go to my thesis advisor Dr. Thomas Staub. He was always willing to listen to my ideas, thoughts and problems and spent much effort in supporting my project. Furthermore, I am very thankful for his assistance in polishing my presentations and this thesis.

Thanks also to the mailing lists ath5k-devel@lists.ath5k.org, devel@lists.open80211s.org, and hostap@lists.shmoo.com, and especially to Wojciech Dubowik who supported me with hints and patches to get the ath5k driver working on our hardware.

I am very grateful to my girlfriend Martina Walker, my entire family and all my friends and colleagues for supporting me in many different ways. My studies would not have been possible without you.

Simon Morgenthaler
May 2012

Contents

Chapter 1

Introduction

Wireless Mesh Networks (WMNs) are growing in attention both in research projects and in commercial and private applications. They provide an efficient and cheap way to deploy large communication networks, as well as to interconnect separated existing networks. Depending on the network scenario and the area of application it can be difficult to deploy and maintain the participating mesh nodes. Additionally, it may be crucial to set up the network in the shortest possible time, e.g. in emergency and disaster recovery scenarios like earthquakes, avalanches and floodings. The deployment and maintenance of current mesh network solutions may be difficult, inefficient and time consuming, or even impossible if the affected area is inaccessible.

This Master thesis introduces and evaluates the framework *UAVNet*, a highly adaptive and mobile WMN using small Unmanned Aerial Vehicles (UAVs). It includes a concept and a prototype implementation of an autonomously deployable temporary WMN, using UAVs with attached wireless mesh nodes. The deployed communication network enables the connectivity between different clients like notebooks, smartphones and tablets and even other wireless or wired networks.

The structure of this introductory chapter is as follows: Section 1.1 gives a short overview of the history of wireless networking, followed by an introduction in WMNs in Section 1.2. After a short description of UAVs in Section 1.3, the motivation for the development of *UAVNet* is discussed in Section 1.4. The chapter concludes with an overview about the remaining chapters of this thesis in Section 1.5.

1.1 History of Wireless Networks

In 1888, the first radio waves were discovered and produced by Heinrich Hertz [1]. During World War II, the United States used the first radio signals for data transmission. The world's first wireless computer communication network, ALOHAnet [2], was developed by Norman Abramson, a professor at the University of Hawaii in 1971. It interconnected six computers on four islands and the central one on the Oahu Island in a bi-directional star topology. Ten years later, amateur radio operators developed the first generation of wireless data modems. In 1990, the 802.11 Working Group was established by the Institute of Electrical and Electronics Engineers (IEEE) 802 Executive Committee to create a Wireless Local Area Network (WLAN)

standard. Finally, the standard data communication format for wireless local area networks, IEEE 802.11, was approved in 1997. Since then, newer standards and amendments have been developed and approved, namely the IEEE standards 802.11a, 802.11b, 802.11g and 802.11n. They all cover the same network scenario, where the clients connect to one or multiple central access points (APs). To establish other network scenarios like so-called ad-hoc networks, which does not need a central management station, IEEE specified the additional Independent Basic Service Set (IBSS) mode. However, these improvements are still not enough to meet the nowadays desired adaptable, mobile and decentralized network scenarios. Therefore, new standards have been developed. Among others, these are IEEE 802.11s (Mesh networking) and IEEE 802.16 (WMAN/WiMAX).

1.2 Wireless Mesh Networks

Nowadays WMNs are growing in attention and are currently the subject of many research projects around the world. They are considered to be a special type of wireless ad-hoc networks. Its main purpose is to avoid the centralized and managed approach of today's common network architectures. Traditional wireless networks consist of one or multiple APs, which manage the entire network, as shown in Figure 1.1. WMNs consist of multiple hosts, which have the same functionality and responsibility in respect of the network topology. This decentralized approach of WMNs (Figure 1.2) is much more flexible and dynamic. It allows the setup of large, inexpensive, reliable, and redundant networks, even in hardly accessible regions. Furthermore, it allows the interconnection of different networks and allows the integration of Wireless Sensor Networks (WSNs). A WMN may involve fixed and mobile nodes. Mobile Ad-hoc Networks (MANETs) combine the mesh topology of WMN with highly mobile and independent nodes, joining, traversing and leaving the network at any time and in any direction.

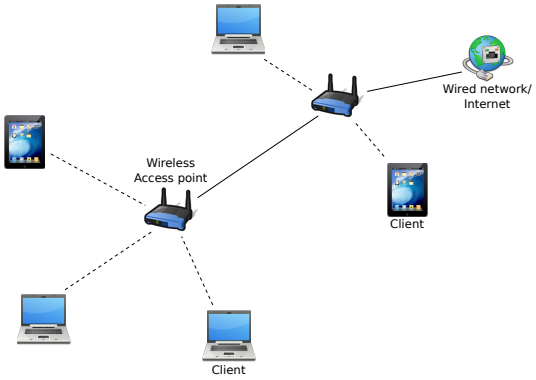

Figure 1.1: Common managed wireless network.

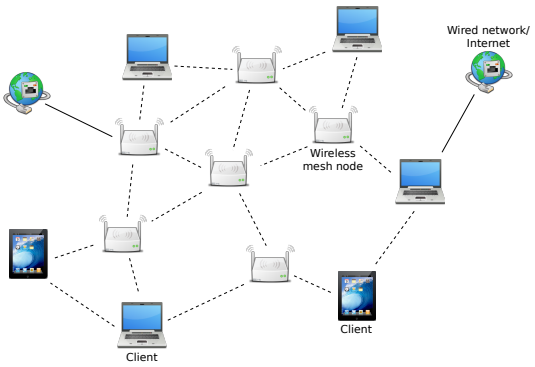

Figure 1.2: Wireless mesh network.

An overview of WMN technology and its applications is provided in [3, 4, 5]. WMNs consist of two different node types: mesh clients and mesh routers. Often the mesh routers are static and are powered by the electricity network. Therefore, they often are equipped with more sophisticated hardware such as multiple radio interfaces and offer bridge and gateway function-

alities. Mesh clients often are mobile devices which connect over multi-hop communications to the WMN. The main characteristics of WMNs are the following [6]:

- Multi-hop wireless communication.
- Ad-hoc networking.
- Self-configuration, self-organisation, self-healing.
- Mobility
- Various types of network interfaces.
- One to multiple radios.

According to [3, 4], WMNs are considered to be a useful communication technology in the following scenarios:

- Metropolitan area networks
- Enterprise networking
- Community and neighbourhood networking
- Broadband home networking
- Building automation
- Health and medical systems
- Surveillance systems
- Emergency/disaster systems
- Transportation systems
- Vehicular networks

The drawbacks of such highly mobile and adaptable network topologies are the need for complex and challenging routing and management algorithms, such as Optimised Link State Routing (OLSR) [7] or Ad-Hoc On-Demand Distance Vector Routing (AODV) [8]. Additionally, mobile devices have limited power resources and need highly optimized software and hardware to reduce the energy consumption.

1.3 Unmanned Aerial Vehicles

Per definition, a UAV is "a powered, aerial vehicle that does not carry a human operator, uses aerodynamic forces to provide vehicle lift, can fly autonomously or be piloted remotely, can be expendable or recoverable, and can carry a lethal or nonlethal payload" [9]. UAVs exist in a lot of different shapes, configurations, sizes and serve for various purposes. A few variations are shown in Figure 1.3. UAVs exist in two different varieties: some are remote controlled, others are flying completely autonomous. Most of them carry a lot of different sensors and cameras, providing data for other applications or used for adjusting current flight parameters.

Author: U.S. Air Force, Public domain

Author: Nicolas Halftermeyer, CC BY 3.0

Author: DARPA, Public domain

Figure 1.3: Different kinds of UAVs.

As well as the WMNs, UAVs are growing in attention. They are used in many different research projects, as well as in private, commercial, and military products. UAVs are used in a wide variety of application scenarios. The majority of the functions they perform are some form of remote sensing and measuring. Common functions they perform among others are the following [10]:

Remote Sensing UAVs may carry a wide variety of sensors, different cameras and other measurement equipment. The gathered data can be stored locally for further analysis or transmitted directly to the base station.

Surveillance and Exploration Using cameras with different spectrum as well as radar systems, allow the exploration and surveillance of large areas and all kind of objects or living creatures.

Transportation Depending on its size and form, the UAV can carry different amount and kind of payload.

Search and Rescue UAVs with attached cameras and other measurement tools are often used for search and rescue missions, as they can fly in inaccessible areas and are much cheaper and safer than standard rescue operations using helicopters and human beings.

Scientific Research UAVs are involved in a wide variety of research projects.

Armed Attacks More and more often UAVs are used by the military for armed attacks of targets in sensitive areas.

1.4 Motivation

Nowadays, networks - communication in general - are becoming more and more important. Sadly, due to the global warming, more and more often, severe first response scenarios such as earthquakes, avalanches, and floodings happen. To rapidly organize and coordinate the rescue forces, a working communication infrastructure is essential. Unfortunately, often the entire communication infrastructure is destroyed during the disaster or was never present in some urban regions. Therefore, it would be very helpful and most-likely lifesaving, if a broadband and reliable communication infrastructure could be deployed quickly and maintained. This temporary infrastructure should be adaptable to different scenarios and should be deployable as fast and easy as possible. The currently existing solutions for such scenarios are often ground-based, which makes it very difficult to establish a working communication infrastructure in a destroyed and inaccessible area. Our proposed solution avoids all these disadvantages: An autonomously deployable and highly adaptable flying WMN could support the rescuers and help them saving lives. The main goals of this study are the following:

- Create a detailed concept how a flying WMN should be designed, developed, deployed and maintained to provide an adaptable, mobile, scalable and robust communication infrastructure. It should be deployable in a fast and easy way, even in inaccessible areas.
- Implement and evaluate different possible network scenarios and topologies. *UAVNet* should cover a wide range of different applications such as a simple connection between two client devices or the coverage of large areas by multiple UAVs. Analyse the feasibility of an implementation of a working prototype within the limits of this thesis.
- Implement a working prototype to show the feasibility of a flying communication network. Keep the prototype as generic and expandable as possible to simplify the development and implementation of future extensions and application scenarios.
- Use common standard software and hardware to keep the system as cheap, compatible and lightweight as possible.
- Use a user-friendly Graphical User Interface (GUI) on a mobile user device to simplify the deployment and monitoring of the network.
- Evaluate the built prototype and the developed concepts and compare them to land-based approaches.

1.5 Thesis Outline

The thesis is structured as follows. In Chapter 2, the work of other researchers is discussed in relation to the developments described in this thesis. This includes underling software and hardware as well as other projects, directly connected to this work. The general architecture and concepts of this Master thesis are presented in Chapter 3. It explains how the entire system works and how the different parts of *UAVNet* are connected to each other. Additionally, it describes the capabilities of the built prototype. The concrete implementation of the system and the prototype is presented in detail in Chapter 4 and 5. In Chapter 6, the prototype of *UAVNet* and different parts of the *UAVNet* system are evaluated by comparing the performance of our approach with other networks and systems. Finally, Chapter 7 concludes the thesis. It proposes also improvements for *UAVNet* and provides an outlook for possible future work.

Chapter 2

Related Work

To develop and implement a complex and large system like *UAVNet*, a lot of different hardware and software components have to interact with each other and fit together. Most of these components have already existed in a more or less usable way, but must be adapted, extended and combined in the right way to build a complete and working system.

This chapter discusses related work and contains background information about the involved hardware and software components. It provides the basic knowledge to understand how *UAVNet* works and how the existing projects have been included, adapted and extended. Section 2.1 presents the used UAV platform. This includes the hardware of the UAV as well as the software running on the flight electronics. The used wireless mesh nodes are described in Section 2.2. The first part of this section covers the hardware, the second part discusses the software running on the nodes. Finally, *Remote Control App*, an other project which builds a remote control client to configure and monitor *UAVNet*, is presented in Section 2.3.

2.1 Quadrocopter

An important part of a flying WMN are the UAVs carrying the wireless mesh nodes. As the flying nodes should stay pretty stable in the air, quadrocopters are used in this work for implementing *UAVNet* and not fixed-wing planes. Due to their moderate costs, good availability, open software, and community support, quadrocopters from the Mikrokopter.de community project [11] have been chosen.

A quadrocopter's principle of flight is as follows: Four brushless motors are mounted in the same horizontal plane at the ends of the frame cross. They are controlled by four brushless controllers and drive the fixed propellers. The front and back rotors turn in opposite directions than the left and right rotor to prevent the torque about the yaw axis (see Figure 2.1). The flight speed and direction are controlled only by increasing and decreasing the speed of the rotors. To fly in a specific direction, the speed of the motor being opposite of the desired flight direction is increased. This gets the quadrocopter in an inclined position and it flies into the desired direction. To turn the quadrocopter around its vertical axis (yaw), the speed of the front and back rotors is increased and the speed of the left and right rotors is decreased.

The following subsections present the used hardware and software components of the quadrocopters in detail.

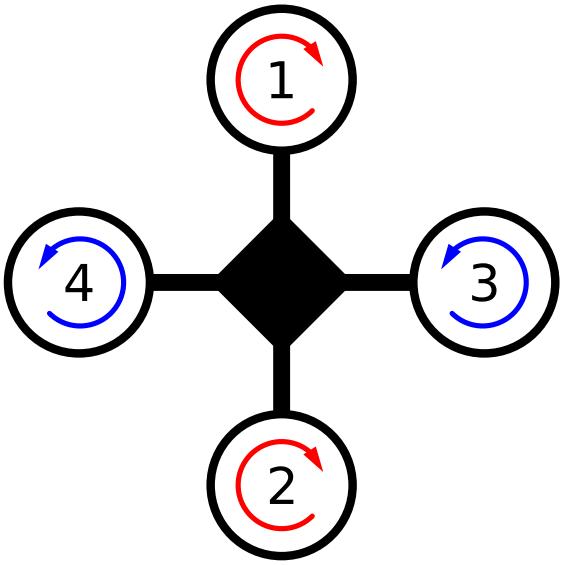

Figure 2.1: A quadrocopter's principle of flight.

2.1.1 Hardware

The UAVs used in the *UAVNet* project are built from self-assembly kits of the Mikrokopter.de project [11]. The assembled quadrocopter is shown in Figure 2.2. It is powered by a 2200 mAh Lithium-ion Polymer (LiPo) battery, which provides energy for about 10-20 minutes, depending on the flight style and the attached payload. The used UAV kit consists of the following components [12]:

- MK Basisset M3 ME
  - 1x preassembled flight controller (FlightCtrl ME 2.0)
  - Altitude sensor
  - 4x preassembled brushless controllers (BL-Ctrl V1.2)
  - 1x switch for power supply
  - 1x Mikrokopter Universal Serial Bus (USB) adapter (MKUSB)
  - 1x Frame set L with anodized (coloured) riggers 1x (red, black 3x)
  - 1x LiPo 2200er/4s
  - 5x pairs of propellers suitable for multicopter setups (e.g. Maxx Products EPP1045)
  - 4x brushless motors (Roxxy 2824-34)
  - 1x connection cable for receiver
  - 1x extensive set of cables
  - Data cable (twisted pair cable - no silicone wire)

  - 4x vibration dampers M3x15
  - Heat shrink tubing for other solder joints
  - 2 pieces of Velcro strips
  - 4 pieces flex Light-emitting Diodes (LED) tape for lighting
  - 100 cable ties (black)
- Wi232-Module + adapter board
- Navigation controller (NaviCtrl V1.1)
- 3-axis compass (MK3Mag)
- Global Positioning System (GPS) receiver using Surface-Mount Device (SMD) technology (MKGPS)
- Remote control transmitter (Graupner MX-16s 35/35B MHz)
- Quartz crystal oscillator (Graupner 35 MHz)
- 3 LEDs (2x in white color, 1x in red color)

Figure 2.2: Assembled Mikrokopter kit.

A standard quadrocopter consists of four main components:

- Frame that carries the flight electronics and motors
- Flight electronics with different components and sensors
- Four brushless motors controlled by four brushless controllers

- Lithium-ion Polymer (LiPo) battery

The flight electronics of the Mikrokopter consists of several different parts (Figure 2.3). First, the *FlightCtrl* controls the speed of the four brushless motors by separate brushless controllers. To stabilize the UAV, the *FlightCtrl* board employs the values of the three integrated rotation speed sensors (gyroscopes), the 3-dimensional acceleration sensor and a height sensor. The second controller board called *NaviCtrl*, provides data from an additional GPS receiver and a 3-dimensional compass.

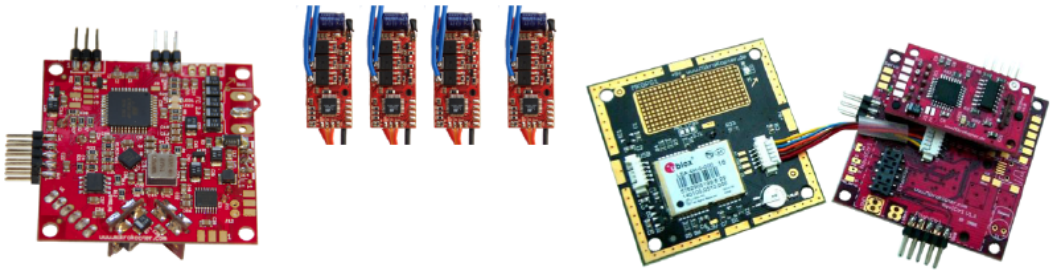

Figure 2.3: Flight electronics of a Mikrokopter (from left to right): the main controller board FlightCtrl, brushless controllers, GPS board and the navigation controller board NaviCtrl with an attached three dimensional compass [6].

Equipped with all these sensors, the UAV is capable of calculating permanently its exact location and position in the air. This ensures a very stable and exact flight. Additionally, the UAV is also capable of holding autonomously its position or even fly on a predefined route using several GPS waypoints, thanks to the GPS receiver.

2.1.2 Software/Firmware

On the flight electronics, different pieces of software and firmware are running. These are software packages for the FlightCtrl board, the NaviCtrl board, the compass MK3Mag, the brushless controllers and the Mikrokopter-Tool. The last one is not running on the Mikrokopter, but is used to flash new software on the UAV and to manage, monitor and configure the quadrocopter using a personal computer. All software is maintained and downloadable from Mikrokopter.de [11]. To keep *UAVNet* compatible, maintainable, and simple, we developed a concept to not alter any software running on the flight electronics. All our custom software runs on the mesh node and communicates with the stock software on the flight electronics using the provided serial communication protocol [13], which is described in detail in Section 5.2.3. Currently, we are running the following software versions on the flight electronics:

- FlightCtrl 0.78b
- NaviCtrl 0.18c
- MK3Mag 0.23a

Newer versions are available but have not been tested yet.

The main functionality of the interface to the flight electronics is the possibility to create, modify, delete and monitor GPS waypoints and locations as well as receiving and interpreting navigation and other useful data over the serial line.

2.2 Wireless Mesh Node

Besides the UAV, the attached mesh nodes are the other main part of *UAVNet*. They are connected with a serial interface to the flight electronics of the UAV and communicate with each other by setting up an IEEE 802.11s WMN between them. Additionally, they are working as an IEEE 802.11g wireless AP to get connected to regular client devices like notebooks, smartphones, tablets, etc.

To be usable in the *UAVNet* project, mesh nodes must meet the following requirements:

- Being capable of setting up an IEEE 802.11s wireless mesh network and an IEEE 802.11g wireless infrastructure network.
- Being as small and lightweight as possible.
- Being powered by a battery.
- Fulfilling the hardware requirements to run our embedded Linux distribution ADAM (Administration and Deployment of Adhoc Mesh Networks) as operating system (OS).
- Consuming low energy.

The following sections provide detailed information about the hardware and the software running on the wireless mesh node.

2.2.1 Hardware

The node used for the *UAVNet* project is the "Professional Mesh OM1P" from Open-Mesh [14] shown in Figure 2.4. It measures 9.5cm x 7cm x 2.5cm and weighs around 86 grams. It is a very low-cost router built on an Atheros AR2315(A) System on Chip (SoC) with a 180 MHz MIPS 4KEc CPU, 32 MB RAM and 8 MB NAND storage. It contains an IEEE 802.11b/g wireless interface (Atheros RF2316), an Ethernet interface (Atheros AR8012), and an internal Universal Asynchronous Receiver/Transmitter (UART) serial port. Additionally, a hardware watchdog is included, which reboots the node if the timer is not reset regularly. It offers recovery if the software crashes. The antenna is a 2.5dbi RP-SMA. The node needs to be powered by 12V DC, which matches the LiPo batteries that we are using for our UAV. Except for the additional hardware watchdog, the OM1P node is built upon the same hardware as the Meraki Mini [15], which is already supported by the ADAM framework. The serial port, which is used for the connection with the flight electronics of the UAV, is not directly accessible from the outside of the node and is not operated on the same voltage level as the remaining UAV flight electronics. Therefore, some slight modifications had to be done, which are described in Chapter 4.

Figure 2.4: Professional Mesh Node OM1P from Open-Mesh [6].

2.2.2 ADAM

The operating system running on the mesh node is ADAM, the embedded Linux distribution developed at University of Bern [6, 16, 17, 18]. ADAM includes a management architecture for MANETs and WMNs. It provides a platform to setup, manage, configure and monitor networks of different sizes. It contains a decentralized distribution mechanism for software updates and remote network configurations. This avoids costly on-site reconfigurations and repairs. Its self-healing mechanisms ensure a stable and safe network.

The nodes fetch (pull) available software or configuration updates from their direct neighbors. This epidemic distribution mechanism ensures the deployment of new software and configuration images within the entire network. If a node is down or if it has no connection to the network during an ongoing update distribution, it fetches the new software or configuration automatically as soon as it is reintegrated in the network. ADAM is very flexible by using a modular approach, including full Internet Protocol Version 4 (IPv4) and Internet Protocol Version 6 (IPv6) support. To guarantee a permanently working network, ADAM provides fall back mechanisms that recover the node, even from a faulty software or configuration update.

ADAM does not only include a framework to set-up and maintain a heterogeneous WMN, but also provides an intuitive, adaptable and simple build system for building an adapted embedded Linux system. It supports a variety of embedded platforms and architectures, such as the WRAP and Alix platforms from PC Engines [19], Meraki and others, by using different build profiles. The included build scripts automate all the steps to cross-compile the appropriate source code and generate an adapted Linux system, optimized for WMNs.

Additionally, ADAM provides an user-friendly web interface for displaying the node status and monitoring the WMN as well as generating and distributing new software and configuration images in the entire network [18]. Figure 2.5 shows the web interface to configure a WMN. It shows all currently deployed nodes, allows the user to generate and deploy a network configuration and to start the automatic distribution of software and configuration image updates. Figure 2.6 depicts the web interface to configure an individual node in the WMN.

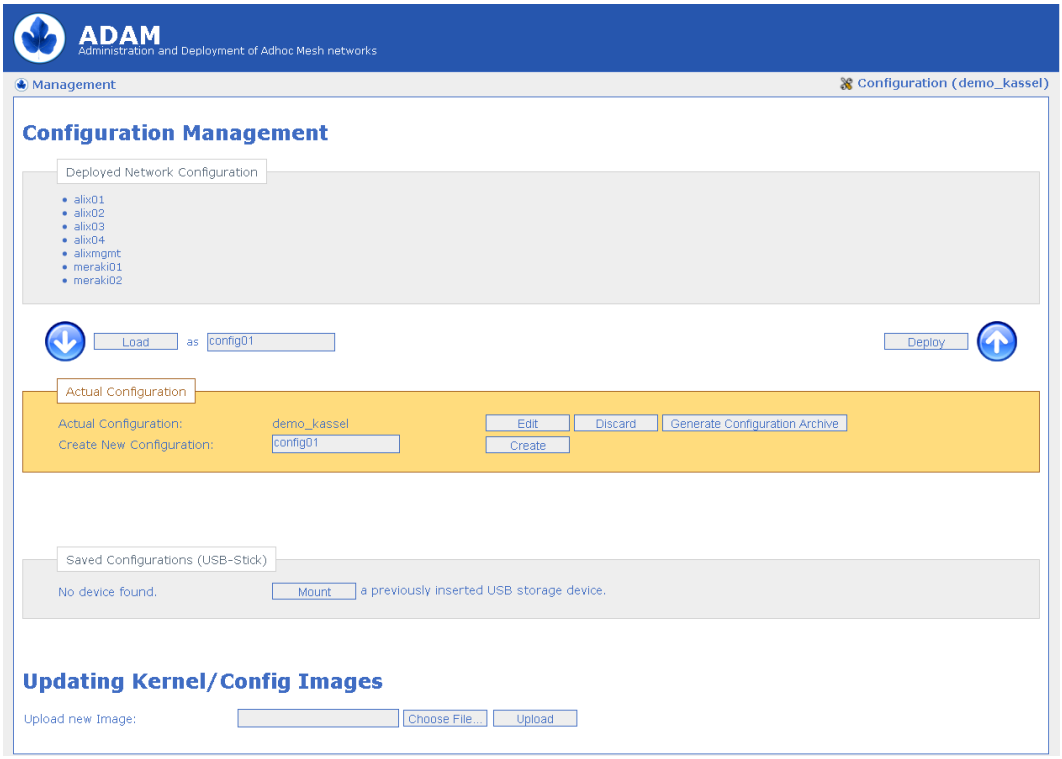

Figure 2.5: ADAM: Network management [6].

To reduce the data transfered in the WMN and, thus, to reduce the probability of network and node failures, ADAM uses separate image files for software and configuration updates. Therefore, only a few KB of data must be distributed in the network, if its configuration changes or an additional node is added to the network. Separating software and configuration data also helps to reduce redundancy, as not a complete image must be distributed for every single node. It is sufficient to deploy one single image file that contains the OS kernel and the binaries for all nodes of a similar type.

In *UAVNet*, ADAM is used to build, configure and set up the mesh nodes attached to the UAVs. The modifications, adaptations and extensions done to ADAM are described in detail in Chapter 5.

2.2.3 Linux kernel / compat-wireless / ath5k

ADAM generates an embedded Linux operating system, which is highly optimized for WMNs and nodes with limited hardware equipment. It uses a Linux kernel which is compiled from the standard sources with an adapted kernel configuration. As all the wireless network drivers, algorithms and protocols are still under heavy development and the AR2315 SoC used in this project is somehow special and not yet supported very well, the Linux kernel used in our project had to be extended with the compat-wireless package [20]. It adds all the latest versions of

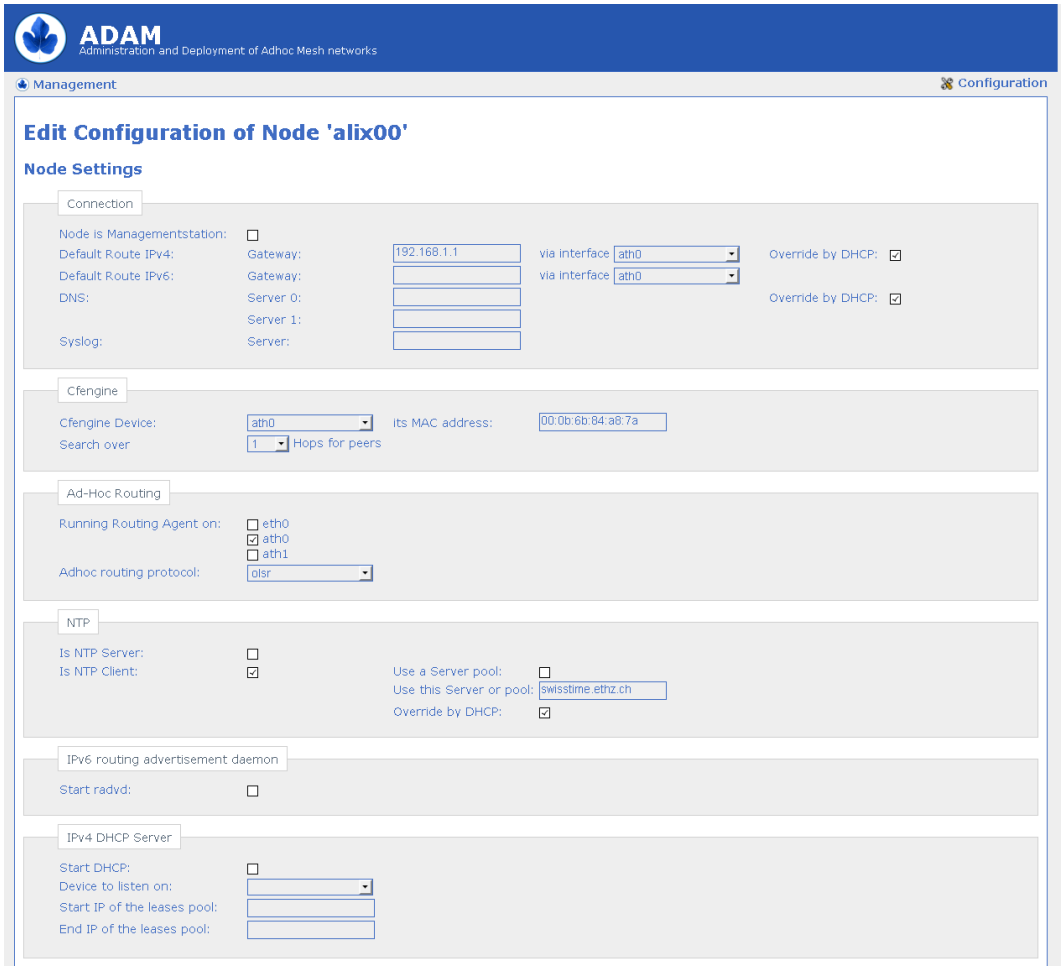

Figure 2.6: ADAM: Node configuration [6].

wireless tools and drivers to a stable and older base kernel. This includes the ath5k wireless driver [21] that replaces the older madwifi project [22], an IEEE 802.11s implementation [23, 24, 25, 26] which is described in the next subsection and other necessary and useful tools like for example *iw* [27].

2.2.4 IEEE 802.11s

As mentioned in Section 1.2, highly adaptable and mobile WMNs need sophisticated routing algorithms to build a performant and robust network. As currently all the existing IEEE 802.11 standards have been developed and approved in earlier days, when centrally managed wireless networks (See Figure 1.1) were state of the art, they are not designed to deal with the fast path and topology changes of the nowadays upcoming ad-hoc networks. Even the additional IBSS extensions are no more sufficient today. Therefore, the new standard IEEE 802.11s has been

started in September 2003 as a Study Group of IEEE 802.11 [28]. IEEE 802.11s is an extension to the IEEE 802.11 standard and is specifically designed to work in current and upcoming network scenarios using mesh networking. It defines how multiple wireless devices can be interconnected in WMNs without the need of a central AP managing the network and works in static, as well as in dynamic ad-hoc networks. Although it is in a preliminary development stage, the IEEE 802.11s draft is already supported by many different products. *open80211s* is a consortium of companies who are sponsoring (and collaborating in) the creation of an open-source implementation of the emerging IEEE 802.11s wireless mesh standard [29]. It claims to "create the first open implementation of 802.11s". As part of the mac80211 layer, a reference implementation of the IEEE 802.11s draft is available in the Linux kernel.

In *UAVNet*, we use the IEEE 802.11s protocol to interconnect the flying mesh nodes with each others. The needed kernel and driver configuration, extensions and modifications are included in the Linux system by using the compat-wireless package, which adds the newest wireless kernel developments to an older stock kernel.

2.2.5 IEEE 802.11g

To get standard clients such as notebooks, smartphones and tablets connected to *UAVNet*, it is not sufficient to provide only an IEEE 802.11s network. Often these clients do not support the new IEEE 802.11s draft yet. Therefore, each node provides also IEEE 802.11g access, which is supported by all modern client devices. To provide this access point functionality, the *hostapd* user space daemon [30] runs on every node. It implements IEEE 802.11 access management, different authenticators like WPA, WPA2, EPA and others and supports different wireless cards and drivers. It works with bridged network interfaces, what is crucial for our combined IEEE 802.11g and IEEE 802.11s network.

2.3 Remote Control App on iOS Devices

To configure, deploy, and monitor the *UAVNet* prototype, the *Remote Control App* has been developed [31]. It runs on iOS devices such as iPhones and iPads. Figure 2.7 shows the *Remote Control App* running on an iPad. The functionality is the same on all devices, the interface adapts to the different screen sizes. It offers three main functions in a convenient and nice looking interface to manage a *UAVNet*:

- Configuring, setting up and deploying one to several UAVs building a *UAVNet* network.
- Monitoring a deployed *UAVNet*, including the involved UAVs and ground based clients.
- Reviewing saved flights and deployments.

2.3.1 Configuring and Deploying the Network

To configure and deploy the *UAVNet*, the user can choose between different possible scenarios, searching and positioning algorithms, which are provided by the *UAVNet* prototype. Possible

Figure 2.7: *Remote Control App* on an iPad: Selection of the deployment scenario [31].

deployment scenarios are *Airborne relay* with one or multiple UAVs, *Area coverage* and *Monitoring*. Right now, the area coverage scenario is not yet supported by our *UAVNet* prototype, but it is already implemented in the *Remote Control App*. The selectable algorithms are either a manual or an autonomous searching mode and a location based or signal strength based positioning algorithm. Figure 2.8 shows the *Remote Control App* on an iPhone. The details of the supported scenarios, positioning, and searching algorithms are described in Chapter 3.

Figure 2.8: *Remote Control App* on an iPhone: Selection of the deployment scenario [31].

2.3.2 Monitoring the Network

Remote Control App is aware of nearby UAVs due to their broadcasted ping messages and is capable to subscribe to their notification service (See Section 3.2.4 and Section 3.2.5 for more details). All the participants of a deployed *UAVNet* are shown on an interactive electronic map. This includes the own position, all UAVs and the involved clients. Additionally, some important UAV data such as battery level, flight direction, speed and height over ground are shown on the map (See Figure 2.9). The map is always automatically rotated towards North direction, thanks to the implemented electronic compass. The electronic map can display online or pre-rendered offline maps from different sources, for example from the OpenStreetMap project [32].

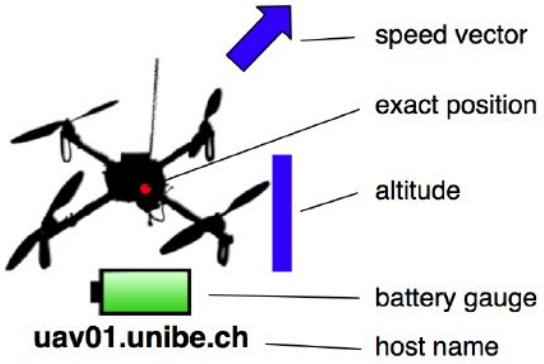

Figure 2.9: Current state of a UAV in the *Remote Control App* [6].

Chapter 3

UAVNet: Architecture and Concepts

UAVNet is a framework, which provides the autonomous deployment of highly adaptive and flexible WMNs. *UAVNet* uses common techniques, established standards and new technologies to build an expandable, robust and highly mobile communication network using UAVs. *UAVNet* is a collaborative project with [6] and [31].

In first response and worst-case scenarios, such as avalanches, earthquakes or floodings, the entire communication infrastructure is often destroyed during such an event, overloaded or nonexistent in outlands. For an efficient and fast rescue it is important to establish a working and robust communication infrastructure as fast and easy as possible. It helps the rescuer to coordinate their rescue efforts and provides the possibility to exchange information and multimedia data among the rescue squads, the coordinator and with the rest of the world. A WMN can provide such a communication network. It is sufficient to distribute some wireless mesh nodes in the affected area. However, as after worst-case scenarios often the terrain is impassable, it is almost impossible to set up a regular ground based communication network in the first phase of a rescue operation within reasonable time.

UAVNet is a prototype implementation of an autonomously deployable WMN using UAVs. The mesh nodes are attached to UAVs, building a kind of "flying mesh nodes". These highly mobile and fast distributed mesh nodes can be flown to sites, that cannot or only hardly be reached by land robots. The distributed network, consisting of several UAVs equipped with wireless mesh nodes, is very mobile and adaptable to different situations. Additionally, the mesh nodes in the air provide a better coverage and availability than mesh nodes placed on the ground.

A disadvantage of UAVs is their pretty high and continuous energy consumption, where land robots can go to sleep when they have reached their final position. A real-live implementation of *UAVNet* has to deal with this drawback and has to implement some kind of replacing and recharging mechanism of the UAVs, which is discussed in more detail in Chapter 7.

This chapter introduces the used architecture for the entire communication network, including clients, wireless mesh nodes and UAVs and concepts to provide solutions for different scenarios, positioning and searching algorithms.

In Section 3.1, the architecture and the different parts of the entire system are discussed. Section 3.2 then shows different concepts and the solutions to make the network as efficient as possible.

3.1 Architecture of UAVNet

UAVNet consists of three different components that are interconnected and communicate with each other. These are clients on the ground, wireless mesh nodes and the UAVs.

3.1.1 Participants of a UAVNet

Participants of *UAVNet* are clients (notebooks, iPhones, iPads, etc.), wireless mesh nodes and the UAVs. The central point of the entire communication infrastructure are the wireless mesh nodes. They are attached to the UAVs and communicate directly with the flight electronics of the UAV over a serial interface. Additionally, they are also connected with the other mesh nodes and the clients over WLAN using the IEEE 802.11g and IEEE 802.11s protocols. The clients are used to configure and monitor the network and the UAVs and can share information and data with other clients using *UAVNet* as the communication network.

Figure 3.1 shows a typical setup of *UAVNet*. Two notebooks use two flying UAVs to communicate with each other. Other clients like iPads and iPhones can be used to configure and monitor the flying network. The dashed arrows symbolize the wireless communication and the solid arrows represent the serial communication.

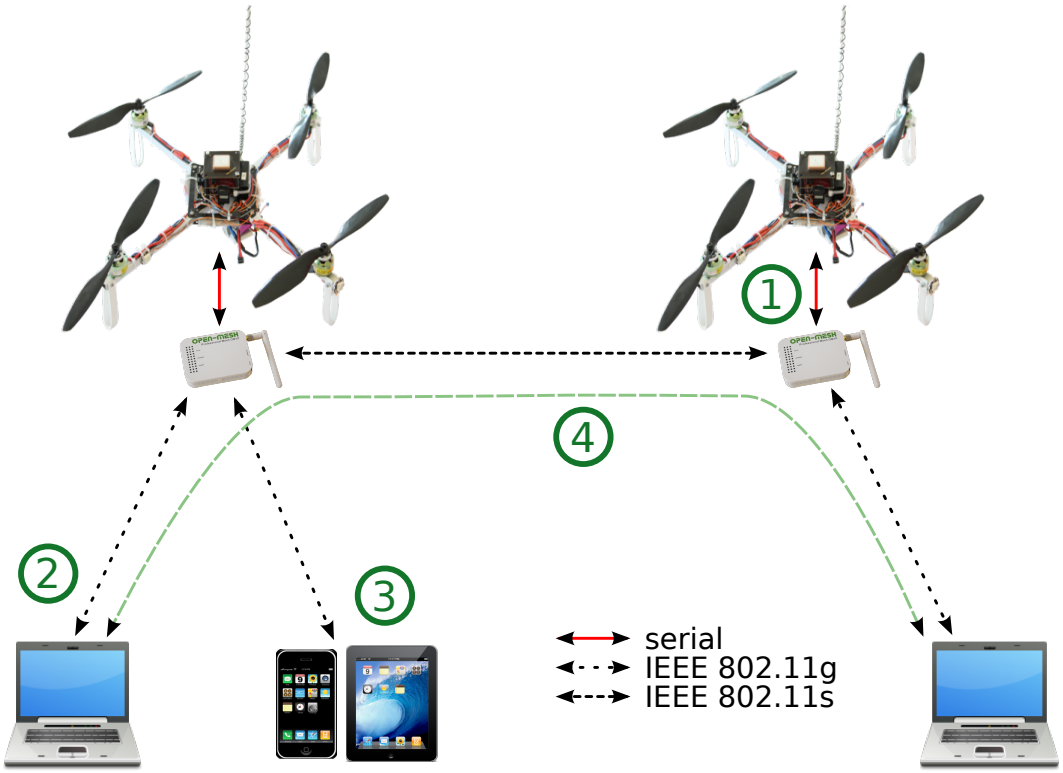

Figure 3.1: Typical setup of *UAVNet* with two UAVs with attached wireless mesh nodes and multiple clients.

1. The wireless mesh nodes are directly connected to the flight electronics of the UAVs using a serial connection.

2. The devices of the rescuers are connected to the mesh network using a standard IEEE 802.11g wireless connection. The wireless mesh nodes act as ordinary APs.

3. Clients like iPhone and iPad are used to configure and monitor the network and its participants. They use also an IEEE 802.11g wireless connection to interact with the *UAVNet*.

4. Traffic between the end devices is forwarded over the IEEE 802.11s wireless mesh network, automatically set up by the UAVs.

3.1.2 Main Components of UAVNet

Figure 3.2 shows the schematic architecture and communication interfaces of *UAVNet*. The dashed arrows symbolize the wireless communication and the solid arrows represent the serial communication.

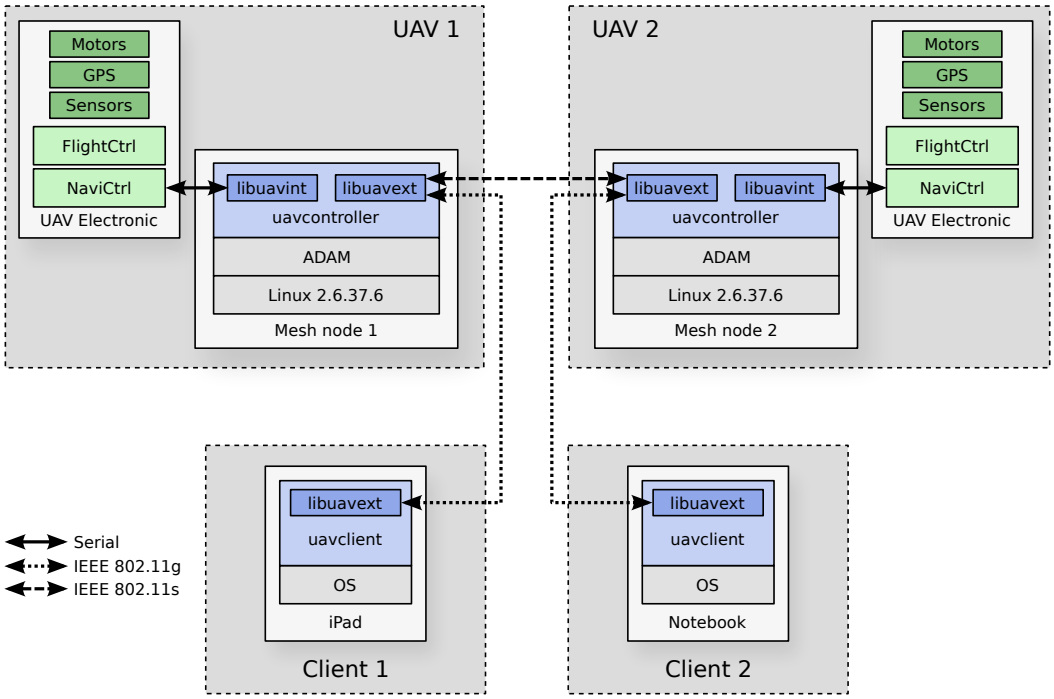

Figure 3.2: Architecture and communication interfaces of *UAVNet*.

The base system software running on the mesh nodes is a Linux 2.6.37.6 kernel [33], extended with the compat-wireless package [20]. The driver from the *ath5k* project [21] is used for the WLAN chip. On top of the kernel ADAM is running. It provides the base system of the mesh nodes. The *uavcontroller* is the software written in C that works as the main component of *UAVNet*. It uses two libraries *libuavext* and *libuavint*. They handle the wireless and

serial interfaces between the different components of *UAVNet*. *libuavint* handles the internal communication between the mesh node and the flight electronics of the UAV using the serial port. *libuavext* handles the external communication between the mesh nodes and the clients using Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) sockets over an IEEE 802.11g and IEEE 802.11s wireless network. On the client (notebooks, iPads or iPhones) is running *uavclient* that uses also the library *libuavext* to handle the communication between the client and the mesh node. In order to develop *UAVNet*, we updated the Linux kernel in the ADAM distribution and extended it with the compat-wireless package, which contains the required WLAN driver and additional protocols. Moreover, ADAM has been extended and the four different pieces of software (*uavcontroller*, *uavclient*, *libuavext* and *libuavint*) have been developed and implemented. All the details of the made modifications, extensions and developments are described in Chapter 5.

3.1.3 Network Configuration

The assignment of IP addresses in *UAVNet* can be separated in two sections. First, all UAVs use static IP addresses. The addresses are defined during the common configuration and setup procedures in ADAM, as described in [17, 16, 18]. Second, the clients such as the notebooks on the ground and the monitoring iPad or iPhone devices, use dynamically assigned IP addresses. On every wireless mesh node runs a Dynamic Host Configuration Protocol (DHCP) server, which allocates IP addresses to the connected clients. This setup ensures on one side a well structured and well defined wireless mesh network. On the other side, the rescuers on the ground do not need to configure their devices in a special way to connect to the network.

The following section describes the different communication interfaces and used protocols and messages.

3.1.4 Communication between UAVNet participants

UAVNet uses two different kinds of interfaces to establish the communication between the UAVs, wireless mesh nodes and clients. The first one is a serial connection between the serial interface of the mesh node and the *NaviCtrl* of the UAV. The second kind of communication is using the IEEE 802.11g and IEEE 802.11s protocol. The mesh nodes are configured as Mesh Access Points (MAPs), which are able to interconnect different kinds of WLANs and also Local Area Networks (LANs). The wireless connection is used by the wireless mesh nodes to communicate with each other and with the clients on the ground. These two interfaces are described in detail in the next sections.

Connection Wireless Mesh Node - UAV

The *uavcontroller* on the wireless mesh node uses *libuavint* to communicate directly with the flight electronics of the UAV using a serial connection.

The *NaviCtrl* is directly connected to the *FlightCtrl* and has a debug port, which is used for the connection with the wireless mesh node. The UAV, or to be more precise, the *NaviCtrl*, sends periodically *NaviData* messages to the mesh node that contain information about its current

position, flight direction and speed and other important data. The *uavcontroller* receives and processes these messages. Afterwards, it sends them to other wireless mesh nodes and clients or it instructs the UAV with new commands.

Using the serial connection, *uavcontroller* is able to send commands to the flight electronics and firmware of the UAV, e.g., the command to fly directly to a specific GPS position.

The details and implementation of the different messages and commands are described in Chapter 5.

Connection Wireless Mesh Node - Wireless Mesh Node

The wireless mesh nodes communicate with each other using a wireless connection and the new IEEE 802.11s mesh protocol. They establish the connection using TCP and UDP sockets, depending on the purpose of the connection. The sockets and the messages are managed by the library *libuavext* which is a part of the *uavcontroller* running on the mesh nodes.

There are two different reasons for the communication between the mesh nodes. The first one is to forward the traffic between two or multiple clients on the ground. The second reason for the connection between the mesh nodes is that the UAVs use these sockets to communicate with each other. The mesh nodes talk to each other and exchange information from the UAVs such as its positions, its flight directions and speeds or the current status. The *uavcontroller* on one wireless mesh node constantly receives data from it's own UAV and also from the other wireless mesh nodes. It uses all this information to act appropriately and send new data and instructions to the electronics of the UAV or the other mesh nodes.

Connection Wireless Mesh Node - Client

The clients communicate with the wireless mesh nodes in the same way as the mesh nodes do it with each other. The sockets are also handled by the library *libuavext* that is also a part of *uavclient*. As most of the clients do not implement the IEEE 802.11s protocol yet, the standard wireless protocol IEEE 802.11g is used for the connection between the clients and the wireless mesh nodes. For the communication between the wireless mesh nodes and the clients, the following different sockets and protocols are used:

- Unicast messages, using a TCP socket on port 7654 for transmitting regular *control* messages.
- Broadcast messages, using a UDP socket on port 7655 for transmitting *ping* messages.
- Unicast messages, using a UDP socket on port 7656 for transmitting *notification* messages.

The details of the implementation of the different sockets and messages are described in Chapter 5.

3.2 Concepts of UAVNet

UAVNet is capable of building up different network scenarios using various positioning and searching algorithms.

UAVNet is built to provide a network infrastructure that includes one to several UAVs and that can be used by multiple different clients. In the following subsections, the different scenarios, positioning, and searching algorithms are described. The details of how the different algorithms are implemented can be found in Chapter 5.

3.2.1 Network Scenarios

UAVNet is capable of establishing networks that provide coverage for two different scenarios. The first one is the *Airborne Relay* scenario where one or several UAVs hover in between two clients to provide a network bridge in the air. The second scenario is the *Area Coverage* scenario, where several UAVs cover autonomously a given area. The *Area Coverage* scenario is not implemented completely in the prototype. In the following the two different scenarios are described in detail.

Airborne Relay Scenario

The *Airborne Relay* scenario offers a solution to establish a wireless connection between two clients (probably two notebooks), which do not get a direct connection, as they are positioned too far away from each other or because there are obstacles between them.

Therefore, one or multiple UAVs autonomously position themselves between the two clients. The attached wireless mesh nodes forward the traffic and thus enable the communication between the two clients. If the two clients are too far away from each other and one UAV is not sufficient to establish a connection, multiple UAVs can build a kind of a forwarder-chain.

When a third client is located in the transmission range, it could provoke a wrong deployment of the UAVs by submitting its own position before the official second client can do it. To prevent this scenario, the user transmits the Media Access Control (MAC) addresses of the two clients participating in the *Airborne Relay* to the UAV, before the deployment process starts. All transmissions from unauthorised clients are ignored then by the UAVs.

In the following paragraphs, the *Airborne Relay* scenario using one or multiple UAV(s) are described in detail.

Airborne Relay Scenario with One UAV

In the fist scenario, one UAV is sufficient to establish a connection between two clients on the ground. The UAV flies autonomously between the two clients and let the attached mesh node forward the traffic between the clients. In Figure 3.3 the *Airborne Relay* scenario with one UAV is shown.

The UAV starts near the first client and begins to broadcast regularly *ping* messages, which are described in detail in Chapter 5. The first client receives these messages and sends its own GPS position to the UAV. Depending on the defined searching algorithm, which is described in

Figure 3.3: *Airborne Relay* scenario with one UAV [31].

detail in Section 3.2.2, the UAV begins to fly in a given direction or on a spiral track around the first client to search the second client. If the second client receives the broadcasted *ping* messages, it sends again its own GPS position to the UAV. Depending on the positioning algorithm, which is described in detail in Section 3.2.3, the *uavcontroller* calculates and measures the position between the two clients and directs the UAV to this calculated center position. When it arrives at its calculated destination, it can be reached by both clients and forward the traffic between them. The details of the set up process using the simple searching and positioning algorithms are shown and described in Figure 3.9 in the Section 3.2.3.

Airborne Relay Scenario with Multiple UAVs

If one UAV is not sufficient to bridge the distance between the two notebooks on the ground, multiple UAVs can be used. They build a chain to forward the traffic between the clients over several wireless mesh nodes. Such an *Airborne Relay* scenario with multiple UAVs is shown in Figure 3.4.

Figure 3.4: *Airborne Relay* scenario with multiple UAVs [31].

The process of setting up such a chain of multiple UAVs is shown in Figures 3.5-3.7 and works as follows:

1. The first starting UAV acts as a kind of a scout spotting the exact position of the second client. Therefore, it uses the same searching algorithms as described in Section 3.2.2 and its behaviour of positioning differs from the succeeding UAVs. As soon as it gets the *position* message from the second client, it broadcasts this position with the *ping* and *notification* messages. This mechanism ensures that all succeeding UAVs already know the location of the second client and the chain is set up in the correct direction.

2. The first UAV then positions itself directly in the middle between the two clients using the *location positioning* algorithm, described in Section 3.2.3.

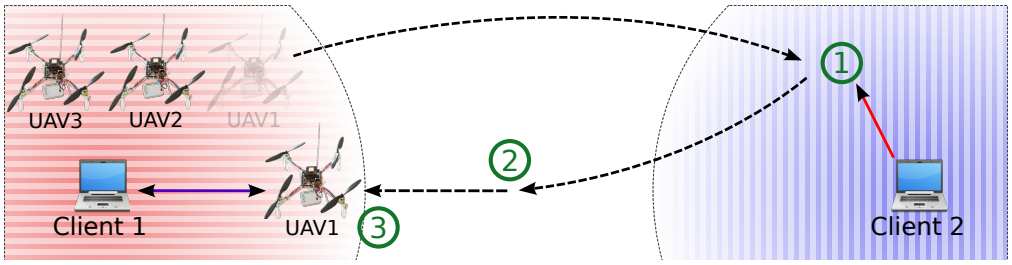

Figure 3.5: *Multi-hop Airborne Relay* scenario: setup UAV 1.

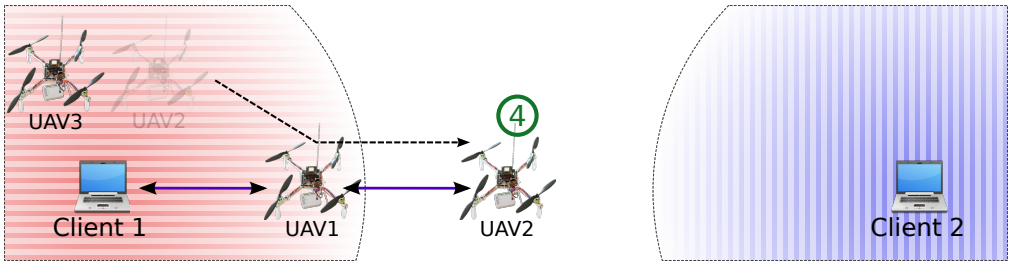

Figure 3.6: *Multi-hop Airborne Relay* scenario: setup UAV 2.

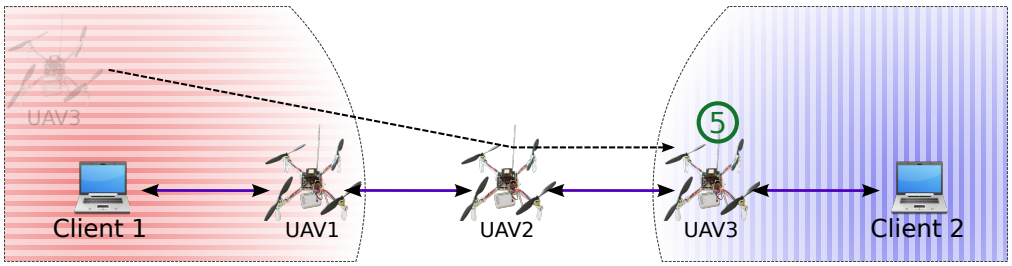

Figure 3.7: *Multi-hop Airborne Relay* scenario: setup UAV 3.

3. When it arrives at this center position, it begins to fly into the direction of the first client until it receives a predefined signal strength (which is explained and evaluated in Section 6.1) from this first client. This location is locked then as the final position of the first UAV. Flying back to the first client and using then the same algorithm as the succeeding UAVs would consume more energy and time and is unnecessary.

4. The second UAV flies directly to the location of the first UAV and moves then into the direction of the second client until it reaches the defined signal strength to the first positioned UAV. This mechanism ensures a straight line of UAVs towards the location of the second client.

5. This is repeated as long as all UAVs are positioned accordingly. This process ensures that the entire chain is built up in the correct direction to ensure optimal connectivity. To avoid collisions, the UAVs are placed at different altitudes. There is no collision avoidance algorithm included in this prototype.

Area Coverage Scenario

This mode is not yet implemented in the prototype, but it is included in the planning and concepts of the different parts of *UAVNet*. It could be implemented in a next step. The idea is to define a given area using a polygon on an interactive map. *UAVNet* positions then autonomously several UAVs to ensure an optimal coverage of the given area. The decision where to place the UAVs depends heavily on the number of available UAVs. If not enough UAVs are available to cover the complete area with a high performance network, some trade-offs between coverage and performance must be made. To deploy such a scenario, sophisticated swarm and collision avoidance algorithms must be implemented. For additional thoughts on this topic see also Section 7.2.

3.2.2 Searching Modes

To ensure an optimal connectivity between the clients on the ground, it is important that *UAVNet* knows its exact locations. To achieve this, the clients transmit their exact GPS coordinates to the UAV. To receive also the position of the second client, the UAVs must get in its transmission range, to receive their sent *position* message. The *uavcontroller* uses two different searching algorithms to find the second client. Both are implemented in the prototype of *UAVNet*.

The first one is the *manual searching* algorithm, in which the first client or the user must tell the UAV in which direction it should fly to find the other client. The second algorithm is the *autonomous searching* mode. Using this mode, the UAV can find the second client without any hint of its location. Both algorithms are implemented and functional in the prototype implementation, as well as in the *Remote Control App*. In the following subsections, these two algorithms are described in detail.

Manual Searching Mode

If the *uavcontroller* has been told to use the *manual searching* mode, it has to know in which direction the second client is located. This information is transmitted from the first client or the mobile device to the UAV using a *flightDirection* message. This information can be transmitted by entering the direction directly on the command line of the client or by touching the approximate location of the second client on the map on an iPhone or iPad. Listing 3.1 shows a typical command to start the *uavclient* on the first notebook. The parameters of the command are described in Table 3.1.

```
> ./uavclient 46.95599 7.43878 359 0.074 1 ath0
```

Listing 3.1: Command to start the *uavclient* on the first notebook.

When the UAV gets the position from the first client, it depends on the configured searching mode how the UAV proceeds. If it is in *manual searching* mode it sends a *getFlightDirection* message back to the client. Then the client transmits in which direction the UAV should fly to find the second client. If the UAV is in *autonomous searching* mode, which is described in the

Parameter	Example value	Description
lat	46.95599	Latitude of the GPS location of the first client
lon	7.43878	Longitude of the GPS location of the first client
dir	359	Direction in degrees to look for the second client
dist	0.074	Maximum distance of the second client in km
conf	1	Defines if *UAVNet* is configured by a mobile user device
iface	ath0	Used WLAN interface

Table 3.1: Description of the parameters of *uavclient*

next subsection in detail, the UAV starts directly to look for the second client without expecting a *flightDirection* message.

When the UAV knows in which direction the second client is located, it starts its flight in the given direction until it gets a *position* message from the second client or until it reaches the defined maximum distance. The details of the message flow of the *manual searching* mode between the clients and the UAV is described in Section 5.2.3

Autonomous Searching Mode

The UAVs of *UAVNet* are able to find the second client autonomously. If the submitted *startConfiguration* message contains the *autonomous searching* mode, the UAV does not request a flight direction. Instead it calculates multiple waypoints lying on a spiral track around the start position. Figure 3.8 shows the concept of the *autonomous searching* mode with two notebooks and one UAV. After the first notebook submitted its position and the start command is given (1), the UAV starts flying on this calculated spiral like route (2) until it gets a *position* message from the second client (3). Then, it positions itself between the two clients (4) as described in Section 3.2.3. There is no risk for collisions in a single-hop airborne relay scenario, because only one UAV is flying in the air. In a multi-hop scenario, the UAVs are placed manually at different altitudes to avoid collisions. The details of the message flow of the *autonomous searching* mode between the clients and the UAV is described in Section 5.2.3.

3.2.3 Positioning Algorithms

UAVNet uses two positioning algorithms to place the UAVs between the clients.

The first one is the *location positioning* mode. It uses the submitted GPS locations of the notebooks and directs the UAV to the exact position between these two GPS coordinates. The second one is the *signal strength positioning* algorithm. It extends the *location positioning* mode and includes also the received signal strength of the two notebooks to calculate a more accurate position for the UAV. This takes the quality of the wireless connection and other environmental influences into consideration. The used mode is transmitted to the UAV at the beginning of the configuration process using the *submitStartConfiguration* message. In the following subsections, these two positioning algorithms are described in detail.

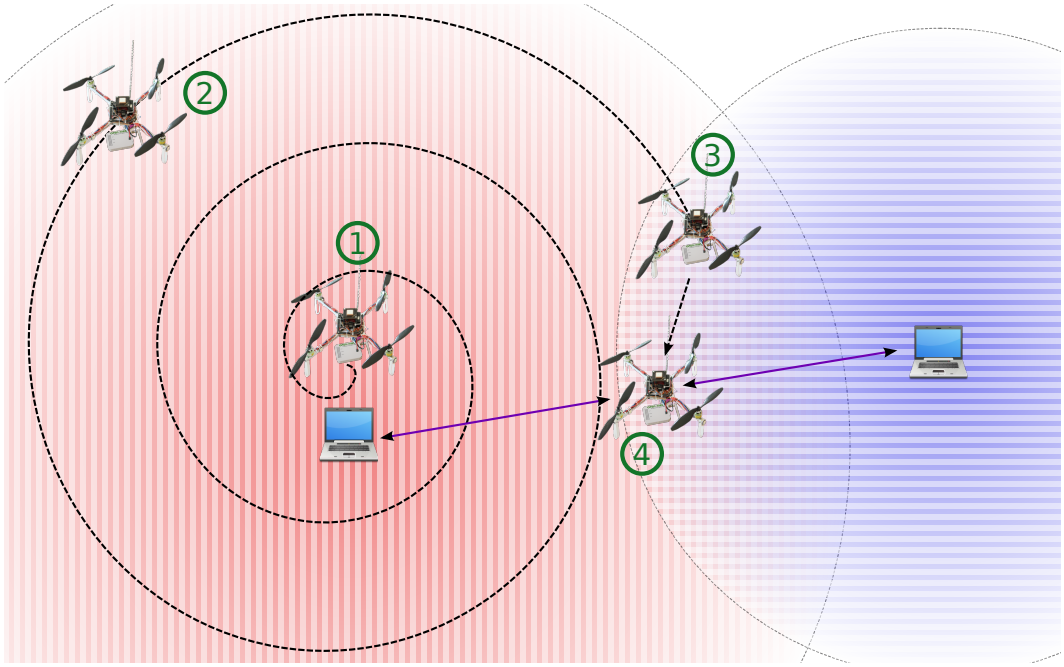

Figure 3.8: Concept of *autonomous searching* mode.

Location Positioning Mode

The *location positioning* mode uses the transmitted GPS locations of the two notebooks to calculate the exact position between them. The calculation of this target location is simply the average of two latitude and longitude values:

$$\begin{aligned} Lat_{center} &= \frac{1}{2}\left(Lat_{pos1} + Lat_{pos2}\right) \\ Lon_{center} &= \frac{1}{2}\left(Lon_{pos1} + Lon_{pos2}\right) \end{aligned} \tag{3.1}$$

Lat_{center}	=	Latitude of center position	Lon_{center}	=	Longitude of center position
Lat_{pos1}	=	Latitude of client 1	Lon_{pos1}	=	Longitude of client 1
Lat_{pos2}	=	Latitude of client 2	Lon_{pos2}	=	Longitude of client 2

The advantage of this algorithm is its simplicity and that the UAV is always positioned exactly between the two notebooks, taking the inaccuracy of GPS signal into account. The drawback is that the quality of the wireless signal and other environmental influences are not considered. It is possible that the wireless signal of the one notebook has a lower range than the one of the other notebook. This can happen due to obstacles, different wireless chips, antennas with different gains or other technical and environmental influences. In this case, the UAV would be positioned in the middle between the two notebooks, but not in the middle of the two wireless coverages. That would result in a worse connection quality than possible.

Figure 3.9 shows the positioning process of the UAV between two notebooks with the same transmission range using the *location positioning* mode.

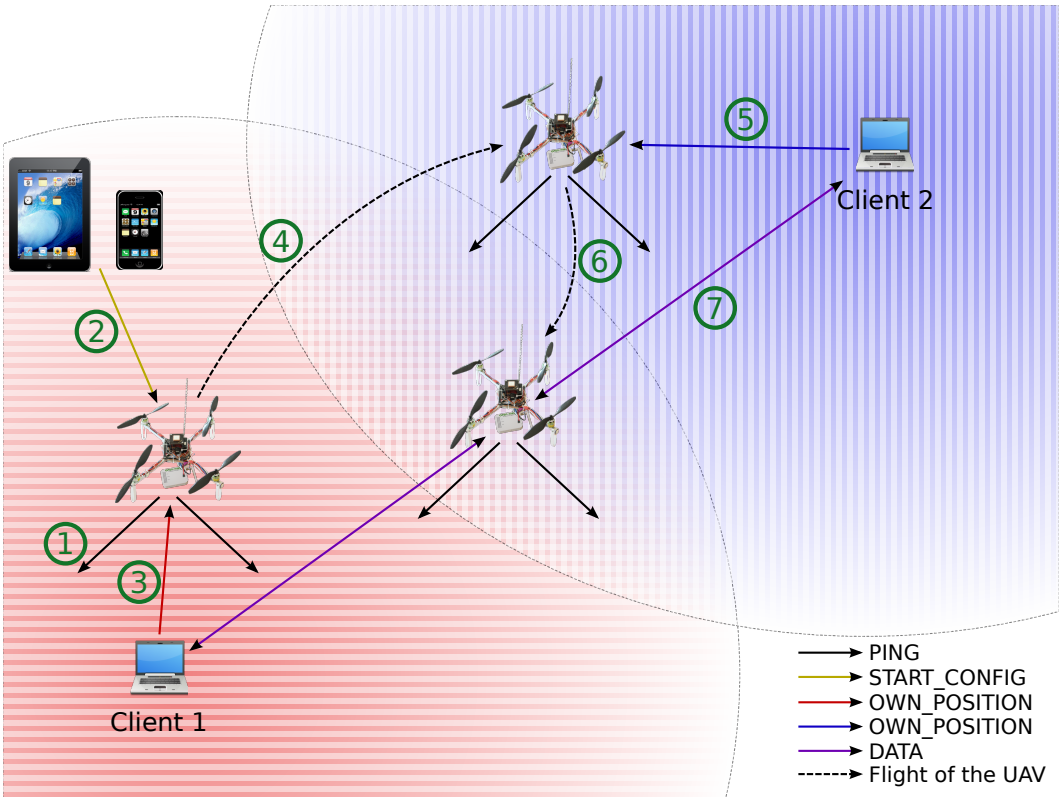

Figure 3.9: Procedure of the *Location Positioning* mode.

The deployment process of *UAVNet* using the *location positioning* and *manual searching* mode consists of the following seven numbered steps, shown in Figure 3.9:

1. When the UAV and the mesh node is switched on and booted up, it announces its presence by sending periodically a *ping* message.

2. A remote control client discovers the presence of the UAV by receiving that message. The user selects then the deployment scenario and modes and transmits a *submitStartConfiguration* to the UAV. Additionally, it defines the involved clients and it can subscribe to the notification service.

3. Client 1 also receives the *ping* messages and transmits its position to the UAV. Furthermore, it transmits also the direction in which the UAV should fly to discover client 2.

4. The UAV now searches for the second client by flying into a predefined direction.

5. When client 2 receives the *ping* messages, it transmits its own GPS position to the UAV.

6. The UAV now positions itself between the two client.

7. Both clients can now communicate with each other, using the UAV as an airborne relay.

Signal Strength Positioning Mode

To overcome the drawbacks of the *location positioning* algorithm, the UAV can be configured to use the received signal strength for positioning (*signal strength positioning* mode). It is an extension of the *location positioning* algorithm that considers also the quality of the wireless signal to calculate the optimal location between the clients for the UAV.

The beginning of the positioning process is the same as in the *location positioning* mode, shown in Figure 3.9. When the UAV is located exactly between the two notebooks using the GPS coordinations as described in Section 3.2.3, the extended functionality of the *signal strength positioning* algorithm jumps in. As the UAV is positioned in the middle of a direct connection line between the two notebooks, it is sufficient to direct the UAV towards the first or the second notebook, depending on their signal strengths. Therefore, the signal strength values of both notebooks are continuously monitored. The best location is reached when both signal strength values are the same. To avoid permanent back and forth movements of the UAV, possibly wrongly measured values are ignored and a threshold value is introduced. It is used to calculate the range in which the measured signal strength values are treated as equal and, therefore, no further positioning adjustments happen.

Listing 3.2 shows the simplified pseudo code of the *signal strength positioning* algorithm. S1 and S2 are the signal strengths of client 1 and 2, h is the signal strength threshold, evaluated in Section 6.1.

```
while (true) {
  if(S1 < S2 - h) {
    Flight one meter in direction of Client 1
  } else if(S1 - h > S2) {
    Flight one meter in direction of Client 2
  } else {
    Stay where you are
  }
}
```

Listing 3.2: Pseudo code of the *signal strength positioning* algorithm.

Figure 3.10 shows the positioning process of the UAV between two notebooks with different transmission ranges using the *signal strength positioning* mode.

3.2.4 Presence Announcement of the UAVs

UAVNet is designed to operate in large areas with not always having direct connections between the participant mesh nodes and devices. Therefore, it is important that the UAVs announce their presence and submit some state information about the network and their own status. To

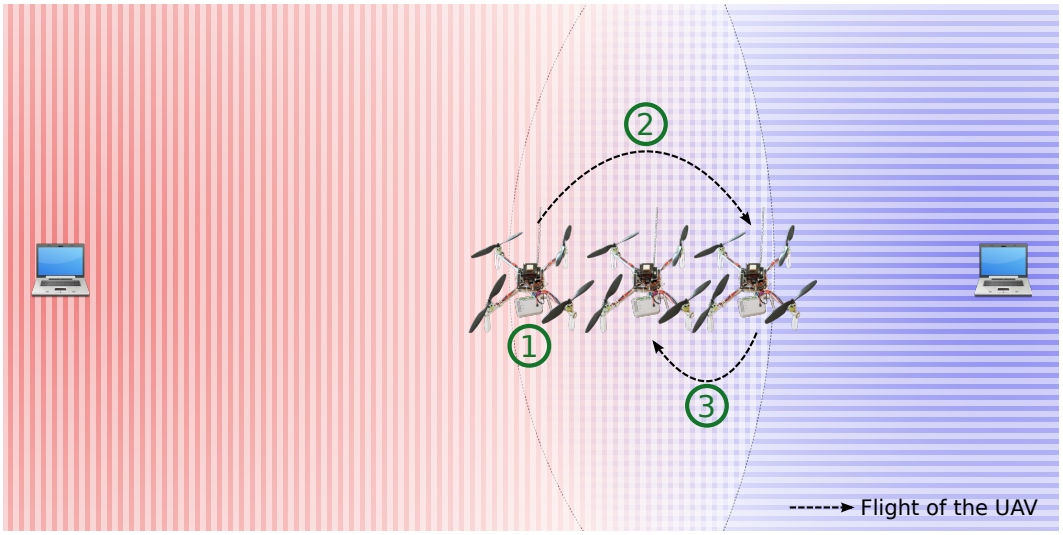

Figure 3.10: Procedure of the *Signal Strength Positioning* mode.

accomplish this, every UAV periodically broadcasts a *ping* message every few seconds. The *ping* message can be received by all other participants like UAVs, smartphones, notebooks, etc... It contains some data about the network and the UAV. The details can be found in Chapter 5. By receiving such a message, the devices know that a UAV is in transmission range. In addition, the status and the network state are announced.

3.2.5 Notifications

To monitor the entire network, *UAVNet* contains a notification service that provides information about the UAVs and the locations of the clients. It sends data about the UAVs and the involved clients to every device that explicitly subscribed to the service.

UAVNet can be monitored using an iPhone or iPad with the *Remote Control App* [31], which is described in more detail in Section 2.3. The notification function works as a subscription service. When the monitoring iPhone or iPad receives a *ping* message of a UAV, it can send a *notificationSubscription* message to the UAV. From that moment the UAV sends periodically *notification* messages to the subscribed client. The *notification* message contains data about the positions of the different notebooks and information like the position, flight direction, battery status, etc. of the UAV. The provided data can be used to draw the positions and status of the different components of a *UAVNet* on an interactive map. The software on the iPhone and iPad is capable of handling and displaying *notification* messages from several different UAVs. This allows the monitoring of a complete *UAVNet* with several UAVs. Figure 3.11 shows a screenshot of such a map, showing different information about the UAV and the clients. The message flow of the notification subscription and the sent *notification* messages is described in detail in Section 5.2.3.

Figure 3.11: Screenshot of a map showing *UAVNet* [31].

3.2.6 Start Procedure

The current prototype implementation of *UAVNet* does not contain any kind of autonomous starting and landing procedure. The UAVs must be started manually and are placed at the desired altitude before the autonomous network deployment can be started. Figure 3.12 shows the remote control unit (RCU) of the UAV.

The complete start procedure works as follows:

- The mesh node and the UAV are switched on. The flight electronics of the UAV need about ten seconds to boot up, the mesh node takes up to one minute. The mesh node has booted completely when the WLAN LED are turned on.
- First, the UAV needs an accurate GPS location, with signals from at least six satellites. Therefore, the GPS switch on the RCU has to be set to “hc:on gps:ch” ((3) in Figure 3.12) until the UAV stops beeping. Then the switch is set back to “hc:off gps:off” (1).
- The motors are started and the UAV can fly manually to the desired start location and

Figure 3.12: Remote Control Unit of the UAV.

altitude.

- When the UAV reaches its start location, the switch on the remote control unit is set to "hc:on gps:hp" (2), which keeps the UAV autonomously at the current position
- When the network is configured using the *Remote Control App* on the iPad and the network deployment should start, the switch is set to "hc:on gps:ch" (3).

Additional information on the calibration, starting, and GPS topic can be found in the Mikrokopter.de wiki [34, 35, 36].

3.2.7 Simulation Script

The *uavcontroller* provides a simulation mode that can be used to test the monitoring client without the need of a real flying UAV.

To test the monitoring client such as an iPhone or iPad the *uavcontroller* can be started in the simulation mode. This mode uses simulated GPS locations and data about the UAV that are defined in a separate file. To start the *uavcontroller* in the simulation mode the *-s* flag must be added to the start command:

```
> ./uavcontroller -s
```

Listing 3.3: Starting *uavcontroller* in the simulation mode.

Chapter 4

UAVNet: Hardware Implementation

In the previous chapter, the general concepts and the architecture of *UAVNet* were introduced. The different components of *UAVNet* were described and the functionality of the entire network as well as the individual components were shown. This and the following chapter cover the implementation details. This includes the modifications on the involved hardware such as the UAV and the mesh nodes as well as the detailed implementation and adaptations on the software running on the mesh nodes and the clients.

To implement *UAVNet* several different hardware components are used. A minimal *UAVNet* uses at least one flying UAV. It carries an attached small and lightweight wireless mesh node, providing the WLAN connection between two clients such as notebooks, smartphones and tablets. Additionally, an iPhone or iPad can be used to configure the network and monitor the complete *UAVNet*, including the UAV in the air and the clients on the ground.

The structure of this chapter is as follows: These four sections, Section 4.1 - 4.4, describe hardware related components and modifications, used to develop a "flying wireless mesh node". This includes the UAV as well as the attached mesh nodes and the serial connection between the flight electronics of the UAV and the mesh node and the clients on the ground.

4.1 Unmanned Aerial Vehicle

The most important components of *UAVNet* are the UAVs carrying the wireless mesh nodes. They should be equipped with all the required electronics such as GPS, compass, acceleration sensors, gyroscopes, height sensors, brushless motors and controllers. Additionally, the UAVs should be agile, but also fly pretty stable in the air and, therefore, be not too heavy.

The following components are used to implement a complete "flying wireless mesh node":

- 1x UAV (Mikrokopter)
- 1x OM1P wireless mesh node
- 1-2x 2200mAh/11.1V LiPo
- 1x 1250mAh/3.7V LiPo for lighting

The UAVs used for this project are based on the Mikrokopter.de platform. The Mikrokopter was assembled by ourselves, using the Mikrokopter.de kits. All the instructions how to assemble the different parts as well as other useful tips, hints and configuration instructions can be found in the wiki and the forum of Mikrokopter.de [11]. The assembly took some time, but was straight forward with the mentioned online instructions. To test the different *UAVNet* scenarios, three UAVs have been built.

To increase the visibility of the position and location of the UAV in the air, several LEDs in different colors are attached to the ends of the riggers of the quadrocopter. The Mikrokopter.de platform proposes several different lighting schemes on [37]. The lighting used in this project is shown in Figure 4.1. To reduce the energy consumption, only two white and one red LEDs are attached to the UAV. This is sufficient, as no manual controlling of the UAV is required. The LEDs are powered by an additional 3.7V LiPo battery.

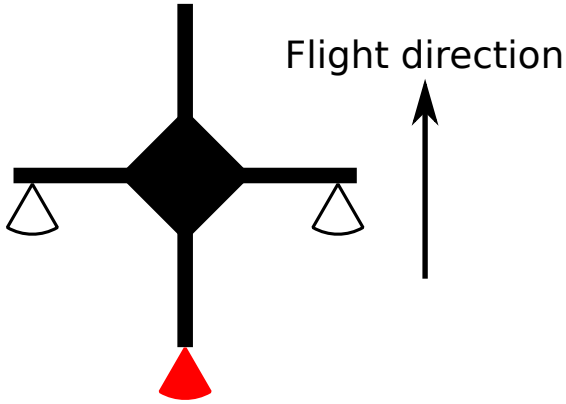

Figure 4.1: Used lighting scheme of a UAV.

4.2 Mesh Node

The "Open-Mesh Professional Mini" is a small, lightweight and low-cost wireless mesh router [38]. It is equipped with a Atheros AR2315(A) SoC with a 180MHz MIPS 4KEc CPU, 32MB RAM and 8MB NAND storage. It has an Ethernet interface (Atheros AR8012), an IEEE 802.11b/g wireless interface (Atheros RF2316) and an internal serial interface UART. Additionally the OM1P is equipped with an onboard hardware watchdog chip to ensure a higher reliability.

As the OM1P operates with a voltage between 6V and 18V, it can be powered directly with the same LiPo battery that is used for the UAV. The drawback of using the same battery to power the UAV and the mesh node is the high energy consumption of the UAV. This means, that also the mesh node becomes unpowered, if the UAV drained the battery. To avoid this, we used two separate batteries to power the UAV and the mesh node.

Power provisioning for the wireless mesh node required some customized battery cables. They are shown in Figure 4.2. There are different configurations, using different types of con-

nectors like “Dean plug” [39], “DC power connector” [40], and “Banana connector” [41] and also a cable to connect a single battery to both, the UAV and the mesh node.

Figure 4.2: Different cables to provision electric power to the UAV and the mesh node.

4.3 Connection UAV - OM1P

The connection between the electronics of the UAV and the OM1P is implemented by using a serial connection between the UART interface on the OM1P and the debug port on the *NaviCtrl* of the flight electronics of the mikrokopter. As the serial interface on the wireless mesh node works with 3.3V and the *NaviCtrl* operates on 5V, a logic level converter is used to establish the connection [42]. It converts signals using 5V base voltage level to signals using 3.3V base voltage level and vice versa. This enables the communication between these two components. Figure 4.3 shows how the Logic Level Converter is used to connect the serial interface of the mesh node with the flight electronics. The red circles indicate the Logic Level Converters. On the left side, the Logic Level Converter is presented on its own. In the middle it is shown how the Logic Level Converter was connected during the development and tests. On the right side the Logic Level Converter is shown fully integrated into the mesh node case of the running *UAVNet* prototype.

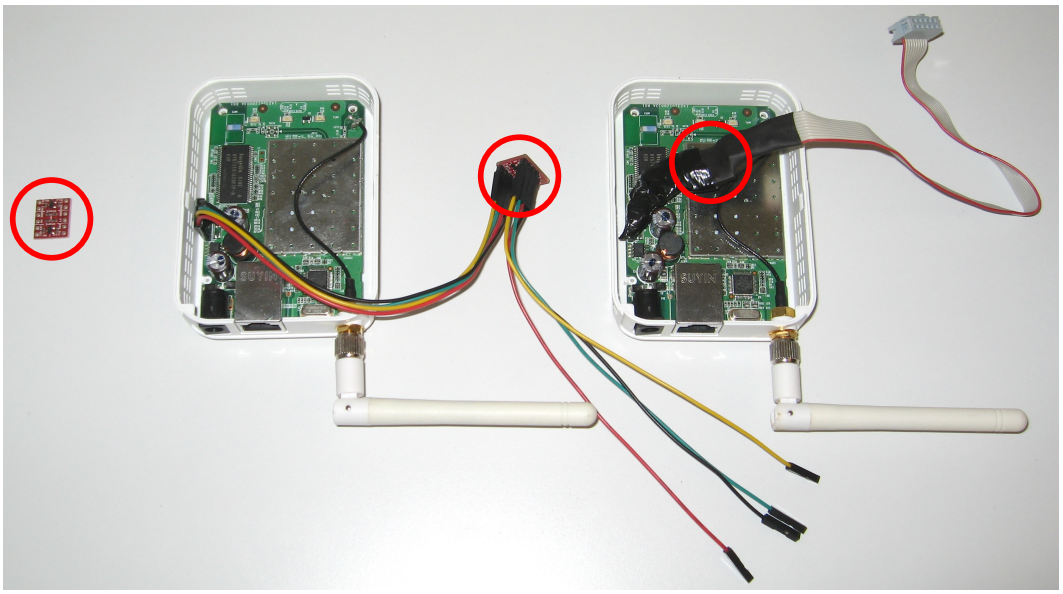

Figure 4.3: Logic Level Converter during the development phase and in the final prototype implementation.

4.4 Client (Notebook/iPhone/iPad)

UAVNet is able to interconnect different types of wireless devices, such as notebooks, smartphones and tablets. The hardware of these client devices does not need to be modified. They simply need a configured and working wireless interface to connect to the *UAVNet* and the appropriate software, which is described in the following chapter.

Chapter 5

UAVNet: Software Implementation

In the previous chapter, hardware related implementations, adaptations, and modifications were described. This chapter covers the software related work.

The software of *UAVNet* consists of different components running on the mesh nodes and on the clients. To guarantee the highest possible compatibility, the software on the flight electronics of the UAV has not been modified or extended. The original software from the mikrokopter.de project can be installed and used on the flight electronics of the UAV. The versions used in this project can be found in Section 2.1. This firmware provides an interface for other hardware and software components to communicate directly with the *FlightCtrl* and *NaviCtrl* and is used by *UAVNet* to implement its functionality [13].

The following sections, Section 5.1 - 5.2, cover software related implementations. This includes the complete *UAVNet* software architecture, the communication protocol, the modifications on ADAM and the Linux system, as well as the used configuration tools and other involved software packages.

5.1 Wireless Extensions of ADAM

A lot of modifications and additions to the software on the node have been done to meet the defined requirements. At first, ADAM had to be slightly adopted to support the new node type. As the Meraki node, which uses almost the same hardware as the OM1P node, is already supported by ADAM, only a few modifications had to be made.

In order to use the new IEEE 802.11s protocol as routing mechanism in the mesh network, ADAM had to be extended by several different components.

First, the Linux kernel had to be updated. Nowadays, the Linux kernel 2.6.37.6 is used [33]. To ensure that the newest possible WLAN drivers can be used, also the compat-wireless packet is configured and installed, currently the version 2011-12-24 [20]. Second, the old madwifi WLAN driver [22] is replaced by the new ath5k driver [21], which supports also the AR2315 SoC now [43]. Additionally, the software iw [27] was updated and installed to configure the mesh network and wireless interfaces. All the drivers, protocols and tools are still under development. It is possible that future updates will provide even better performance and stability. To prevent the mesh node from rebooting every five minutes due to a not reset hardware watchdog, an

additional kernel module has to be installed. This module generates a /proc/gpio device that can be triggered by crontab to reset the watchdog. A new version of *hostapd* [30] is used to set up a standard IEEE 802.11b/g AP, for clients that do not support IEEE 802.11s yet.

5.1.1 Wireless Mesh Network - IEEE 802.11s

To build up the mesh network and manage the communication between the flying mesh nodes, the new IEEE 802.11s standard is used. It is still in a preliminary development stage, but already supported by several soft- and hardware products and projects. As part of the *mac80211* layer, a reference implementation of the IEEE 802.11s draft is included in the Linux kernel.

The IEEE 802.11s standard is included in the *mac80211* kernel module [44]. This is a framework that can be used by driver developers to write drivers for wireless devices. The kernel module *mac80211* uses the underlying new Linux wireless configuration Application Programming Interface (API) *cfg80211* [45]. This API replaces the *Wireless-Extensions* [46] and uses *nl80211* [45] to configure *cfg80211* devices.

These new modules are included in the ADAM build system using the compat-wireless package to always stay up-to-date with the latest driver developments.

5.1.2 The Wireless Driver - ath5k

Ath5k is a new wireless driver for Atheros based wireless chipsets [21]. It has evolved out of MadWiFi [22], OpenHAL [47], and the open-sourced HAL code of Atheros and Sam Leffler [48]. Recently ath5k has been extended to support also the Atheros AR2315 SoC [43]. However, it is still under development and not everything works smoothly until now. The used chipset AR2315 seems to be a little different than other Atheros chipsets and needs sometimes special treatment in the code of ath5k.

Although the Atheros chipsets should be supported by the ath5k driver, the first tries to get this driver work with our AR2315 SoC failed completely. No wireless device was created. Only after applying a few special patches from the ath5k development community, especially from Wojciech Dubowik, the driver provides the full functionality now [43]. However, there still seems to exist some incompatibilities with this chipset.

Ath5k is compiled as a kernel module staying on top of the module chain and depends on the modules *ath*, *mac80211*, *cfg80211* and *compat*.

5.1.3 Bridging Wireless Interfaces

To forward the traffic between the clients over the IEEE 802.11s network, a bridging interface is set up on the nodes. It forwards the traffic from the IEEE 802.11b/g to the IEEE 802.11s network and vice versa.

On every node, two virtual wireless interfaces (VIF) are configured. First, there is an interface called *mesh0* that is participant of the mesh network using the IEEE 802.11s protocol. The second interface is called *wlan0* and handles the IEEE 802.11g traffic between the wireless mesh node and the client on the ground. *hostapd* uses this second interface to set up its access point functionality. A bridge interface called *br0* bridges the VIFs *mesh0* and *wlan0* to forward

the traffic between these two different networks. Figure 5.1 shows a schematic presentation of the two bridged wireless interfaces.

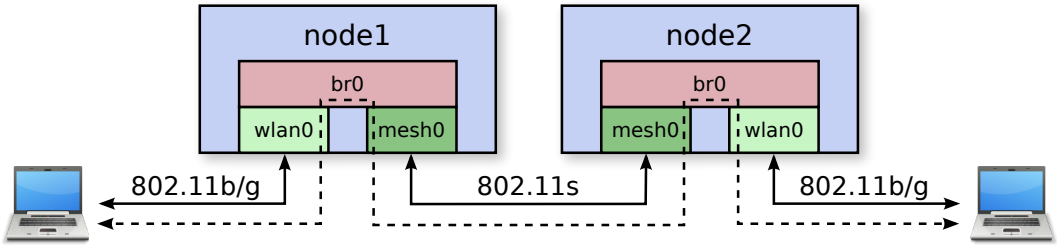

Figure 5.1: Bridged wireless interfaces.

5.1.4 Configuration of Wireless Devices - iw

To configure the wireless device and the network, *iw*, the new *nl80211* based command-line interface (CLI) configuration utility for wireless devices, is used [27]. It is still under development and replaces in ADAM the older tool *iwconfig* from the *Wireless Tools for Linux* package [49]. Currently, the version 0.9.20 is used in *UAVNet*, but newer versions are available.

The tool *iw* is used to set up the network devices, to configure it as mesh point and a wireless AP, to configure the channels and mesh id and to apply other configuration parameters. It is included in the ADAM build and configuration process and is called automatically when the mesh node starts up. To configure the devices, *iw* uses the configuration values from the *network.conf* file, introduced in ADAM. A possible configuration of such a setup with two wireless interfaces, one bridge interface and a running *hostapd* is shown in Listing 5.1.

```
> iw phy phy0 interface add wlan0 type __ap
> iw phy phy0 interface add mesh0 type mp mesh_id uavmesh
> brctl addbr br0
> ip link set down dev wlan0
> ip link set down dev mesh0
> iw dev mesh0 set channel 1
> ip link set address 00:12:cf:d2:dc:23 dev wlan0
> ip link set address 00:12:cf:d2:dc:24 dev mesh0
> brctl addif br0 wlan0
> brctl addif br0 mesh0
> ip addr add 0.0.0.0/0 dev mesh0
> ip addr add 0.0.0.0/0 dev wlan0
> ip link set address 00:12:cf:d2:dc:24 dev br0
> ip link set dev mesh0 up
> ip addr add 192.168.230.13/24 dev br0
> ip link set up dev br0
> hostapd -B /etc/hostapd.conf
```

Listing 5.1: Setting up two wireless interfaces combined by a bridge and a running *hostapd*.

5.1.5 Infrastructure/Managed Wireless Network - hostapd

As not every client supports the new IEEE 802.11s protocol, a standard IEEE 802.11b/g AP is set up on the nodes. Clients can connect to it by using the published Service Set Identifier (SSID) “uavmesh”.

Setting up the AP is done by *hostapd* [30], an IEEE 802.11 AP and IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator. It runs on every node in the mesh network to guarantee a highest possible connection availability and best performance. As this is just a prototype, currently the network is protected only with a WPA2 encryption and no further authentication or security is included. Activating additional security features seems to cause no big problems. An exemplary running *hostapd* configuration is shown in Listing 5.2.

```
interface=wlan0
bridge=br0
driver=nl80211
ssid=uavwlan
hw_mode=g
channel=1
ctrl_interface=/var/run/hostapd
ctrl_interface_group=0
ignore_broadcast_ssid=0
dump_file=/root/hostapd.dump
logger_syslog=-1
logger_syslog_level=0
logger_stdout=-1
logger_stdout_level=0
wpa=1
wpa_passphrase=asdfasdf
wpa_key_mgmt=WPA-PSK
wpa_pairwise=CCMP TKIP
eapol_version=1
auth_algs=3
eapol_key_index_workaround=0
eap_server=0
```

Listing 5.2: Exemplary *hostapd.conf*.

5.1.6 Network configuration - network.conf

As mentioned before, the VIFs are configured and set up according to values stored in the *network.conf* file. This simplifies the process of configuring multiple devices and the reconfiguration of the network. Additionally, all the common processes and *cfengine* distribution mechanisms of ADAM can be used to build the images and update existing networks. To make this possible, the *network.conf* file has been extended by some additional configuration values. The newly introduced variables are described in Table 5.1.

Variable	Description	Default Value
UAVNET_BRIDGE	Set this to “yes” if this particular node is a *UAVNet* bridge, which is generally the case, and bridges the network traffic between regular IEEE 802.11g WLAN clients and the *UAVNet* IEEE 802.11s mesh network. This generates two interfaces, one for the WLAN clients, the other one for the mesh network. If set to “no”, only a mesh interface is generated.	“yes”
UAVNET_WLAN_DEVICE	Defines the name of the IEEE 802.11g WLAN interface if UAVNET_BRIDGE=“yes”.	“wlan0”
UAVNET_WLAN_MAC	MAC address of the IEEE 802.11g WLAN interface.	
UAVNET_MESH_DEVICE	Defines the name of the IEEE 802.11s mesh interface.	“mesh0”
UAVNET_MESH_MAC	MAC address of the IEEE 802.11s mesh interface.	
UAVNET_MESH_CHANNEL	Used channel for the IEEE 802.11s mesh network. Must be the same on all mesh nodes.	“1”
UAVNET_MESHID	Defines the IEEE 802.11s mesh id. This must be the same on all mesh nodes.	“uavmesh”
UAVNET_BRIDGE_DEVICE	Defines the name of the bridge interface if UAVNET_BRIDGE=“yes”, bridging the UAVNET_WLAN_DEVICE and the UAVNET_MESH_DEVICE.	“br0”
UAVNET_BRIDGE_IP	IP of the bridge interface.	“192.168.230.10”
UAVNET_BRIDGE_MASK	Netmask of the bridge interface.	“255.255.255.0”

Table 5.1: Newly introduced variables in network.conf

5.2 UAV Management

The key innovation of *UAVNet* is the software controlling and coordinating the UAVs. It consists of different components, installed on the different participating UAVs, clients and configuration devices of *UAVNet*. The most important software component of *UAVNet* is running on the mesh node. Other components are running on the clients, either a notebook or an iPad/iPhone controlling and monitoring the network. The software on the flight electronics provides an interface to all needed functionality. Therefore, it does not have to be modified. This ensures a highest possible compatibility also with newer versions of the firmware.

The software of *UAVNet* consists of three main components. These are the *uavcontroller* running on the mesh nodes, the *uavclient* running on the devices on the ground and the software running on the flight electronics of the UAVs. As the firmware running on the flight electronics did not need to be modified, it is not described in detail in this thesis. Both components, the *uavcontroller* and the *uavclient*, make use of two libraries. The first one is *libuavext*, which handles the external communication, meaning the communication between the mesh nodes and the clients using the wireless interface. The second library is *libuavint*. It handles the UAV

internal communication, meaning the communication between the mesh node and the flight electronics using the serial interface. The *uavcontroller* includes both libraries, the *uavclient* uses only *libuavext*. These two libraries work as the interfaces between the mentioned three main software components of *UAVNet*.

This modular approach allows the implementation of other clients or controllers by including the provided libraries. One example is the iPhone/iPad client to configure and monitor *UAVNet*. Another one could be the implementation of an Android version (see Section 7.2). In the following subsections the *uavcontroller*, *uavclient*, *libuavext*, and *libuavint* are described in detail.

5.2.1 Uavcontroller

The *uavcontroller* is the main component of *UAVNet* and runs on every mesh node of the network. It handles all the communication between the mesh nodes, the clients and the flight electronics. Additionally, it controls and monitors the different parameters from the UAV and sends appropriate commands and information to the flight electronics, clients and other mesh nodes.

The *uavcontroller* runs as a daemon on the Linux wireless mesh node. It is crucial that the *uavcontroller* is always running, otherwise there is no more control over the UAV. Therefore, the daemon is configured to automatically be restarted, if there should be a problem with it or if it is shutted down unexpectedly. This is achieved by using the respawn mechanism of *init* which restarts a process automatically, if it gets killed [50]. Additionally, a mechanism can be implemented that restarts *uavcontroller*, if it becomes unresponsive. This could be some kind of a software or hardware watchdog and could be realised as future work.

The component *uavcontroller* consists of several different parts: Besides the main part, two libraries are included and several parts are running as separate threads. Figure 5.2 shows a schema of the architecture of the *uavcontroller*. In the following the different parts are described in detail:

main

This is the main part of the *uavcontroller*. It controls the entire system and manages the different running threads, the wireless and serial interfaces and contains the entire logic. Further, it manages the flow of the different information and control messages. It includes the *libuavext* and *libuavint* and spawns the different threads. It contains also the state and data of the UAVs, the current network configuration and status and keeps track of the existing clients, the other UAVs, and subscribed configuration devices.

External Communication - libuavext

libuavext is programmed as a library, used to manage the external traffic in the network. This includes the traffic between the mesh nodes, as well as the traffic between the mesh nodes and the client devices. It is included in the *uavcontroller* software, as well as in the software on the clients and on monitoring and configuration devices like iPhones or iPads. It manages both, the outgoing and incoming traffic and is responsible for the following tasks:

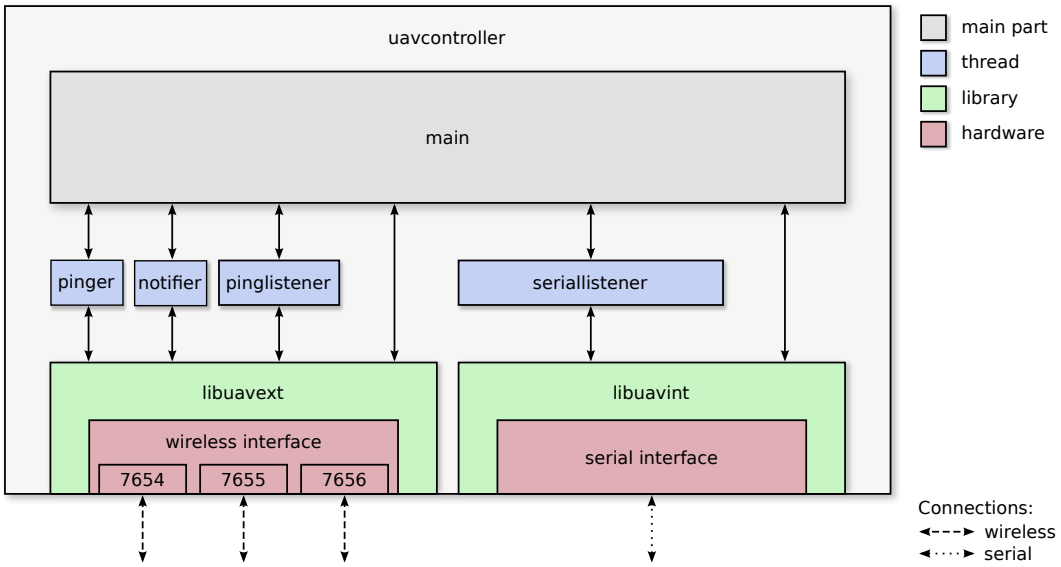

Figure 5.2: Schema of the architecture of the *uavcontroller*.

- Establishing, listening on, managing, handling and closing all different kind of sockets on different ports to enable the communication between the different devices.
- Handling and translating the different hostnames, Internet Protocol (IP), and MAC addresses, using standard operating system procedures like static hostname - IP mappings in the */etc/hosts* file, Domain Name System (DNS) using the */etc/resolv.conf* file, and the Address Resolution Protocol (ARP).
- Sending and receiving all different kind of TCP and UDP packets on the different sockets. This includes the automatic retransmission on failed submissions.
- Handling, decoding, and encoding all the different outgoing and incoming notification, ping and control messages. This includes the Cyclic Redundancy Check (CRC) calculation and validation, as well as the base64-encoding and -decoding among other tasks.

Internal Communication - libuavint

libuavint is a library similar to *libuavext*, but it handles the internal communication between the mesh node and the flight electronics of the UAV using the serial interface. It is included only in the *uavcontroller* software running on the mesh node. Like the *libuavext*, it manages both, the outgoing and incoming traffic on the serial interface and is responsible for the following tasks:

- Managing the serial connection between the mesh node and the flight electronics of the UAV.
- Sending and receiving the different messages between the mesh node and the UAV.

pinger

pinger is a thread that broadcasts every few seconds some information about the UAV using a *sendPing*-message. It uses a multicast address and the port 7655. All clients and configuration devices receiving such a message know that the UAV is in transmission range, its current status and the positions of already found registered other clients.

notifier

notifier is a thread sending regularly *notification*-messages to subscribed clients or configuration devices. It uses an unicast address and the port 7656 to transmit the message containing a *Notification_t*-structure as payload. It contains the GPS coordinates, the altitude, battery voltage, heading, speed and the status of the UAV and network. To receive the *notification*-messages, a client or configuration device must subscribe itself to the notification service at first, using a *notificationSubscription*-message. The same message can be used to unsubscribe from the service.

pinglistener

pinglistener is a thread, listening for broadcasted *ping*-messages by other UAVs on the port 7655. It keeps track of all the UAVs, including their current states and found client positions. This mechanism ensures that every UAV knows exactly the current network state and the positions of the other UAVs and clients, even if another UAV discovered the notebooks or smartphones on the ground.

seriallistener

seriallistener is a thread, listening for serial transmissions from the flight electronics. These are regular messages like current navigation and status information or answers on sent serial commands. *seriallistener* collects all incoming messages and passes them to the *main* part of the *uavcontroller* for further evaluation and processing.

5.2.2 Uavclient

The *uavclient* is the software running on the clients or configuration devices on the ground. It includes the *libuavext* to communicate with the mesh nodes in the air, using a wireless IEEE 802.11g connection. There are different possibilities to implement this part of software. The proposal we make in this thesis is only one way to do it and works as follows:

The devices on the ground are divided in two groups, playing different roles in the *UAVNet*. The first group are the notebooks trying to communicate with each other. The second group contains a monitoring and configuration device, in our case an iPhone or iPad. It is also possible to write a client software containing both components, without the existence of an iPad/iPhone or an Android version. The two following subparagraphs describe the role of the two different device groups.

Notebook

In general, the *uavclient* running on the notebook on the ground, listens for a near UAV and transmits its position to the flying mesh node.

In detail it works as follows: The *uavclient* listens on the port 7655 for a *ping*-message from a UAV. As soon as it receives such a message it knows that a UAV is in transmission range. The *ping*-message contains information about the current state of the UAV and the *UAVNet*. Depending on this information, the *uavclient* transmits its own position to the UAV using a *submitPosition*-message. This type of message contains a *GPS_Pos_t* structure and the MAC-address of the notebook. If the transmission is successful, the *uavcontroller* running on the mesh node registers the client and its position and continues deploying the *UAVNet*, depending on the current state and configuration, as described in more detail in Section 3.2.

Configuration Devices

The iPad/iPhone is used to configure and deploy the network as well to monitor it. The *Remote Control App* running on it includes also the *libuavext* library and listens on the port 7655 for a *ping*-message from a nearby UAV. If a UAV is discovered, the app provides the possibility to define the network scenario and the used positioning and searching modes. After submitting this configuration by transmitting a *submitStartConfiguration*-message, the deployment process, as described in Section 3.2, is initialized. Additionally the iPad/iPhone can subscribe to the UAV's notification service, using a *notificationSubscription*-message. From this moment on, the software listens on port 7656 for the *notification*-messages from the UAVs. These messages contain all needed information to show the positions of the UAVs and clients on an interactive map, with some additional values like the altitude, speed, direction and battery level of the UAVs.

5.2.3 Communication Protocol

To transmit messages between the mesh nodes, the clients, and the flight electronics, a special protocol is used. For both types of messages (for internal and external messages) the same protocol with slight modifications is used. The software running on the flight electronics provides already an interface and a protocol to transmit messages [13]. This message protocol is implemented in *UAVNet* for the internal messages. To keep it simple and standardized, the same protocol is used also for the external messages with a slight adaption. The following paragraphs describe how the protocol is implemented and what messages are defined.

Protocol Messages

Every sent and received message, either between the mesh nodes or between the mesh node and the flight electronics, is a special string of characters. Figure 5.3 shows the structure of a typical internal message in detail:

It starts with the dedicated 1-byte sign '#' to signalize the start of a message. The second byte describes the part of the flight electronics, that is the sender or receiver of this message and can be one of the following characters:

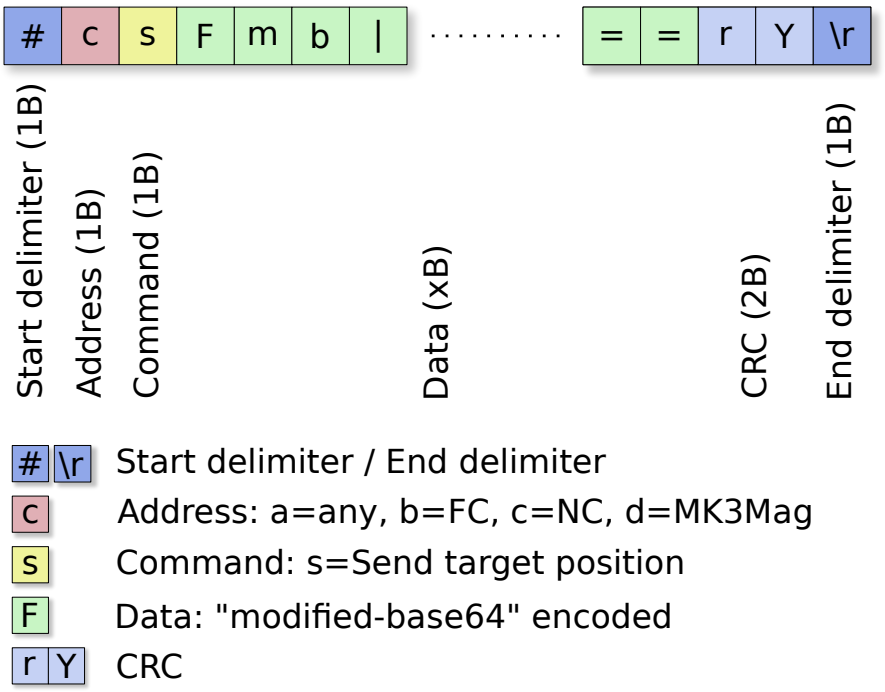

Figure 5.3: Internal message format.

- 'a': No specific component
- 'b': FlightCtrl
- 'c': NaviCtrl
- 'd': MK3Mag

The third byte is an 'id' that defines the command or kind of message. The following bytes are the actual payload, encoded by a modified Base64 algorithm. The next two bytes are CRC characters to ensure the correct encoding and transmission of the message and finally the end-byte '\r' signalizes the end of the message.

The structure of a typical external message is very similar to an internal message and is shown in Figure 5.4. The only difference is the omitted 'id'-byte, which describes the flight electronics part in internal messages.

The payload of such a message can contain anything from a simple 1-byte value to a complex nested structure. The *UAVNet*-software provides methods to simplify the process of generating and extracting transmission messages. To transmit complex data like navigation and position information, the data is packed and handled in c structures [51], which can be converted into transmission messages. The off-the-shelf software on the UAV flight electronics uses the same mechanism to handle the data.

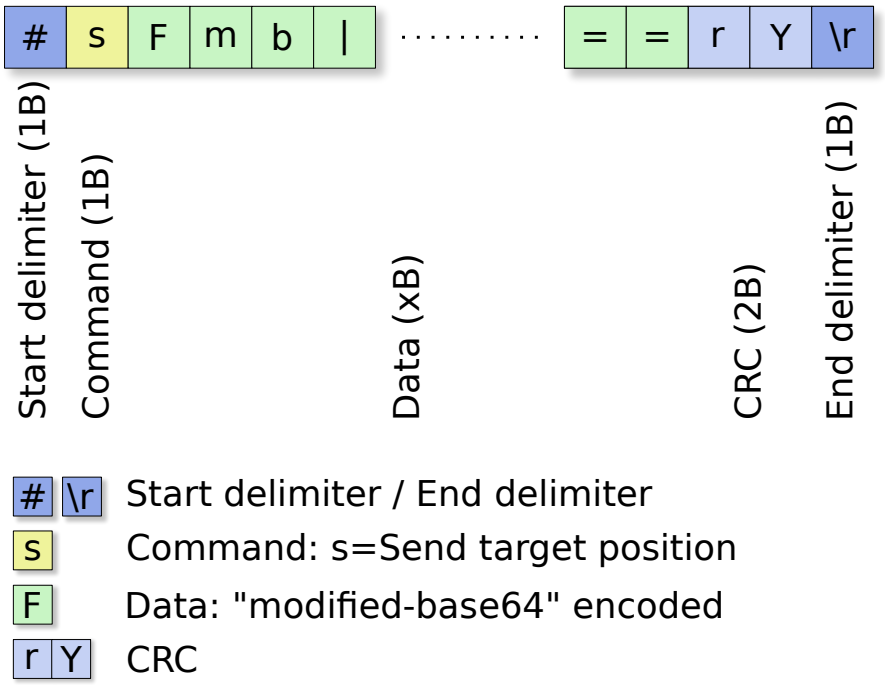

Figure 5.4: External message format.

In-UAV Messages

In the following, the existing internal messages are described in detail. The firmware of the flight electronics implements some more messages, which are not used by *UAVNet* and, therefore, not implemented and supported. Every specific message is sent only in one direction, from the mesh node to the flight electronics or vice versa. There exists similar messages in one or the other direction but every 'id'-byte is unique for one message-type.

outSerialLinkTest

The *outSerialLinkTest* message, shown in Figure 5.5, is sent from the mesh node to the flight electronics and is used to test the serial connection. It contains a two-byte integer as payload. This message is sent periodically until it gets an *inSerialLinkTest* message with the same integer as payload. From this moment, the software knows, that the serial connection is established and the flight electronics booted up and is running and functional.

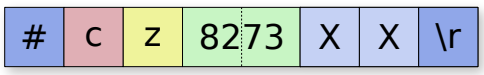

Figure 5.5: Message outSerialLinkTest.

inSerialLinkTest

Figure 5.6 shows the *inSerialLinkTest* message. It is sent from the flight electronics to the mesh node as an answer on a received *outSerialLinkTest* message. It contains the same two-byte integer value, that was sent from the mesh node. This test ensures that the *UAVNet*-software waits to send data to the flight electronics until it is booted up and the serial connection is working.

Figure 5.6: Message inSerialLinkTest.

outSerialOSDInterval

The message *outSerialOSDInterval*, shown in Figure 5.7, is sent from the mesh node to the flight electronics and contains as payload a 1-byte integer. It configures the sending interval in 10ms steps of the *inSerialOSDData* messages, sent from the flight electronics to the mesh node. The *outSerialOSDInterval* message must be resent at least every four seconds to ensure a regular reception of *inSerialOSDData* messages.

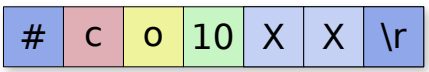

Figure 5.7: Message outSerialOSDInterval.

inSerialOSDData

Figure 5.8 shows the message *inSerialOSDData*, which is sent periodically from the flight electronics to the mesh node with the interval defined by the *outSerialOSDInterval* message. Usually, it happens a few times per second. The message contains a *NaviData_t* structure, which is shown in Figure 5.9. The structure *NaviData_t* contains a lot of different positions and values from the *NaviCtrl* of the flight electronics of the UAV. These are for example the current and next GPS positions, flight altitude, flight direction and speed, battery voltage and many more. Some of these values are sent in the *sendPing* and *sendNotification* messages to the clients and configuration devices.

Figure 5.8: Message inSerialOSDData.

NaviData_t

Version	(uint8)
CurrentPosition	(GPS_Pos_t)
TargetPosition	(GPS_Pos_t)
TargetPositionDeviation	(GPS_PosDev_t)
HomePosition	(GPS_Pos_t)
HomePositionDeviation	(GPS_PosDev_t)
WaypointIndex	(uint8)
WaypointNumber	(uint8)
SatsInUse	(uint8)
Altimeter	(int16)
Variometer	(int16)
FlyingTime	(uint16)
UBat	(uint8)
GroundSpeed	(uint16)
Heading	(int16)
CompassHeading	(int16)
AngleNick	(int8)
AngleRoll	(int8)
RC_Quality	(uint8)
FCFlags	(uint8)
NCFlags	(uint8)
Errorcode	(uint8)
OperatingRadius	(uint8)
TopSpeed	(int16)
TargetHoldTime	(uint8)
RC_RSSI	(uint8)
SetpointAltitude	(int16)
Gas	(uint8)
Current	(uint16)
UsedCapacity	(uint16)

Figure 5.9: NaviData_t structure.

outSerialRequestWaypoint

The message *outSerialRequestWaypoint*, shown in Figure 5.10, is sent from the mesh node to the flight electronics and requests a given waypoint stored in the waypoint list of the UAV. The payload is a 1-byte integer containing the index of the requested waypoint. The answer from the flight electronics on an *outSerialRequestWaypoint*-message is an *inWaypoint*-message containing the requested waypoint structure.

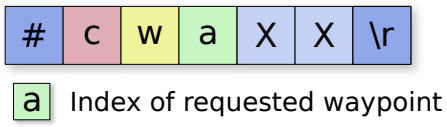

Figure 5.10: Message outSerialRequestWaypoint.

inWaypoint

Figure 5.11 shows the message *inWaypoint*, which is sent from the flight electronics to the mesh node in answer to a received *outSerialRequestWaypoint* message. As payload the message contains a *Waypoint_t* structure, shown in Figure 5.12. This structure contains the GPS coordinates of the requested waypoint and some additional information.

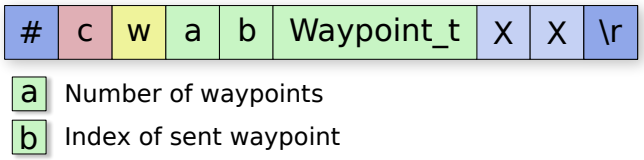

Figure 5.11: Message inWaypoint.

Waypoint_t	
Position	(GPS_Pos_t)
Heading	(int16)
ToleranceRadius	(uint8)
HoldTime	(uint8)
Event_Flag	(uint8)
Index	(uint8)
Reserve	(uint8[11])

Figure 5.12: Waypoint_t structure.

outSerialSendWaypoint

The message *outSerialSendWaypoint*, shown in Figure 5.13, is sent from the mesh node to the flight electronics and submits an additional waypoint to the waypoint list stored on the UAV. It contains a *Waypoint_t* structure, shown in Figure 5.12. If the *GPS_Pos_t* structure (shown in Figure 5.12), included in the *Waypoint_t* structure, contains a 0 as the *Status* value, the flight electronics clear the entire waypoint list. This mechanism is used when a new waypoint route must be submitted to the UAV, or if the flight should be aborted. The answer from the flight electronics on a received *outSerialSendWaypoint* message is an *inNbrOfWaypoints* message.

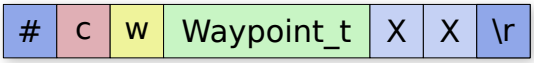

Figure 5.13: Message outSerialSendWaypoint.

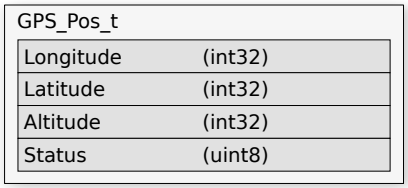

Figure 5.14: GPS_Pos_t structure.

inNbrOfWaypoints

Figure 5.15 shows the message *inNbrOfWaypoints*, which is sent from the flight electronics to the mesh node after processing the received *outSerialSendWaypoint* message. It contains only a 1-byte integer value, showing how many waypoints currently are stored on the UAV flight electronics.

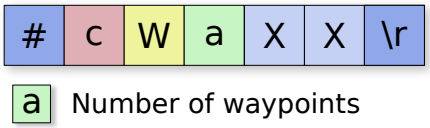

Figure 5.15: Message inNbrOfWaypoints.

outSerialSelectFC

The message *outSerialSelectFC*, shown in Figure 5.16, is sent from the mesh node to the flight electronics and is used to change the main component. It is implemented in the *UAVNet* software only for compatibility and future purposes and implementation. It is not used for the current functionality.

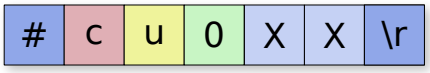

Figure 5.16: Message outSerialSelectFC.

outSerialSelectNC

The message *outSerialSelectNC*, shown in Figure 5.17, is sent from the mesh node to the flight electronics and is used to change the main component of the UAV. It is a kind of a special message and has not the same structure as all other messages. It has to be sent before any *outSerialSendWaypoint* message. Otherwise the flight electronics would not accept the transmitted waypoint.

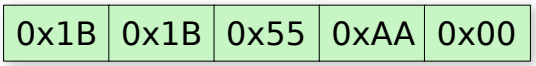

Figure 5.17: Message outSerialSelectNC.

External Messages

In the following, all the existing external messages are described in detail. All of these messages are sent between the mesh nodes and the clients or the configuration devices. The structure of an external message is shown in Figure 5.4. It looks very similar to the one of an internal message. The only difference is the omitted 'id'-byte.

submitStartConfiguration

The message *submitStartConfiguration*, shown in Figure 5.18, is sent from the client or the monitoring/configuring device to the wireless mesh node. It is the first message that has to be transmitted to start the deployment of a *UAVNet*. Its payload contains a *StartConfig_t* structure, shown in Figure 5.19. This structure contains the used scenario and searching and positioning algorithms, described in Section 3.2. Additionally, it contains an *AllowedClients_t* structure, shown in Figure 5.20. This structure contains the MAC addresses of the two clients involved in an *Airborne Relay* scenario. Only these two clients are allowed to transmit their position to the UAV. Once a *submitStartConfiguration* message is sent to the UAV, it cannot be changed anymore, except an *abort* message is sent and the deployment process is restarted. After receiving a *submitStartConfiguration* message, the *UAVNet* starts to deploy the network, depending on the submitted configuration values.

Figure 5.18: Message *submitStartConfiguration*.

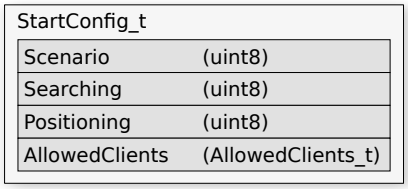

Figure 5.19: StartConfig_t structure.

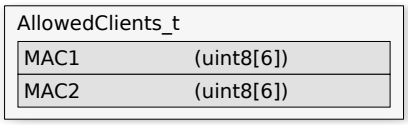

Figure 5.20: AllowedClients_t structure.

submitPosition

Figure 5.21 shows the message *submitPosition*, which is sent from the client notebook the the mesh node. It contains a *Submitted_Pos_t* structure as shown in Figure 5.22. A *Submitted_Pos_t* structure contains a *GPS_Pos_t* structure with the GPS coordinates and the MAC address of the submitting client.

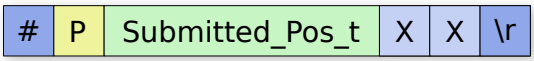

Figure 5.21: Message *submitPosition*.

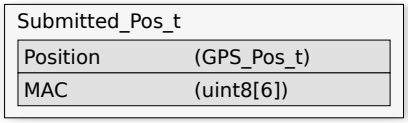

Figure 5.22: Submitted_Pos_t structure.

sendNotificationSubscription

The message *sendNotificationSubscription*, shown in Figure 5.23, is sent from the client or the monitoring/configuring device to the mesh node to subscribe to the UAV's notification service. Normally it is sent either after the client received a *ping* message, or if the user selected it on the configuration device. The payload of this message is a *NotificationSubscription_t* shown in Figure 5.24. This structure contains a 1-byte integer to define if the sending device should be subscribed or unsubscribed and the IPv4 or IPv6 address of the subscribing client or configuration device.

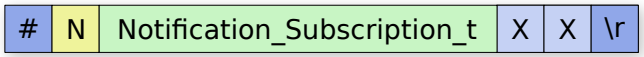

Figure 5.23: Message *sendNotificationSubscription*.

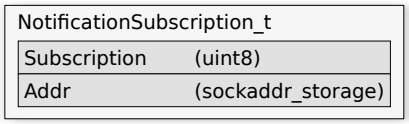

Figure 5.24: NotificationSubscription_t structure.

sendFlightDirection

Figure 5.25 shows the message *sendFlightDirection*, which is sent from the client or configuration device to the mesh node. It tells the UAV to which GPS location the UAV should fly at maximum to reach the second notebook. Therefore, the payload of the *sendFlightDirection* message is a *Submitted_Pos_t* structure, shown in Figure 5.22.

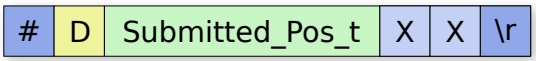

Figure 5.25: Message *sendFlightDirection*.

sendNotification

The message *sendNotification*, shown in Figure 5.26, is sent regularly from the mesh node to subscribed clients or configuration devices. The payload of this message is a *Notification_t* structure shown in Figure 5.27. It contains a *GPS_Pos_t* structure (see Figure 5.14), a *Status_t* structure (see Figure 5.29) and some additional information like the altitude, battery voltage, heading and speed. The *sendNotification* message is sent periodically every few seconds to all devices, that subscribed to the notification service using a *sendNotificationSubscription* message.

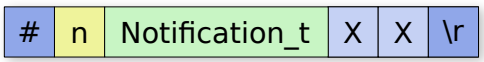

Figure 5.26: Message *sendNotification*.

Notification_t	
Position	(GPS_Pos_t)
Height	(uint16)
Battery	(uint8)
Heading	(int16)
Speed	(uint16)
Status	(Status_t)

Figure 5.27: Notification_t structure.

sendPing

Figure 5.28 shows the message *sendPing*, which is broadcasted regularly by the mesh node. The payload of that message is a *Status_t* structure as shown in Figure 5.29. This structure contains information about the configuration of the network, the positions of the clients and the hostname of the UAV, as well as its position, IP and MAC address. The *sendPing* message is broadcasted every few seconds using a multicast address and the port 7655.

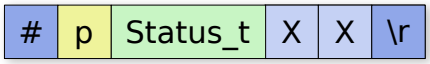

Figure 5.28: Message *sendPing*.

Status_t	
Scenario	(uint8)
State	(uint8)
Positioning	(uint8)
Searching	(uint8)
Position	(GPS_Pos_t)
Position Client1	(Submitted_Pos_t)
Position Client2	(Submitted_Pos_t)
Hostname	(char[])
IPv4	(char[])
MAC	(uint8[6])

Figure 5.29: Status_t structure.

outSocketResponse

The message *outSocketResponse*, shown in Figure 5.30, is sent in answer to almost every received control message. It contains a 1-byte integer that signalizes if the transmission and com-

mand was successful or if it failed. As answer on a *submitStartConfiguration* message, a special answer is sent if the chosen scenario is *Airborne Relay*, to let the client know that a *sendFlightDirection* message is expected.

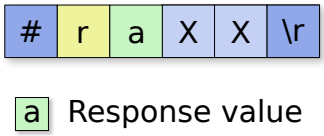

Figure 5.30: Message *outSocketResponse*.

sendAbort

Figure 5.31 shows the message *sendAbort*, which is sent from the client or configuration device to abort the entire setup process. It contains only a 1-byte integer as payload. If the UAV receives such a message it aborts everything and flies back to his home position.

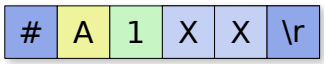

Figure 5.31: Message *sendAbort*.

Message Flow

In the following, the three main message flows are described in detail. These are the *UAVNet* deployment processes using the *Manual Searching* and *Autonomous Searching* algorithm and the message flow for the *Notification Service*. The shown scenario is the standard *Airborne Relay* consisting of a UAV, a monitoring iPhone and two clients on the ground. All the messages (except the broadcasted *ping* and the *notification* messages) are acknowledged with an *outSocketResponse* message. For reasons of clarity these acknowledgement messages are not shown in the following figures.

Manual Searching Mode Message Flow

Setting up a *UAVNet* using the *manual searching* algorithm is shown in detail in Figure 5.32 and works as follows:

- The user starts the Remote Control client.
- Then he/she switches on the UAV.
- As soon as the software on the flight electronics has booted, which takes some seconds, and the mesh node has connected successfully to the flight electronics, the UAV begins to broadcast periodically the *ping* message to indicate its presence and its current state (1).

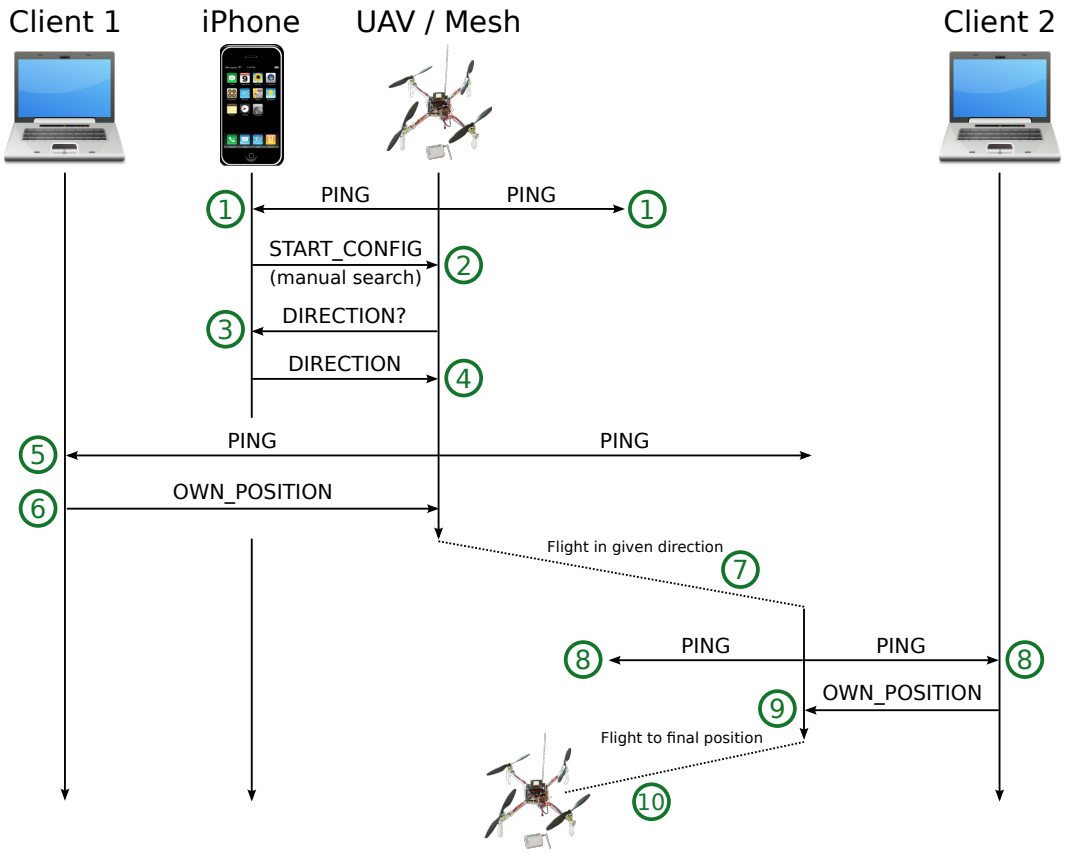

Figure 5.32: Message flow of *manual searching* mode.

- These messages are received by the iPhone, and the UAV is shown as a newly detected UAV in the GUI.
- The user configures the *scenario*, *searching*, and *positioning* mode, defines the MAC addressed of the participating clients and transmits this information to the UAV using a *submitStartConfiguration* message (2).
- In the *manual searching* mode, the UAV asks now for the direction of the expected location where it should get connection to the second client (3).
- After selecting this direction on the iPhone and transmitting it to the UAV with a *sendFlightDirection* message, the configuration is done (4). From this moment, the *ping* messages, broadcasted by the UAV, contains the *awaiting position 1* state.
- As soon as the *client 1* receives such a message (5), it sends its own position to the UAV, using a *submitPosition* message (6).
- The UAV starts the flight in the given direction (7), broadcasting an *awaiting position 2 ping* message (8).

- When *client 2* receives such a message, it transmits again its own position to the UAV (9).
- The *uavcontroller* software calculates now the center coordinates between these two GPS positions and flies to this location (10).
- The *UAVNet* deployment is done.

Autonomous Searching Mode Message Flow

The setup process of a *UAVNet* using the *autonomous searching* mode is similar to the *manual searching* one and is shown in detail in Figure 5.33. The differences are the following:

- The UAV does not require a flight direction. Therefore, the user does not need to define a flight direction on the *Remote Control App* (iPhone).
- The *uavcontroller* switches directly to the *awaiting position 1* state after receiving a *submitStartConfiguration* message, containing the *autonomous search* mode.
- The UAV does not fly on a direct line to the second client, but along a spiral around the first client. As the number of stored waypoints on the flight electronics of the UAV is limited, it does not fly on a smooth spiral, but on a spiral shaped line of waypoints.

Once the UAV received the position of the second client, it uses the same algorithms as in the *manual searching* mode to calculate the center position and flies to this location.

Notification Service

The message flow of the notification service, shown in Figure 5.34, is straight forward. When the configuration device receives a *ping* message of a UAV and, therefore, knows that it is present and in transmission range, it can register to the service by sending a *sendNotificationSubscription* message. Unsubscribing is done by sending the same message but with a different payload. While being subscribed, the registered client or configuration device receives periodically a *sendNotification* message.

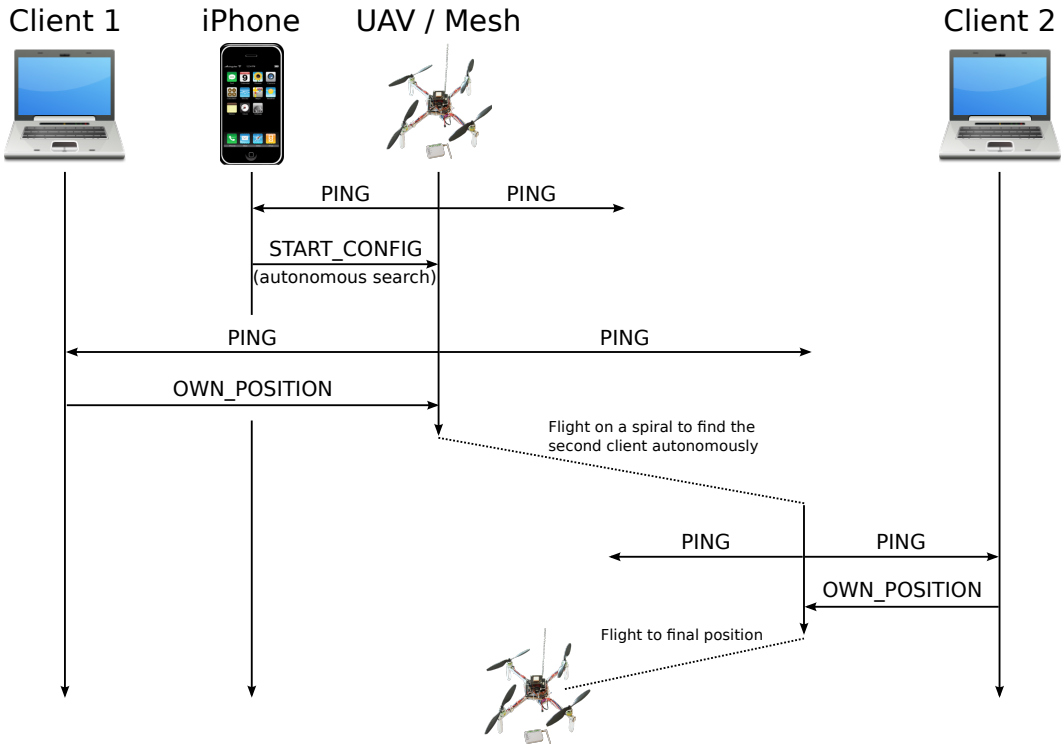

Figure 5.33: Message flow of *autonomous searching* mode.

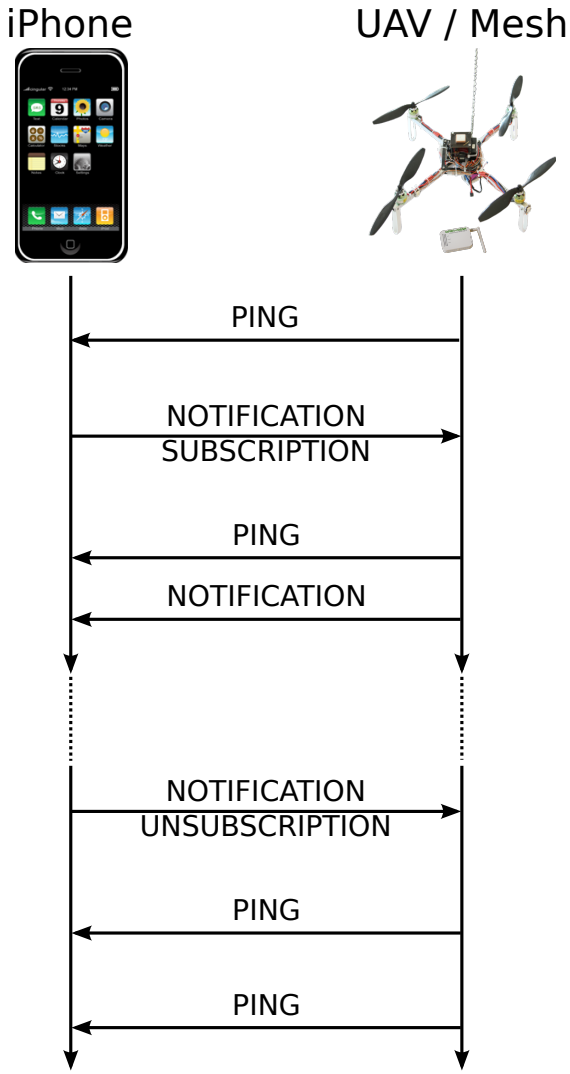

Figure 5.34: Message flow of a *notification subscription*.

Chapter 6

Evaluation

The main goal of *UAVNet* is to provide a solution to deploy a performant and reliable network in a fast and easy way. To evaluate the performance of *UAVNet* and to compare it to ground-based approaches, different Round-trip Time (RTT) and TCP/UDP throughput measurements have been performed. RTT measurements were done by using the *ping* tool from the *iputils* package [52]. The throughput of the network was measured by using *netperf* [53]. Different scenarios have been set up to evaluate the different parts and networks of *UAVNet*.

The evaluation of *UAVNet* is structured as follows: Section 6.1 evaluates the optimal signal strength threshold between two UAVs in terms of optimal network performance and reliability. The mesh network performance including multiple nodes is measured and discussed in Section 6.2. Section 6.3 evaluates a complete *UAVNet* one-hop *Airborne Relay* scenario. It shows several measurements with UAVs located at different positions and compares the results between flying mesh nodes and ground-based approaches. The chapter concludes with the evaluation of a multi-hop *Airborne Relay* scenario in Section 6.4.

6.1 Optimal Signal Strength Threshold

To set up a performant and reliable network, the distances and signal quality between the network participants is crucial. To measure the optimal distance between the UAVs, the optimal signal strength threshold has been evaluated by measuring the RTT and the TCP and UDP throughput with decreasing distance between the two nodes.

6.1.1 Evaluation Setup

To determine the optimal signal strength threshold, we performed multiple measurements between two mesh nodes while the distance between them was constantly decreased, as shown in Figure 6.1. In order to obtain meaningful results, the nodes have been placed considering the signal strength between them, independent of the real distance between the nodes. We measured the TCP and UDP throughput with the *netperf* tool, using its TCP_STREAM and UDP_STREAM tests. The RTT was evaluated by using the *ping* tool, performing 1000 measurements with a measurement interval of 0.1 seconds and a payload of 56 bytes. To measure the signal strength, we used the *iw* tool [27].

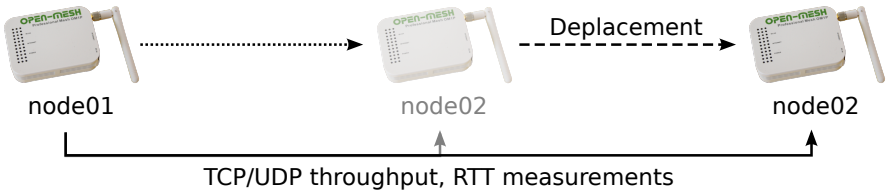

Figure 6.1: Evaluation Setup to determine the optimal signal strength threshold.

6.1.2 Results

Figure 6.2 shows the average TCP throughput and the standard deviations (whiskers) between two mesh nodes with decreasing distance between them. For signal strengths above -70 dBm the throughput reaches a pretty stable value of about 10 Mbps. For a value below -70 dBm, the TCP throughput drops significantly. The reason for this is the higher number of packet retransmissions, caused by more lost packets on weak connections. The lists of all achieved measurements can be found in the Tables B.1, B.2, B.3, and B.4 in Appendix B.1.

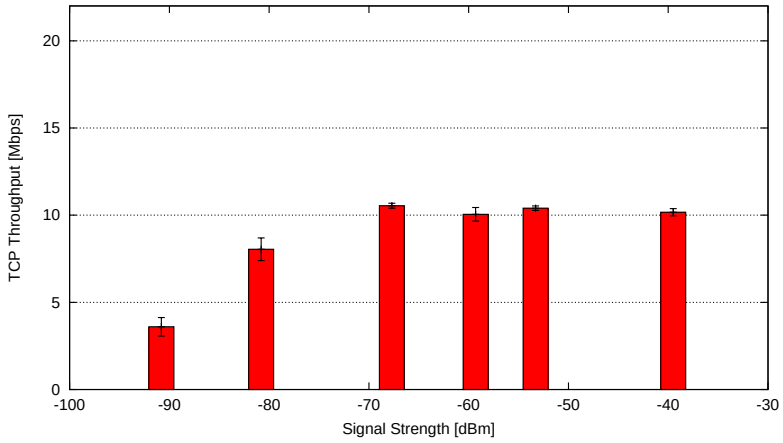

Figure 6.2: TCP throughput between two mesh nodes depending on signal strength.

The results of the UDP throughput measurements, shown in Figure 6.3, depict a very similar graph. The throughput decreases significantly below a signal strength of -70 dBm and stays very stable at around 16-17 Mbps above this threshold. As UDP has no flow and congestion control, the measured receive rate is slightly lower than the send rate. Thus, some packets are lost. This

is marked in green color in Figure 6.3. Overall, the UDP throughput is higher than the TCP throughput as expected.

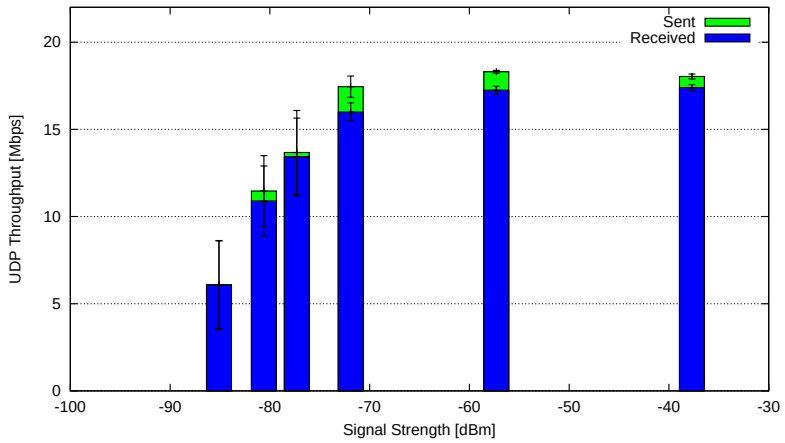

Figure 6.3: UDP throughput between two mesh nodes depending on signal strength.

During the TCP and UDP throughput measurements also the RTT was evaluated. Figure 6.4 depicts the average RTT and the standard deviations depending on the received signal strength on the mesh node. The RTT increases with a signal strength of below -70 dBm.

Based on the described TCP and UDP throughput and RTT measurements, we defined a signal strength threshold of -60 $\pm$ 10 dBm for an optimal positioning of the UAVs. This threshold guarantees a performant network with fast and reliable connections.

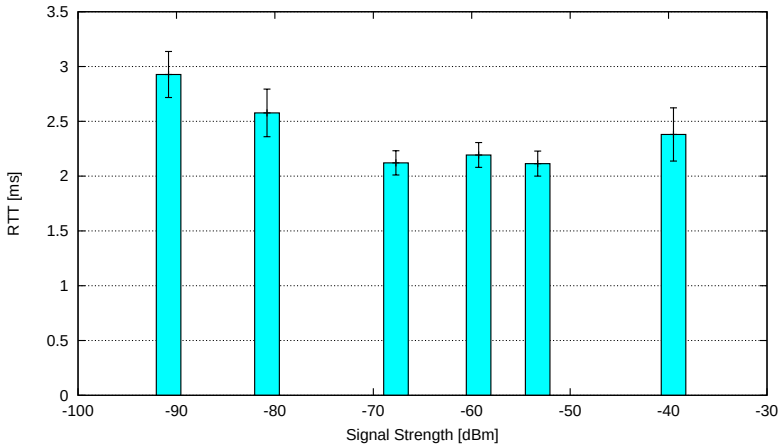

Figure 6.4: RTT between two mesh nodes depending on signal strength.

6.2 Mesh Network Performance

To evaluate the performance of the wireless mesh network, multiple UDP and TCP throughput and RTT measurements have been performed. First, the influence of the number of involved nodes on the network performance has been tested. Second, the effect of too far away nodes has been measured, to evaluate the influence of wrongly placed UAVs. A failure or removal of a center node would result in a similar situation and network performance.

6.2.1 Multi-Hop Performance

UAVNet is designed to deploy large networks, consisting of multiple wireless mesh nodes. Therefore, it is crucial to know the expectable network performance and how the connections behave, depending on the number of involved network members. As the nodes of our *UAVNet* prototype contain only one radio and a single channel communication, a decrease in the performance is expected for multi-hop connections.

Evaluation Setup

To evaluate the influence of multiple mesh nodes on the performance of a network connection, four nodes were placed outdoors in a chain topology with the evaluated signal strength threshold of -60 $\pm$ 10 dBm between them. We measured the TCP and UDP throughput between the

first node (node01) and the other nodes (node02, node03, node04), as shown in Figure 6.5. The measurements were done with the *netperf* and *ping* tool, using the same configuration and parameters as in the tests before.

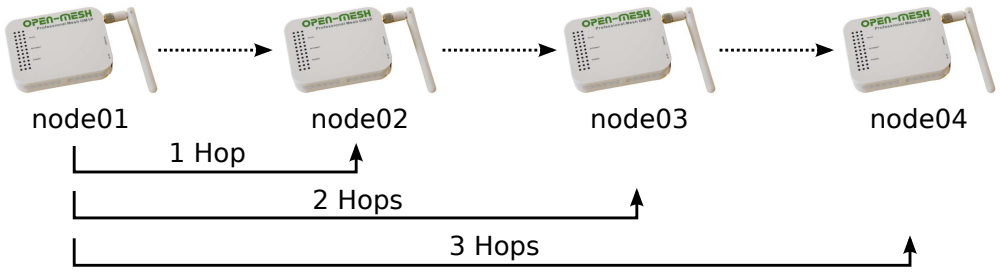

Figure 6.5: Evaluation Setup of Multi-Hop Performance Measurements.

Results

The results of the measurements are shown in Figure 6.6. They are listed in detail in the Tables B.5, B.6, and B.7 in Appendix B.2. The graph includes the average TCP and UDP throughput over one to three hops and the corresponding standard deviations (whiskers). The measured TCP and UDP throughputs over one hop are a bit lower but similar to the results gained in the Optimal Signal Threshold (Section 6.1) experiment. As expected, the throughput over multiple hops is significantly lower than if only two nodes are involved. This is a known phenomenon in ad-hoc networks for a small number of hops and is due to the one radio and single channel communication of our prototype nodes. The reason for this is that node02 forwards the packets to node03 using the same radio as for the communication with node01. Using a three-hops-connection shows only a small decrease in the throughput, compared to the two-hops-communication. The reason for this is that the nodes are positioned far enough from each other, so the link between node01 and node02 does not affect the communication between node03 and node04 too much. As in all experiments the UDP throughput is higher than the TCP throughput. But due to the lacking retransmission in the UDP protocol, some packets are lost.

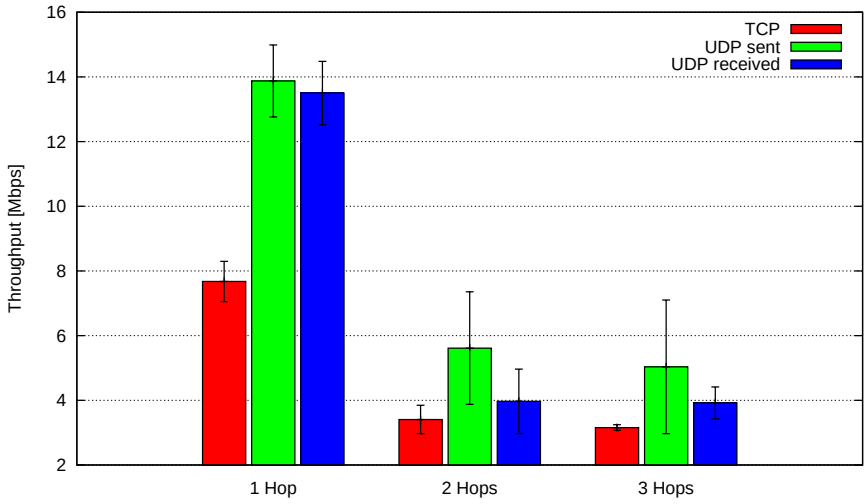

Figure 6.6: TCP and UDP throughput over multiple mesh nodes.

6.2.2 Too Far Away Node

As seen in the experiments before, the positioning of the UAV is crucial for the network performance. As *UAVNet* is highly mobile, it is likely that the UAVs are not placed totally accurate all the time. An other aspect needs to be considered as well: in possible future implementations of *UAVNet* a UAV must leave the network to recharge its battery or it can fail completely. To measure the influence of such not optimal UAV positioning or the failure of an intermediate mesh node, the following additional scenario has been set up and evaluated.

Evaluation Setup

Three mesh nodes (node01, node02, node03) have been placed outside, stationary on the ground in a chain setup, as shown in Figure 6.7. We measured the TCP and UDP throughput between node01 and node03, using node02 in-between as traffic forwarder. To simulate a node failure, node02 has been removed afterwards and the throughput measurements have been repeated. Node01 and node03 were kept at the same place and had only a very weak connection, because of the big distance between them.

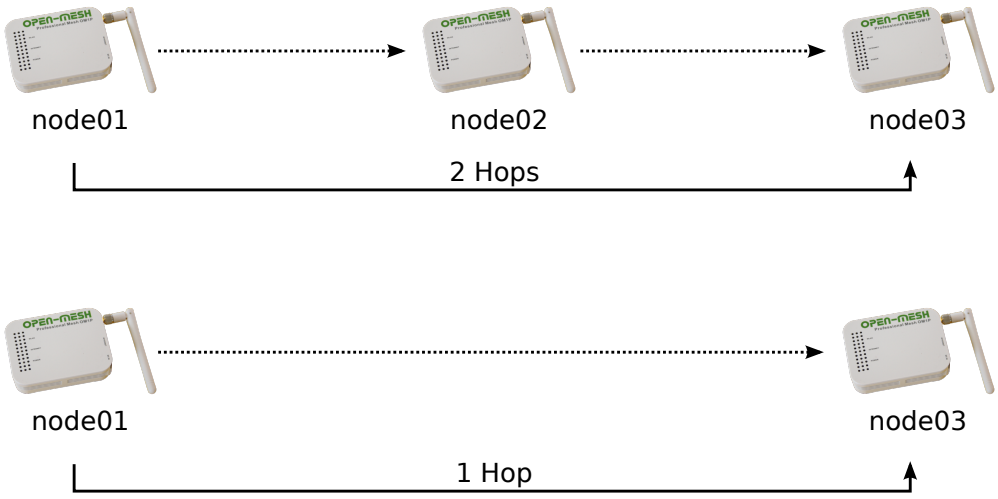

Figure 6.7: Evaluation Setup of the "Too far away node"-Scenario.

Results

The results shown in Figure 6.8 are expected. The measured throughput with the included center node node02 is very similar to the 2-hops result in the Multi-Hop Performance test. With a removed center node, the TCP throughput decreases drastically by a factor of 10 from 2.7 Mbps to 0.2 Mbps. Also the UDP throughput drops significantly. Additionally, a lot of packet loss occurs in the second UDP experiment. However, communication was still possible and affirms the optimal positioning of the nodes in respect of deploying a robust network. The measured values can be found in the Tables B.8, B.9, and B.10 in Appendix B.3.

Based on these results, three different strategies for recharging the batteries of the UAVs are conceivable:

- UAVs leaving the network to recharge its batteries are not replaced by other UAVs and the network does not adapt. This would be the simplest solution to implement. The evaluation results show that communication in the network is still possible but the network performance might decrease drastically.

- The leaving UAV could be replaced immediately by a fully charged UAV. This strategy would ensure a high network performance all the time, but it needs some additional UAVs in reserve. Maybe the overall network performance would be higher if these additional UAVs are integrated in the network instead of keeping them in reserve to replace other UAVs.

- The third strategy could be to not replace the leaving UAV, but to adapt the positions of the remaining UAVs, to close the "hole". After the recharging of the battery, the UAV could be reintegrated in the network. This strategy would result in a slightly lower network

performance because gaps between the nodes are larger than in an optimal scenario, but no additional UAVs are required.

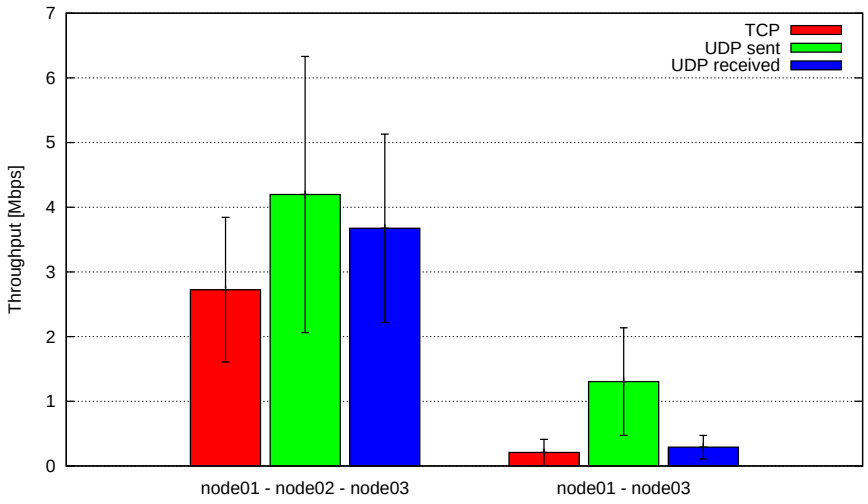

Figure 6.8: TCP and UDP throughput over multiple mesh nodes.

6.3 End-to-End Throughput in a Single-Hop Airborne Relay Scenario

The measurements described above evaluate the wireless mesh network, not including the end systems. This is only one part of a complete *UAVNet* setup. Therefore, some evaluations with a fully functional *Single-Hop Airborne Relay* setup have been done. This includes two stationary placed notebooks as clients and a flying UAV with an attached wireless mesh node, working as an airborne relay. We evaluated the advantage of our flying *UAVNet* approach over setups where the wireless mesh nodes were placed on the ground. Additionally, the difference between the location based and the signal strength positioning algorithm have been evaluated.

6.3.1 Evaluation Setup

In this setup several TCP throughput measurements have been performed. Two notebooks (shown as *client1* and *client2* in Figure 6.9) were placed on the ground, with a distance of

75 m between them. They stayed at the same place during all measurements. The UAV was placed and moved at an altitude of about 3-5 meters over ground to different locations. Both notebooks were connected to the UAV using a wireless IEEE 802.11g connection. The setup is shown in Figure 6.9. The TCP throughput measurements between *client1* and *client2* have been performed multiple times with the same configuration and parameters, but with the UAV located at the following positions:

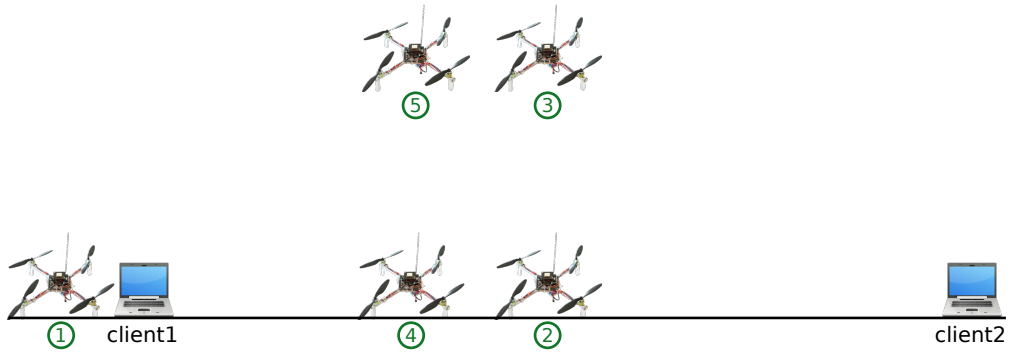

Figure 6.9: End-to-end throughput setup.

1. The UAV was placed on the ground, directly beside *client1* to simulate no flying UAV between the two clients. As the clients are configured to communicate using an AP, using a different technology to interconnect the two clients with no involved mesh node could result in measurements, which are not comparable to the desired network scenario.

2. The UAV was placed on the ground, exactly between the two notebooks, to simulate a ground based approach, using the *Location positioning* algorithm.

3. The UAV was hovering exactly between the two notebooks, to measure the *UAVNet* approach, using the *Location positioning* algorithm.

4. The UAV was placed on the ground, with the same signal strength to both notebooks to simulate a ground based approach, using the *Signal Strength positioning* algorithm.

5. The UAV was flying with the same received signal strength from both notebooks to measure the *UAVNet* approach, using the *Signal Strength positioning* algorithm.

As the two notebooks have different hardware and software, the strengths of their transmitted wireless signals were not the same. This is the reason why the UAV was not positioned exactly between the notebooks when it received the same signal strength from both notebooks.

6.3.2 Results

Figure 6.10 depicts the average TCP throughput from one notebook to the other one with the corresponding standard deviation (whiskers), depending on the location of the UAV. The achieved

results are listed in Table B.11 in Appendix B.4. Measurements with location 1) resulted in a very low TCP throughput of 0.064 Mbps, which indicates that the notebooks had almost no direct connection. The evaluation shows that there is a huge difference between ground based approaches and our flying *UAVNet* proposal. With a flying UAV, we reached a 5.3 to 6.3 times higher throughput, than when the UAV is positioned on the ground at the same location. Also the used positioning algorithm has significant influence on the reached throughput. The *Signal Strength Positioning* algorithm resulted in a 24% - 46% higher throughput compared to the *Location Positioning* mode. The best measured throughput was achieved, when the airborne relay was set up using the *Signal Strength Positioning* algorithm. The throughput was 127 times higher than the measurement with no UAV positioned between the notebooks.

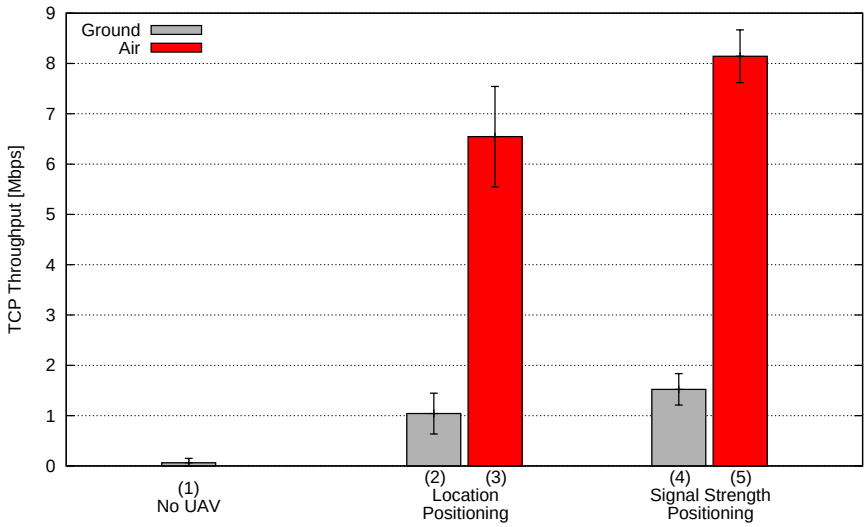

Figure 6.10: TCP throughput between two notebooks in an airborne relay setup.

6.4 End-to-End Throughput in a Multi-Hop Airborne Relay Scenario

As seen in Section 6.2, in which the performance of the mesh network was evaluated, the number of involved nodes in a communication path has a remarkable influence on the network performance. Therefore, we also evaluated a *Multi-hop Airborne Relay* scenario and compared these

results with the measurements done during the “End-to-end throughput” evaluation of a *Single-hop Airborne Relay* scenario described in Section 6.3.

6.4.1 Evaluation Setup

Figure 6.11 shows this evaluation setup. To compare the results to a ground based approach, two different scenarios have been deployed. In scenario 1 the UAVs were placed on the ground. Scenario 2 uses a real *Multi-hop Airborne Relay* setup with two flying UAVs. Several TCP throughput measurements have been performed between the two notebooks *client1* and *client2*. The two UAVs were deployed in-between the clients, using the *Multi-hop Airborne Relay* scenario. The traffic from *client1* was forwarded over both UAVs to *client2*.

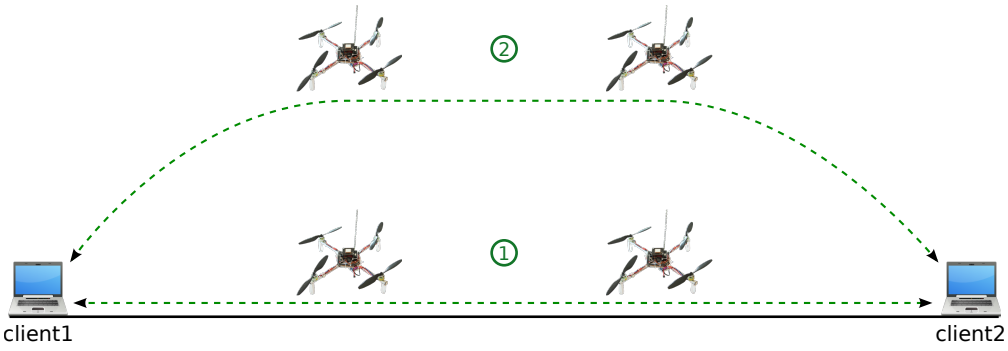

Figure 6.11: End-to-end throughput setup using a *Multi-hop Airborne Relay* scenario.

6.4.2 Results

Figure 6.12 depicts the average TCP throughput of ten measurements from one notebook (*client1*) to the other one (*client2*) with the corresponding standard deviation (whiskers). The achieved results are listed in Table B.12 in Appendix B.5. As seen in the evaluations before, the measured throughput in scenario 1 is much lower than in scenario 2. With the UAVs placed on the ground, we achieved an average throughput of 0.266 Mbps, whereas the flying nodes produced an average throughput of 1.868 Mbps, which is a factor of seven higher. The comparison with the single-hop airborne relay evaluation reveals expected results as well. As seen already in the multi-hop mesh network performance evaluation in Section 6.2 and Figure 6.6, the performance of the multi-hop airborne relay setup is lower than the performance of the single-hop airborne relay scenario, where one UAV is sufficient to cover the shorter distance between the two clients. However, the multi-hop airborne relay scenario achieves a significantly higher network throughput compared to a single-hop scenarios where the distances between the nodes are too large (“Too Far Away Node”-evaluation in Section 6.2). The results of such measurements heavily depend on several factors like used hardware, distances and the accuracy of the GPS signal.

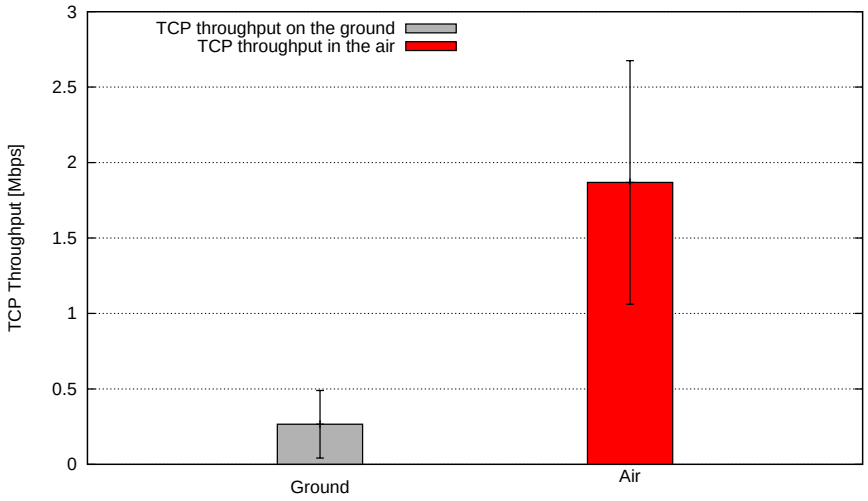

Figure 6.12: TCP throughput between two notebooks in a multi-hop airborne relay setup.

Chapter 7

Conclusions and Future Work

This chapter concludes the gained insights and experiences made during the implementation, testing, and evaluation of *UAVNet*. Furthermore, it shows some ideas how *UAVNet* could be improved in future work. Additionally, it shows up some needed extensions to evolve *UAVNet* from a prototype to a usable system in real-world scenarios.

7.1 Conclusions

This Master thesis introduces *UAVNet*, a concept and an implemented prototype of an autonomously deployable temporary flying IEEE 802.11s WMN. The main purpose of this framework is to provide a possibility to deploy a complete communication network in emergency and disaster recovery scenarios in an easy and fast way. The concept is based on small quadrocopter UAVs with attached lightweight wireless mesh nodes. The mesh nodes are connected directly to the flight electronics of the UAVs and control the autonomous network deployment using a decentralized approach. The entire deployment process can be configured with the integrated user-friendly *Remote Control App* running on an iPad or iPhone. Additionally, the *Remote Control App* monitors the entire *UAVNet* and displays all involved participants such as UAVs and clients on an interactive map.

We have proven the feasibility of an autonomously deployable flying WMN. The OpenMesh OM1P mesh nodes set up an IEEE 802.11s WMN and provide access for any kind of IEEE 802.11g wireless devices such as notebooks, tablets or smartphones. The integrated IEEE 802.11s wireless mesh network protocol guarantees optimal performance in a highly mobile network. This prototype implementation is capable of autonomously interconnect two distant clients by deploying an airborne relay, consisting of one or multiple UAVs. It provides sophisticated searching and positioning algorithms and can be extended with additional functionality. The entire network can be configured, deployed, and monitored by a single user, using a user-friendly application on an iPad or iPhone. Additional scenarios and usages are included in the concept, as well as in the *UAVNet* code and the *Remote Control App*.

We achieved to implement *UAVNet* without having to modify the original firmware running on the flight electronics of the UAVs. This guarantees an optimal compatibility and expandability of the system. The system could be adapted to be deployed on UAVs using different flight

electronics. *UAVNet* uses a uniform communication protocol to exchange commands and data between the flight electronics, attached mesh nodes and wirelessly connected devices.

UAVNet has some significant advantages, compared to a ground-based network approach: It has been shown that the flying wireless mesh nodes result in an up to 6.3 times higher network throughput compared to a ground-based approach. The UAVs are highly mobile and the network is adaptable for different network scenarios and requirements. Its placement is not restricted by the environment and the deployment is simple and fast and can be performed by a single person.

To achieve a best possible network performance, an optimal distance between the UAVs has been evaluated. Results of other tests confirm this optimal placement. The nodes are placed near enough to each other to achieve a good received signal strength, but are sufficiently far away from each other to get not too much interferences, caused by the single channel radio chip. If an intermediate node fails, the evaluation of *UAVNet* shows that communication between the nodes is slow, but still possible.

Almost all of the defined goals in Section 1.4 have been achieved. The *area coverage* scenario could not be implemented completely, but it is included in the overall system concept and could be implemented in a future work, based on the provided *UAVNet* prototype. The UAVs of the current prototype implementation are able to communicate with each other and exchange all kind of data. To deploy a functional *area coverage* scenario, some self-organizing distribution, replacement, and collision avoidance algorithms and strategies have to be developed.

Unfortunately, not all the drivers and used tools were totally stable during the development of *UAVNet*. However, the used build system ADAM makes it fairly easy to upgrade the Linux kernel, the wireless drivers and the other software running on the mesh nodes. As the IEEE 802.11s standard, especially its implementation in the Linux kernel, the wireless driver and other tools are still under heavy development, it can be expected that the overall performance and the robustness of the entire network may increase with advancing development.

Several flights, measurements, and deployment tests during the development and evaluation of *UAVNet* have shown that the entire system heavily depends on strong and accurate GPS signals. If the signals are weak or even wrong, the inaccuracy of the automatic deployment and placement of the UAVs can amounts to several meters or even a few dozen meters.

A big drawback of the entire system is the fact that the UAVs cannot stay longer than 10 - 15 minutes in the air, due to their high energy consumption. This might not be enough for a real system, but it is sufficient to prove the feasibility of a flying WMN. In addition, battery technology is making huge progresses in the last decade and may provide better batteries in the future.

7.2 Future Work

The provided prototype of *UAVNet* offers a lot of different possible extensions and improvements. The first step to evolve the prototype to a fully functional real world system would be to introduce an autonomous replacement and recharging strategy to keep the system alive for several hours or days. UAVs with low battery capacity should autonomously leave the network and fly to a recharging station. The missing UAV must be replaced by another one, or the formation must adapt to ensure optimal network connectivity. After recharging, the UAV can be reintegrated into the network. The recharging of a UAV would include some automatic landing and starting procedure. Additionally, the possible flight duration, which is currently 10 - 15 minutes, should be increased. That could be achieved by either using better and maybe larger batteries or by optimizing the needed amount of energy.

Other enhancements include the implementation of the *area coverage* scenario. Several UAVs should autonomously position themselves over a user-defined area to maximise the network coverage of this region. This includes some self-organizing distribution and collision avoidance algorithms.

Another subject of research could be the increase of the network performance and reliability by positioning the UAVs accordingly. Questions such as “How many UAVs are needed to guarantee an optimal network performance over a given area?”, or “Should more UAVs placed in regions with more clients?” should be answered. The system could constantly measure the achieved network performance and adapt autonomously, if needed.

Furthermore, *UAVNet* provides a good prototype for the research project “Opportunistic Routing for Highly Mobile Ad-hoc Networks” (ORMAN). New opportunistic multi-channel routing protocols and topology controlling algorithms for highly mobile networks must be developed. They guarantee that the UAVs are always connected to each other and maximise the overall network performance and stability. Newly developed opportunistic routing protocols should include all available data from the UAVs, such as flight direction, speed, altitude, location, and battery voltage and predict future topology configurations and changes.

Another possible enhancement is to not keeping the UAV swarm stationary on a given region, but let the swarm autonomously follow a predefined route. This could be useful, if the covered area is replaced by another one. Instead of taking all UAVs down and redeploy the network at the other location, the swarm could just move to the new region.

Besides the Remote Control App for iOS devices, other clients could be implemented. This includes for example an Android version for tablets and smartphones or a client running on Linux, Mac or Windows notebooks or netbooks.

List of Acronyms

ADAM Administration and Deployment of Adhoc Mesh Networks, 11–13, 21, 22, 39–42, 76
AODV Ad-Hoc On-Demand Distance Vector Routing, 3
AP Access Point, i, 1, 2, 11, 14, 21, 39, 41, 42, 71
API Application Programming Interface, 40
ARP Address Resolution Protocol, 45

CLI Command-line Interface, 41
CPU Central Processing Unit, 11, 36
CRC Cyclic Redundancy Check, 45, 48

DHCP Dynamic Host Configuration Protocol, 22
DNS Domain Name System, 45

GNU GNU's Not Unix, 93
GPS Global Positioning System, 9, 10, 23, 24, 27–31, 33–35, 46, 50, 51, 55, 60, 73, 76, 81
GUI Graphical User Interface, i, 5, 58

IBSS Independent Basic Service Set, 1, 14
IEEE Institute of Electrical and Electronics Engineers, 1, 11, 13–15, 20–23, 36, 39–42, 46, 75
IP Internet Protocol, 22, 45, 57
IPv4 Internet Protocol Version 4, 12, 55
IPv6 Internet Protocol Version 6, 12, 55

LAN Local Area Network, 22
LED Light-emitting Diodes, 9, 33, 36
LiPo Lithium-ion Polymer, 7–9, 11, 35, 36

MAC Media Access Control, 24, 45, 47, 54, 55, 57, 59
MANET Mobile Ad-hoc Network, 2, 11
MAP Mesh Access Point, 22

OLSR Optimised Link State Routing, 3
ORMAN Opportunistic Routing for Highly Mobile Ad-hoc Networks, 77
OS Operating System, 11, 12

RAM Random access memory, 11, 36
RCU Remote Control Unit, 32, 33
RTT Round-trip Time, 63, 65, 66, 82

SMD Surface-Mount Device, 9
SoC System on Chip, 11, 13, 36, 39, 40
SSID Service Set Identifier, 41

TCP Transmission Control Protocol, 21, 23, 45, 63–68, 70–73, 82, 83, 85, 95, 98–100

UART Universal Asynchronous Receiver/Transmitter, 11, 36, 37
UAV Unmanned Aerial Vehicle, i, 1, 3–5, 7, 9–11, 13, 15, 17, 19–37, 39, 42–48, 50, 52–55, 57–60, 63, 65, 66, 68–71, 73, 75–77, 81
UDP User Datagram Protocol, 21, 23, 45, 63–68, 70, 82, 85, 96, 98, 99
USB Universal Serial Bus, 8

VIF Virtual Wireless Interface, 40, 42

WiMAX Worldwide Interoperability for Microwave Access, 1
WLAN Wireless Local Area Network, 1, 20–22, 27, 33, 35, 39, 42
WMAN Wireless Metropolitan Area Network, 1
WMN Wireless Mesh Network, i, 1, 2, 4, 5, 7, 11–14, 19, 75, 76
WSN Wireless Sensor Network, 2

List of Figures

List of Tables

List of Listings

Bibliography

[1] "The Discovery of Radio Waves - 1888," http://www.sparkmuseum.com/BOOK_HERTZ.HTM, Jan. 2008.

[2] N. Abramson, "Development of the ALOHANET," *IEEE Trans. Inform. Theory*, vol. 31, no. 2, pp. 119–123, Mar. 1985.

[3] I. F. Akyildiz and X. Wang, "A Survey on Wireless Mesh Networks," *IEEE Commun. Mag.*, vol. 43, no. 9, pp. 23–30, 2005.

[4] I. F. Akyildiz, X. Wang and W. Wang, "Wireless Mesh Networks: a Survey," *Computer Networks Journal*, vol. 47, no. 4, pp. 445–487, Mar. 2005.

[5] R. Bruno, M. Conti, and E. Gregori, "Mesh Networks: Commodity Multihop Ad Hoc Networks," *IEEE Commun. Mag.*, vol. 43, no. 3, pp. 123–131, Mar. 2005.

[6] T. Staub, "Development, Testing, Deployment and Operation of Wireless Mesh Networks," Ph.D. dissertation, 2011, Doctoral dissertation, University of Bern.

[7] "RFC 3626: Optimized Link State Routing Protocol (OLSR)," http://tools.ietf.org/html/rfc3626, Oct. 2003.

[8] Ian D. Chakeres and Elizabeth M. Belding-Royer, "AODV Routing Protocol Implementation Design," in *International Workshop on Wireless Ad Hoc Networking (WWAN)*, Tokyo, Japan, Mar. 2004.

[9] "Unmanned Aerial Vehicle - Definition," http://www.thefreedictionary.com/Unmanned+Aerial+Vehicle, Nov. 2011.

[10] "Unmanned aerial vehicle - Wikipedia," http://en.wikipedia.org/wiki/UAV, Nov. 2011.

[11] H. Buss and I. Busker, "Mikrokopter Platform," http://www.mikrokopter.de, Nov. 2011.

[12] "Mikrocontroller-Shop," https://www.mikrocontroller.com, Nov. 2011.

[13] "Mikrokopter Serial Protocol," http://mikrokopter.de/ucwiki/en/SerialProtocol, Nov. 2011.

[14] Open-Mesh, "Open-Mesh OM1P," http://www.open-mesh.com, 2011.

[15] Meraki, "The Meraki Mini / Indoor Wireless Platform," http://www.meraki.com, 2007.

[16] D. Balsiger, “Administration and Development of Wireless Mesh Networks,” Master’s thesis, University of Bern, Institute of Computer Science and Applied Mathematics, 2009.

[17] D. Balsiger and M. Lustenberger, “Secure Remote Management and Software Distribution for Wireless Mesh Networks,” Sept. 2007, Computer Science Project, University of Bern.

[18] S. Morgenthaler, “Management Extensions for Wireless Mesh and Wireless Sensor Networks,” 2010, Bachelor Thesis, University of Bern.

[19] P. Engines, “ALIX system boards,” http://www.pcengines.ch, 2011.

[20] “compat-wireless,” http://www.linuxwireless.org/en/users/Download, Nov. 2011.

[21] “ath5k - Wireless Driver for Atheros Based Wireless Chipsets,” http://www.linuxwireless.org/en/users/Drivers/ath5k, Nov. 2011.

[22] “MadWifi project - Linux kernel driver for WLAN devices with Atheros chipsets,” http://www.madwifi-project.org, Nov. 2011.

[23] J. Camp and E. Knightly, “The IEEE 802.11s Extended Service Set Mesh Networking Standard,” *IEEE Commun. Mag.*, vol. 46, no. 8, pp. 120–126, 2008.

[24] G. Hiertz, D. Denteneer, S. Max, R. Taori, J. Cardona, L. Berlemann, and B. Walke, “IEEE 802.11s: The WLAN Mesh Standard,” *Wireless Communications, IEEE*, vol. 17, no. 1, pp. 104–111, Feb. 2010.

[25] IEEE P802.11 Task Group S, “IEEE P802.11sTM / D5.0, draft amendment to standard IEEE 802.11TM : Mesh Networking,” IEEE, Apr. 2010.

[26] Open80211s Consortium (Nortel, cozybit, one laptop per child, Google), “open80211s - A Reference Implementation of the Upcoming IEEE 802.11s Standard on Linux,” http://www.open80211s.org, Apr. 2011.

[27] “iw - nl80211 based CLI configuration utility for wireless devices,” http://www.linuxwireless.org/en/users/Documentation/iw, Nov. 2011.

[28] “IEEE 802.11s - Wikipedia,” http://en.wikipedia.org/wiki/802.11s, Nov. 2011.

[29] “open80211s,” http://www.open80211s.org, Nov. 2011.

[30] “hostapd,” http://hostap.epitest.fi/hostapd, Nov. 2011.

[31] A. Hänni, “Remote Control App,” 2011, Bachelor Thesis, University of Bern.

[32] “OpenStreetMap,” http://www.openstreetmap.org/, Nov. 2011.

[33] “Linux Kernel,” http://www.kernel.org, Nov. 2011.

[34] “Mikrokopter Get Started - Mikrokopter.de,” http://mikrokopter.de/ucwiki/en/Mikrokopter-Get-started#The_MikroKopter_is_ready._What_next.3F, Nov. 2011.

[35] "GPS Mode Control - Mikrokopter.de," http://mikrokopter.de/ucwiki/en/NaviCtrl_2.0?highlight=%28gps%29#GPS-Mode_control, Nov. 2011.

[36] "How Do You Start - Mikrokopter.de," http://mikrokopter.de/ucwiki/en/NaviCtrl_1.1?highlight=%28gps%29#How_do_you_start.3F, Nov. 2011.

[37] "Lighting schemes," http://www.mikrokopter.de/ucwiki/BeleuchtungsSchema, Nov. 2011.

[38] Open-Mesh, "OM1P 802.11g Mid Power Mini Router," http://www.open-mesh.com/index.php/professional/professional-mini-router-us-plugs.html, 2011.

[39] "Deans Ultra Plug," http://www.wsdeans.com/products/plugs/ultra_plug.html, Nov. 2011.

[40] "DC connector - Wikipedia," http://en.wikipedia.org/wiki/DC_connector, Nov. 2011.

[41] "Banana connector - Wikipedia," http://en.wikipedia.org/wiki/Banana_connector, Nov. 2011.

[42] "Logic Level Converter," http://www.sparkfun.com/products/8745, Nov. 2011.

[43] W. Dubowik, "Atheros: Fix ath5k Support on ar2315/2317," http://repo.or.cz/w/openwrt.git/commit/cf521fcca87ee5330d41200c3470ca78e6519eb3, apr 2011.

[44] "mac80211 kernel module," http://www.linuxwireless.org/en/developers/Documentation/mac80211, Nov. 2011.

[45] "cfg80211 - New Linux wireless configuration API," http://www.linuxwireless.org/en/developers/Documentation/cfg80211, Nov. 2011.

[46] "Wireless-Extensions," http://www.linuxwireless.org/en/developers/Documentation/Wireless-Extensions, Nov. 2011.

[47] "OpenHAL," http://www.madwifi-project.org/wiki/About/OpenHAL, Nov. 2011.

[48] "Open-sourced HAL Code of Atheros and Sam Leffler," http://svn.freebsd.org/base/projects/ath_hal, Nov. 2011.

[49] "Wireless Tools for Linux," http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html, Nov. 2011.

[50] "init - Linux Man Page," http://linux.die.net/man/5/init, Nov. 2011.

[51] "struct (C programming language) - Wikipedia," http://en.wikipedia.org/wiki/Struct_%28C_programming_language%29, Nov. 2011.

[52] A. Kuznetsov and Y. Hideaki, "Linux iputils," http://www.linuxfoundation.org/collaborate/workgroups/networking/iputils, Nov. 2011.

[53] "netperf - Network Performance Benchmark," http://www.netperf.org/, Nov. 2011.

Appendix A

Evaluation Setup

A.1 Notebook 1

Model:	IBM ThinkPad X41
Operating System:	Debian GNU/Linux (i686)
Kernel:	2.6.16.1
Wireless Chip:	Atheros Communications Inc. AR5212 802.11abg (rev 01)
Wireless Driver:	MadWifi 0.9.4
	wpa_supplicant 0.6.4
Wireless Kernel Module:	ath_pci

A.2 Notebook 2

Model:	IBM ThinkPad T43
Operating System:	Ubuntu 10.04 (Lucid Lynx, i686)
Kernel:	2.6.32-24-generic
Wireless Chip:	Intel Corporation PRO/Wireless 2200BG
Wireless Driver:	Intel(R) PRO/Wireless 2200/2915 Network Driver 1.2.2kmprq
	wpa_supplicant 0.6.9
Wireless Kernel Module:	ipw2200

A.3 Mesh Nodes

Model:	Open-Mesh OM1P
Operating System:	ADAM
Kernel:	2.6.37.6-om1p
	compat-wireless-2011-12-24
Wireless Chip:	Atheros Communications Inc. AR2315 802.11bg
Wireless Driver:	ath5k
Wireless Kernel Module:	ath5k

Appendix B

Evaluation Results

B.1 Optimal Signal Strength Threshold Evaluation Results

To simplify the layout, the following tables depict only the averages of ten measurements performed at every location.

B.1.1 TCP Throughput

Signal strength [dBm]	Minimal Throughput [Mbps]	1st quartile Throughput [Mbps]	Median Throughput [Mbps]	3rd quartile Throughput [Mbps]	Maximal Throughput [Mbps]	Stddev Throughput [Mbps]	Average Throughput [Mbps]
-39.5	9.77	10.09	10.19	10.29	10.52	0.21	10.17
-53.3	10.29	10.31	10.37	10.42	10.71	0.13	10.40
-59.3	9.00	10.02	10.19	10.26	10.31	0.39	10.05
-67.7	10.29	10.50	10.55	10.59	10.84	0.14	10.54
-80.8	7.36	7.50	7.80	8.67	9.110	0.65	8.04
-90.8	2.58	3.74	3.81	3.89	3.950	0.53	3.60

Table B.1: TCP throughput between two nodes, placed at different locations.

B.1.2 UDP Throughput

Sent

Signal strength [dBm]	Minimal Throughput [Mbps]	1st quartile Throughput [Mbps]	Median Throughput [Mbps]	3rd quartile Throughput [Mbps]	Maximal Throughput [Mbps]	Stddev Throughput [Mbps]	Average Throughput [Mbps]
-37.7	17.72	18.01	18.03	18.13	18.22	0.14	18.04
-57.3	18.22	18.24	18.27	18.37	18.41	0.08	18.3
-71.9	16.32	17.18	17.59	17.89	18.16	0.61	17.45
-77.6	10.56	11.71	13.41	15.79	16.95	2.41	13.67
-80.3	8.78	9.63	11.25	13.17	14.3	2.03	11.46
-85.1	2.23	4.6	5.57	7.33	11.22	2.52	6.09

Table B.2: UDP throughput (sent) between two nodes, placed at different locations.

Received

Signal strength [dBm]	Minimal Throughput [Mbps]	1st quartile Throughput [Mbps]	Median Throughput [Mbps]	3rd quartile Throughput [Mbps]	Maximal Throughput [Mbps]	Stddev Throughput [Mbps]	Average Throughput [Mbps]
-37.7	16.99	17.33	17.4	17.5	17.6	0.18	17.39
-57.3	16.94	17.06	17.22	17.42	17.58	0.23	17.24
-71.9	14.99	15.81	16.08	16.25	16.83	0.53	15.99
-77.6	10.53	11.51	13.27	15.43	16.27	2.22	13.42
-80.3	8.62	9.21	10.61	12.51	13.88	2.02	10.89
-85.1	2.02	4.6	5.57	7.33	11.22	2.56	6.06

Table B.3: UDP throughput (received) between two nodes, placed at different locations.

B.1.3 RTT

Signal strength [dBm]	Minimal RTT [ms]	1st quartile RTT [ms]	Median RTT [ms]	3rd quartile RTT [ms]	Maximal RTT [ms]	Stddev RTT [ms]	Average RTT [ms]
-39.5	2.037	2.276	2.339	2.385	2.981	0.243	2.380
-53.3	2.035	2.067	2.087	2.101	2.433	0.114	2.114
-59.3	2.033	2.124	2.156	2.260	2.419	0.113	2.193
-67.7	2.014	2.065	2.094	2.125	2.415	0.111	2.121
-80.8	2.265	2.451	2.566	2.691	2.906	0.217	2.577
-90.8	2.742	2.780	2.860	3.004	3.387	0.210	2.927

Table B.4: RTT between two mesh nodes, placed at different locations.

B.2 Multi-Hop Mesh Network Performance Evaluation Results

B.2.1 TCP Throughput

Number of hops	Minimal Throughput [Mbps]	1st quartile Throughput [Mbps]	Median Throughput [Mbps]	3rd quartile Throughput [Mbps]	Maximal Throughput [Mbps]	Stddev Throughput [Mbps]	Average Throughput [Mbps]
1	6.650	7.168	7.780	8.093	8.470	0.622	7.674
2	2.790	3.063	3.335	3.685	4.190	0.441	3.408
3	3.030	3.073	3.190	3.225	3.270	0.089	3.158

Table B.5: TCP throughput depending on hop count.

B.2.2 UDP Throughput

Sent

Number of hops	Minimal Throughput [Mbps]	1st quartile Throughput [Mbps]	Median Throughput [Mbps]	3rd quartile Throughput [Mbps]	Maximal Throughput [Mbps]	Stddev Throughput [Mbps]	Average Throughput [Mbps]
1	11.280	13.458	14.060	14.418	15.230	1.115	13.875
2	3.460	4.148	5.245	7.058	8.260	1.740	5.615
3	3.250	3.695	4.245	5.908	8.940	2.067	5.035

Table B.6: UDP throughput (sent) depending on hop count.

Received

Number of hops	Minimal Throughput [Mbps]	1st quartile Throughput [Mbps]	Median Throughput [Mbps]	3rd quartile Throughput [Mbps]	Maximal Throughput [Mbps]	Stddev Throughput [Mbps]	Average Throughput [Mbps]
1	11.200	13.133	13.650	14.128	14.620	0.980	13.499
2	3.010	3.110	3.485	5.060	5.210	0.999	3.967
3	3.140	3.615	4.025	4.313	4.530	0.494	3.920

Table B.7: UDP throughput (received) depending on hop count.

B.3 Too Far Away Node Evaluation Results

B.3.1 TCP Throughput

Node setup	Minimal Throughput [Mbps]	1st quartile Throughput [Mbps]	Median Throughput [Mbps]	3rd quartile Throughput [Mbps]	Maximal Throughput [Mbps]	Stddev Throughput [Mbps]	Average Throughput [Mbps]
n01 - n02 - n03	1.100	2.330	2.585	3.353	4.530	1.118	2.725
n01 - n03	0.000	0.010	0.190	0.365	0.570	0.203	0.209

Table B.8: TCP throughput (missing node).

B.3.2 UDP Throughput

Sent

Node setup	Minimal Throughput [Mbps]	1st quartile Throughput [Mbps]	Median Throughput [Mbps]	3rd quartile Throughput [Mbps]	Maximal Throughput [Mbps]	Stddev Throughput [Mbps]	Average Throughput [Mbps]
n01 - n02 - n03	1.910	2.545	3.495	5.605	8.700	2.135	4.198
n01 - n03	0.810	0.848	1.035	1.248	3.300	0.832	1.305

Table B.9: UDP throughput (sent, missing node).

Received

Node setup	Minimal Throughput [Mbps]	1st quartile Throughput [Mbps]	Median Throughput [Mbps]	3rd quartile Throughput [Mbps]	Maximal Throughput [Mbps]	Stddev Throughput [Mbps]	Average Throughput [Mbps]
n01 - n02 - n03	1.860	2.478	3.445	4.453	5.970	1.456	3.675
n01 - n03	0.080	0.160	0.265	0.395	0.600	0.181	0.291

Table B.10: UDP throughput (received, missing node).

B.4 End-to-End Throughput in a Single-Hop Airborne Relay

Setup/ Location	Minimal Throughput [Mbps]	1st quartile Throughput [Mbps]	Median Throughput [Mbps]	3rd quartile Throughput [Mbps]	Maximal Throughput [Mbps]	Stddev Throughput [Mbps]	Average Throughput [Mbps]
1	0.000	0.013	0.035	0.070	0.290	0.086	0.064
2	0.630	0.790	0.860	1.125	1.800	0.406	1.041
3	4.900	6.095	6.380	7.365	8.040	0.998	6.546
4	1.140	1.370	1.400	1.568	2.210	0.313	1.523
5	7.170	7.818	8.115	8.505	8.870	0.525	8.143

Table B.11: TCP throughput (end-to-end, single-hop).

B.5 End-to-End Throughput in a Multi-Hop Airborne Relay

Scenario	Minimal Throughput [Mbps]	1st quartile Throughput [Mbps]	Median Throughput [Mbps]	3rd quartile Throughput [Mbps]	Maximal Throughput [Mbps]	Stddev Throughput [Mbps]	Average Throughput [Mbps]
1	0.010	0.090	0.210	0.495	0.560	0.224	0.266
2	0.290	1.588	2.110	2.250	2.980	0.807	1.868

Table B.12: TCP throughput (end-to-end, multi-hop).

Erklärung

gemäss Art. 28 Abs. 2 RSL 05

Name/Vorname: .....

Matrikelnummer: .....

Studiengang: ……

Bachelor ☐ Master ☐ Dissertation ☐

Titel der Arbeit: .....

.....

.....

LeiterIn der Arbeit: .....

.....

Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe o des Gesetztes vom 5. September 1996 über die Universität zum Entzug des auf Grund dieser Arbeit verliehenen Titels berechtigt ist.

.....
Ort/Datum

.....
Unterschrift