Automated Deployment of a Wireless Mesh Communication Infrastructure for an On-site Video-conferencing System (OViS)

Masterarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Stefan Ott
2011

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik und angewandte Mathematik

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank Prof. Dr. Torsten Braun for giving me the opportunity to realize this thesis in his *Computer Networks and Distributed Systems* group at the University of Bern. His support for and interest in my project has been a great source of enthusiasm.

Then, I would like to express my gratitude to Thomas Staub who supervised my work. He provided much of his valuable time giving highly appreciated input, ideas for improvement and the occasional hint to get my brain unstuck. He also provided me with numerous mesh nodes and other hardware to work from the comfort of my home, organized machines at the institute when needed and generally showed a lot of patience. Last but not least, he spent countless hours proofreading this very thesis.

In addition, I would like to thank my friends Claude Henchoz, Julien Fierz, Reto Gantenbein, Anselm Strauss and Marc Landolt for providing much-needed distraction on a regular basis, Daniel Balsiger for giving me additional insight into the ADAM system in particular and the world of embedded Linux in general and Philipp Bunge for various technical inputs.

Finally, my biggest thanks go to Claudia Asti who was very understanding whenever I got obsessed over some minor detail and neglected her in favour or trying out yet another crazy new idea and to my family, which supported me throughout my studies.

Stefan Ott
July 2011

Abstract

The availability of means for wireless communication is taken for granted today. With the proliferation of high-speed mobile phone infrastructure and wireless networks, more and more areas become accessible for audio and increasingly also for video transmissions. Yet even in densely populated areas there remain some locations where the signal strength for cell phones and wireless networks is too weak and thus the available bandwidth is very limited. This affects remote rural areas as well as locations in modern buildings at urban sites.

In many situations, a proverbial picture is worth more than a thousand words. The original problem we were faced with was to illustrate a challenging technical situation at a location without network coverage to a qualified off-site engineer. Only a very limited understanding of wireless technologies can be expected from the person on site. We therefore propose OViS, a platform that brings wireless networking into buildings and other remote sites where a wired Internet connection is available somewhere nearby. We extend the range of that connection by using a Wireless Mesh Network (WMN) that can easily be deployed by a non-technical user guided by a graphical user interface. Furthermore, we provide an implementation that integrates this deployment process with a means of communicating with the outside world by audio and/or video conferencing.

We implemented a prototype of OViS consisting of the following components: the custom Linux distribution that powers our mesh nodes and our modifications to it, the topology of the WMN, the deployment process and the various user-facing applications that were written to guide the inexperienced user through the deployment of our network. We present a solution that relies on open standards and free software for the network in combination with a platform-independent graphical user interface that integrates a well-known proprietary video-conferencing system. We also show the flexibility of OViS with the help of a second client application that works on the Android mobile platform. It replaces the video-conferencing application with a mechanism to quickly take pictures and send them to the off-site user while at the same time using an established voice-over-IP standard for audio communication.

Chapter 1

Introduction

Nowadays, wireless network connectivity is omnipresent: mobile phones and laptop computers connect to cellular broadband networks and *Wireless Local Area Networks (WLANs)* to not only enable voice and video calls but to deliver all kinds of multimedia content to their users. Gaps in the network coverage can be very disruptive; the user is left without access to crucial information and, more importantly, without the possibility of communicating with others in important situations.

We are going to show a real-life situation where the ability to communicate at remote sites is important but often impossible with current technology and, later in this thesis, present our solution to that problem.

1.1 Motivation

The motivation for our work is a simple situation often found in real life: a worker on a construction site needs to install a complicated piece of equipment (such as an electrical switch board) and requires outside help to do so. Unfortunately, this equipment is often located in unfavourable parts of the building (e.g., the basement) where cell phone signals cannot penetrate the thick walls. This situation is illustrated by Figure 1.1. The worker has to go outside, call the qualified engineer, explain the problem over the phone and then go back and try to fix it with the hints received on the phone – a process that is prone to misunderstandings on both sides. If the worker fails to fix the problem, another call has to be made, possibly even more until the qualified remote engineer decides to travel to the construction site to have a look at the problem. All things considered, this is a highly disruptive and costly exercise caused by a simple lack of network coverage.

1.2 Wireless Mesh Networks

A *Wireless Mesh Network (WMN)* is a wireless network that consists of a number of mesh routers and clients. Those nodes communicate among each other without the need of any centralized or preexisting infrastructure. In particular, WMNs feature mechanisms for routing: any node on a mesh network can talk to any other node on that network without any manual routing configura-

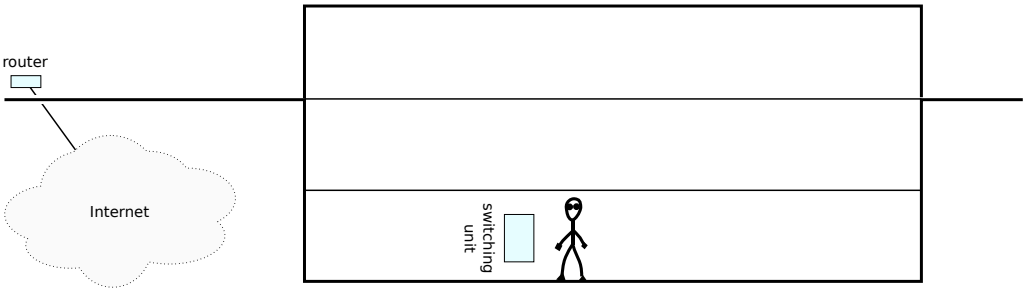


Figure 1.1: The problem.

tion. The routing protocols used in Wireless Mesh Networks usually also allow nodes that have Internet connectivity to let other nodes connect to the outside world through that connection. A WMN can thus be connected to the Internet through one or more gateway nodes and share the connection not only with all other devices within the WMN, but also with devices that are not themselves part of the WMN but merely connected to it through one of the participating nodes.

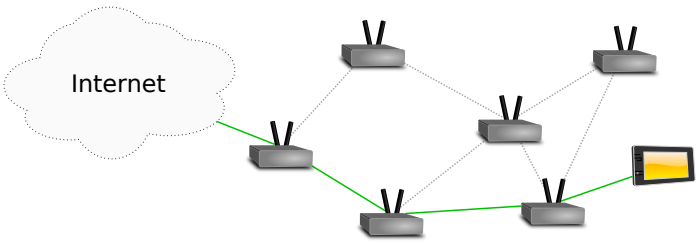


Figure 1.2: A wireless mesh network.

Figure 1.2 shows a typical WMN. Several nodes are connected to each other, one of them has a connection to the Internet. The client machine can connect to the gateway node through the mesh network and access the Internet that way.

1.3 Our Solution

Our solution to the problem described in Section 1.1 is a temporary WMN. The idea is to guide the worker at the construction site through the process of deploying a mesh network that covers the area between the closest Internet connection point (usually construction sites have an office with Internet access these days) and the troublesome equipment. After deploying the network, the user proceeds to initiate a video call to the qualified off-site engineer. Figure 1.3 shows this setup: the worker in the basement can now use the WMN and its gateway node to talk to the remote engineer over the Internet.

With this solution, the user never has to leave the installation site and can even show the faulty equipment to the engineer through the video camera, thereby saving considerable amounts of time, money and effort for everyone involved.

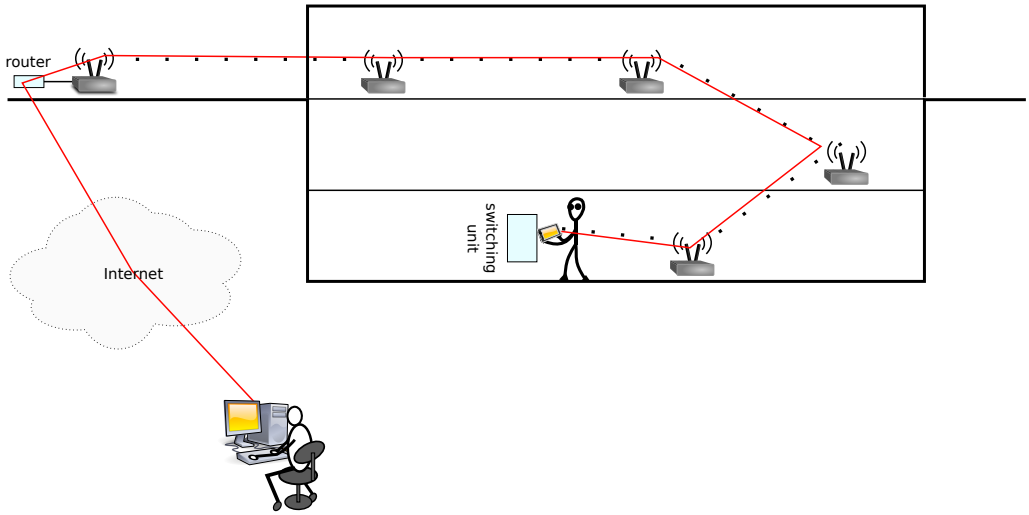


Figure 1.3: The solution: a wireless mesh network.

1.4 Document Structure

This thesis is structured as follows: Chapter 2 explains several technologies, mechanisms and applications that are related to our work. In particular, we talk about the underlying Linux distribution used on the mesh nodes and its wireless networking subsystem. Then, in Chapter 3, the architecture and design of the WMN and the network deployment process are explained and several alternative design ideas are discussed. Afterwards, Chapter 4 focuses on the mesh nodes: we explain the operating system that is running on the nodes, what the deployment process looks like from a node's perspective and the additional software that was written and added to support the deployment process. The last part of the implementation details are explained in Chapter 5, which talks about the client application (or, to be more precise, the various client applications) and how they interact with the rest of the system to guide the user in his task of deploying the OViS network and start communicating over it. Finally, in Chapter 6, we evaluate OViS by measuring the network throughput and comparing it to a manually deployed network and we then continue to present our conclusions and some suggestions for future work in Chapter 7.

Chapter 2

Related Work

A lot of time and effort was put into OViS and the software components it comprises. Those programs will be presented and explained in great detail in Chapters 3, 4 and 5.

Our components do, however, rely on many third-party tools and applications – from the Linux distribution to the video client and on various levels in between. Before we discuss OViS, we would first like to present some of the related projects that we consider particularly relevant for OViS.

2.1 Networking Technologies

A core aspect of OViS is the underlying network. The following technologies in particular play a vital role in how our network works or present viable alternatives to the technologies used for this project.

2.1.1 Automatic Network Configuration

Manually assigning *Internet Protocol (IP)* addresses to network hosts can be a tedious process, especially in large networks and in situations where hosts may join or leave the network at any time such as WMNs. To make life easier for network administrators, several protocols have been created to automatically configure the hosts on a network.

DHCP

In IPv4 networks, the *Dynamic Host Configuration Protocol (DHCP)* [1] is normally used to automatically configure hosts. DHCP requires a designated server responsible for dynamically assigning addresses from a configured address pool. The server listens for requests sent by machines that need an IP address (*DHCP discovery* packets) and sends back *DHCP offers* that contain the IP address the machine should use and some other information such as the default router, DNS servers and the *lease duration*, i.e., how long the address can be used. The DHCP server needs to keep track of all the addresses that have been assigned on the network and the client has to renew its address whenever the lease expires.

IPv6 Stateless Address Autoconfiguration

In IPv6 networks, stateless address autoconfiguration can be used instead of DHCP. As the name implies, this mechanism does not require any saved state concerning the assigned IP addresses anywhere in the network. Instead, a server periodically broadcasts *router advertisement* messages that contain the IPv6 address of the router and an IPv6 network prefix (the "first half" of an IPv6 address). A host that gets connected to this network can then automatically generate the "second half", usually based on the interface's *Medium Access Control (MAC)* address, optionally using *privacy extensions* [2] to prevent unique identification of the machine by changing the address over time. IP address collisions are virtually impossible thanks to the vast address space available in each IPv6 network and such conflicts can be detected by the host, therefore the role of a centralized guardian over the list of assigned IP addresses is no longer necessary.

In a decentralized mesh network, DHCP is obviously not an option. IPv6 autoconfiguration could make sense; however, the *Linux IPv6 Router Advertisement Daemon* (radvd [3]) refuses to operate on interfaces that are themselves auto-configured.

2.1.2 Routing

Routing in WMNs differs significantly from routing in wired networks and wireless infrastructure networks: the topology is highly dynamic (nodes may join or leave the network at any time), no dedicated preconfigured infrastructure exists and manual intervention is often impossible. Multiple specialized routing protocols have been designed and implemented to address the issues specific to WMNs, often based on pre-existing protocols for wired networks.

OLSR

The *Optimized Link State Routing protocol (OLSR)* [4] is a widely-used routing protocol for WMNs. As the name implies, OLSR is a *link-state* routing protocol, a proactive routing protocol where each node keeps state and topology information about the network regardless of the data traffic generated, as opposed to reactive routing protocols such as AODV [5] which only discover routes through the network when they are needed.

Compared to classical link-state routing, OLSR brings several improvements specifically designed for wireless networks. Figure 2.1 shows one of these improvements: the use of *Multipoint Relays (MPRs)*, special nodes that are responsible for forwarding broadcast packets. MPRs are chosen by each node from among its one-hop neighbours in a way that makes sure that each two-hop neighbour can be reached over at least one of the MPRs while keeping the number of relays as small as possible. This way, messages sent over the MPRs will eventually reach every node on the network. OLSR nodes have a policy of only forwarding a packet if they are an MPR of the sending node and only if they have not already resent the packet. This mechanism guarantees that all nodes on the network receive control traffic that is flooded to the network while keeping the number of retransmits at a minimum.

Nodes on an OLSR network exchange state information through so-called *HELLO* messages. Those messages are crucial for OLSR in order to find neighbours (i.e., nodes in communication range), to detect the link quality to those nodes and to announce information about the selected

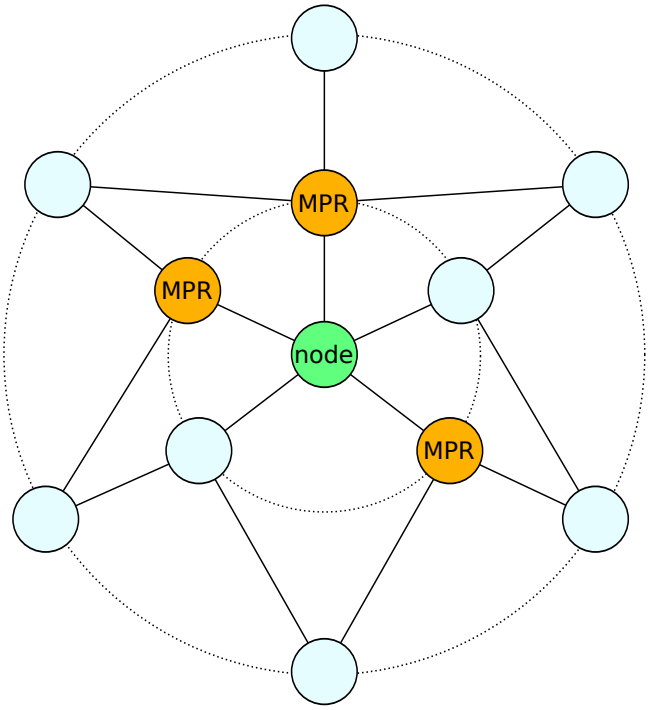


Figure 2.1: Multipoint relays in OLSR.

MPRs. HELLO messages are periodically broadcast to the whole network where they will be overheard by other nodes and used to create / update the model of the network topology held by each node. OLSR can therefore work without any centralized infrastructure whatsoever.

Another interesting feature of OLSR is the ability to inject additional routes into the OLSR network through the use of *Host and Network Association (HNA)* messages. HNA messages contain information about any additional networks that are connected to the originating node and are flooded to the network similar to MPR messages. In this thesis, this allows us to not only announce the routing information about the gateway node to the network but also the connection between the last node and the hand-held device. This is explained in more detail in Section 3.3.

Thanks to the efficient and completely decentralized nature of OLSR, it was chosen as the routing protocol for OViS. More precisely, OViS uses the OLSR implementation from olsr.org [6]. This implementation was chosen due to its portability, its widespread use and because of the successful use for other projects at our institute [7]. The version employed for our project is *0.5.6-r8*.

While our setup utilizes OLSR as its routing protocol, OViS itself (including all the software written for this project) is completely unaware of the underlying routing protocol. Once the ADAM embedded Linux distribution (see below) supports other routing protocols, it should be possible to use any other ad-hoc routing protocol with OViS.

IEEE 802.11s Mesh Networking

IEEE 802.11s is a work-in-progress wireless mesh standard. It differs from most other wireless routing protocols (including OLSR) in a fundamental way: while other projects are *layer-3* (i.e., IP) [8] routing protocols, IEEE 802.11s is part of the IEEE 802.11 standard for wireless networking [9], implemented directly as part of the MAC layer. The mesh network is therefore transparent to IP routing mechanisms and, as a direct consequence, handles IPv4 and IPv6 at the same time.

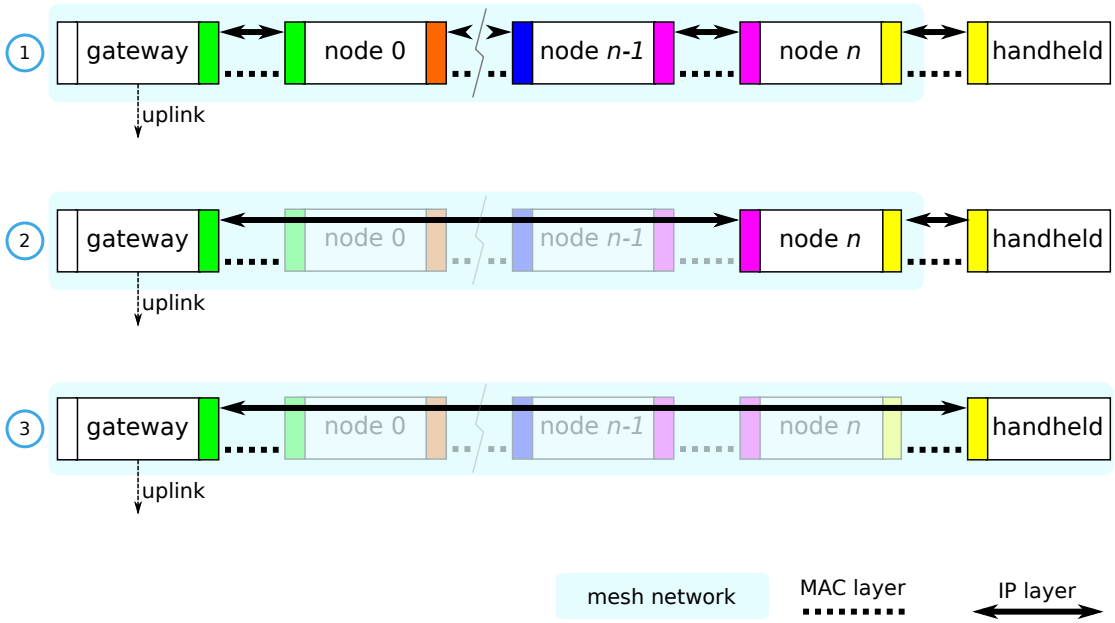


Figure 2.2: IEEE 802.11s mesh networking compared to OLSR.

Figure 2.2 shows the effect of using *layer-2* mesh networking on the network topology. With classical IP-level mesh routing protocols (1), nodes are addressed by their IP addresses and routing happens according to these addresses. This introduces additional overhead as each packet has to pass through the IP stack on every node. For OViS, a setup where the hand-held device is not part of the actual mesh network (2) was considered in order to retain interoperability with legacy operating systems and other devices that do not support IEEE 802.11s mesh networking. Once 802.11s is supported by all major platforms, the hand-held device can become part of the mesh network itself (3). Such a setup would not only reduce the overhead for IP packet processing but also abolish the need for IPv4 configuration on the intermediate nodes altogether – from an IP layer perspective, those nodes are nonexistent.

IEEE 802.11s mesh networking was evaluated by using the implementation found in Linux 2.6.35. Initial tests looked promising, i.e. the mesh network seemed to work when we used a single wireless interface on each node. For our multi-channel setup described in Section 3.4.1, we did, however, need to use multiple interfaces on each node. Since the IEEE 802.11s implementation we used does not support multiple interfaces and attempts at simply bridging the wireless cards failed [10], it was finally decided to use OLSR instead.

2.1.3 Automatic Rate Control

IEEE 802.11 wireless networks can operate at various data rates [9]. In 802.11a/g networks, data rates of 6, 9, 12, 18, 24, 36, 48 and 54 Mbit/s are supported. If the data rate of a wireless interface is not manually specified, the interface will communicate at the highest speed which is supported by all members of the network.

If a wireless node is configured to use automatic rate control, it will dynamically adjust its data rate depending on the measured packet loss. I.e., an interface that suffers from bad connectivity will be reconfigured to transmit at a lower data rate in order to achieve a slower but more reliable connection.

2.2 General OViS Concepts

In this section, we would like to present some general concepts and ideas that were helpful in designing various aspects of the OViS architecture.

2.2.1 State Pattern

The *state pattern* is a well-known behavioural design pattern in software engineering [11]. It describes a mechanism to encapsulate state information in objects. Operations are delegated to the corresponding state object. That object can then decide on what action to take as a consequence of the requested operation.

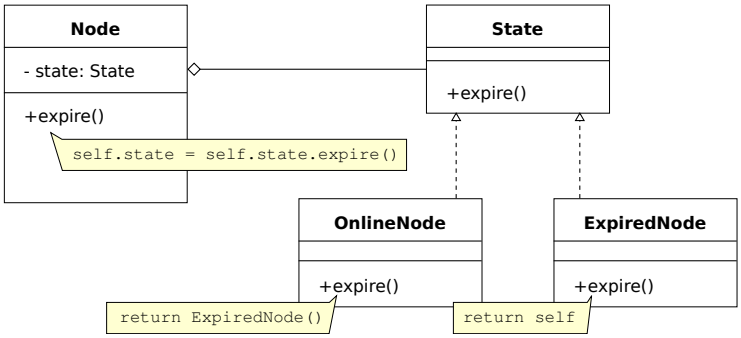


Figure 2.3: The state pattern.

Figure 2.3 shows a simple example of this pattern: `Node` objects have an `expire` method. Any calls to this method will be forwarded to the object's *state*, an object of a class that implements the `State` interface. If the previous state was `OnlineNode`, it is changed to `ExpiredNode`. Otherwise, the state remains unchanged. While this example is highly unspectacular, it shows the general idea of delegating the decision on which action to take as a result of a particular message to an object's *state* instead of relying on a chain of `if .. else` statements.

2.2.2 Adjacent Channel Interference

One major disadvantage of wireless communications on the 2.4 GHz band according to the IEEE 802.11b standard is that there is significant overlapping of the communication channels [12]. Figure 2.4 shows how the frequencies overlap – neighbouring channels suffer from significant interference. In a multi-channel ad-hoc network, channels have to be carefully selected not to interfere with each other since otherwise a node's outgoing traffic would collide with the incoming packets.

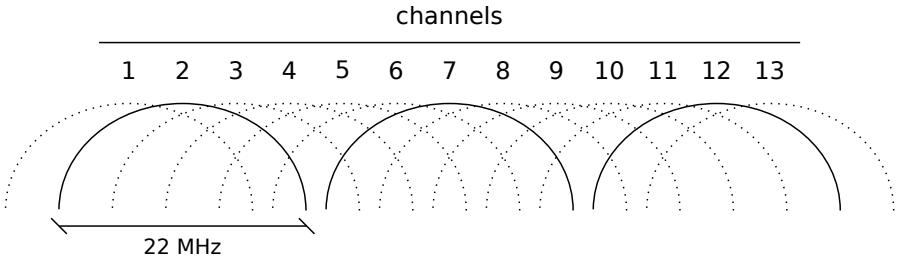


Figure 2.4: Overlapping frequencies in IEEE 802.11b.

Fortunately, the channels on the 5 GHz band specified in the IEEE 802.11a standard were chosen without overlapping frequencies. However, in [13], [14], [15] and [16], previous research has shown that, while the channels should not overlap in theory, actual implementations differ from that. In practice, traffic on a channel in the 5 GHz band still causes interferences with neighbouring channels. It is therefore advisable not to choose consecutive channels for a node's wireless interfaces.

2.3 Third-Party Software

While multiple programs and scripts were written specifically for OViS, we also rely heavily on third-party software: our Linux distribution, the video client application and several other important parts of OViS have been written by other people. On the following pages you will find a selected choice of those applications.

2.3.1 ADAM

As part of the *Administration and Deployment of Adhoc Mesh networks (ADAM)* [17] project, a Linux distribution specialized in embedded wireless networking platforms was created at our institute. The ADAM distribution is cross-compiled entirely from source for a particular type of hardware. The distribution comprises about 90 software packages, written as simple build-scripts. The packages are hand-picked (and patched) to be optimized for embedded systems, i.e., to keep a low memory footprint and to preserve CPU cycles by not running any software that is not strictly required.

The ADAM distribution proved to be very well suited for OViS as it already contains a lot of software related to the detailed configuration of wireless networking, including for instance

OLSR. Thanks to the good documentation and the clean design, only minimal effort was needed to add our custom software to the distribution (see Section 4.3), thereby creating a system that is highly efficient and optimized for our hard- and software needs.

2.3.2 Skype

Skype is a well-known application for voice and video communication over the Internet. The Skype client application is available for several different operating systems such as Windows, Mac OS X, Linux and several mobile platforms.

The Skype client offers remote-control abilities ("Skype API" [18]) that allow other programs to query the Skype application's state and initiate various activities, e.g. start a call. On Linux, the Skype application uses the Qt widget toolkit [19].

Also, Skype has a reputation for working through most firewalls thanks to *UDP hole punching* [20], a technique to allow bidirectional communications through firewalls.

2.3.3 Central Regulatory Domain Agent

The regulations regarding the wireless spectrum (i.e., the allowed channels for wireless networking and the maximum allowed power on those frequencies) vary greatly from country to country. Modern wireless drivers on Linux rely on the *Central Regulatory Domain Agent (CRDA)* [21] to determine the limits that are imposed upon them according to the country they are operated in.

CRDA uses a cryptographically signed regulatory database that contains information on those restrictions. Figure 2.5 shows how CRDA is used. When an application wants to change the *regulatory domain* (i.e., configure the network card to obey a specific country's regulations regarding wireless networking), it sends a message to the kernel via *netlink* [22] (1). The kernel then generates a *uevent* [23] to announce that change (2). This message will be picked up by a user space daemon (in our case *hotplug2*, see Section 2.3.4) which in turn spawns CRDA (3). CRDA then checks its database for the country-specific regulatory parameters (4) and applies them through another netlink message (5).

Thanks to the signed database file that is part of CRDA, the user can be sure not to violate country-specific regulations without the need of manually tweaking the wireless card's transmission power settings or explicitly avoiding forbidden frequencies.

2.3.4 hotplug2

Hotplug2 [24] is a very small replacement for *udev* [23], an application that processes events from the Linux kernel and thereby makes them available to user space applications. Hotplug2 can be used to handle changes to the regulatory settings in combination with CRDA (see Section 2.3.3). It is significantly smaller than udev and provides all the features needed for CRDA to work. It is therefore an ideal replacement for embedded systems where disk space is crucial.

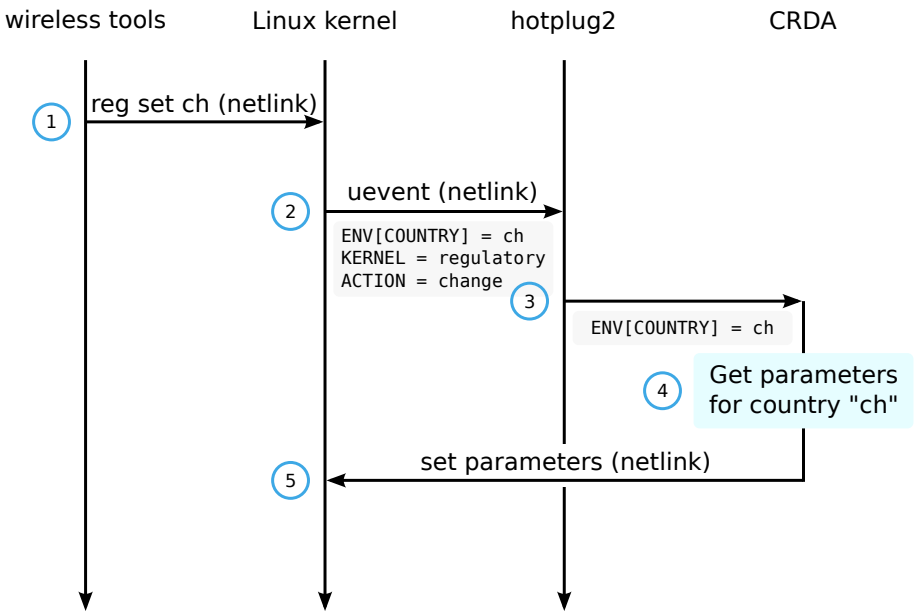


Figure 2.5: CRDA and hotplug2.

2.3.5 wxPython

wxPython [25] is a wrapper for the Python programming language around *wxWidgets* [26], a cross-platform library for writing *Graphical User Interfaces (GUIs)* that integrate smoothly into the operating system's native look and feel. wxPython facilitates the development of Python programs that resemble native applications on any operating system without requiring platform-specific code.

2.3.6 TkInter

TkInter [27] is a Python binding to Tk [28], a cross-platform toolkit for writing graphical applications, originally developed for the Tcl scripting language. TkInter brings that cross-platform compatibility to Python similar to wxPython (see above). TkInter is older than wxPython and has proven its strengths over the decades.

2.3.7 Android Platform

Android [29] is an open-source operating system for smartphones based on the Linux kernel, a collection of software libraries and an application framework centred around *Dalvik* [30], a custom Java virtual machine. Android applications are *sandboxed*, i.e. they run in containers that prevent access to other running applications and to low-level system functions. Figure 2.6 shows an overview of Android's architecture. Applications can typically only use functionality provided by the application framework and the included programming libraries. Direct access to the Linux kernel and other low-level system functionality is usually not possible.

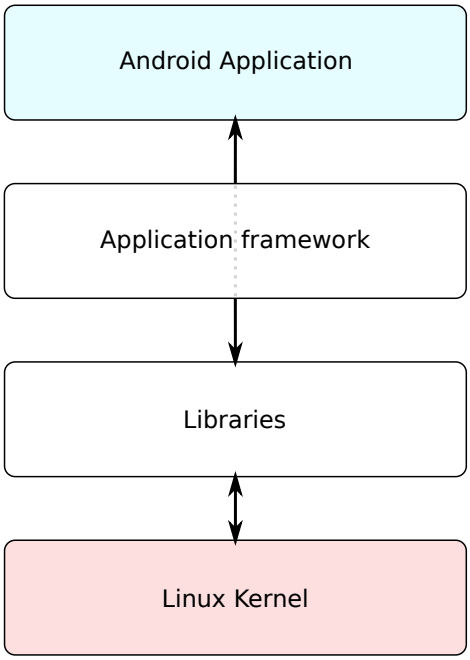


Figure 2.6: Android architecture.

Android is available in several versions for a large range of smartphones and tablet PCs. The Android applications for OViS were developed on/for an HTC *desire* [31] smartphone running a modified version of the Android 2.2 operating system. At the time of writing, Android 2.3 has been released to the public but is not yet available for the HTC desire. Some of our work (i.e. the support for ad-hoc networking described in Section 5.3.2) is specific to this hand-held device and might not work on other devices or other versions of Android. Porting should be relatively easy, though.

Android Native Development Kit

In some cases, Android applications need to get access to the Linux user-space that goes beyond what can be done through the application framework. For these situations, the Android *Native Development Kit (NDK)* has been made available to developers. With the NDK, parts of an application can be written in the C or C++ programming language and compiled natively for the underlying Linux platform (specifically, the programs will be cross-compiled for the ARM architecture on which current Android devices are based). Through the use of natively-compiled code, Android applications can run system-level commands with limited user permissions, i.e. the code that is compiled with the NDK still runs with the same limited privileges as the rest of the application.

Superuser Access

An Android application typically runs with limited permissions. Each application runs as a different system user and is thereby prevented from accessing files and memory that belong to a different application / user. More importantly, applications cannot run any operation that requires superuser permissions, such as loading kernel modules and setting networking parameters that are not supported by Android's API.

Figure 2.7 shows how these restrictions can be circumvented. A small C program runs as a native application on the phone, the application calls the program's functions through the NDK. These functions may then call the `su` system command which can be used to run a command with superuser privileges. In order for this to work, the phone has to be *rooted*, i.e. a modified version of the Android operating system must be installed.

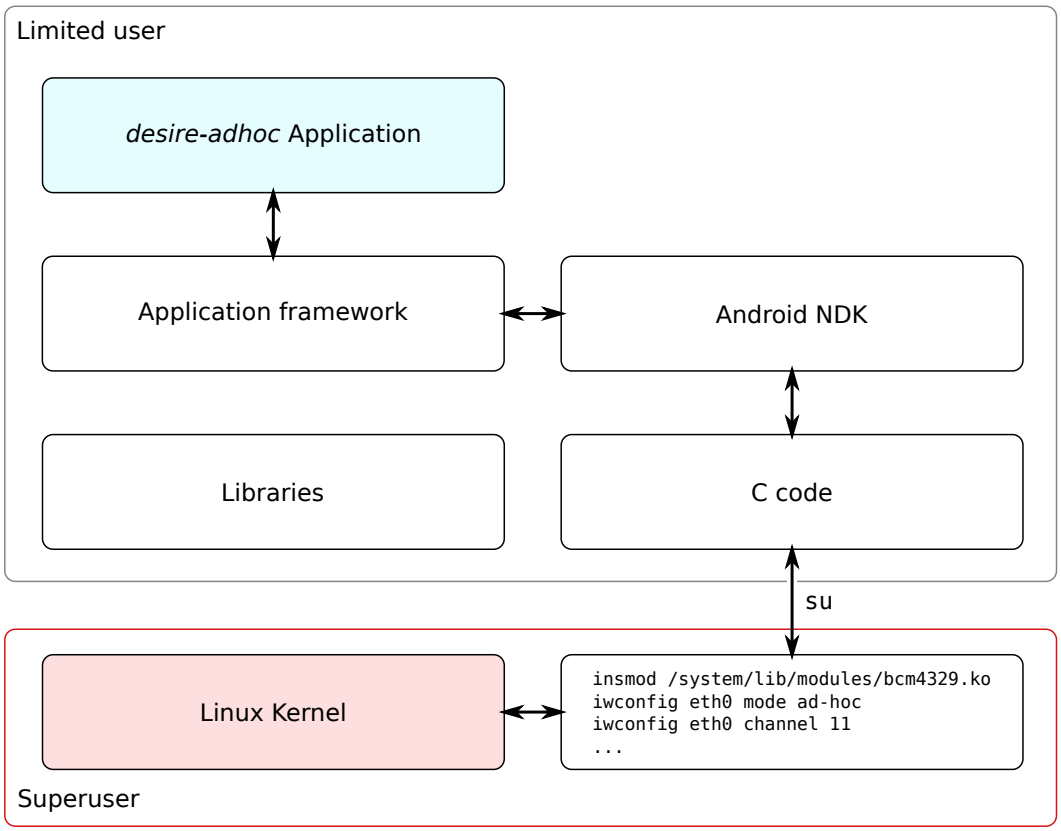
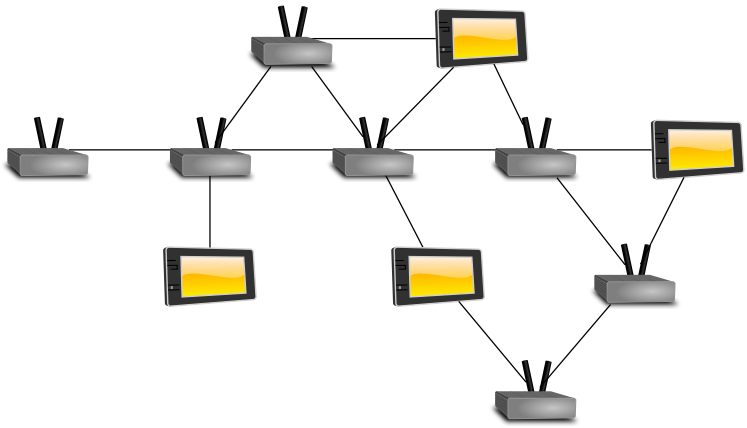


Figure 2.7: Root access through the Android NDK.

(a) OViS: single line of nodes.



(b) Rapidly-Deployable Mesh Network Testbed: full mesh.

Figure 2.8: OViS mesh topology as compared to the Rapidly-Deployable Mesh Network Testbed.

2.4 Competing Approach: Rapidly-Deployable Mesh Network Testbed

A while ago, a different approach to the problem of limited wireless coverage was presented [32]. The paper proposes an implementation where the user carries a mesh node that constantly monitors the signal strength to the network and instructs the user to deploy the node whenever the signal becomes too weak.

With this approach, the mesh network can be extended at any time. The user walks through the area that needs to be covered and deploys additional mesh nodes whenever / wherever they are needed. This allows multiple people to use the network at the same time and even lets some users extend the network while others are already using it to communicate. The deployment process is not necessarily an explicit, self-contained activity but can be paused and resumed as the user progresses through the area.

While the project shares some ideas with OViS (for instance it also uses OLSR), one major difference is that the network uses a single frequency for all links. The advantage of such a design is that it results in a full mesh topology with multiple paths through the network. Nodes can compensate for temporarily bad links and even for node failures by switching to a different path – the OViS network consists of a simple chain which breaks when a single node fails. Since the authors specifically address situations where emergency response personnel needs to cover a potentially large area, such a proper mesh topology makes a lot of sense.

In OViS, the scenario is slightly different: instead of covering a potentially large *area* for several people, the goal is to cover a potentially large *distance* for a single person. The use of multi-channel communications reduces interference in this scenario at the cost of some flexibil-

ity: end-user devices can only connect to the network at the very last node since the intermediate nodes use frequencies which might not be supported by general-purpose hardware.

Figure 2.8 shows the different resulting topologies: in an OViS network (a), end user devices will typically only connect to the last node, intermediate nodes are only used to forward traffic. While in theory nothing would prevent a device from connecting to any of those nodes, they communicate with each other on various channels in the 5 GHz band which is not generally supported by consumer-grade devices. Also, since each link uses a different frequency (see Section 3.4.1), a device cannot connect to multiple nodes at the same time. In the Rapidly-Deployable Mesh Network Testbed (b) on the other hand, the entire network communicates on the same frequency. End user devices can therefore connect to any number of randomly chosen nodes at the same time and use which ever link works best for them.

While the Rapidly-Deployable Mesh Network Testbed approach could be used for an OViS scenario, the absence of multi-channel communication and, as a result, the limitation to the 2.4 GHz band would probably increase the interference from other devices (e.g., wireless networks). On the other hand, due to the network topology used in OViS, our proposal would not be suitable for the scenario for which the Rapidly-Deployable Mesh Network Testbed was designed.

2.5 Related Publications

OViS has been published in [33], [34] and [35]. An iPhone/iPad client for OViS has been implemented by another student at our institute [36].

Chapter 3

OViS Architecture and Design

The core philosophy of OViS is to keep things as simple as possible. Emphasis was placed on re-using existing technologies and integrating them into a user-friendly application, adding simple custom-written software only where it is inevitable.

This chapter describes the network topology and the corresponding deployment process, two areas where the philosophy described above is particularly evident. Furthermore, we discuss some design alternatives that were previously used as well as the reasons for their dismissal.

3.1 Requirements

First, some high-level requirements to our system need to be formulated as a list of goals to serve for orientation purposes during OViS' development. Our requirements were:

- Develop a system that allows an inexperienced user to communicate visually at locations that are not currently covered by a communication network.
- The client software for that system has to be available for all major operating systems.
- Any networked consumer-grade device must be able to serve as a hand-held client, OViS cannot rely on uncommon, specialized hardware.
- Multi-channel communication should be used in order to reduce interference.

These requirements put some constraints on the technical side as well as the user-facing applications. Having such explicit requirements was helpful during the process of designing the general architecture for OViS.

3.2 Overview

Figure 3.1 shows a general overview of the OViS deployment process. The user arrives on site with the OViS system packed away (1). The first node is then connected to the local router (2) and additional nodes are added while the user walks towards the position where the conversation is supposed to take place ("switching unit" in the Figure) (3). When the destination has been reached, the video conference with the qualified off-site engineer is started (4).

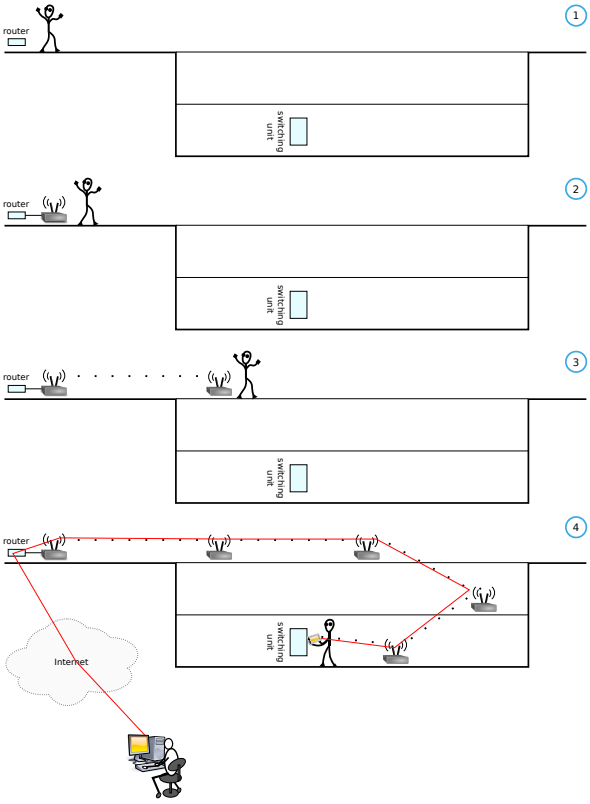


Figure 3.1: OViS deployment.

3.3 Logical Network Topology

The network topology describes the design of the underlying network that OViS is built on. OViS is designed to be network-agnostic, i.e., the client does not require any special features from the network other than the ability to send IP traffic over it. In addition, the requirements specified in Section 3.1 were taken into account.

Specifically, the first requirement means that we cannot assume any prior knowledge about wireless networking in general or ad-hoc networks in particular from the user. Also, the local situation will be different at each site. The topology has therefore been designed with flexibility in mind – the network is supposed to work regardless of local obstacles.

Since the hand-held device does not generally have any information on its absolute position, mechanisms like GPS cannot be used to acquire detailed positioning information and possibly display it or utilize it to calculate ideal positions for the nodes.

Lastly, multi-channel communication simplifies the task of predefining a network topology: nodes can only communicate with each other if they use the same frequencies, therefore the available links on the network can be defined by setting the frequencies accordingly.

On the following pages, we are going to present several design ideas for the OViS network topology and discuss their advantages and drawbacks.

IPv6 Topology

The topology shown in Figure 3.2 creates a "pure" IPv6 network between the mesh nodes – none of their wireless interfaces have an IPv4 address configured except for the very last node that communicates with the hand-held device. Preliminary tests showed that this works quite well thanks to the excellent IPv6 support in olsrd (see Section 2.1.2).

To stay compatible with IPv4 user space applications and the IPv4 Internet, the hand-held device is connected to the last node through a private IPv4 network and an IPv4-in-IPv6 [37] tunnel through the OLSR network bridges that private network with the IPv4 network managed by the local router.

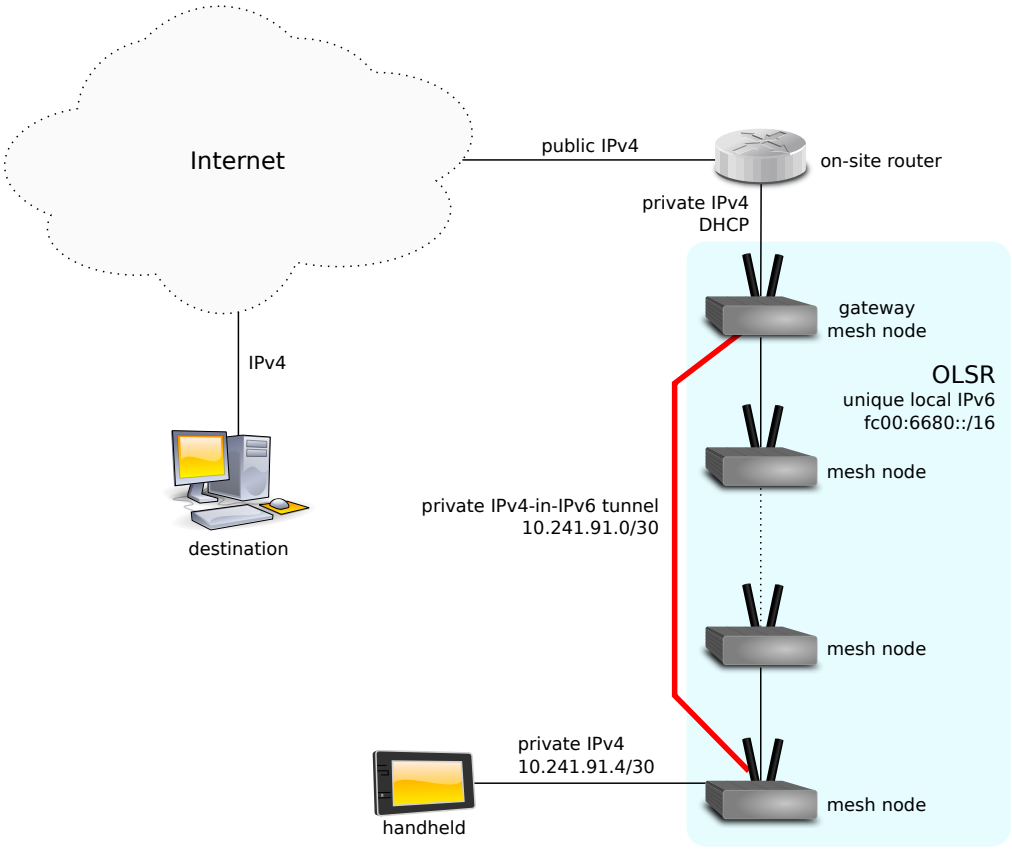


Figure 3.2: Network topology with an IPv4-in-IPv6 tunnel.

An IPv6 topology would have the benefit of integrating seamlessly into the Internet once IPv6 becomes more widely used. In particular, the need for *Network Address Translation (NAT)* [38] would fall away and the hand-held device would be properly addressable and fully reachable from any other host connected to the Internet. This would make it significantly easier to establish connections for services such as video conferencing.

On the other hand, the obvious drawback of this topology is the complexity due to the IP tunnel, in particular the network's reduced *Maximum Transmission Unit (MTU)* as a result of the

extra headers that are required for tunnelled packets. The tunnel has an MTU that is smaller than the one generally used in the wireless network. Therefore, the hand-held device needs to have its MTU reduced manually. Otherwise the additional packet fragmentation might significantly lower the network's throughput.

Full OLSR Topology

Figure 3.3 shows another design idea: extend the OLSR network to include the hand-held device instead of having a private network between it and the last node.

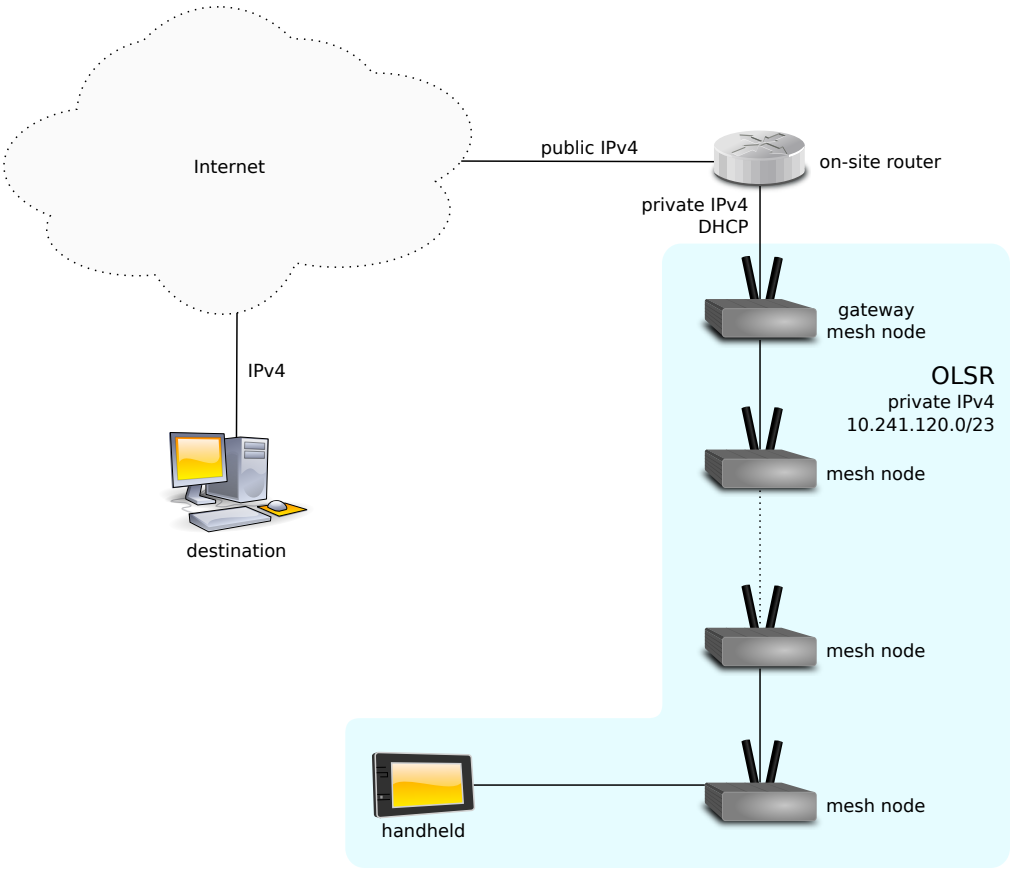


Figure 3.3: Network topology with OLSR.

This approach significantly simplifies the routing setup for our network. OLSR automatically takes care of all the routes, no additional configuration is needed on the hand-held device. Furthermore, this topology is easy to port to IPv6 since only the OLSR configuration has to be updated and node discovery can be done simply by looking at OLSR's neighbour list.

Running OLSR on the hand-held device might not seem like a big problem. However, when mobile phones enter the picture, the situation becomes a whole lot more complicated. While ports of *olsrd* for various mobile platforms such as Android and the iPhone exist [39], that software requires superuser privileges to work. Such privileges are not normally available to the

user and would require modifications to the phones' operating system. It was therefore decided not to pursue this approach any further.

IPv4 Topology

A plain IPv4 topology is shown in Figure 3.4. The OLSR network between the nodes uses a private IPv4 subnet, another private subnet is used for communications between the hand-held device and the last node. The second private subnet is announced through the OLSR network by using the HNA mechanism (see Section 2.1.2) in order to make it visible (and routable) for the other nodes and in particular for the gateway node.

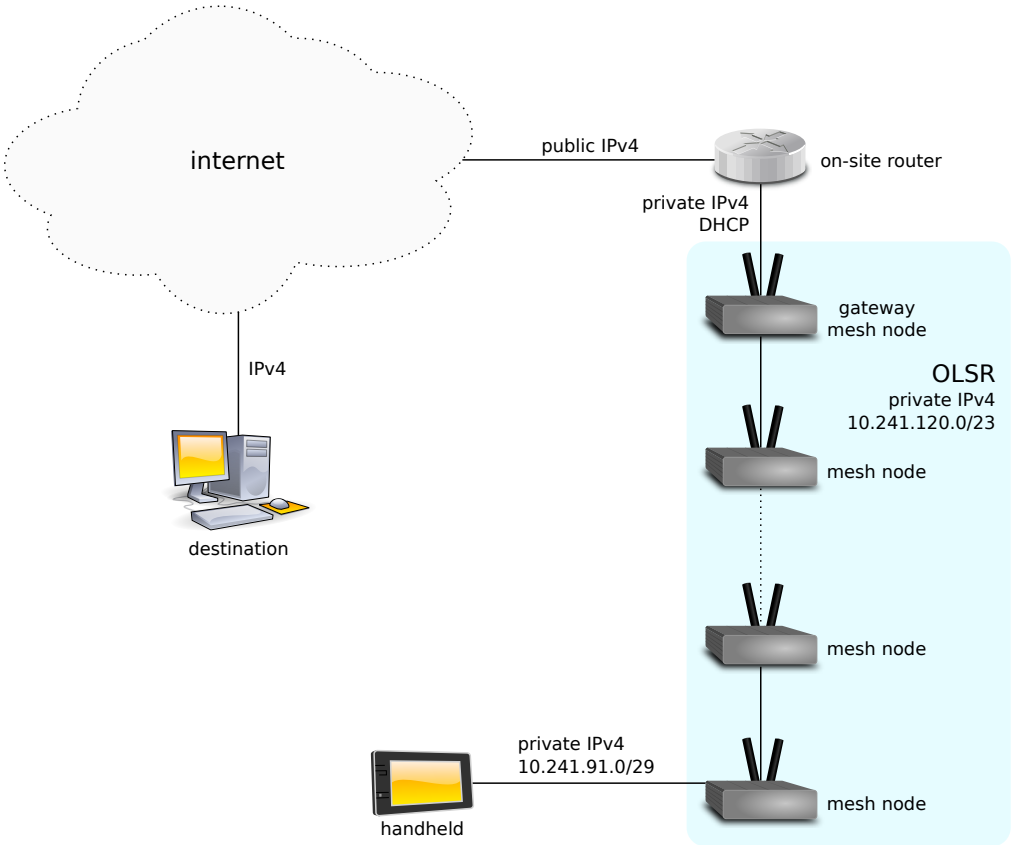


Figure 3.4: IPv4 Network topology.

The advantage of this topology is its simplicity: the hand-held device can use IPv4 without the need for any special configuration or, even worse, superuser permissions. All that is required is the ability to change the IP configuration – a feature that is offered by all platforms considered.

The plain IPv4 topology requires special configuration on the last mesh node in order for the HNA mechanism to work. Additionally, it means that traffic from the hand-held device to the Internet will in most cases have to pass through two hosts that do NAT: once on the last mesh node and once more on the on-site router. This effectively destroys any chance for the

node to ever be addressable directly by other machines on the Internet and means that no direct communication with the node is possible.

The Final Topology

After careful evaluation, it was decided that OViS use the IPv4 topology shown in Figure 3.4 and only use IPv6 during the deployment process (see Section 3.4.2). The requirement for OLSR on the hand-held device makes the OLSR topology hard to realize and the additional tunnel required for an IPv6 network needs too much tweaking that would be hard to do on a mobile phone. All communications between the hand-held device and the mesh nodes during the initial configuration of the network happen via IPv6 though, since these messages do not need to be routed through the Internet and the general availability of IPv6 is therefore irrelevant in this case.

The mesh nodes are deployed in a single line where each node can only communicate with exactly one other node per network interface. This topology allows us to cope with each of the requirements mentioned in Section 3.1:

- By requiring nothing but a single line of nodes, the user can be instructed to walk from the Internet access point to his destination, dropping nodes at convenient locations in between. The user does not have to worry about details concerning a mesh network or about which nodes can communicate with each other due to the different network channels involved.
- No absolute positional data is needed for this setup. The ideal position of a node can be computed solely as a result of the signal strength to the previous node.
- Communication frequencies can be configured on a per-link basis without affecting the rest of the network.
- This topology is entirely transparent to the client host and looks like any other IP-based network.

As for the IP layer, the OViS network consists of two private IPv4 networks. One (`10.241.120.0/23`) between the mesh nodes managed by OLSR (see Section 2.1.2) and the other one (`10.241.91.0/29`) between the last node and the hand-held device. Each node's first wireless networking interface is configured to be in the `10.241.120.0/24` network, the second interface receives the corresponding address in the `10.241.121.0/24` network. The local network that the gateway node connects to is assumed to also be a private IPv4 network with a properly configured DHCP server (see Section 2.1.1).

During the deployment process (see Section 3.4), the last node's secondary interface is reconfigured to an address in the `10.241.91.0/29` network in order not to be affected by OLSR's routing mechanism. As mentioned above, the additional subnet is then announced to the other nodes through OLSR's HNA mechanism to have appropriate routes created on every node on the OLSR network.

This leads to a fully operational network consisting of two different subnets that are routed through the last mesh node and connected to the Internet through the gateway node, thereby creating the foundation on which the OViS application described in Chapter 5 can be built.

3.4 Network Deployment

The OViS network as described above is a mesh network consisting of numerous nodes. We are now going to explain the process of configuring these nodes after bootup and having them placed at the right position to get the OViS network up and running.

3.4.1 Multi-channel Communication

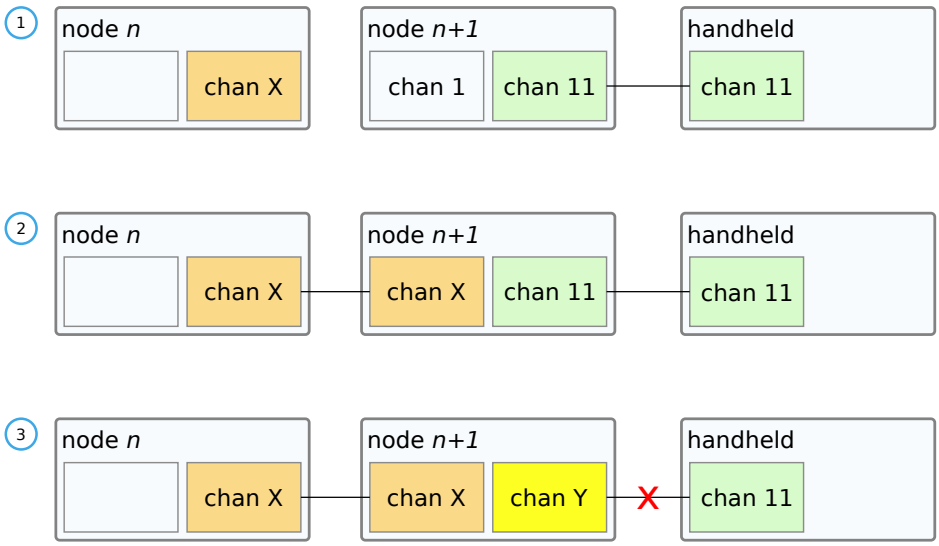


Figure 3.5: Deploying node $n + 1$.

As discussed in Section 3.3, one of the requirements for the OViS network is to support multi-channel communication. The reason behind this requirement is the idea to prevent the traffic on a network link from interfering with the signal on other links. The links between the nodes use the 5 GHz band because the 2.4 GHz band is already being used by countless other devices such as Bluetooth devices and other IEEE 802.11 wireless networks to name a few.

Figure 3.5 explains the basic mechanisms used to connect a node to the multi-channel network. It is assumed that at this point the network already contains n nodes and that it will at the end comprise at least $n + 2$ nodes. That is, the node being connected is neither the first nor the last node in the network; the process differs slightly for those nodes as explained below.

The hand-held device keeps a list of all available channels and the order in which to use them. Whenever a channel is assigned to an interface, the corresponding frequency is removed from the top of the list and re-added at the end. This way, the first list element always indicates the frequency for the next link and the last element can be used to determine the frequency that was used for the previous link.

When the new node $n + 1$ boots up, its first wireless interface is set to channel 1 (2.412 GHz) and its second interface is communicating on channel 11 (2.462 GHz), the same channel that is used by the hand-held device's wireless interface (1). It is worth mentioning here that the channel on the hand-held device is never changed as a) this would require superuser permissions

and b) the hardware we use does not support wireless networking in the 5 GHz range anyway. At this stage, the new node can only communicate with the hand-held device and is not connected to the rest of the mesh network.

The first interface is then reconfigured to the same frequency that the previous node uses on its second interface (2), a frequency in the 5 GHz band (called *chan X* in the figure). As mentioned above, the client application can find out which frequency to use by looking at the end of its frequency list. The node is now connected to the mesh network as well as to the hand-held device. At this point, the user will be instructed to find the appropriate location for the node as described in Section 5.2.3.

Finally, when the user chooses to deploy the node, the second interface is reconfigured to use the next unused channel (*chan Y*) in the 5 GHz range (3), causing the connection to the hand-held device to be lost. Due to this connection loss, the client cannot verify whether this action actually succeeds or not. The entire process is explained in detail in Section 3.4.2.

As mentioned above, minor modifications are required for the first and last nodes. On the first node, step (2) is skipped, the first network interface remains on channel 1. Since we are deploying a chain of nodes, this interface will never be used and by leaving it on channel 1 we make sure that no other node will ever talk to it. Similarly, on the last node we omit step (3), thereby allowing the node to keep communicating with the hand-held device over its second interface. In theory, it would be possible to also use a frequency in the 5 GHz band for communications between the last node and the hand-held device. In reality, however, consumer-grade network devices are often limited to the 2.4 GHz range. Thus, in order to be compatible with a large range of devices (such as mobile phones), channel 11 seemed like a good choice.

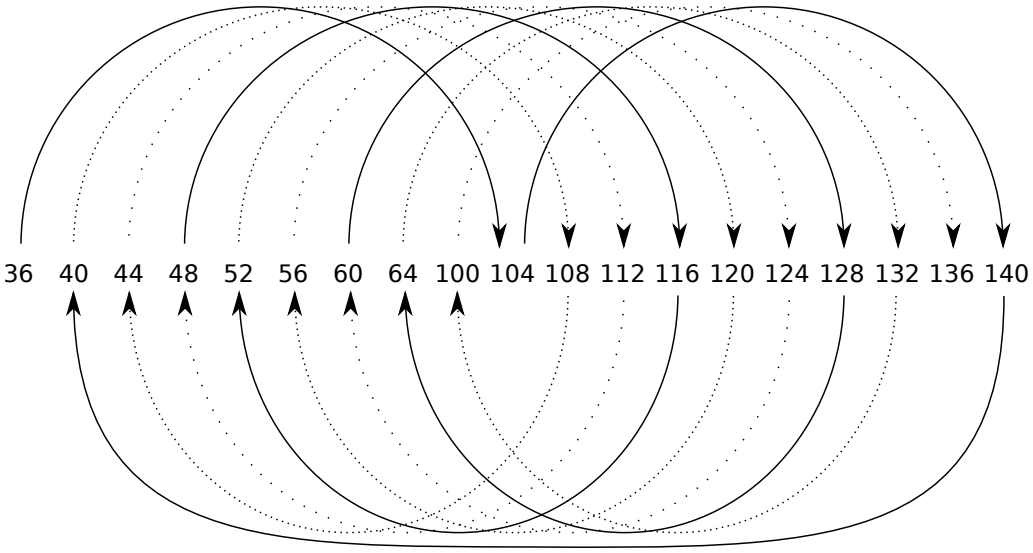


Figure 3.6: IEEE 802.11a channel allocation in OViS.

Section 2.2.2 explains that it is not advisable to use consecutive channels on the 5 GHz band for both interfaces of a node since data traffic on one channel may cause interference on its neighbouring channels. OViS, therefore, allocates channels as shown in Figure 3.6. Figure 3.7

illustrates how to interpret the allocation mechanism: the first link uses channel 36, the second link channel 104, the third link channel 140, etc. – this allocation scheme allows us to spread our data traffic over the entire range of available frequencies while minimizing the interference from neighbouring channels.

Figure 3.7: Example of OViS channel allocation.

3.4.2 Deployment Process

A general overview of the deployment process has been shown in Figure 3.1. We are now going to explain the mechanism used to connect a node to the network, i.e., what happens from the moment the node first talks to the network until it is fully configured.

Please note that the ideas discussed here lack the mechanisms required for multi-channel communications discussed in Section 3.4.1 since adding them to the figures would only clutter them needlessly.

Deployment with DHCP

Figure 3.8 shows the first design idea:

- The nodes boot up without any preexisting IP configuration and are configured using DHCP (1) with the hand-held device acting as a DHCP server.
- The hand-held device can discover new nodes by looking at the DHCP leases and then get additional details about the nodes (such as the host name and whether the node is a gateway connected to a local network) through an HTTP API (2), possibly encoded using JSON [40] or YAML [41].
- When the user / the client application reaches the stage where details about the node are needed (3), those details are available thanks to the previously made HTTP request.
- The user is then directed to find the right position for the node (4).
- Finally, the node is deployed through the GUI running on the hand-held device (5).

The design was ultimately dismissed as the application is supposed to be platform-independent, that is run at least on the Linux, MacOS and Windows operating systems. This would have required us to either find and configure a cross-platform DHCP server or write one of our own – neither of which seemed like a good and/or feasible idea at the time.

This decision was later proven correct when the idea came up to use smartphones as the hand-held device. A DHCP server process needs to open a socket on port 67 and this operation can only be done with superuser privileges [42]. Running software with such permissions on

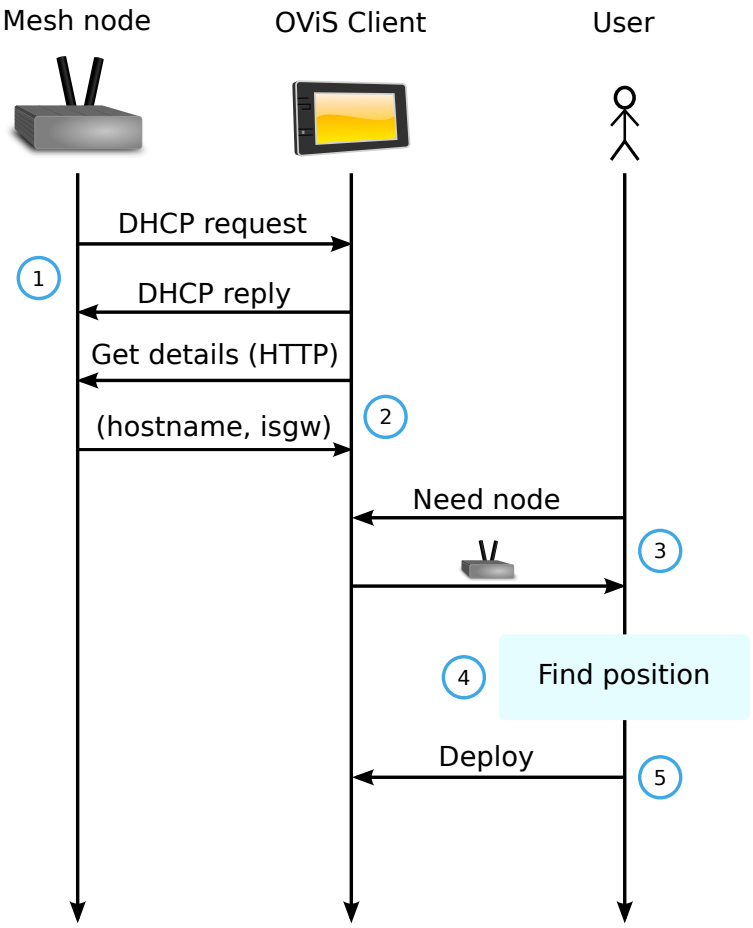


Figure 3.8: Network deployment with DHCP (dismissed).

smartphones is, however, tedious at best and requires significant modifications ("rooting") to the phone's operating system.

Deployment with IPv6 Auto-configuration

Our second approach shown in Figure 3.9 involves IPv6 auto-configuration and was designed with the IPv6 topology described in Section 3.3 in mind.

- After startup, the IP addresses are configured automatically using the IPv6 auto-configuration mechanism described in Section 2.1.1.
- The nodes then start sending ICMPv6 [43] *Echo Request* ("ping") packets to the hand-held device (1).
- Details about the node are again fetched via HTTP (2).

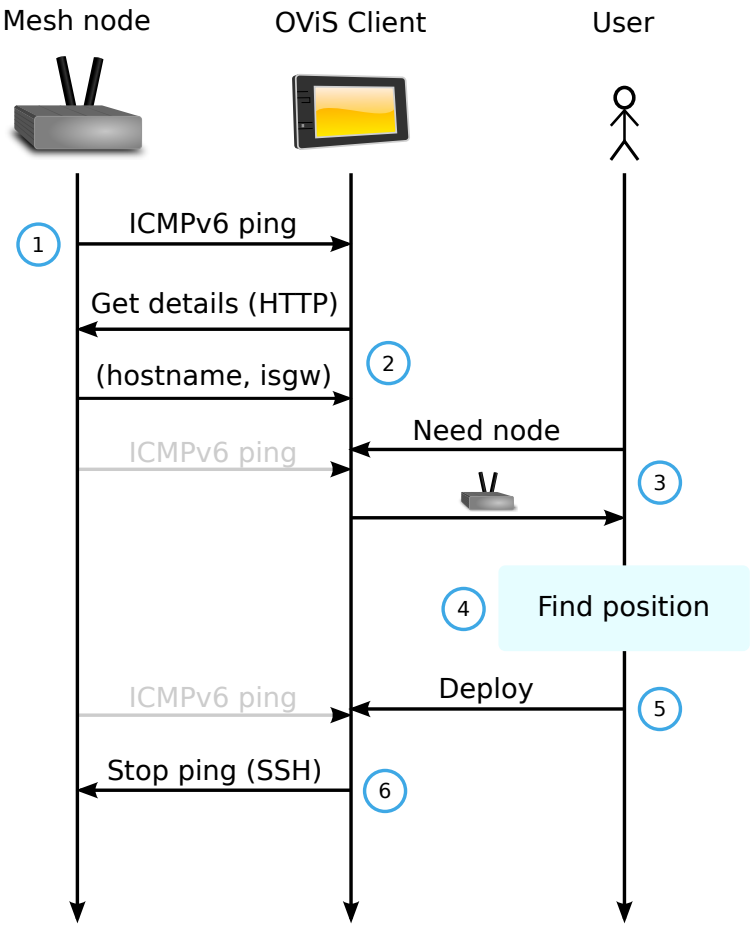


Figure 3.9: Network deployment with IPv6 auto-configuration (dismissed).

- The user interaction (3-5) remains the same as described in the DHCP deployment scenario.

- After deploying the node, the ping process is terminated through an Secure Shell (SSH) session (6).

However, this design also has several drawbacks. First of all, in order to receive ICMP packets, a "raw" socket has to be opened. This operation is again restricted to superusers [44]. Also, initial tests have shown that while this works just fine on Linux, the application does not seem to receive any data on Mac OS (this was not investigated any further). Moreover, in a mesh network, this would mean that every node has to run the *Linux IPv6 Router Advertisement Daemon* (radvd, see Section 2.1.1) and auto-configure itself using another node's advertisement messages, thereby introducing a bootstrap problem on the first node. Those tests also made us realize that radvd cannot be run on network interfaces that are themselves auto-configured.

Deployment with a UDP Pinger

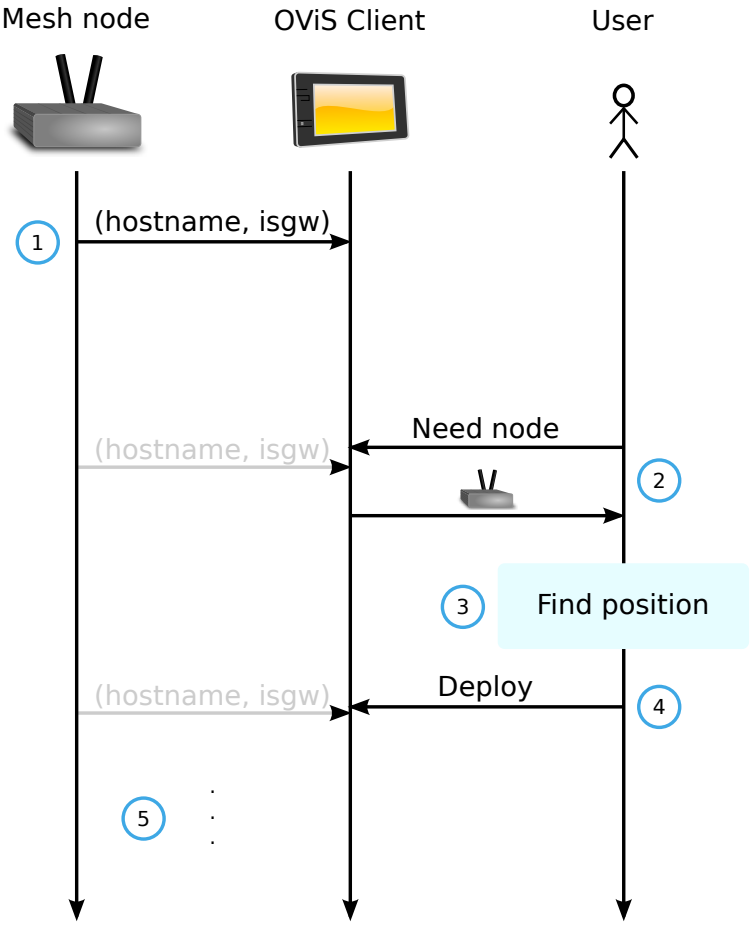


Figure 3.10: Network deployment with UDP pinger (accepted).

A third idea for the deployment process is shown in Figure 3.10. For this scenario the nodes need preconfigured IPv4 and IPv6 addresses.

- In order to indicate its presence on the network, a node starts a custom-written program (*pinger*) after bootup that will broadcast some information about the node over UDP on an unprivileged port (1) (see Section 4.3.1). This information contains the node's host name as well as a flag indicating whether the node is a gateway node or not. All the information needed in step (2) is thus broadcast by the *pinger*.

- The user interaction (2-4) remains as described in the previous two approaches

- The ping process is left alive "forever" (5) – this was deemed less problematic than starting SSH sessions in the background.

The downside of this approach is the need for an additional piece of custom-written software to announce the nodes' presence to the network.

The final deployment process

While all the designs proposed in this chapter share a philosophy of interconnecting simple, lightweight mechanisms to deploy the OViS network in a reliable and efficient way, only the last one can be implemented with limited user permissions on the hand-held device. This advantage outweighs the slight drawback of using a custom-written software for announcing new nodes to the network, therefore that design was chosen as a basis for the OViS network deployment process.

Figure 3.11 shows the process augmented by the mechanisms needed for multi-channel communication as described in Section 3.4.1.

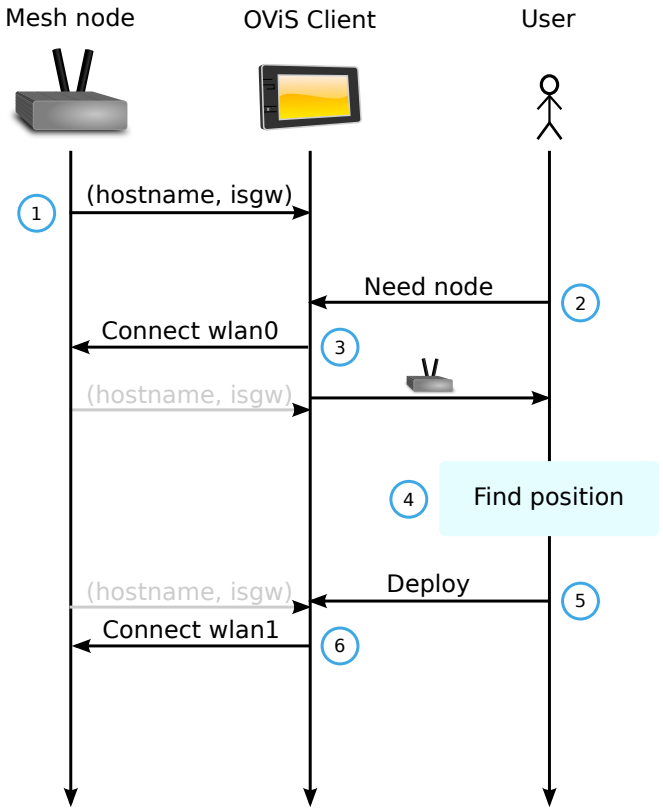


Figure 3.11: Final network deployment process, including mechanisms for multi-channel networking.

For the sake of completeness, let us recapitulate:

- After bootup, the nodes start broadcasting some information about themselves. That information contains the node's host name and a flag that indicates whether the node is a gateway, i.e., whether it has a link on its Ethernet interface. This data is sent via UDP to the link-local multicast address, `ff02::1` (1).

- When a new node is requested by the user / client application (2), it gets connected to the previous node as described in Section 3.4.1 (3): namely the first interface's frequency is changed to a channel in the 5 GHz band via an HTTPS configuration interface (see Section 4.3.1 for details).
- The user is then guided into finding the optimal position for the new node (4).
- Once the system (OViS) is satisfied with the node's position, the user chooses to deploy the node (5).
- The second interface is then also configured to switch to the 5 GHz range (6).

Note that the UDP "ping" packets as well as any other messages between the node and the hand-held device during the deployment process are sent via IPv6. Since ADAM already uses IPv6 for management tasks, it makes sense to also employ the new version of the IP protocol for OViS-specific management operations.

As explained in Section 3.4.1, things are slightly different for the first and last node. The steps in Figure 3.11 that are left out are (3) on the first node and (6) on the last one.

This process yields a simple multi-channel topology while only requiring minimal interaction with the user. All operations on the hand-held device can be done with limited user permissions and the number of network connections is reduced to the bare minimum.

Chapter 4

OViS Mesh Nodes

The OViS mesh nodes are running the Linux distribution created for the *Administration and Deployment of Adhoc Mesh networks (ADAM)* project described in Section 2.3.1. In this chapter, we are going to explain the modifications that were done to the standard ADAM distribution and why they are required in order for OViS to work.

We will also explain the deployment process as seen from the mesh node's perspective, that is what happens on the node from the moment it boots up until it is a fully deployed part of the OViS mesh network.

4.1 Hardware

The mesh nodes are based on the "ALIX" embedded computer platform from PC Engines [45] with a 500 MHz AMD Geode CPU, 256 MB RAM, an Ethernet interface and two wireless cards with Atheros AR5213A chips. Figure 4.1 shows such a node.

While in general OViS does not care too much about the hardware it runs on, it was relevant in this case since the ADAM embedded Linux distribution (see Section 2.3.1) recently substituted the old `madwifi` driver for `ath5k`, a new driver that is part of the official Linux kernel and that uses the new Linux wireless subsystem [46]. To use this new driver with OViS, some additional software was added to ADAM (see Section 4.3.2).

A side-project of the OViS development was to add batteries to the mesh nodes. The batteries are contained in a chassis identical to the one normally used for the node. These chassis-contained batteries are then bolted to individual nodes, creating what looks like a "double node" as shown on Figure 4.2.

With the additional batteries, we have a powerful, open and transportable platform the can be used for the OViS mesh network.

4.2 Deployment Process

Section 3.4 explains the deployment process from the client's perspective. We are now going to show what happens "behind the scene", on a mesh node, as the network is being deployed.

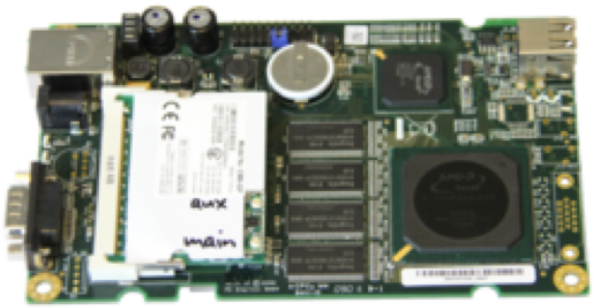


Figure 4.1: An OViS mesh node.

When a node is started, its first wireless network interface communicates on channel 1. As discussed in Section 3.4.1, the second interface uses channel 11 by default. This channel is used by the hand-held device to communicate with the node, i.e. to reconfigure the network interfaces and to read the signal strength of the connection to the previous node.

At the end of the bootup process, the node starts announcing its presence via the *User Datagram Protocol (UDP)* to `ff02::1`, the *link-local* IPv6 multicast address (see Section 4.3.1 for details). As described in Section 3.4.2, these messages are then received by the hand-held device where the OViS application (see Section 5.2) uses them to detect the node. The messages contain the node's host name and some information about its (wired) network connectivity that is used by the OViS application to determine whether the node can be used as a gateway. Section 4.3.1 describes this in more detail.

The node then continues to send presence announcements and waits for the OViS client to take initiative. The client is expected to configure the node through an HTTPS *Common Gateway Interface (CGI)* [47]. Section 4.3.1 explains the remote network configuration API in more detail.

Table 4.1 shows a typical sequence of commands used to deploy a node, Figure 4.3 illustrates this in more detail. On the first (the gateway) node, the deployment process starts by enabling IP masquerading (a special kind of NAT [38]) on the wired interface (1). Afterwards, the second wireless card is set to a frequency in the 5 GHz range and its ESSID is changed (2) – the gateway is now ready.

The process is a little more complex on intermediate nodes (non-last, non-gateway nodes): first, the frequency and ESSID are changed on the first interface in order to connect the node to the previous one (3). Afterwards, the client application queries the node about the connection

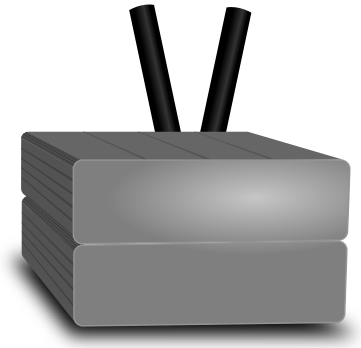


Figure 4.2: A battery-powered OViS mesh node.

Node type	Commands
Gateway	`masq_iface=eth0`
	`iface=wlan1, freq=5180, essid=fdca:140b:acdf`
Intermediate node	`iface=wlan0, freq=5180, essid=fdca:140b:acdf`
	`iface=wlan1, freq=5200, essid=fdca:140b:acdf`
Last node	`iface=wlan0, freq=5200, essid=fdca:140b:acdf`
	`iface=wlan0, mask=23`
	`iface=wlan1, ip=10.241.91.1, mask=29`

Table 4.1: Typical sequences of network configuration commands.

strength to its previously deployed neighbour. Once the user chooses to deploy the node, the second interface is reconfigured to use the next unused frequency (4).

Finally, when the last node is deployed, there are again some particularities. The first interface is configured just as it happens on any other non-gateway node (5). However, in order to make sure that OLSR routes all traffic to any of the other nodes through the first interface, its network mask is reduced to `/23` (6) – this way, the interface covers both `/24` networks that are normally spread over the two wireless interfaces. The second card's IP configuration is then modified to use an entirely different private IPv4 network to communicate with the hand-held device (7).

4.3 OViS Mesh Node Implementation

The OViS mesh nodes run a modified version of the ADAM Linux distribution (see Section 2.3.1). This distribution was selected because of its simple yet flexible architecture and due to its proven reliability on our hardware in similar projects.

The standard ADAM Linux distribution already includes a lot of software related to (wireless) networking. For OViS, we added a few extra tools: some of them are custom-written software required by the OViS deployment process, others are third-party software that is of use to other projects based on ADAM as well.

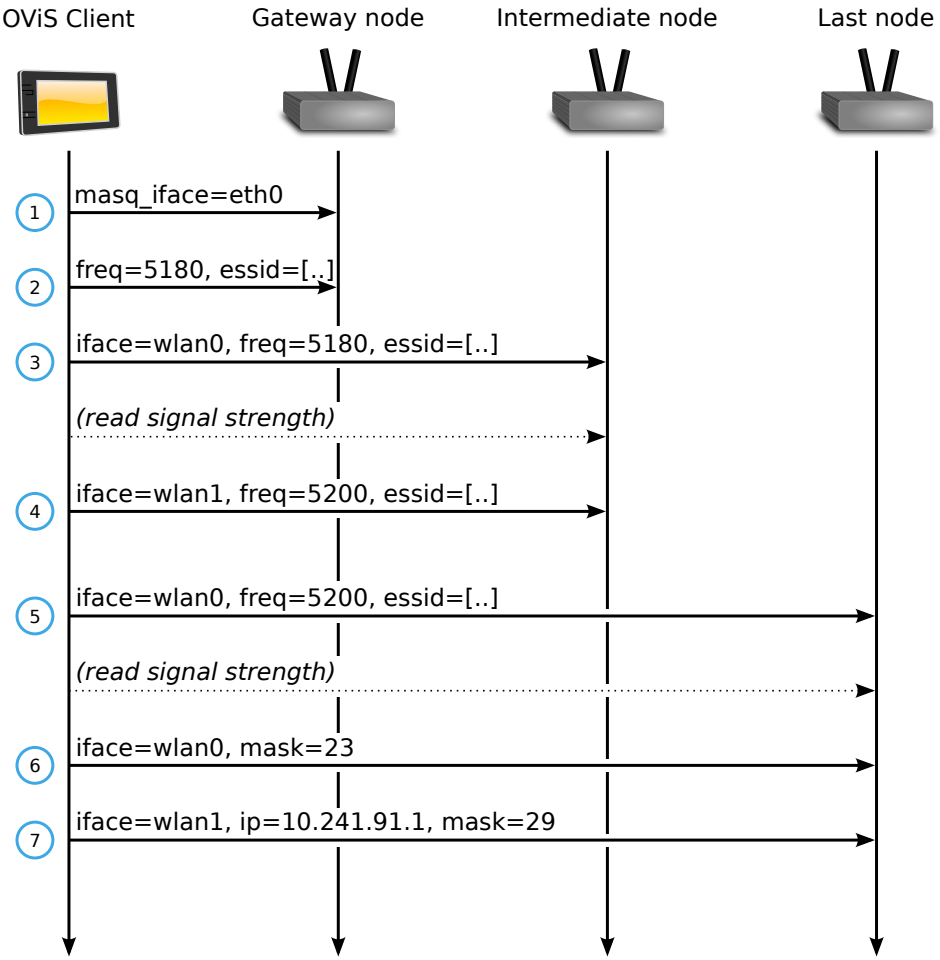


Figure 4.3: Typical sequences of network configuration commands.

In this section, we are going to explain the most important programs that were added to ADAM. We discuss why they are needed and how they work – the latter is explained in detail for our custom software, the third-party tools mentioned are well-known open-source software that come with good documentation and are only explained briefly.

4.3.1 OViS Mesh Node Utilities

The deployment process as explained in Section 3.4 imposes some specific requirements on the mesh nodes that are part of the OViS network:

- The nodes are required to announce their presence using UDP broadcast messages.
- Remote network reconfiguration is done over HTTPS/CGI, must therefore be possible for unprivileged users.

To fulfill these requirements, several additional programs were added to the ADAM build system and installed on the nodes. Emphasis was put on keeping the programs small, simple and robust.

Figure 4.4 depicts the role of two programs that were written specifically for OViS and how they interact: the CGI script *netcfg.sh* that receives and processes the configuration commands and the *OViS network watcher*, a daemon that applies the changes made through the CGI script to the running system.

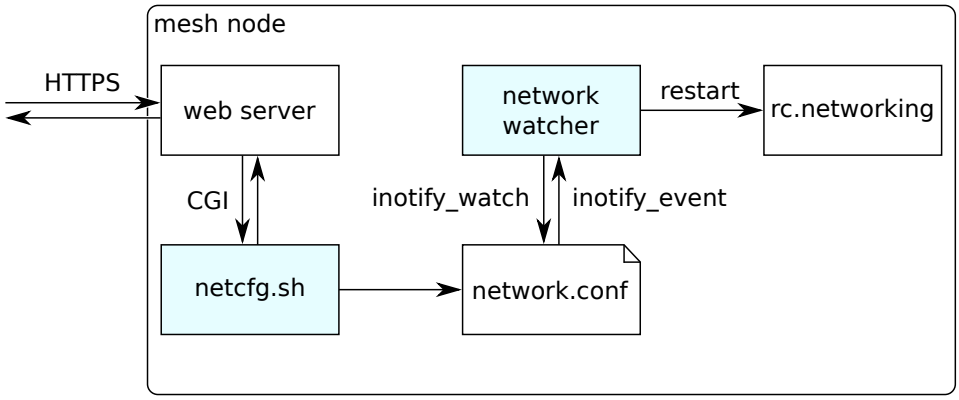


Figure 4.4: OViS remote network configuration.

Theoretically, those two programs could be combined into a single application that receives the configuration requests and directly applies them. The decision to split the tasks into two separate tools was made for several reasons:

- To prevent random observers from running network configuration commands on our nodes, some authentication mechanism is needed. Therefore, the `pass` parameter in table 4.2 is required to run any configuration commands. By using a CGI interface we can utilize the web server's *Secure Sockets Layer (SSL)* functionality to encrypt the commands and thereby hide the password from curious observers.

- The downside of using a CGI script is that it is executed with the limited privileges of the user that the web server is running as. This user does not have permission to reconfigure the network. Running the web server with increased security clearance (i.e., as the *root* user) would have solved this, but it would also open the system to attacks if/when security holes in the web server are found.

- By implementing the (comparatively) complex task of parsing and applying the configuration options in a shell script, we can use ADAM's integrated features to manage the network configuration file while limiting the part that requires superuser permissions to the task of detecting changes in the configuration file and restarting the network when needed.

In the following, the OViS utilities for remote and local network configuration are explained in detail. These utilities implement the requirements described above.

Parameter	Description	Example
pass	Authentication token required for configuration	s3kr1t
iface	The interface to configure	wlan1
freq	The frequency to set the interface to	5180
essid	The ESSID for the interface	ovismesh
ip	The IPv4 address for the interface	192.168.1.19
mask	Network mask for the interface	24
tun6ip6local	Local IPv6 address for the tunnel	fc00:6680::3:1
tun6ip6remote	Remote IPv6 address for the tunnel	fc00:6680::3:2
tun6ip4local	Local IPv4 address for the tunnel	192.168.2.1
tun6ip4localmask	Network mask for the tunnel	24
tun6ip4remote	Remote IPv4 address for the tunnel	192.168.2.2
tun6destnet	IPv4 network behind the tunnel	192.168.3.0
tun6destmask	Network mask for the network behind the tunnel	24
tun6route	Set the default route via the tunnel	yes
ipfwd	Enable IP forwarding	yes
masq_iface	Outgoing interface for IP masquerading	eth0

Table 4.2: OViS remote network configuration commands.

Remote Network Configuration: CGI script

The remote network configuration is handled by a CGI shell script. Table 4.2 shows a list of all the commands that are supported by the script and explains what they do.

As shown in Figure 4.4, the CGI script does not actually modify the network settings. Instead, it writes the changed values to the node's network configuration file (`/etc/conf.d/network.conf`) and relies on the *network watcher* (see below) to notice and apply those changes.

For this to work, a minor modification to the standard ADAM image was required: the permissions for the network configuration file were modified to grant write access to the web server.

By implementing this as a CGI script, we can use the web server's encryption and socket management features and focus on the actual task of modifying the network configuration file, thereby eliminating a lot of redundant code. Since this task is mostly about parsing arguments and writing files, a shell scripting language seemed like the natural choice rather than a more complex programming language such as C.

Local Network Configuration: OViS Network Watcher

The second part of the network configuration is handled by the *OViS network watcher* (*onw*), a small program that runs as a system *daemon* (i.e., a background process). This daemon observes the network configuration file and reconfigures the network whenever modifications to that file are detected. The program is started during the bootup process.

To detect modifications to the configuration file, *onw* uses *inotify* [48], a mechanism to

monitor file system events that was added to the Linux kernel with version 2.6.13 [49]. Figure 4.5 describes the use of *inotify* within the OViS network watcher. First, *onw* registers itself with the kernel (1). It then proceeds to registering its interest in the network configuration file by calling `inotify_add_watch` (2). This call has two relevant arguments: the path of the file to be monitored (in our case, the network configuration file) and the operations we are interested in; for *onw* that is `IN_CLOSE_WRITE`, the event that occurs when a file that has previously been opened for writing gets closed. Afterwards, whenever a call to the CGI script (or, theoretically, any other operation) results in changes to the network configuration file (3), *onw* receives an `inotify_event` (4) and will, as a result, restart the node's network.

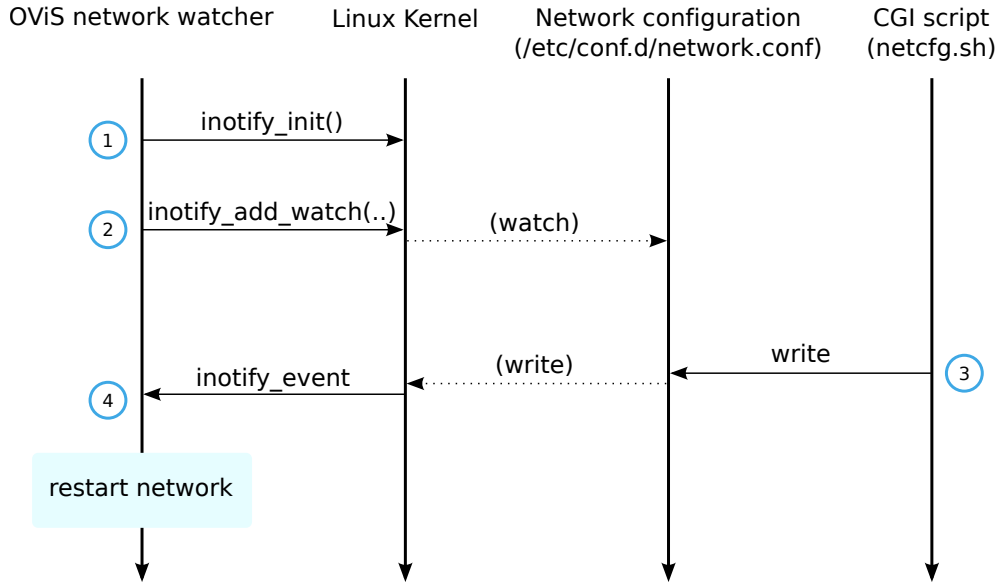


Figure 4.5: OViS remote network configuration.

onw is implemented in the C programming language as it needs to communicate directly with the Linux kernel. Restarting the network is done by calling the corresponding *init* script through C's `system` command.

We follow a strictly modular approach. The system-specific commands used by *onw* are cleanly separated from the OViS-specific network configuration mechanism. If at any point *inotify* becomes obsolete or OViS is ported to a platform that does not support it, only *onw* needs to be changed; the CGI script that parses and applies the configuration commands can remain unchanged. On the other hand, this makes *onw* re-usable for other projects based on ADAM that might require a mechanism to automatically restart the network when its configuration changes.

OViS UDP Pinger

The OViS UDP pinger (henceforth called *pinger*) is a small program that runs on the OViS mesh nodes in order to announce the node's presence and state to the hand-held device. During the deployment process described in Section 3.4.2, the OViS client application listens for those

Name	Value	Meaning
IF_PHY_UP	1	Physical link detected (cable plugged-in)
IF_IP_UP	2	IPv4 address configured

Table 4.3: Node status flags in OViS presence announcements.

announcements to detect new nodes. The pinger is started as a background daemon during the node's bootup process and is never stopped.

The message payload is a simple string containing the host name of the node and some flags that indicate its network connectivity, separated by a colon. Table 4.3 shows the valid flags and their meaning. The flags are combined by a logical OR operation. A typical message might look like `meshnode19:3` where `meshnode19` is the host name and `3` is the combination of the `IF_PHY_UP` and `IF_IP_UP` flags. Figure 4.6 depicts such a packet as captured by the *Wireshark* network protocol analyser [50].

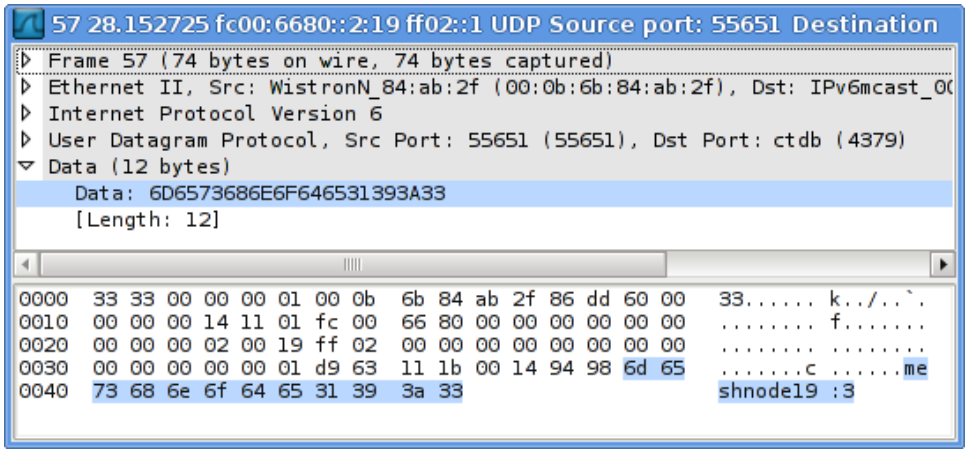


Figure 4.6: OViS node presence announcement.

OViS presence announcements are addressed to `ff02::1`, the IPv6 link-local multicast address, and sent to UDP port 4379.

It is worth mentioning that for each packet sent, the pinger refreshes its knowledge about the network connectivity and redetermines the best source address to use for the packet. In situations where the user mistakenly turns on the gateway node without its network cable plugged in and only remembers to connect the cable after bootup, constantly re-checking the connectivity allows the pinger to gracefully detect the newly established connection and immediately announce it to the network. The client application on the hand-held device can then react to this change by promoting the node to a gateway (see Figure 5.1).

During the course of our implementation, we realized that, if not specified otherwise, the packets sent by the pinger would have their source address set to the node's link-local IPv6 address. In order to communicate with such a link-local address, the sender needs to specify the network interface to be used since there is no way for the network stack to know who the address belongs to, i.e., on which interface the address can be reached. However, the OViS

client application uses the source address of the presence announcements to communicate with the node. As this would require the application to be aware of the hand-held device's various network interfaces, the pinger was modified to *bind* the socket over which the packet is sent to the network interface's non-link-local (i.e., *site-local* or *global*) IPv6 address [51]. Since this operation is done independently for each packet, the pinger will react immediately to any changes done to the node's IPv6 configuration.

While the pinger application described above was written specifically for OViS, its simplicity and the fact that it does not actually do anything specific to OViS might make it useful for other similar projects.

4.3.2 Third-party Software

The ADAM embedded Linux distribution received several enhancements during our work. Most of them were updates for components that had already been part of ADAM, such as the user space tools to manage wireless network interfaces and the Linux kernel, others were added specifically for OViS.

For the change from the old `madwifi` driver to its new replacement `ath5k` that was already mentioned in Section 4.1, one program in particular had to be added: CRDA, a piece of software that manages the country-specific regulations regarding the wireless spectrum. This is explained in more detail in Section 2.3.3.

Along with CRDA, several libraries and the *hotplug2* package were added to ADAM. Hotplug2 (see Section 2.3.4) is a light-weight program that handles and processes events from the Linux kernel – in our case, the events that occur as a result of setting the regulatory domain for the network interfaces to comply with Swiss regulations.

Since our network cards came without any regulatory settings predefined, the kernel would fall back to the default (most restrictive) settings and refuse to communicate on many of the frequencies that are permitted in Switzerland. We therefore patched the kernel to completely ignore the regulatory settings stored in the card's EEPROM [52] and instead accept the list of allowed frequencies provided by CRDA and chosen during the node's network configuration.

During the development of OViS, we frequently encountered situations where the node image would fail to compile on systems where it had not been compiled before. This would typically be caused by errors during the initialization of the build environment and would, therefore, not occur on machines where the system had been compiled before. Thus, an auto-build system was created that would compile the OViS branch of ADAM once every night. This allows us to be informed whenever a modification causes build-time errors. It also allows other projects that use OViS to pick precompiled system images directly from the server instead of installing the build environment on their machine and compiling everything from source.

Most of the modifications that were done to the ADAM distribution during the development of OViS have since been merged into the ADAM source code repository and are thus available to all other projects that are based on ADAM.

Chapter 5

OViS Client Application

The OViS client application is the user-facing part of OViS. The client comprises a "wizard" to guide the user through the deployment process (see Section 3.4) and a second part that establishes a video conference with some other person.

The deployment process was designed to hide the technical details from the user since the idea was to make OViS operable by people without any specific technical background or training. In this chapter, we present the various client applications that were implemented and talk about the differences between those applications, specifically regarding the platforms that they run on and the different communication technologies used.

5.1 Software Architecture

One of the challenges was to keep the application's architecture from becoming exceedingly complex despite the many different states that a node might be in and the various operations that can or cannot be applied to a node in any given state. In order to achieve this, we relied on the *state pattern* as described in Section 2.2.1.

Figure 5.1 shows a graphical representation of the states and their transformations found in the OViS client. Note that in the interest of clarity, only those events that modify the state are shown in the figure. Any other transitions are considered illegal and are routed back to the originating state, i.e., no state change occurs.

The first time a discovery packet (see Section 4.3.1) from a new node is detected, the node enters the system in the *online node* state. Afterwards, the following transitions can happen, provided that the node is in an appropriate state (e.g., an *expired* node cannot be *deployed*):

- Whenever an additional discovery packet is received, the node gets *poked*.
- If no discovery packets are received from a particular node for a predefined interval (e.g., 3 seconds), the node *expires*.
- As soon as the node gets picked by the application as the next node to be deployed, it is *connected*.

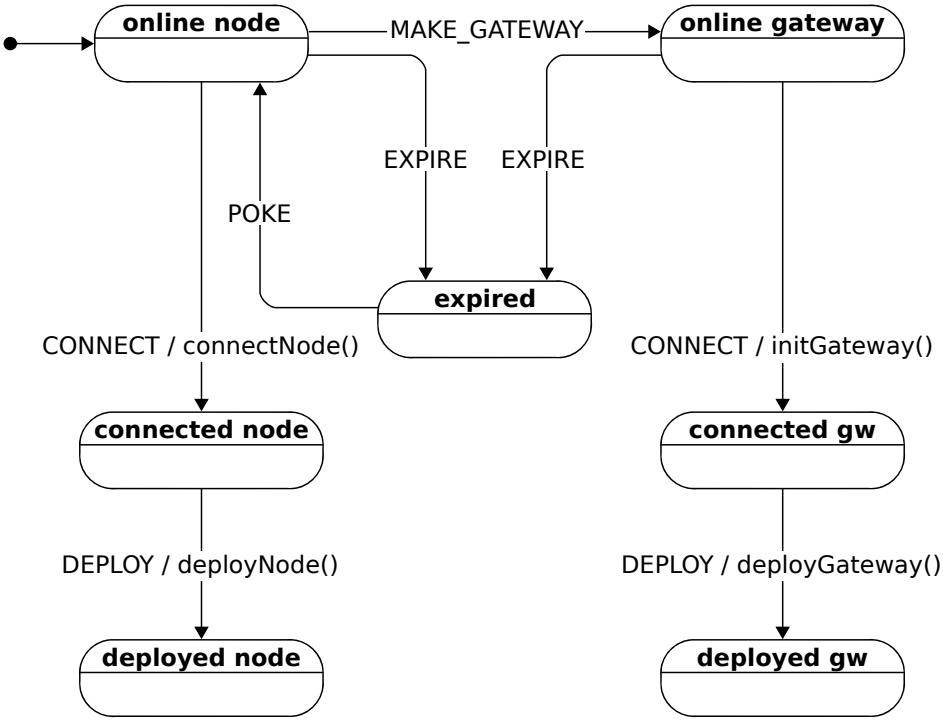


Figure 5.1: Node states as modelled in the OViS application.

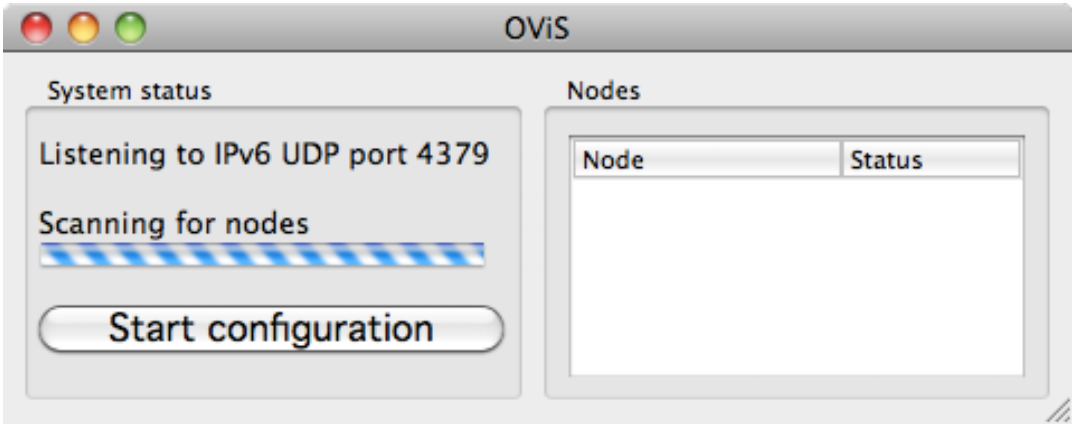
- If a discovery packet indicates that the originating node is a gateway (i.e., connected to a wired network), *make_gateway* gets called.

- Finally, when the user chooses to do so in the GUI, the node is *deploy*ed.

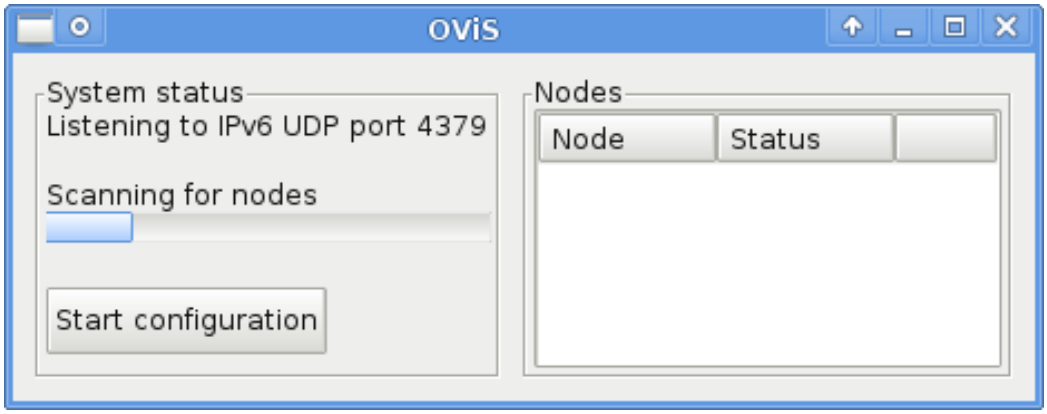
5.2 Desktop Client

The PC client is the main client application for OViS. The goal was for it to be platform independent, i.e., to run at least on the Linux, Mac and Windows operating systems. The client is meant to be used by an end-user without any special training and without prior knowledge about mesh networks or networking in general.

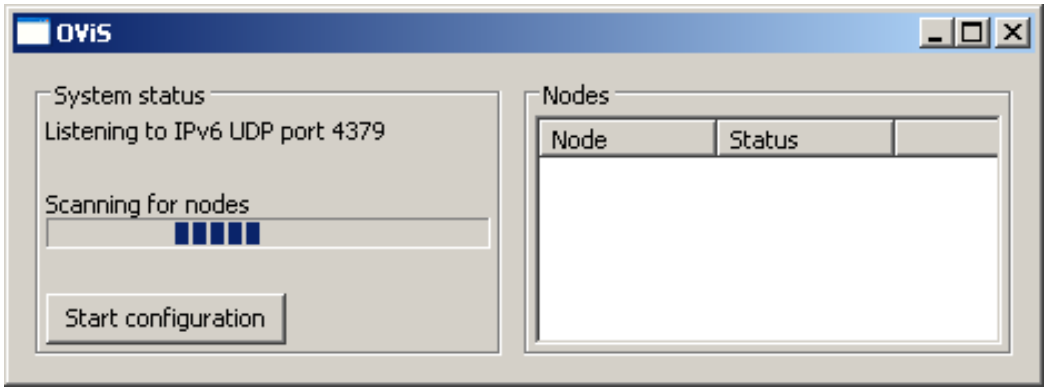
It is worth mentioning that the client application is one of the very few elements of OViS that do not serve one and only one very specific task: the application is responsible to guide the user through the node deployment process as well as to initiate the conversation with the off-site expert. This combination of two basically unrelated tasks was chosen in order to make the system easier to use by non-technical people. In our opinion, the benefits of having an integrated application that comprises those activities outweigh the negative impact on maintainability that is caused by arbitrarily combining seemingly unrelated tasks.



(a) Mac OS X.



(b) Linux / GTK+.



(c) Windows.

Figure 5.2: The wxPython OViS client on different operating systems.

5.2.1 Prototype in wxPython

A first version of the PC client was implemented in wxPython (described in Section 2.3.5). Figure 5.2 shows what the wxPython OViS client looks like in different operating system environments.

Preliminary tests with the wxPython-based client implementation did, however, reveal serious usability issues. Namely, the standard widgets were too small to hit reliably on our UMPC's touch screen (see Section 5.2.4). Significant modifications to the layout would have been necessary to work around this issue, thereby destroying the operating system's native appearance and with it wxPython's major advantage. The client was hence scrapped and future development went into a completely different direction.

5.2.2 Python CLI Client

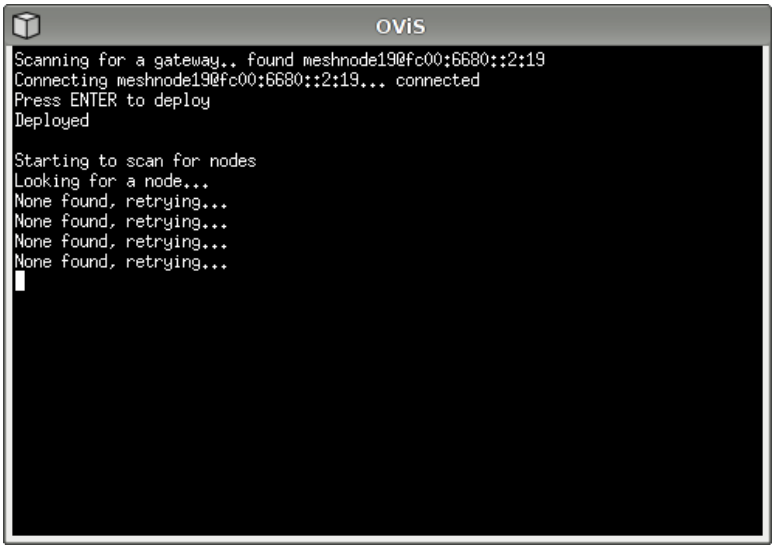


Figure 5.3: OViS command-line client.

At this point, some fine-tuning of the deployment process described in 3.4 was required. A *Command-Line Interface (CLI)* was considered more useful for that job, since repeatedly clicking the same GUI elements is much more tiresome than repeatedly typing the same keys. Thus, the OViS CLI client shown in Figure 5.3 came to life. While this client itself lacks the video-conferencing part, it is still worth mentioning as it shares a great deal of code with the final GUI client.

Figure 5.4 shows the class hierarchy of the common model. The top-left box represents the class of which the UI needs to instantiate an object to get a reference to the model, the *DiscoveryDaemon*. As the name implies, this is a daemon that works in the background. It listens for UDP datagrams on port 4379 (IPv6) as sent by the OViS pinger (see Section 4.3.1). Whenever a new node is found via this mechanism, it is added to the daemon's list of known

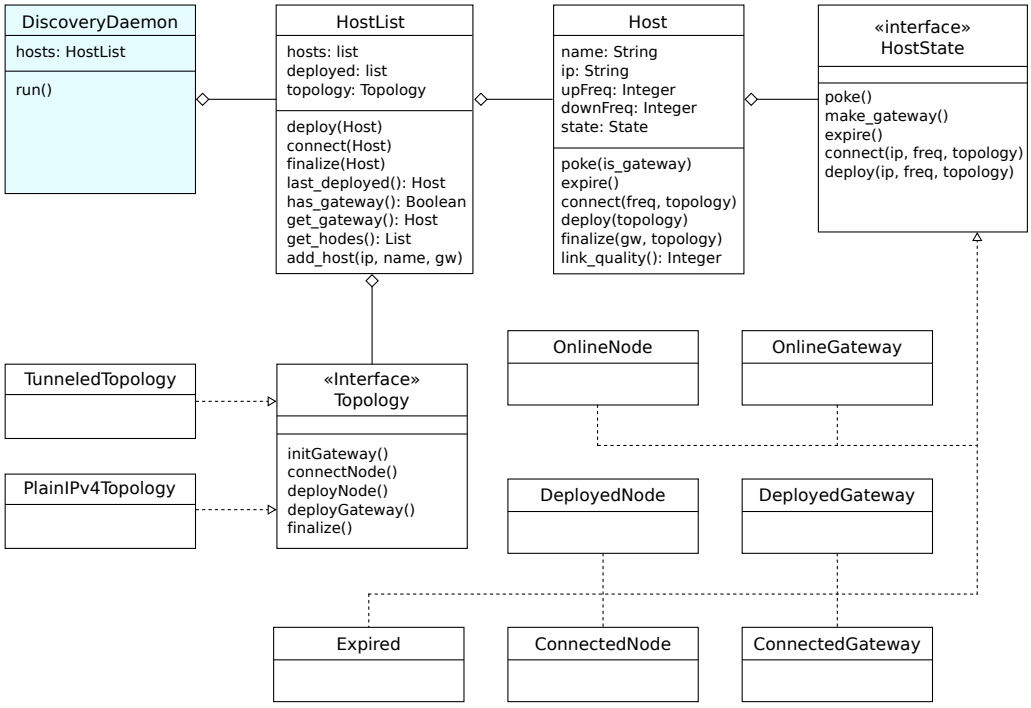


Figure 5.4: OViS CLI / GUI model.

hosts. The application can then perform operations on those nodes through the *HostList* object, which is in turn available as an attribute of the discovery daemon.

Operations to the nodes such as `connect` and `deploy` are performed as a function of the node's `state` (see Section 5.1) and the `Topology` object in use. The state objects encapsulate the higher-level logic ("what to do") while the topology objects contain the lower-level instructions ("how to do it"). This allows the coexistence of multiple different network topology implementations (see Section 3.3) without significant modifications to the higher-level logic.

As an example, the `initGateway` method gets called if a node in the `OnlineGateway` state is told to `connect` to the network. This is considered a high-level operation that is independent from the actual implementation, i.e., it is safe to assume that in any network topology a gateway node might have to be initialized in a specific way – even if that specific way is to do nothing at all. In the case of a `PlainIPv4Topology`, the detailed instructions are indeed rather simple: enable NAT on the node. In a `TunneledTopology`, this step also needs to configure the IPv4-in-IPv6 tunnel (see 3.3). The same mechanisms are also used to determine what steps to take for any of the other operations required for the node deployment.

This level of abstraction offers a mechanism to test different network topologies during the development process and, by encapsulating most of the behaviour in topology and node-state objects, helps to keep our code clean and reusable.

5.2.3 The TkInter Client

As a result of our attempts at using the operating systems' native widgets and the ultimate dismissal of that idea, we decided to write the whole GUI ourselves as a full-screen *kiosk* application, i.e., an application that does not look and feel like a "normal" PC program but more like an integrated appliance. This would also allow us to design the widgets in a way that they are useful regardless of our hardware limitations.

We decided in favour of TkInter, a cross-platform Python toolkit described in Section 2.3.6. By implementing the GUI in the Python programming language, we can reuse the back-end code presented in Section 5.2.2 for a platform independent graphical application. The GUI client's model is therefore identical to the one described above and shown in Figure 5.4.

The Application

The application was designed for two basic purposes: guide the user through the process of deploying the ad-hoc network and initiate a video call afterwards. Figure 5.5 outlines the workflow from the moment the application is started until the video call to the qualified off-site engineer is initiated.

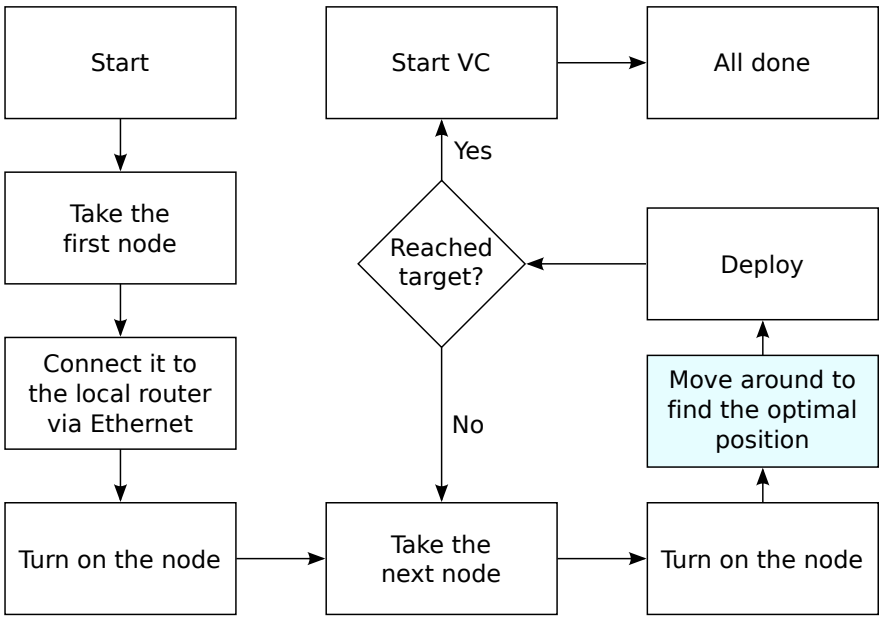
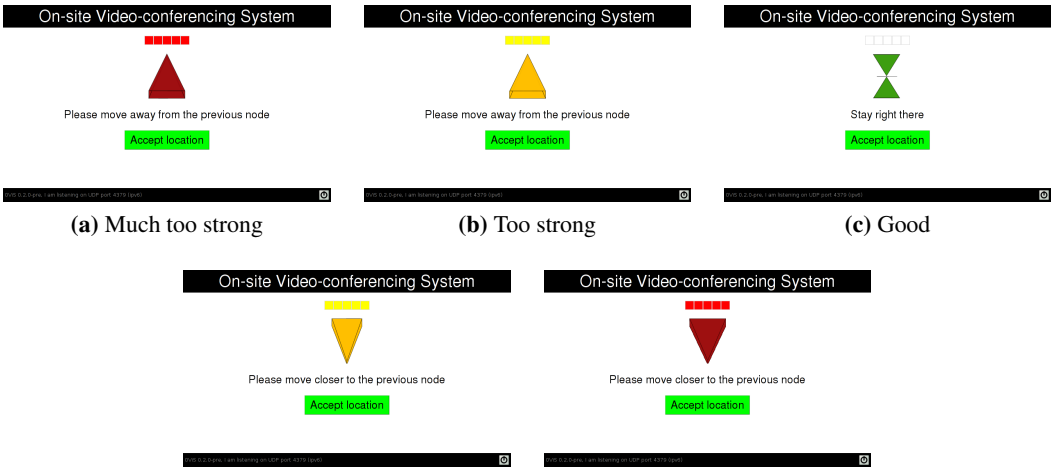


Figure 5.5: Application workflow.

First, the user is instructed to take a node, connect it to the local Internet router and turn it on. This node will be used as the ad-hoc network's gateway node. Afterwards, an iterative process starts where the user takes the next node, turns it on and walks towards the destination. The application now guides the user to find the right spot for the node through visual and acoustic feedback (see below for details). When the user decides to deploy a node, the application asks



(a) Much too strong **(b)** Too strong **(c)** Good

(d) Too weak **(e)** Much too weak

Figure 5.6: Instructions to the user as a function of signal strength.

whether the destination has been reached and, if not, repeats the last steps in order to deploy an additional node. Once the destination has been reached, the video call is initiated.

Finding the Right Location

Finding a way to instruct the user to deploy the node at the right position was one of the trickier tasks. The following aspects had to be considered:

- If the distance between the nodes is too short, the number of nodes required to cover the distance from the router to the destination might exceed the number of available nodes. Also, the deployment process would be exhausting since each node takes around 30 seconds to boot.
- If the distance between the nodes is too long, the connectivity might suffer to an extent where the bandwidth is reduced below the minimum required for a decent video call.
- Obstacles and safety concerns on the construction site might prevent the user from placing the node at the exact position indicated by the application.

As a consequence of these considerations, the application was designed to classify the node's position into five categories as shown by Figure 5.6: much too strong, too strong, good, too weak and much too weak.

- If the received signal power from the previous node is at around -60dBm ($\pm$10dBm), two green arrows are presented to the user as an indication to place the node at (or near) the current location (see Figure 5.6c). These values were chosen as a result of bandwidth tests with several different signal qualities (see Section 6.1). The additional margin grants some flexibility regarding the node placement.

- If the signal strength is outside those boundaries but within ±20dBm of the ideal value, yellow arrows as shown on Figure 5.6b (too strong) and Figure 5.6d (too weak), accompanied by slow audible beeps instruct the user to move further away from / closer to the previous node.
- In situations where the signal strength deviates further from the ideal value, the red arrows seen in Figure 5.6a (much too strong) and Figure 5.6e (much too weak), combined with fast beeps encourage the user to move towards the right spot.
- The audio feedback's pitch depends on whether the user is too close (low pitch) or too far away (high pitch) from the previous node.

By adding acoustic feedback to the visual indications, the application allows the user to pay attention to his surroundings instead of focusing on the computer screen – when the beeping stops, the destination has been reached.

Skype Integration

The video communication part of OViS has been implemented using Skype (see Section 2.3.2). Skype was selected for several reasons:

- Thanks to the large number of platforms supported by Skype, the OViS client can communicate with the vast majority of systems that the qualified off-site engineer might be using.
- As explained in Section 2.3.2, Skype can work around most firewalls to allow bidirectional video (i.e., UDP) streams. Since we cannot modify the on-site router's configuration and we have to be prepared for highly restrictive firewall settings, this feature will allow OViS to work in situations where other technologies might fail to establish a connection.
- The free competitors we looked at (such as *Pidgin* [53] and *Empathy* [54]) were still in heavy development and lacked the necessary reliability.
- Implementing a custom video conferencing application would have gone beyond the scope of this thesis.

Thanks to the "Skype API", OViS can use the Skype client without forcing the user to interact with its tiny control buttons on our hardware's mediocrely-precise touch-screen, thereby creating the illusion of an integrated video-call feature in our client. Observing users will still notice that the *look&feel* of the Skype client's user interface differs from the OViS client, though.

5.2.4 UMPC

The OViS client application was developed on an Asus R2H *Ultra-Mobile Personal Computer (UMPC)*, a mobile computer featuring a 900 MHz Intel Celeron CPU, 1.2 GB RAM, a wireless network interface and a touch screen. The screenshots shown in Section 5.2.3 were taken on

that device. While the UMPC might be considered to be a special, limited hand-held device by the end user, software-wise it is a fully-fledged computer, running a "normal" desktop OS (see below). Hence, it runs the same client application that could also be used on a more traditional laptop computer.

Yet, the UMPC imposed some hardware-specific requirements on the client application and on the network topology:

- The device's touch screen is not very precise. Therefore, the client application needs to have big control elements that can be reliably activated even on a moderately-precise touch screen.
- The UMPC's screen operates at a native resolution of 800x480. The application had to be designed to fit all the required information on that limited amount of space.
- The built-in *ZyDAS ZD1211* wireless controller does not support communications on the 5 GHz band and is limited to channels 1-11 in the 2.4 GHz range. Since this is a fairly common limitation of consumer-grade hardware, the link between the last node and the hand-held device uses channel 11.

The hand-held device runs Debian GNU/Linux 6.0 [55] with the Xfce Desktop Environment. Special *themes* (customized UI presets) were made for the *grub* boot loader, the *GNOME Display Manager* (gdm) and the *GIMP Toolkit* (GTK+). The idea was to create a common *look&feel* for everything the user sees from the moment the boot loader starts until the video conference is over. In addition, the system was configured to automatically login a specific user after booting and to immediately start the OViS application afterwards. The user is not supposed to think of OViS as a complex piece of software running on a "normal" PC but instead see the system as a highly integrated appliance.

Thanks to the open nature of the underlying Debian operating system, the user interface could be customized in a way to make such an impression. While modifications to the source code of the client's operating system would also have been possible, no such modifications were necessary.

5.3 OViS Android Client

Android is a Linux-based open-source operating system for hand-held devices and mobile phones. Section 2.3.7 describes the Android platform in more detail. In the light of the shortcomings of our hand-held device described in Section 5.2.4, the OViS client was ported to Android. Unfortunately, the current version of Android has some limitations that needlessly increased the complexity of our application. We used Android 2.2 for our project, the problems are still present in the current version (3.1) though. The next section explains these limitations in detail.

5.3.1 Challenges

As mentioned above, Android is based on the Linux kernel. While this should make Android an ideal platform to use for networking-related projects like OViS, we were faced with several

challenges that required special attention and caused significant differences between the client described in Section 5.2 and the Android client application:

- The first – and probably most challenging – issue was Android's lack of support for ad-hoc networking. An Android device can neither connect to an ad-hoc network [56] nor create one. It was therefore required to manually take control of the underlying Linux system in order to circumvent that restriction as described in Section 5.3.2.
- As a consequence of the missing support for ad-hoc networks, the wireless interface cannot be configured using the mechanisms provided by the Android platform. The need to change network settings came up, because a DNS request is made for every HTTPS connection to a mesh-node. Since the DNS servers cannot be reached during the deployment process, the OViS application would become unresponsive while waiting for those requests to time out. To fix this, the application has to disable all DNS lookups at startup and re-enable them once the network has been deployed, which can only be done directly on the system level. We therefore need superuser privileges. Section 2.3.7 explains how this is done.
- The HTC *desire* smartphone that was used does not have a user-facing camera. Videoconferencing as used in the OViS PC client could be cumbersome.

While these limitations proved to be rather tedious, a solution was found for all of them. In the case of the communication method it even has some advantages over the approach chosen for the PC client.

5.3.2 Ad-hoc Networking Support for Android: The *desire-adhoc* Application

The toughest problem was the aforementioned lack of support for ad-hoc networking (see Section 5.3.1). Thanks to Android's Linux foundations, it was possible to overcome this limitation by manually configuring the network using Linux tools. The *iwconfig* program from the *Wireless Tools for Linux* package [57] was therefore added to the phone. Afterwards, the interface can be put into ad-hoc mode by reloading the wireless device driver and then configuring the ad-hoc mode using the appropriate sequences of *iwconfig* commands. Finally, the interface can be connected to the OViS network through the use of standard Linux IP configuration commands. This process has been described in detail and published on the web [58], since the great number of responses in the bug report about Android's missing support for ad-hoc networking indicates a public interest in the topic.

The steps described above cannot normally be done on an Android device since the user does not have the permissions required to reload kernel modules or run low-level system commands. Section 2.3.7 explains the process used to acquire the needed privileges and the required modifications to our Android device. In particular, a modified version of the Android operating system was installed [59] since the default firmware does not allow superuser access.

In the interest of being able to test our Android client in the field, where manually entering the commands required to configure ad-hoc networking would be tedious, those commands were wrapped into an additional program called *desire-adhoc* [60]. Figure 5.7 shows a screenshot of

this application, which has since been released to the general public under the terms of the *GNU General Public License (GPL)* [61].

Figure 5.7: The *desire-adhoc* application.

The workaround described above and the resulting application allows us to reliably connect the smartphone to the OViS network and illustrate the advantages of using a fully-fledged, free operating system such as Linux on mobile phones – working around similar restrictions would have been significantly harder (or impossible) on a proprietary operating system.

Unfortunately, the modifications required to the phone's operating system make OViS unsuitable for general purpose use at the moment since this process will void the phone's warranty. Once Android starts natively supporting ad-hoc networking, this will no longer be an issue.

5.3.3 OViS Application for Android

The smartphone used during the development of OViS is significantly different from the handheld device described in Section 5.2.4. Particularly, it lacks a user-facing camera; the built-in lens is located on the phone's backside. Thus, we decided to try a different approach at communication for the Android OViS client while copying the deployment process from the PC client.

Figure 5.8 shows the node placement process as presented by the Android application: first, the user is told to grab a node and turn it on. The client then waits for the node to announce its presence. Once the node is found, the client configures the first interface. Afterwards, the user is guided towards the right position for the node, supported by a coloured indicator (and acoustic feedback), similar to what the PC client does (see Section 5.2.3). The user can then decide to accept the current location, at which point the node's second interface will be configured.

The communication method is a combination of voice-chat and photographs instead of a video link. The on-site user can talk to the qualified engineer in a fashion similar to a "normal" phone call and, at the same time, take photographs of the local installation and send them to the

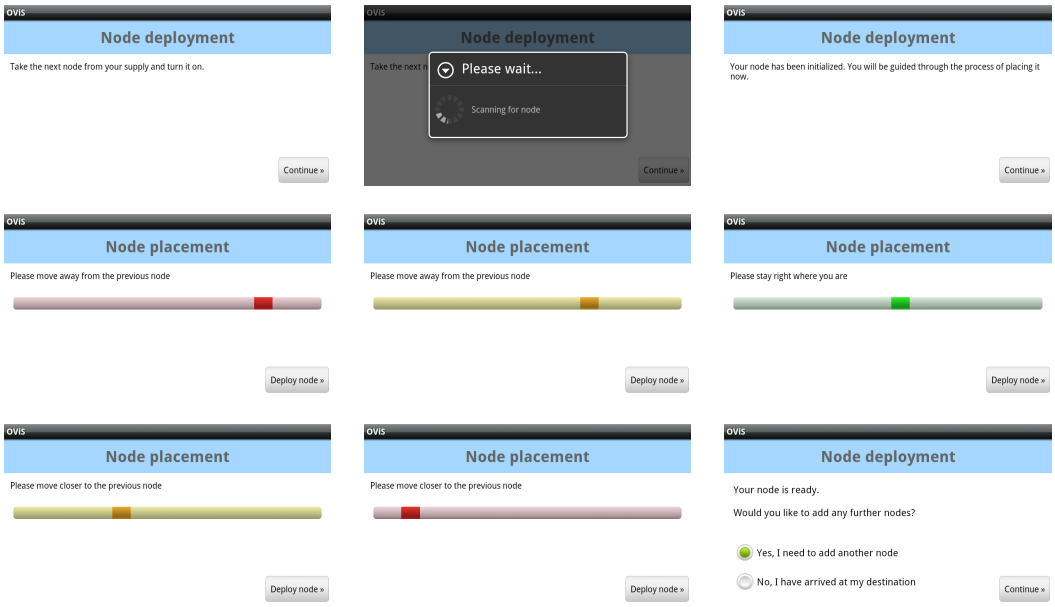


Figure 5.8: Node deployment with the OViS Android application.

remote expert to further illustrate the issue at hand.

The voice communication is done using voice-over-IP (VoIP) through the *Session Initiation Protocol (SIP)* [62]. During the development of OViS, version 2.3 of the Android operating system was released that includes native SIP support [63]. An updated version of the operating system is, however, not currently available for the HTC desire; therefore, we use *sipdroid* [64], an open-source third-party SIP client.

Figure 5.9 shows an overview of the communication process. After deploying the network, the on-site user and the off-site expert communicate via SIP (1). The on-site user can then take a picture and send it via HTTP POST (2) to a custom written application based on the *django* web framework [65] that runs on a remote server (3). Figure 5.10 shows a screenshot of the upload process. When the upload is completed, the server sends a message to the off-site expert that contains a hyperlink to the uploaded picture (4). This message is sent using the *Extensible Messaging and Presence Protocol (XMPP)* [66], an instant messaging protocol formerly known as “Jabber”. Finally, the off-site expert can open that link and fetch the uploaded image via HTTP GET (e.g., with a web-browser).

The Android application shows an important aspect of OViS: thanks to the clean separation of the network deployment process and the actual communication mechanism it is relatively easy to completely replace one of them – in this case the video chat – with an entirely different system. This modularity should also make a system like OViS independent enough from implementation details to be cleanly portable to entirely new software ecosystems, whether it is a different network topology, a new operating system such as Apple’s iOS or a new approach at communication.

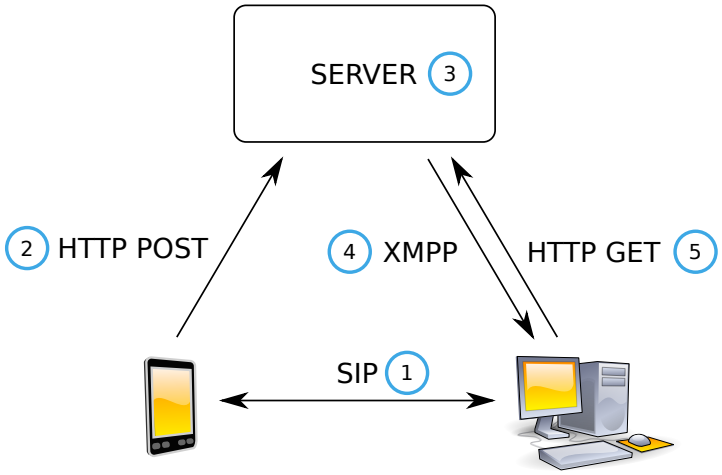


Figure 5.9: Communications using the OViS Android application.

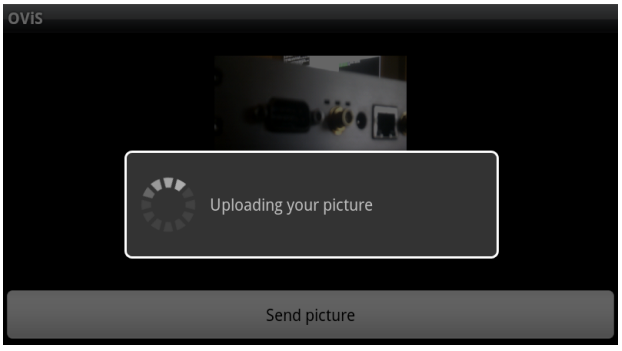


Figure 5.10: Uploading a picture from the Android client.

Chapter 6

Evaluation

OViS was evaluated in various ways: first, a qualitative evaluation was done to find the required signal quality for a link between neighbouring nodes at which video communication is still possible without a disturbing degradation of the audio / video signal. The results from these measurements were then used to calibrate the desired signal strength in the OViS client application. The determined signal quality thresholds are used to generate the instructions for node placement (see Section 6.1).

Afterwards, a quantitative evaluation compares the throughput of non-guided deployments, an OViS network and a manually optimized topology as it could be deployed by an experienced user. These results are described in Section 6.2 along with details about the testbed setup and the software that was used. Additional measurements to test various aspects of the network such as the benefits from using a multi-channel setup and the effect of using automatic rate control vs. a predefined bitrate are presented in the subsequent sections.

Finally, we revisit the requirements from Section 3.1 that were asked from the system to determine which ones have been fulfilled by our implementation and where additional work might be required.

6.1 Signal Quality

The targeted signal quality for a link between two mesh nodes is a trade-off between maximizing the covered distance (or minimizing the number of required mesh nodes) and maintaining enough bandwidth for the network. Tests were made with a simple network consisting of three mesh nodes to see how low the link quality can go before the Skype application loses connectivity or the video signal gets degraded to an extent that makes it unsuitable for practical use. The tests showed that the signal becomes severely distorted when the link quality between two nodes falls below -75 dBm and the connection is terminated at around -80 dBm.

Figure 6.1 shows how the bandwidth over a single hop is affected by the signal strength. This was evaluated using the *Network Protocol Independent Performance Evaluator* (NetPIPE [67]) with a packet size of 512 kB (TCP). As the signal decays beyond approximately -60 dBm, the available bandwidth starts decreasing. Once the -80 dBm threshold is crossed, the connection is at risk of being terminated. The lowest signal strength at which we were still somewhat able to

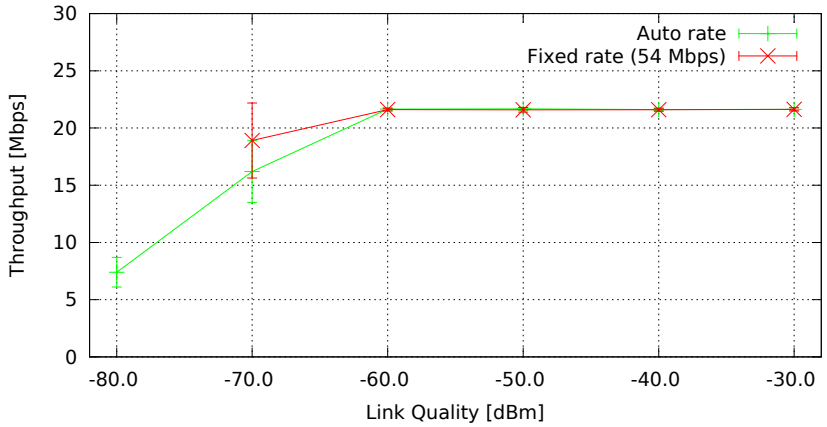


Figure 6.1: Network throughput against link quality.

communicate over the network was around -85 dBm – the connection would be highly unstable at that point, though.

The same tests were then repeated with the network interfaces configured to operate at a fixed bitrate of 54 Mbps instead of using automatic rate control (see Section 2.1.3). Figure 6.1 also contains the results from these measurements. The outcome was very similar for good links, i.e., the bandwidth was virtually identical for links with -60 dBm or better. At -70 dBm, the fixed bitrate shows better throughput than what was achieved with the automatic bitrate settings. However, once the signal becomes weaker than -70 dBm, the connection becomes too unstable and is terminated. This is consistent with the expected behaviour in an IEEE 802.11 wireless network where the bitrate will be reduced as the signal strength becomes worse. By using a fixed bitrate, this mechanism can no longer take effect and the connection is simply terminated instead.

As a result of these measurements, we chose a target link quality of -60 dBm for our network with a tolerance range of $\pm$10 dBm in order to have enough bandwidth to sustain video communication with decent image and audio quality while also covering a relatively large distance with few nodes. For the client application, this meant that the position indicator would be green for signal qualities of -60$\pm$10 dBm, yellow at -60$\pm$20 dBm and red otherwise. By not exhausting the measured range for decent link quality we allow for some temporary fluctuations and some flexibility for the user to choose a more practical location for the node than what the system indicates. Choosing a lower target quality would increase the risk of losing the network link due to short-term interferences. It was also decided to use automatic rate control instead of a fixed bitrate since the benefit of having some additional bandwidth for links between -60 dBm and -70 dBm is outweighed by maintaining a stable network even with poor signal strength.

6.2 Network Bandwidth

To determine the quality of a network deployed through the OViS client, such a network was tested inside a building. We used NetPIPE to measure the bandwidth of a three-hop network consisting of four mesh nodes (as described in Section 4.1) and an IBM ThinkPad X41 Laptop with an Intel PRO/Wireless 2915ABG network controller.

NetPIPE measures the bandwidth by sending *Transmission Control Protocol (TCP)* traffic with incrementing message sizes to another host on the network. We therefore connected a second laptop, an Apple MacBook Air, to the same Ethernet switch that the gateway node was connected to and measured the bandwidth between the two laptops. The ThinkPad was using a wireless link on the 2.4 GHz band to the last mesh node. Figure 6.2 shows this setup in more detail.

The first node was placed in a room on the building's top floor along with the MacBook. The OViS client was then used on the ThinkPad to deploy the additional nodes throughout the building. The target link quality for the deployed network was -60 dBm ($\pm$10 dBm) as described in Section 6.1. The second node was placed on the same floor near the stairway entrance, and the next two nodes covered the stairway and parts of the lowest floor. The other laptop was then placed approximately three meters from the last node where it stayed during the performance evaluation.

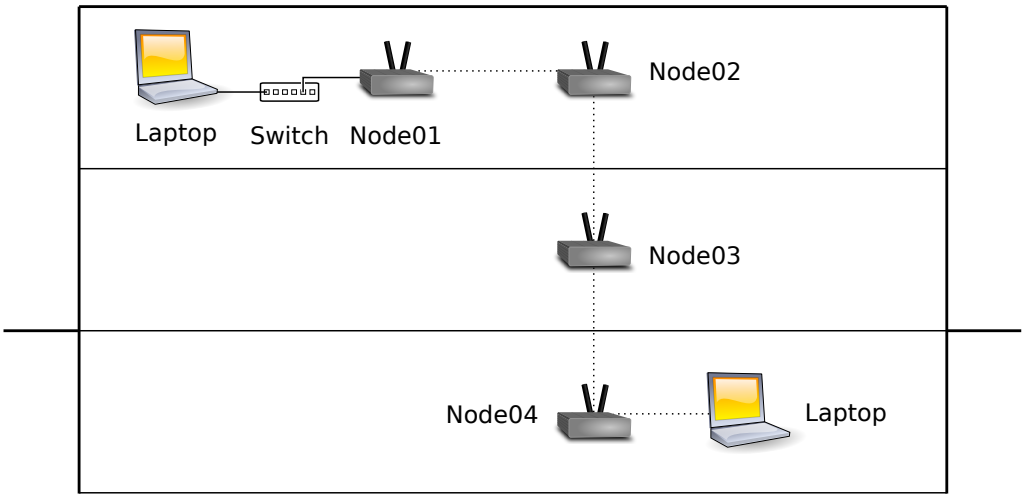


Figure 6.2: OViS testbed setup.

Both laptops were running Debian GNU/Linux 6.0 with the shipped version *2.6.32* of the Linux kernel. The operating system was installed natively on the ThinkPad and in a VirtualBox environment on the MacBook. We used NetPIPE version *3.7.1*.

Figure 6.3 compares the bandwidth measured on two networks (A, B) that were deployed according to the instructions from the OViS client with other deployment methods.

To evaluate the results, the network links from the first OViS deployment (A) were then optimized manually: by properly aligning the antennas and moving the nodes a few centimetres, we tried to improve the signal strength of the individual links while covering the same distance

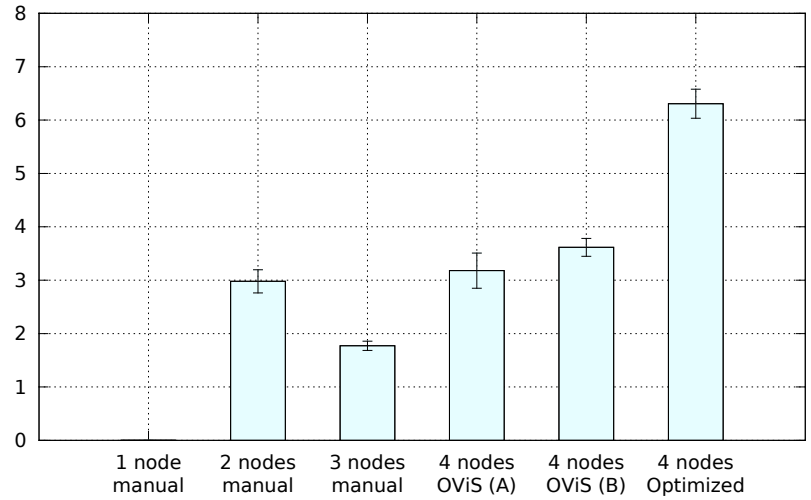


Figure 6.3: OViS network bandwidth.

between the first and the last node. Figure 6.4 shows the result of these manual improvements. The signal was measured on the receiving node of each link (e.g., "Node02" for the first link) through the `iw` command-line tool. Each value represents the average and standard deviation for ten measurements done at 1 s intervals.

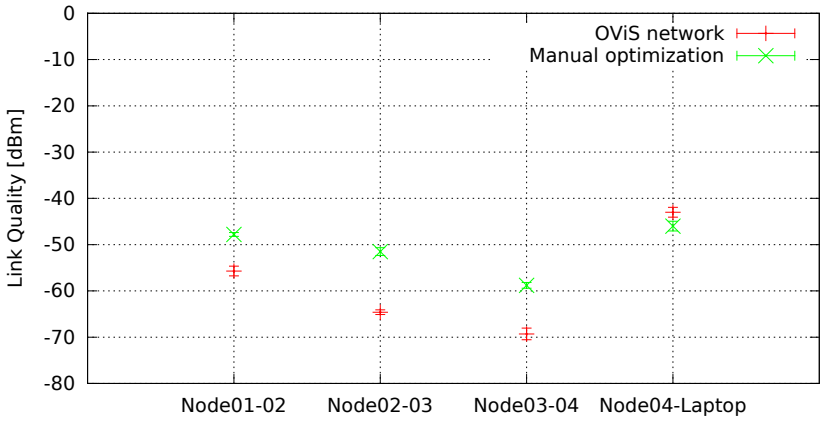


Figure 6.4: Manually improved link quality on OViS network (A).

The manual adjustments resulted in a difference of roughly 10 dBm on each link. As can be seen, this increase is larger on the second and third links – the links where vertical distance was covered. We attribute this particular improvement to the antenna adjustments. On the first link, the improvement was mostly achieved by avoiding some of the larger objects between the nodes.

It is worth mentioning that the link between the last node and the laptop became slightly weaker because the node was moved a couple of centimetres away from the computer to improve the signal strength to its other neighbour and ended up on the other side of a metal door frame.

After these improvements, the same bandwidth tests were run again. The results of these tests are shown in Figure 6.3 as the "Optimized" values. Compared to the guided OViS deployment, the optimized network offers roughly twice the throughput.

A third series of measurements was then made to compare a manually deployed network to the topology that is created by the OViS deployment process. To evaluate this, an attempt was made to cover the same distance spanned by the previous tests with one, two and three mesh nodes. The nodes were naïvely placed at convenient locations between the gateway node and the ThinkPad, spread at even distances. The resulting network throughput can also be seen in Figure 6.3.

With one node, no communication was possible, OViS was right to insist on additional nodes. Interestingly, with only two nodes, the network throughput was already comparable to the result measured over the OViS network, while adding an additional node turned out to have a highly negative impact on the overall performance. These phenomena are, however, easily explained by having another look at Figure 6.2: with only two nodes, the second node was placed near the location of "Node03" in the Figure, a position with both, some horizontal and a slight vertical offset. The node was therefore capable of reaching its two neighbours over a reasonably good link. With three nodes, the situation looks different: the intermediate nodes were placed near the original locations of "Node02" and "Node04", thereby creating a large vertical gap between the nodes. Since the type of antenna we used has a high horizontal gain but is quite inefficient vertically, this caused a very weak link between these two nodes.

The results show that while OViS might not be able to create an ideal network, it is fully capable of guiding an inexperienced user through the deployment of a mesh network that performs reasonably well. By measuring the link quality between mesh nodes during the deployment process, OViS can implicitly compensate for misconceptions on the user's part, such as obstacles or (in our case) poor connectivity over vertical distances.

Specific knowledge about how wireless networks operate, which kinds of objects to avoid and how the orientation of an antenna can affect the signal quality will of course result in a more efficient network topology. Yet the unoptimized network deployed according to the instructions from the OViS client is still good enough to provide reliable bandwidth for video conferencing and will cause additional nodes to be deployed where the inexperienced user would otherwise introduce poor links into the network. With OViS it is possible to deploy a stable, reliable network by blindly following the instructions from the client application and OViS should therefore be usable by anybody without any particular knowledge or special training.

6.3 Impact of Multihop Communication

The previously presented results do, however, require some clarification: Figure 6.1 shows a maximum throughput of around 22 Mbps over a single hop, yet even the optimized network shown in Figure 6.3 only achieves a maximum throughput of a little more than 6 Mbps. We therefore examined the influence of increased hop count on the network's bandwidth.

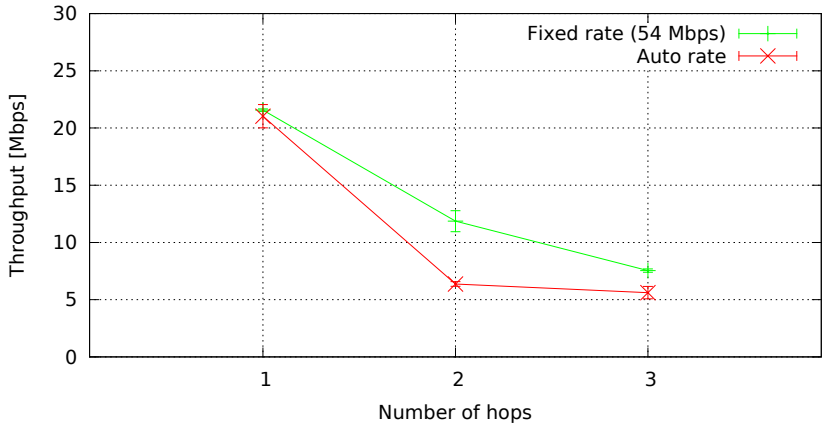


Figure 6.5: Bandwidth over multiple hops.

Figure 6.5 shows the results of those measurements. As before, we tested the network by using NetPIPE with 512 kB TCP packets. At the beginning, two nodes were each connected to a Laptop / PC via their Ethernet ports and to each other through a wireless link. Additional nodes were then added to this setup, connected only through their wireless interfaces. By choosing appropriate frequencies for the interfaces we made sure that the nodes would form a single line and the network traffic would flow through all the nodes. As the figure shows, a single link can carry over 22 Mbps. When an additional node is added to the network, the bandwidth will, however, decrease drastically to around 6 Mbps. Adding further nodes does not have a similar effect, the additional bandwidth reduction is comparatively small.

To investigate this further, the experiment was repeated using a fixed bitrate of 54 Mbps instead of the automatic rate control (see Section 2.1.3) that was used the first time. As can be seen in Figure 6.5, this leads to significantly better performance on a two-hop (three nodes) network, the effect does, however, diminish when a fourth node is added.

These measurements also explain why the final network's bandwidth as shown in Figure 6.3 is much lower than what was measured over a single hop and we conclude that, aside from the negative effects on bad links shown in Section 6.1, even a network with decent links would not benefit from using a fixed bitrate.

6.4 Impact of Multi-Channel Communications

Since considerable effort was put into designing the OViS deployment process to use multi-channel communications as explained in Section 3.4.1, the benefits of using multiple channels instead of a single-channel network were also evaluated. For this, a network over two hops with each wireless interface set to a fixed bitrate of 54 Mbps was created. The nodes were placed to have a signal quality of $>$-50 dBm on both links to make sure they were operating at ideal

bandwidth and that short-term fluctuations in signal strength would not affect the measurements. Both, the single-channel and the multi-channel topology used frequencies in the 5 GHz band. The bandwidth was again measured with the NetPIPE software as described in Section 6.2.

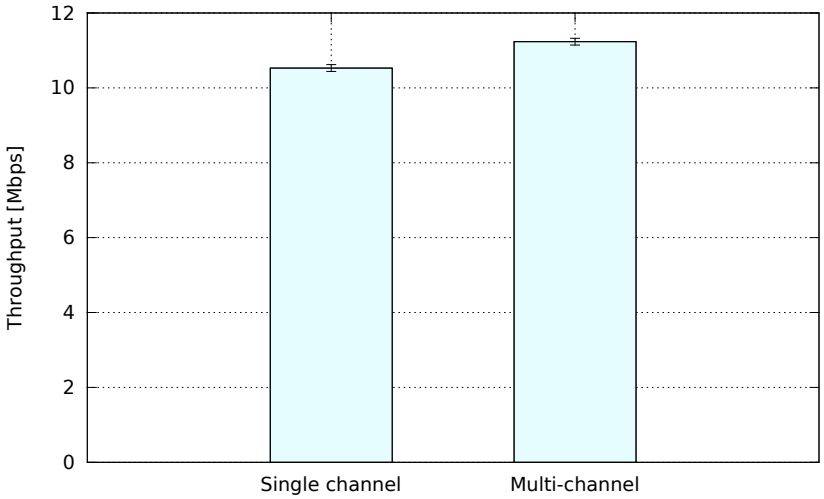


Figure 6.6: Multi-channel bandwidth.

Figure 6.6 shows the result. Our two-hop network gained an additional 1 Mbps of bandwidth when multiple frequencies were used. We attribute this improvement to the reduced interference between the two links. Since this effect will become even more pronounced as the number of nodes grows, we are confident that the decision to use multi-channel communications was beneficial for the overall performance of the OViS network and the time invested for designing and implementing an appropriate deployment process was well spent.

6.5 Requirements Evaluation

Finally, an evaluation of the system requirements as specified in Section 3.1 was done to determine whether the project's goals have been met. Our evaluation yielded the following results:

- The deployment process described in Section 3.4.2 does not require any prior knowledge about wireless networks or networking in general. An inexperienced user should be able to deploy our system by following the on-screen instructions. The resulting network cannot compete with a topology that has been optimized by a highly trained expert, but it offers stable connectivity with a reasonable number of mesh nodes and it is much less prone to user misconceptions than a manually deployed network. The result could be further enhanced by adding a second antenna for each wireless interface to better cover vertical distances.
- Chapter 5 shows that client applications for OViS are available for Linux, Mac OS and Windows as well as for the Android platform. OViS does not only provide a cross-

platform application but also an architecture that should allow for additional client software to be written for any operating system that supports wireless networking according to the IEEE 802.11b/g standard and basic IPv4 and IPv6 connectivity.

- By using a link on the 2.4 GHz band between the last node and the hand-held device and by not relying on positional data, we designed a system that is compatible with any end-user device that features wireless networking hardware compatible with the IEEE 802.11b/g standard. The combination of IPv4 for data payload and IPv6 during the deployment process requires a reasonably modern operating system on the client device.
- In Section 3.4.1 we explain how OViS uses multi-channel communication in the 5 GHz band between the mesh nodes while keeping the connectivity to the hand-held device in the widely-used 2.4 GHz band for improved compatibility with end-user devices. This approach combines the advantage of reduced interference with the required compatibility described above.

We are therefore confident to state that, while further improvements to OViS could be made, all the functional requirements concerning the system have been fulfilled. We have shown that OViS is a modular, highly flexible system that provides stable bandwidth to sustain audio and video conferencing at remote locations where wireless communication would be impossible otherwise.

Chapter 7

Conclusion and Future Work

After describing OViS and its various components in details and presenting an evaluation of the system, we would like to give our conclusions from the evaluation results as well as some possible directions for future work.

7.1 Conclusion

The OViS platform has been shown to be very flexible not only with regards to the client application but also concerning the communication method. This was achieved by cleanly separating the deployment logic from the network payload. The different client applications presented in Chapter 5 show that the system can be deployed by and used with any number of operating systems, including (but not limited to) Linux, Mac OS and Windows. Client applications have been written in the Python and Java programming languages, showing that the OViS API was designed well enough not to depend on any particular language. The Android application presented in Section 5.3.3 uses an entirely different communication mechanism than the standard OViS client and thereby shows that our network serves as a general-purpose network rather than being specialized and limited to video conferencing.

Concerning the mesh nodes, OViS was designed to work with a standard Linux environment and should therefore be portable to any kind of node that supports Linux with only minor effort. By using cleanly designed standard interfaces to the kernel (such as netlink), we prevent limitations concerning the supported hardware. The only concern is that with the current design, any node that is used in an OViS network needs to have at least two wireless interfaces that are properly supported by modern Linux drivers.

The evaluation results presented in Section 6.2 have shown that OViS' user-friendliness comes at a price, namely a reduced efficiency of the resulting network. With some knowledge about wireless networks, the link quality can be significantly increased and the number of required nodes can be reduced. The improvements have been particularly significant where nodes were placed above / below each other. However, such optimizations take a lot of time, even for experienced users. Also, by not reaching the maximum distance between two nodes, OViS provides a buffer against temporary fluctuations in the network connectivity.

The guided deployment process might be improved by adding additional antennas to the

mesh nodes: the antennas that were used have a high horizontal gain but are very limited vertically. This could be compensated by adding extra antennas at a 90 degree angle for increased vertical performance.

However, OViS does not prevent a user with some additional knowledge from applying their experience during the deployment process. Therefore, we believe that despite the suboptimal result of a network deployed by strictly following the client instructions, OViS does a good job at helping non-experts to deploy stable and reliable networks.

As a result of those considerations, we conclude that OViS is a flexible platform which, while favouring simplicity over efficiency, allows end-users to deploy stable network connectivity in previously uncovered areas. By focusing on simplicity, OViS makes mesh networks accessible to users who would otherwise not be able to profit from this technology.

7.2 Future Work

While OViS works quite well in general, further improvements can of course always be made not only to our system and code but also to some of the related third-party projects.

One area that comes to mind is Android and its lack of proper ad-hoc networking. A future project could try to modify the Android platform to natively connect to ad-hoc networks and possibly create a customized version of Android built around and focused on the task of running OViS. Such a platform would ideally use Android 2.3 (or newer) as a basis due to its native support for SIP voice communication and replace the stock Android launcher with a menu specific to OViS and appropriate branding to achieve the appliance experience that was attempted to create on the hand-held device.

Another part that could benefit greatly from future work is the integration of the video chat program. A future project could try to use *SkypeKit* [68], an SDK that should allow for proper integration of Skype into the OViS client. For now, SkypeKit is only available to select developers in a closed beta program and was therefore not an option for OViS. Once it becomes available to the public, this will change. Alternatively, open-source video conferencing programs should keep improving and, due to their source code being publicly available, would probably be reasonably easy to include in the OViS client once they start working reliably. Future work could focus on those tools instead of the proprietary Skype application.

Further efforts should also be put into the IEEE 802.11s mesh networking protocol. The technology looks promising and highly interesting. The benefits of moving the mesh networking logic to a lower layer and having it not only included in the standard Linux kernel but, since it will hopefully become a widely adopted IEEE standard, probably in other operating systems too, could greatly simplify the deployment and management of ad-hoc mesh networks. A future project could try and resolve the current limitations related to the use of multiple network interfaces.

Lastly, OViS is IPv6 ready, i.e., once it becomes more widely adapted, OViS can easily be modified to use IPv6 for its mesh network. This would make the hand-held device properly addressable over the Internet and thereby resolve any issues with free video conferencing implementations that are related to network address translation and the use of private, non-routed IPv4 addresses.

List of Acronyms

ADAM Administration and Deployment of Adhoc Mesh networks

AODV Ad-Hoc On-Demand Distance Vector routing protocol

API Application Programming Interface

ARM Advanced RISC Machine

CGI Common Gateway Interface

CLI Command-Line Interface

CPU Central Processing Unit

CRDA Central Regulatory Domain Agent

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

EEPROM Electrically Erasable Programmable Read-Only Memory

ESSID Extended Service Set ID

GIMP GNU Image Manipulation Program

GNOME GNU Network Object Model Environment

GNU GNU's Not Unix

GPL GNU General Public License

GPS Global Positioning System

GUI Graphical User Interface

HTTP HyperText Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

HNA Host and Network Association

ICMP Internet Control Message Protocol

IEEE Institute of Electrical and Electronics Engineers

IP Internet Protocol

JSON JavaScript Object Notation

LAN Local Area Network

MAC Medium Access Control

MPR Multipoint Relay

MTU Maximum Transmission Unit

NAT Network Address Translation

NDK Native Development Kit

OLSR Optimized Link State Routing protocol

OS Operating System

PC Personal Computer

RAM Random-access Memory

RISC Reduced Instruction Set Computing

SDK Software Development Kit

SSH Secure Shell

SIP Session Initiation Protocol

SSL Secure Sockets Layer

TCP Transmission Control Protocol

UDP User Datagram Protocol

UI User Interface

UMPC Ultra-Mobile Personal Computer

WLAN Wireless Local Area Network

WMN Wireless Mesh Network

XMPP Extensible Messaging and Presence Protocol

YAML YAML Ain't Markup Language

Bibliography

[1] R. Droms, "Dynamic Host Configuration Protocol," RFC 2131 (Draft Standard), Internet Engineering Task Force, Mar. 1997, updated by RFCs 3396, 4361, 5494. [Online]. Available: http://www.ietf.org/rfc/rfc2131.txt

[2] T. Narten, R. Draves, and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6," RFC 4941 (Draft Standard), Internet Engineering Task Force, Sept. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4941.txt

[3] P. Savola, "Linux IPv6 Router Advertisement Daemon (radvd)," retrieved on January 28, 2011. [Online]. Available: http://www.litech.org/radvd/

[4] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," RFC 3626 (Experimental), Internet Engineering Task Force, Oct. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3626.txt

[5] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," RFC 3561 (Experimental), Internet Engineering Task Force, July 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3561.txt

[6] The olsr.org project, "The olsr.org OLSR daemon," 2009. [Online]. Available: http://www.olsr.org/

[7] T. Staub, M. Anwander, K. Baumann, T. Braun, M. Brogle, K. Dolfus, C. Félix, and P. K. Goode, "Connecting Remote Sites to the Wired Backbone by Wireless Mesh Access Networks," in *16th European Wireless Conference*, Lucca, Italy, April 2010, pp. 675–682, IEEE Xplore, ISBN 978-1-4244-5999-5.

[8] A. S. Tanenbaum, *Computer Networks*, 4th ed. Prentice Hall, 2003, ISBN 0-13-066102-3.

[9] "IEEE 802.11: WIRELESS LOCAL AREA NETWORKS (LANs)," retrieved on February 26, 2011. [Online]. Available: http://standards.ieee.org/about/get/802/802.11.html

[10] M. Tabakiernik, "Devel Digest, Vol 36, Issue 7," Sept. 2010, retrieved on February 26, 2011. [Online]. Available: http://open80211s.com/pipermail/devel/2010-September/000526.html

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*. Boston, MA: Addison-Wesley, 1995, ISBN 0-20-163361-2.

[12] IEEE Computer Society LAN/MAN Standards Committee, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band," 1999. [Online]. Available: http://standards.ieee.org/getieee802/download/802.11b-1999.pdf

[13] C.-M. Cheng, P.-H. Hsiao, H. Kung, and D. Vlah, "Adjacent channel interference in dual-radio 802.11a nodes and its impact on multi-hop networking," in *IEEE GLOBECOM 2006*, San Francisco, CA, USA, 27 November - 1 December 2006.

[14] V. Angelakis, S. Papadakis, N. Kossifidis, V. A. Siris, and A. Traganiti, "The effect of using directional antennas on adjacent channel interference in 802.11a: Modeling and experience with an outdoors testbed," in *4th International workshop on Wireless Network Measurements (WiNMee 2008)*, Berlin, Germany, March 2008.

[15] J. Nachtigall, A. Zubow, and J.-P. Redlich, "The Impact of Adjacent Channel Interference in Multi-Radio Systems using IEEE 802.11," in *Wireless Communications and Mobile Computing Conference (IWCMC '08)*, Crete Island, Greece, August 6-8 2008, pp. 874–881.

[16] P. Dely, M. Castro, S. Soukhakian, A. Moldsvor, and A. Kassler, "Practical considerations for channel assignment in wireless mesh networks," *Measurement*, vol. 1, pp. 3–7, 2010.

[17] D. Balsiger, "Administration and Deployment of Wireless Mesh Networks," Master's thesis, University of Bern, Apr. 2009.

[18] Skype ltd., "Skype Accessories (the Skype Public API)," retrieved on January 31, 2011. [Online]. Available: http://developer.skype.com/accessories

[19] Nokia Corporation, "Qt - Cross-platform application and UI framework," retrieved on February 2, 2011. [Online]. Available: http://qt.nokia.com/

[20] J. Schmidt, "Der Lochtrick," *c't Magazin für Computer Technik*, no. 17, pp. 142–143, Aug. 2006.

[21] L. R. Rodriguez, J. Berg, and M. Green, "Central Regulatory Domain Agent (CRDA)," retrieved on February 24, 2011. [Online]. Available: http://wireless.kernel.org/en/developers/Regulatory/CRDA

[22] J. Salim, H. Khosravi, A. Kleen, and A. Kuznetsov, "Linux Netlink as an IP Services Protocol," RFC 3549 (Informational), Internet Engineering Task Force, July 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3549.txt

[23] G. Kroah-Hartman, "udev - Linux Dynamic Device Management," retrieved on February 28, 2011. [Online]. Available: http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev/udev.html

[24] iSteve and T. Janousek, "Hotplug2, a Lightweight udev Replacement," retrieved on February 24, 2011. [Online]. Available: http://code.google.com/p/hotplug2/

[25] wxPython Developers and Contributors, “wxPython, a Blending of the wxWidgets C++ Class Library with the Python Programming Language,” retrieved on January 30, 2011. [Online]. Available: http://www.wxpython.org

[26] wxWidgets Developers and Contributors, “The wxWidgets Project,” retrieved on January 30, 2011. [Online]. Available: http://wxwidgets.org/about/

[27] F. Damiani and P. Giannini, “The TkInter homepage,” retrieved on January 30, 2011. [Online]. Available: http://www.python.org/topics/tkinter

[28] J. Ousterhout, “Tcl Developer Xchange,” retrieved on January 30, 2011. [Online]. Available: http://www.tcl.tk

[29] Google Inc., “Android,” retrieved on February 2, 2011. [Online]. Available: http://www.android.com/

[30] DalvikVM.com, “Dalvik Virtual Machine insights,” retrieved on March 29, 2011. [Online]. Available: http://www.dalvikvm.com/

[31] HTC Corporation, “HTC Desire - Specification,” retrieved on March 29, 2011. [Online]. Available: http://www.htc.com/www/product/desire/specification.html

[32] M. R. Souryal, A. Wapf, and N. Moayeri, “Rapidly-Deployable Mesh Network Testbed,” in *Proceedings of the 28th IEEE conference on Global telecommunications*, ser. GLOBECOM’09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 5536–5541, ISBN 978-1-4244-4147-1.

[33] T. Staub, S. Ott, and T. Braun, “Automated Deployment of a Wireless Mesh Communication Infrastructure for an On-site Video-conferencing System (OViS),” in *4th ERCIM Workshop on eMobility co-located with the 8th International Conference on wired/Wireless Internet Communications (WWIC 2010)*, Lulea, Sweden, May 2010, ISBN 978-91-7439-103-9.

[34] T. Staub, “Development, Testing, Deployment and Operation of Wireless Mesh Networks,” Ph.D. dissertation, University of Bern, Bern, Switzerland, May 2011.

[35] T. Staub, S. Ott, and T. Braun, “Videoconferencia,” *RTI - Redes, Telecom e Instalações, revista brasileira de infra-estructura e tecnologias de comunicação*, vol. XII, no. 133, pp. 80–85, June 2011.

[36] M. Stolz, “iPhone/iPad Mesh Deployment Tool for Onsite Video System (OViS),” Bachelor’s thesis, University of Bern, Bern, Switzerland, to be submitted.

[37] A. Conta and S. Deering, “Generic Packet Tunneling in IPv6 Specification,” RFC 2473 (Proposed Standard), Internet Engineering Task Force, Dec. 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2473.txt

[38] P. Srisuresh and K. Egevang, “Traditional IP Network Address Translator (Traditional NAT),” RFC 3022 (Informational), Internet Engineering Task Force, Jan. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3022.txt

[39] The olsr.org project, “Releases — www.olsr.org,” retrieved on January 28, 2011. [Online]. Available: http://www.olsr.org/?q=download

[40] D. Crockford, “The Application/JSON Media Type for JavaScript Object Notation (JSON),” RFC 4627 (Informational), Internet Engineering Task Force, July 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4627.txt

[41] O. Ben-Kiki, C. Evans, and B. Ingerson, “YAML Ain’t Markup Language (YAML) (tm) Version 1.2,” YAML.org, Tech. Rep., September 2009. [Online]. Available: http://www.yaml.org/spec/1.2/spec.html

[42] W. R. Stevens, B. Fenner, and A. M. Rudoff, *UNIX Network Programming, Vol. 1*, 3rd ed. Pearson Education, 2003, p. 51, ISBN 0-13-141155-1.

[43] A. Conta, S. Deering, and M. Gupta, “Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification,” RFC 4443 (Draft Standard), Internet Engineering Task Force, Mar. 2006, updated by RFC 4884. [Online]. Available: http://www.ietf.org/rfc/rfc4443.txt

[44] W. R. Stevens, B. Fenner, and A. M. Rudoff, *UNIX Network Programming, Vol. 1*, 3rd ed. Pearson Education, 2003, p. 799, ISBN 0-13-141155-1.

[45] PC Engines GmbH, “ALIX System Boards,” retrieved on February 24, 2011. [Online]. Available: http://pcengines.ch/alix.htm

[46] R. Floeter and N. Kossifidis, “ath5k - Linux Wireless,” retrieved on February 25, 2011. [Online]. Available: http://wireless.kernel.org/en/users/Drivers/ath5k

[47] D. Robinson and K. Coar, “The Common Gateway Interface (CGI) Version 1.1,” RFC 3875 (Informational), Internet Engineering Task Force, Oct. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3875.txt

[48] R. Love, “inotify - a Powerful Yet Simple File Change Notification System,” retrieved on February 23, 2011. [Online]. Available: http://www.kernel.org/doc/Documentation/filesystems/inotify.txt

[49] L. Torvalds, “Linux v2.6.13,” retrieved on February 23, 2011. [Online]. Available: http://www.kernel.org/pub/linux/kernel/v2.6/ChangeLog-2.6.13

[50] The Wireshark team, “Wireshark Network Traffic Analyzer,” retrieved on February 24, 2011. [Online]. Available: http://www.wireshark.org/

[51] R. Hinden and S. Deering, “IP Version 6 Addressing Architecture,” RFC 4291 (Draft Standard), Internet Engineering Task Force, Feb. 2006, updated by RFCs 5952, 6052. [Online]. Available: http://www.ietf.org/rfc/rfc4291.txt

[52] P. Fertser, "Overriding Regulatory Domain for ath5k, ath9k and ar9170," retrieved on February 25, 2011. [Online]. Available: http://forum.aircrack-ng.org/index.php?topic=6822.0

[53] Pidgin Developers and Contributors, "Pidgin, the Universal Chat Client," retrieved on June 24, 2011. [Online]. Available: http://www.pidgin.im/

[54] The GNOME Project, "Empathy," retrieved on June 24, 2011. [Online]. Available: http://live.gnome.org/Empathy

[55] S. Ott, "Debian GNU/Linux on the Asus R2H UMPC," retrieved on February 26, 2011. [Online]. Available: http://www.ott.net/knowledge/r2hdebian/

[56] Unknown Author, "Android Issue 82: Wifi - Support Ad Hoc Networking," 2008, retrieved on February 2, 2011. [Online]. Available: http://code.google.com/p/android/issues/detail?id=82

[57] J. Tourrilhes, "Wireless Tools for Linux," retrieved on February 2, 2011. [Online]. Available: http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html

[58] S. Ott, "HOWTO Connect an HTC Desire to an Ad-Hoc Network," retrieved on February 2, 2011. [Online]. Available: http://www.ott.net/knowledge/htc-desire-adhoc/

[59] P. O'Brien, "Android @ MoDaCo - Device Specific - HTC Desire - Desire.MoDaCo.com," retrieved on February 2, 2011. [Online]. Available: http://desire.modaco.com/

[60] S. Ott, "desire-adhoc - Android App to Create / Join an Ad-Hoc Network," retrieved on February 2, 2011. [Online]. Available: http://code.google.com/p/desire-adhoc/

[61] Free Software Foundation, "GNU General Public License," retrieved on February 2, 2011. [Online]. Available: http://www.gnu.org/licenses/gpl.html

[62] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261 (Proposed Standard), Internet Engineering Task Force, June 2002, updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026. [Online]. Available: http://www.ietf.org/rfc/rfc3261.txt

[63] Google Inc., "Android 2.3 Platform Highlights," retrieved on February 21, 2011. [Online]. Available: http://developer.android.com/sdk/android-2.3-highlights.html

[64] P. Merle, "sipdroid - Free SIP/VoIP Client for Android," retrieved on February 2, 2011. [Online]. Available: http://code.google.com/p/sipdroid/

[65] L. Journal-World, "django - The Web framework for perfectionists with deadlines," retrieved on February 2, 2011. [Online]. Available: http://www.djangoproject.com/

[66] P. Saint-Andre, “Extensible Messaging and Presence Protocol (XMPP): Core,” RFC 3920 (Proposed Standard), Internet Engineering Task Force, Oct. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3920.txt

[67] T. B. Dave Turner, “A Network Protocol Independent Performance Evaluator,” retrieved on March 26, 2011. [Online]. Available: http://www.scl.ameslab.gov/netpipe/

[68] Skype ltd., “Introducing the SkypeKit Beta Program,” retrieved on February 28, 2011. [Online]. Available: http://blogs.skype.com/developer/2010/06/skypekit_beta.html

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