VL Zuverlässige Systeme

Projekt 5: Fremdortung mit WLAN

Johannes Semmler, Robert Hilbrich

7. Februar 2007

Im Zuge der zunehmenden Verbreitung von mobilen Geräten gewinnt auch die WLAN Technologie stark an Bedeutung. Die mobile Vernetzung von Endgeräten bietet jedoch mehr Nutzungsmöglichkeiten als die der kabelgebundenen Netzwerktechnik.

Eine dieser neuen Möglichkeiten liegt in der Fremdortung von mobilen Geräten, die in diesem Projekt evaluiert werden sollen. Diese Ortungstechnik soll, unabhängig von GPS, die Ortung eines WLAN Clients in einem Funknetzwerk ermöglichen. Die Auswertung dieser Information könnte dann genutzt werden um zum Beispiel standortabhängige Informationen bereitzustellen.

Praktische Anwendung könnte so etwas zum Beispiel in Museen oder Ausstellungen finden. Auch Gebäudeleitsysteme auf Grundlage dieser Technik wären denkbar.

Inhaltsverzeichnis

1 Projektziel

In diesem Projekt soll eine Demonstration von Fremdortung mittels WLAN entstehen. Der so enthaltene Standort soll dann entsprechend ausgewertet werden, um standortabhängige Informationen zu übermitteln. Im folgenden wird der prinzipielle Ablauf dargestellt.

Zum Anfang gehen wir davon aus, dass der mobile Client mit einem Access Point assoziiert ist um Zugang zum WLAN zu erhalten. Die IP Adresse des Clients und des Webservers sind bekannt, bzw. es ist ein DNS Server installiert der die Namensauflösung übernimmt.

Schickt nun der Client eine Anfrage an den Webserver, so startet dieser vor dem Senden der HTML Antwort eine Standortbestimmung des Clients durch die Access Points. Die Access Points bestimmen nun die von ihnen wahrgenommene Empfangsstärke des Clients, normieren diese und schicken sie zurück an den Webserver. Dieser wertet nun die übermittelten Signalinformationen aus und bestimmt den Standort des Clients. Dabei wird die Auswertung der Signalinformationen durch das Projekt MagicMap[1] übernommen.

Nun könnte die HTML Antwort des Webservers aufgrund des Clientstandortes bestimmt werden und an diesen zurückgeschickt werden.

In Anbetracht des kurzen Realisierungszeitraums, haben wir uns entschlossen, das Projekt auf die Anbindung an *MagicMap* zu beschränken. Durch diese Konzentration auf den Kernbestandteil wird die vollständige Funktionalität von WLAN Fremdortung demonstriert und eine Realsierung im Zeitrahmen ermöglicht.

Zur Realisierung stehen uns die folgenden Mittel zur Verfügung:

- 2x Asus WL 500g WLAN Access Points mit Atheros MiniPCI WLAN Karten
- 3x Netgear WGT 634U WLAN Access Points mit Atheros MiniPCI WLAN Karten
- zwei Notebooks

[1] Ein System zur kooperativen Positionsbestimmung über WLAN

2 Projektrealisierung

2.1 Architektur

Wie im Bild 2.1 zu erkennen ist, gibt es zwei Netzwerkbereiche beim Einsatz von WiSpy. So gibt es zum einen das Netzwerk, auf dem der eigentliche Netzwerkverkehr zwischen dem mobilen Client und dem WebServer abgewickelt wird - das *Content-Network*. In diesem Netzwerk befinden sich nur zwei Geräte: der Mobil Client in Form eines XDA Neo und der Web- und DHCP Server auf einem LinkSys Router mit OpenWRT als Firmware installiert.

Daneben gibt es noch ein weiteres Netzwerk, das alle Access-Points miteinander verbindet. Es dient zum einen der Übermittlung der Signalstärke durch die WLAN Router an den zentralen Sammelpunkt - dem *WiSpy Collector* und zum Anderen der geschickten Fernwartung der Router per *SSH*.

Durch die besondere Trennung des IP Class-C Netzwerks `192.168.17.0/24` können die beiden logischen Netzwerke in diesem physikalischen Netz betrieben werden.

Ziel ist der folgende Aufbau. Die Access Points hören den Verkehr in Reichweite mit und führen Buch über die Signalstärken der vorhanden Wireless LAN Clients, bzw. deren MAC Adressen. Diese Signalstärken können dann von außen mit dem WiSpy Collector abgefragt werden. Dies funktioniert mittels UDP Pakete, um unnötigen Overhead zu vermeiden. Hat der WiSpy Collector alle Signalinformationen gesammelt, werden diese an MagicMap weitergeleitet, wo die eigentliche Standortermittlung des Clients stattfindet.

2.2 Routerkonfiguration

Die verwendeten Router basieren auf einer MIPS Architektur und sind von Hause aus mit herstellereigener Firmware bestückt. Damit wir in der Lage waren, unsere Software auf diesen Router zu benutzen, mussten wir uns zwei Herausforderungen stellen. Zunächst musste die Firmware durch OpenWRT - eine Linux-Distribution für Router - ersetzt

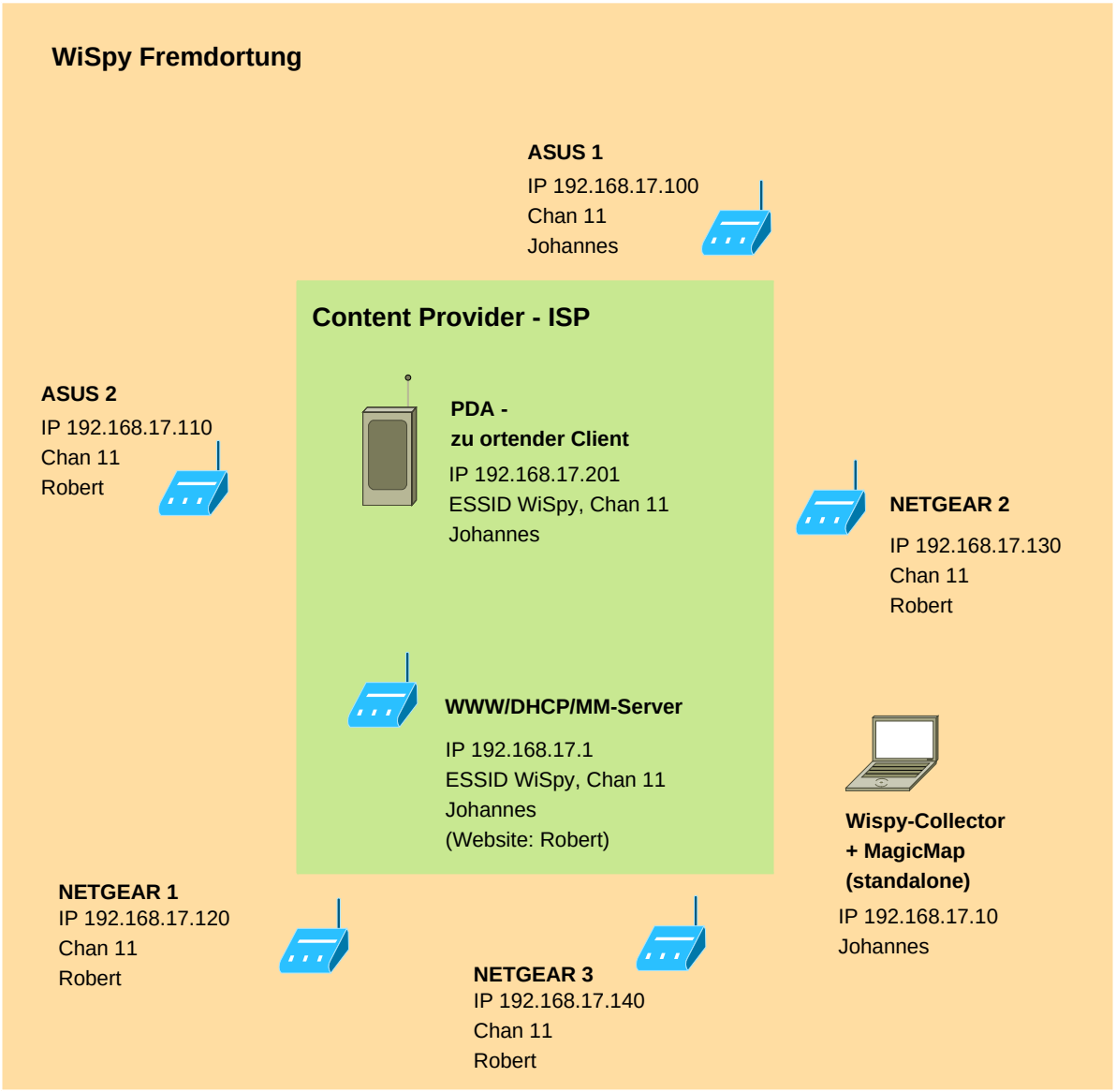

Abbildung 2.1: Aufbau der WiSpy Infrastruktur

werden. Danach brauchten wir einen Compiler, um Binär-Code für die MIPS Architektur erstellen zu können.

2.2.1 OpenWRT Firmware

Da die beiden verwendeten Routertypen (Netgear WGT634U und ASUS WL500G) von OpenWRT [1] sehr gut unterstützt wurden, gestaltete sich das Erstellen und Installieren der Firmware nicht schwierig. Wegen der neuen Version des MadWifi Treibers haben wir uns für die Version *Kamikaze* entschieden. Die ältere, aber stabile Version *WhiteRussian* bringt leider nur den alten MadWifi Treiber mit.

Durch die folgenden Schritte wurde die Firmware erstellt:

```
svn co https://svn.openwrt.org/openwrt/trunk/
cd trunk
make menuconfig
make V=99
```

Im Verzeichnis `trunk/bin/` befand sich nun das OpenWRT-TRX File, das nun auf die Router gespielt werden konnte.

Bei den ASUS Routern haben wir das mit dem kostenlos erhältlichen *ASUS Firmware Restoration Tool* durchführen können. Bei den Netgear Routern mußten wir hingegen ein Console-Kabel anschließen, um das Image dann während des Boot Vorgangs im *CFE* Bereich flashen zu können.

Nach einem Neustart der Router und dem Durchführen der Netzwerkkonfiguration, boten die Router einen SSH Fernwartungszugang und waren damit bereit für weitere Aufgaben.

2.2.2 Cross-Compiler

Zur Programmerstellung auf der Zielplattform MIPS brauchten wir noch einen Cross-Compiler mit der Fähigkeit Binärcode für MIPS zu erstellen. Dazu wählten wir im im `make menuconfig` den Punkt `Build OpenWRT SDK` aus. Durch den anschließenden Lauf von `make` wurde dann der GNU C Compiler und GNU C++ Compiler mit den entsprechenden Optionen gebaut. Nach einem erfolgreichen Build war der Compiler im Verzeichnis `trunk/staging_dir_mipsel/bin/mipsel-linux-uclibc-c++` zu finden. Dieser lies sich nun mit den gleichen Optionen aufrufen, wie der klassische GCC/G++.

[1] `www.openwrt.org`

2.2.3 IPKG Paket erstellen

OpenWRT benutzt zur Softwareinstallation die Software `ipkg`. Diese Paketverwaltungssoftware ist angelehnt an die Syntax von `dpkg`, das man auf Debian basierten Linux Systemen findet. Ziel war es, automatisch ein IPKG-Paket von unserer Software zu erstellen, damit die Installation von neuen Version automatisiert ablaufen kann.

Dazu wurde im Pfad `/trunk/package` ein neues Verzeichnis `WiSpy` angelegt. Damit dieses beim nächsten Aufruf von `make menuconfig` das Paket auswählbar ist, mussten wir ein Makefile entwickeln, das zusätzliche Informationen über das Paket enthält. Dies hat folgenden Inhalt:

```
include $(TOPDIR)/rules.mk
PKG_NAME:=WiSpy
PKG_VERSION:=1
PKG_RELEASE:=1
PKG_BUILD_DIR:=$(BUILD_DIR)/$(PKG_NAME)
PKG_BUILDDEP:=libpthread uclibcxx
include $(INCLUDE_DIR)/package.mk
define Package/wispy
    SECTION:=net
    CATEGORY:=Network
    TITLE:=WiSpy Server
    DEPENDS:=+libpthread +uclibcxx
    DESCRIPTION:=WiSpy - A signal tracker
endef
define Build/Prepare
    mkdir -p $(PKG_BUILD_DIR)
    $(CP) ./src/* $(PKG_BUILD_DIR)/
endef
define Build/Compile
    $(MAKE) -C $(PKG_BUILD_DIR) \
        LD=mipsel-linux-uclibc-ld \
        NM=mipsel-linux-uclibc-nm \
        CC="mipsel-linux-uclibc-gcc" \
        GCC="mipsel-linux-uclibc-gcc" \
        CXX=mipsel-linux-uclibc-g++ \
        CFLAGS="$(TARGET_CFLAGS)"  \
        CXXFLAGS="$(TARGET_CFLAGS) -O0  -fno-builtin -fno-rtti -nostdinc++" \
        CPPFLAGS="-I$(STAGING_DIR)/usr/include -I$(STAGING_DIR)/include \
           -I$(LINUX_DIR)/include" \
        LDFLAGS="-nodefaultlibs -L$(STAGING_DIR)/usr/lib -L$(STAGING_DIR)/lib" \
```

```
        LIBS="-luClibc++ -lc -lm -lgcc"
endef
define Package/wispy/install
    $(INSTALL_DIR) $(1)/usr/local/bin/
    $(INSTALL_BIN) $(PKG_BUILD_DIR)/wispy $(1)/usr/local/bin
endef
$(eval $(call BuildPackage,wispy))
```

Das eigentliche Verzeichnis `trunk/package/wispy` beinhaltet dann das obige Makefile und ein Unterverzeichnis `src`. Dort sind alle Sourcen inklusive Makefile zum Kompilieren enthalten. Es ist dabei zu beachten, dass das Kompilier-Makefile Gebrauch der Variablen `CC/GCC/CXX/CXXFLAGS/...` machen muß, damit diese entsprechend vom IPKG-Makefile gesetzt werden können. Unser Kompilier-Makefile sieht so aus:

```
SOURCES=packetreader.cpp server.cpp wispy.cpp
OBJS=packetreader.o server.o wispy.o
TARGET=wispy
wispy: ${OBJS}
    ${CXX} ${CXXFLAGS} ${CPPFLAGS} ${LDFLAGS} \
           ${LIBS} -lpthread -o ${TARGET} ${OBJS}
.cpp.o:
    ${CXX} -Wall -c ${CXXFLAGS} ${CPPFLAGS} $<
clean:
    -rm *.o ${TARGET}
```

Damit kann unser Paket `WiSpy` schnell für verschiedene Architekturen kompiliert und in das OpenWRT Package Format umgewandelt werden. Durch eine Einstellung beim Aufruf von `make menuconfig` kann das Paket ebenfalls gleich in das Firmware-Image eingebunden werden.

Ein Aufruf von `make package/wispy-compile V=99` führt zum Neukompilieren des Paketes, ohne dass das komplette SDK von OpenWRT neu gebaut wird. Ein schnelles Kompilieren zu Debug-Zwecken ist so einfach möglich.

2.3 Auslesen der Signalstärke

WLAN Clients sind im Gegensatz zu den Access Points typischerweise unsichtbar und durch Standardtools nicht anzeigbar. Wir hatten zunächst überlegt, ob wir mit der Adhoc Netzwerk Infrastruktur arbeiten, um dieses Problem zu umgehen. Leider ist dafür

eine Rekonfiguration des Clients notwendig, so dass wir diese Idee wieder verworfen haben.

Wesentlich vielversprechender ist jedoch der Monitor-Modus eines WLAN Interfaces. In einem Prototyp konnten wir hier mit dem Programm KISMET auch die Signalstärken von Clients auslesen, die Netzwerkverkehr auslösen. Stille Clients wurden nicht erkannt. Dies ist jedoch kein Problem, da der Client ausreichend Netzwerkverkehr beim Absenden der Web-Anfrage an den Webserver erzeugt und somit nicht mehr still ist.

2.3.1 Erster Versuch - Reduktion von Kismet

Da Kismet keinen direkten Zugang zu Signalstärke und MAC Adresse bot, haben wir zunächst versucht, aus dem Quellcode die Funktionalität herauszulösen und mit unserem Code zu ergänzen. Angefangen haben wir mit einem aktuellen Abzug der Quellen mittels `svn co http://svn.kismetwireless.net/code/trunk kismet-devel`. Nach einem Entwicklungsaufwand von etwa 4 Wochen hatten wir einen ersten Prototypen erstellt, der für X86 kompilierte und die gewünschten Ausgaben lieferte. Dessen Funktionalität basierte auf dem Code von Kismet und beinhaltete wie geplant nur die wirklich benötigten Teile. So wurde u.a. der GPS Support und die Unterstützung für nicht Atheros-basierte WLAN Karte herausgelöst.

Ein erster Test auf einem Router verlief jedoch wenig erfreulich. Ein mehrfaches Lesen im Monitor-Modus per `recv` schien das Programm zum Abbruch zu zwingen. Ein Bug Report im Forum von OpenWRT brachte bisher noch keine weiteren Erkenntnisse. Trotz intensiven Bemühungen kamen wir hier nicht weiter. Da das Auslesen der Signalstärke allerdings integraler Bestandteil unseres Projektes war, beschlossen wir einen alternativen Ansatz zu wählen.

2.3.2 Zweiter Versuch - Patch von Kismet

Die Idee dazu lieferte eine Unterhaltung mit einem der Entwickler von Kismet im IRC Chatroom auf `irc.freenode.org` im Raum #kismet. So ist in Kismet ein sehr mächtiges Client-Server Protokoll implementiert, mit dem TCP/IP Clients sich mit einem laufenden Kismet Server verbinden und die Benachrichtigung bei bestimmten Ereignissen anfordern können. Ein solches Ereignis kann das Mitlesen eines Paketes sein. Leider wurde das Senden der Signalstärke über dieses Protokoll nicht vorgesehen.

Inständiges Bitten und Betteln von uns wurde jedoch belohnt, so dass der selbe Entwickler noch live diese Änderungen in das Protokoll überführte und ebenfalls in den

aktuellen SVN Sourcetree integrierte. Da ein neues Release auch unmittelbar bevorstand brauchten wir nur noch kurz warten, bis dieses veröffentlich wurde und ein offizielle OpenWRT Paket dafür existierte. Dies ist nun geschehen, so dass unser WiSpy Client einfach nur eine lokale TCP/IP Sitzung zum laufenden Kismet Server aufmachen muss, um die Pakete, deren Signalstärke und die MAC Adresse des Absenders auszulesen.

Der Client aktiviert die benötigten Daten durch Senden des folgenden Texts an den Server:

```
!0 ENABLE PACKET type,subtype,sourcemac,signal,noise
```

Daraufhin schickt der Server bei jedem neuen Paket eine Antwort der Form:

```
*PACKET: 0 8 00:0C:41:9C:C4:87 -35 0
*PACKET: 0 8 00:0C:41:9C:C4:87 -35 0
*TIME: 1169054686
```

Dies musste dann nur noch entsprechend verarbeitet werden. Zuständig dafür ist das Programm *WiSpy*, das zusätzlich zum Kismet Server auf dem Router läuft.

2.4 MagicMap

MagicMap ist eine Softwarelösung zur Positionsbestimmung von WLAN. Dabei ist lediglich der MagicMap-Client auf einem WLAN-fähigen Notebook nötig und einige Access Points. Auf diesen sind keine Modifikationen nötig, so dass MagicMap in nahezu jeder WLAN-Umgebung funktionsfähig ist. Zur Bestimmung der Position werden die Signalstärken von jedem sichtbaren Access Point verwendet, welche MagicMap unter Windows aus dem Programm NetworkStumbler und unter Linux aus iwlist bezieht. Aus den Signalstärkeinformationen berechnet MagicMap die aktuelle Position des Clients und stellt diese auf einer Karte dar. Zur Erhöhung der Genauigkeit lassen sich Referenzpunkte setzen, für welche die Signalstärken gespeichert werden und als Referenzen herangezogen werden können.

Die Übergabe der Daten von NetworkStumbler bzw. iwlist an MagicMap funktioniert über ein Script, welches UDP-Pakete über Port 2446 an MagicMap schickt. Diese haben das folgende Format
`|1*|Unknown*|BSSID*|Signal*|100*|Time *|hostname*|mac*|`
Darin enthalten sind Informtionen zu BSSID, hostname und Signalstärke.

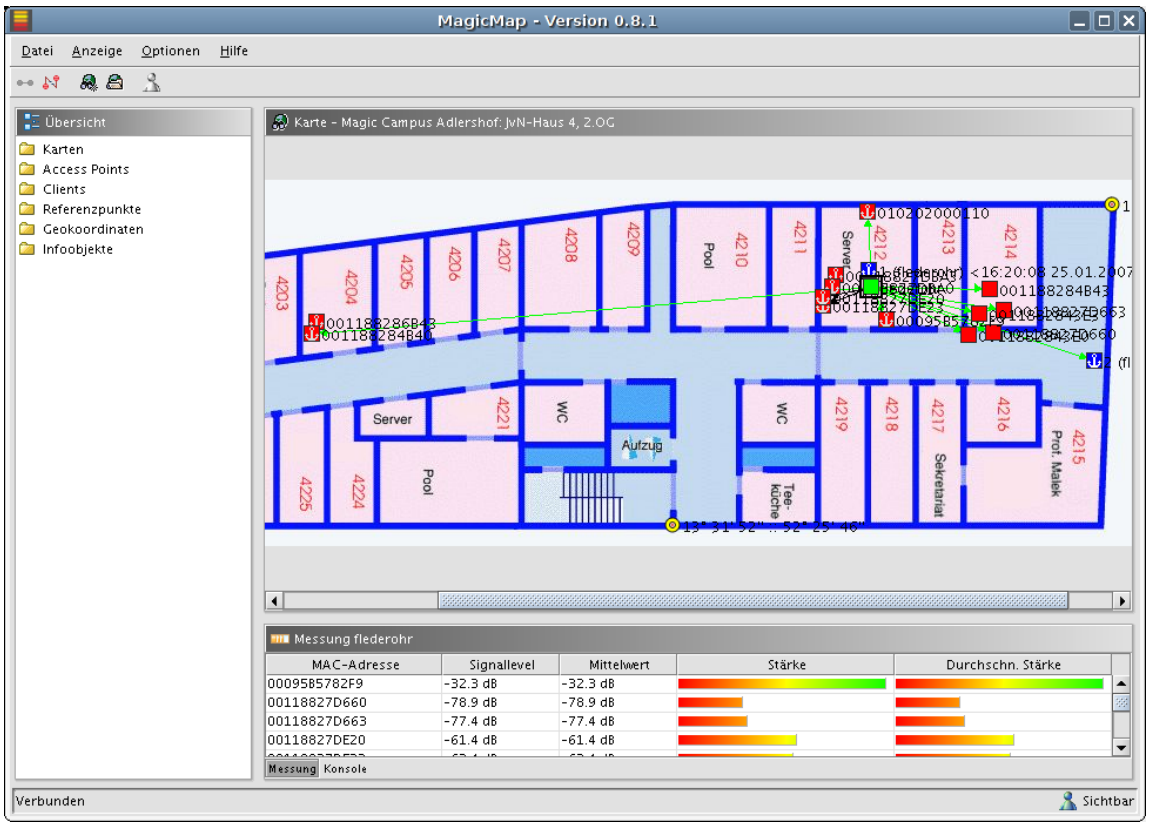

Abbildung 2.2: Software MagicMap zur Ortung des Clients

An dieser Stelle setzt nun WiSpy ein. Statt die Signalstärkeinformationen von einem lokalen Skript an MagicMap zu schicken werden Pakete im selben Format generiert, welche jedoch nicht die Signalstärken des MagicMap-Rechners enthaltem sondern jene die zu einem anderen Client gehören und von den AccessPoints gemessen wurden. Dadurch ist es möglich, MagicMap als Backend für die Auswertung und Visualisierung der Daten zu nutzen, als Frontend für die Messungen jedoch die Access Points. Damit lässt sich MagicMap nicht nur zur Selbst- sondern auch zur Fremdortung nutzen.

2.5 WiSpy Collector

Der WiSpy Collector stellt die Schnittstelle zwischen MagicMap und WiSpy dar. Er sammelt die Signalstärkeinformationen von den Messstationen ein, wandelt sie in ein MagicMap-kompatibles Format und schickt sie dann an MagicMap. In der Konfigurationsdatei `wispycollect.conf` wird festgelegt, unter welchen Adressen die Monitore und der MagicMap-Client erreichbar sind:

```
monitor=192.168.17.100
monitor=192.168.17.110
monitor=192.168.17.120
monitor=192.168.17.130
monitor=192.168.17.140
client=127.0.0.1
```

2.5.1 WiSpy-Anfrage

Beim Programmaufruf wird die MAC-Adresse des zu ortenden Clients übergeben. Nach dem Start schickt WiSpy Collector periodisch alle 0,5 Sekunden an jede der in der Konfigurationsdatei eingetragene Monitorstation UDP-Pakete mit dem Inhalt `WISPYGETMAC XXXXXXXXXXXX` an Port 2447. Dabei steht `XXXXXXXXXXX` für die MAC-Adresse des Clients.

2.5.2 Paket-Umwandlung

Parallel läuft ein Thread, welcher an Port 2448 auf die Antworten der Monitorstationen gewartet. Für jede eintreffende Nachricht mit Signalstärkeinformationen wird ein Paket im MagicMap-Format generiert, mit den Signalinformationen versehen und an den MagicMap-Client gesendet.

2.6 WiSpy

Das Programm WiSpy läuft auf den Routern und sorgt dort für eine Aufbereitung der von Kismet stammenden Daten. Die Informationen über die Signalstärken der Clients werden lokal gespeichert und auf Anforderung an den WiSpy Collector geschickt. Da eine solche Anforderung durchaus zur gleichen Zeit eintreffen kann wie die Daten von Kismet, wurde das Programm mit drei parallelen Threads geschrieben.

2.6.1 Hauptthread

Die Aufgabe des Hauptthreads ist (neben dem Starten der anderen Threads) die Verwaltung der Informationen über die Signalstärken. Dazu wird eine Tabelle angelegt, welche Datensätze mit folgendem Inhalt speichert:

mac Speichert die MAC-Adresse eines Clients. Dies ist die eintige Möglichkeit, einen Client zu identifizieren, da die IP-Adressen in der Regel nicht von Kismet ausgelesen wird. Für spätere Anwendungen, in denen eine Verwendung von IP-Adressen vorteilhafter ist, ließe sich die IP durch ein einfaches ARP-Request im entsprechenden Netz erfragen.

signals Speichert eine Liste von Werten, die den Signalstärken der letzten von Kismet empfangenen Pakete entspricht.

Der Zugriff auf die Daten erfolgt mit Hilfe der folgenden Funktionen:

`putTableEntry(string mac, int signal)` Diese Funktion erweitert den Datensatz für die angegebene MAC um einen Eintrag mit dem mitgegebenen Signalwert und der aktuellen Systemzeit. Sollte die Anzahl der Datensätze für die MAC die Maximalzahl überschritten haben, so wird der älteste Eintrag gelöscht.

`getTableEntry(string mac, int n)` Diese Funktion sucht in der Tabelle nach der angegebenen MAC-Adresse und gibt die mit dem Parameter n gegebene Anzahl an Werten in einem String zurück.

2.6.2 Packetreader-Thread

Dieser Thread ist für die Kommunikation mit Kismet zuständig. Dazu wird zunächst ein TCP-Socket erzeugt eine Verbindung mit dem lokalen Port 2501 aufgebaut, auf welchem Kismet lauscht. Durch Senden einer Nachricht mit dem Inhalt
`!0 ENABLE PACKET type,subtype,sourcemac,signal,noise\n`
über dieses Socket wird Kismet dazu veranlasst die Signalstärkedaten zurückzuschicken. Daraufhin wird für jede von Kismet gesendete Nachricht überprüft, ob sie Signalstärkeinformationen enthält, d.h. ob sie mit `*PACKET` anfängt. Ist dies der Fall, so werden MAC-Adresse und Signalstärke ausgelesen und durch Aufruf der Funktion `putTableEntry()` in die Tabelle eingetragen.

2.6.3 Server-Thread

Der Server-Thread ist Zuständig für die Kommunikation mit dem WiSpy Collector. Dafür wird ein UDP-Socket auf Port 2447 aufgemacht und auf Nachrichten gewartet. Wird eine Nachricht empfangen, so wird deren Inhalt überprüft. Lautet der Nachrichtentext `WISPYGETMAC XXXXXXXXXXXX` so handelt es sich um eine gültige Abfrage, ansonsten wird sie verworfen. Ist die Abfrage gültig, so werden mit der Funktion `putTableEntry()`

die Signalstärkeinformationen für die durch `XXXXXXXXXXXX` gegebene MAC-Adresse abgefragt. Ist für den angeforderten Client kein Datensatz vorhanden, so wird eine Nachricht mit dem Inhalt `WISPYNODAT XXXXXXXXXXXX` mit der entsprechenden MAC an den Absender zurückgeschickt. Sind Daten für diese MAC vorhanden so wird eine Nachricht mit folgendem Inhalt geschickt: `WISPYDAT XXXXXXXXXXXX YYYYYYYYYYYY AA BB`

`XXXXXXXXXXXX` steht für die MAC-Adresse des Routers.

`YYYYYYYYYYYY` steht für die MAC-Adresse des angeforderten Clients.

`AA` steht für die Signalstärke des angeforderten Clients.

2.7 WWW Server

Damit der PDA Client stetigen Verkehr erzeugt wurde eine WebSite in HTML entwickelt, die sich im Sekundentakt neu lädt. Diese wurde auf dem WWW-Server Access-Point in das Verzeichnis `/www` kopiert und durch den Browser des PDAs aufgerufen. Die Webseite hat den folgenden Inhalt:

```
<html>
<head>
        <meta http-equiv="refresh" content="1; URL=index.html">
        <title> WiSpy </title>
</head>
<body>
        <h1>We are tracking you!</h1>
</body>
</html>
```

3 Experimente und Tests

3.1 Aufbau

Vereinfacht wurde der gesamte Aufbau durch die Nutzung von WLAN Karten mit dem Atheros Chipsatz. Damit ist es uns möglich, mehrere virtuelle Interfaces auf eine Netzwerkkarte zu binden und damit sowohl den "Infrastruktur"-Modus als auch den "Monitor" Modus gleichzeitig auf dem gleichen WLAN Router zu betreiben.

Damit konnte das Infrastrukturnetzwerk zum Übermitteln der Client-Signalstärken und das überwachte Netzwerk gleichzeitig von einer physikalischen Karte benutzt werden. Der zusätzliche Aufwand für die Verkabelung entfiel damit.

Das folgende Skript sorgte für den automatischen Aufbau dieser Verbindungen und virtuellen Access-Points und wurde zum automatischen Start nach dem Boot in `/etc/init.d/` entsprechend verlinkt.

```
wlanconfig ath0 create wlandev wifi0 wlanmode monitor
wlanconfig ath1 create wlandev wifi0 wlanmode sta nosbeacon
iwconfig ath1 mode managed essid WiSpy channel 11
ifconfig ath1 192.168.17.X netmask 255.255.255.0
```

Die Konfiguration des Kismet Servers erfolgte durch Anpassen der Datei `/etc/kismet/kismet.conf`. Dort wurden die folgenden Einstellungen vorgenommen

```
source=madwifi_g,wifi0,wireless
channelhop=false
sourcechannels=wireless:11
gps=false
logtypes=none
```

Nach den ersten Experimenten mit der Server Version von MagicMap hat sich herausgestellt, dass die Vielzahl der schon vorhanden AccessPoints eine gute Kalibrierung sehr

schwierig machten. Da MagicMap auch als Stand-Alone Applikation verwendet werden kann, haben wir uns dafür entschieden. Auch die Router ließen sich so wesentlich schneller und komfortabler auf der Karte platzieren.

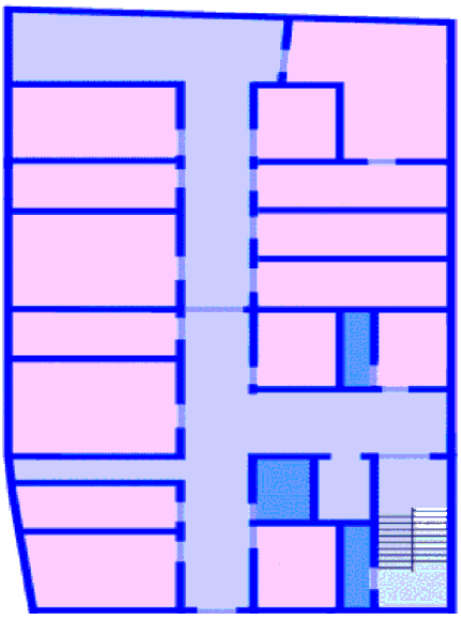

Abbildung 3.1: MagicMap-Karte für den Experimentierbereich

3.2 Ergebnisse

Das Zusammenwirken aller Komponenten des Gesamtkonzeptes wurde sehr gut in unseren Experimenten deutlich. So sammelte der WiSpy Collector alle ankommenden Datenpakete ein und reichte sie zuverlässig an MagicMap weiter.

Die Ergebnisse zur Ortung waren aber leider enttäuschend. Sehr grobe Ortung ist möglich, aber eine genauere Standort-Bestimmung, wie man es von GPS kennt ist nicht realisierbar. Die Signalstärke schwankte im regulären Betrieb so stark um den Mittelwert, dass eine präzise Ortung nicht durchführbar war. Allein schon die Abschirmung des Signals durch einen vorbeilaufenden Menschen beinflusste die Signalstärke beträchtlich. Eine Ortung basiert jedoch auf zeitlich stabilen Signaleigenschaften, die sich mit der Entfernung zuverlässig ändern. Die Signalstärke alleine scheint kein ausreichendes Mittel zu sein, um Entfernungen zu bestimmen. Möglicherweise läßt sich dies durch Verwendung der Signallaufzeit beheben. Allerdings haben wir aus programmiertechnischer Sicht keinen Zugriff auf die Physikalische Schicht der Atheros Netzwerkkarten, so dass die Integration dieser Größe in MagicMap sehr komplex sein dürfte.

Dieses Thema ist vermutlich nicht brennend neu, da sogar schon Paper existieren, die ebenfalls zu dem Schluß kommen, dass WLAN Ortung mittels Signalstärke und Rauschabstand nur schwer durchführbar ist. So ist vermutlich auch das Aufstellen von mehr AccessPoints nicht hilfreicher, da Messungen eines Meßgerätes nie genauer sein können, als das Meßgerät selbst.

In `http://www4.in.tum.de/~dornbusc/pubs/RvPiWLAN_final.pdf` wird das Thema aufgegriffen und so zusammengefasst:

> "Aussagen über die Genauigkeit lassen sich für WLANs immer nur relativ machen. Zum einen sind die Ergebnisse von der Zelldichte abhängig, die gerade bei WLANs sehr stark variieren kann.
>
> [...]
>
> Zum anderen kann die Genauigkeit bei unterschiedlicher Struktur der Umgebung erheblichen Schwankungen unterliegen. Dennoch sind wir zu dem Ergebnis gekommen, dass eine Genauigkeit die über 25m liegt nicht zu erreichen ist, auch wenn man ein Netz mit maximaler Zelldichte voraussetzt."

Zusätzlich erschwert wurden unsere Messungen durch den zufällig auftretenden Ausfall einzelner Router. Ein Neustart behob das Problem immer, aber die wegfallende Signalinformation gestaltete das Setzen von Referenzpunkten sehr schwierig.