DESIGN AND IMPLEMENTATION OF WLAN SUPPORT FOR CELLULAR ASSISTED HETEROGENEOUS NETWORKING

Diplomarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

Vorgelegt von:

Ehsan Maghsoodi

2004

Leiter der Arbeit:
Prof. Dr. Torsten Braun
Forschungsgruppe Rechnernetze und Verteilte Systeme (RVS)
Institut für Informatik und angewandte Mathematik

Betreuer der Arbeit:
Marc Danzeisen
Forschungsgruppe Rechnernetze und Verteilte Systeme (RVS)
Institut für Informatik und angewandte Mathematik

This diploma thesis is dedicated to
Abbas and Mansoure

ABSTRACT

CAHN (Cellular Assisted Heterogeneous Networking) [1] is an approach for managing several mobile nodes in a heterogeneous environment. CAHN separates the data plane from the signalling plane for the establishment of a secured and automated heterogeneous data link between the mobile nodes. A cellular network like GSM [2] is applied for the signalling plane, which is provided by a cellular operator. The existing trust relation between the user and the operator of a cellular network (Subscriber Identity Module) builds the fundament for authentication in CAHN applications. For the data plane, CAHN desires to apply mobile technologies which give the user the possibility to share services and resources with others. To achieve this purpose, CAHN intents to extend the usage of the data channel to a wide range of today's technologies such as Bluetooth [3] and WLAN [4].

This diploma work engages with defining a new CAHN protocol using protocol engineering principles. These principles help studying and improving the existing CAHN Bluetooth protocol [5] as well as developing further enhancements and improvements to this protocol for additional and supplementary services. All in all, the goal of this diploma work is to provide an integration of the CAHN approach and the WLAN technology. To realize this integration, a solution approach based on the Bloom filter [6] concept is suggested. It uses the mathematical idea of the Bloom filter to reduce the CAHN specific information (MSISDN) and to provide the CAHN ability to mobile nodes.

ZUSAMMENFASSUNG

Bei der Betrachtung heutiger Kommunikationsmethoden fällt auf, dass mobile und drahtlose Netzwerktechnologien immer mehr an Bedeutung gewinnen. Viele Geräte wie Laptops, PDAs oder Mobiltelefone verfügen bereits über eingebaute Adapter für derartige Netze. Die Vielfalt dieser Technologien macht es oftmals unmöglich, bei Bedarf einen gemeinsamen Kommunikationskanal zwischen zwei Benutzern aufzubauen. Das wiederum hat zur Folge, dass die nötigen Sicherheitsparameter nicht ausgetauscht werden können, um einen sicheren Datenkanal zu etablieren. Man stelle sich folgende Situation vor: Mehrere Geschäftsleute treffen sich in einem Konferenzraum und möchten Daten zwischen verschiedenen mobilen Geräten austauschen. Wegen der zunehmenden Anzahl und Komplexität der Kommunikationstechnologien wie Infrarot, Bluetooth, WLAN oder gar Ethernet wird es für gewöhnliche Benutzer immer schwieriger, unterschiedliche Geräte zu verbinden.
Falls eine Verbindung zwischen zwei Geräten hergestellt wird, sollte diese oft auch sicher sein. Die mobilen Geräte müssen dazu über gemeinsame Sicherheitsparameter verfügen. Ein Verschlüsselungsalgorithmus und ein Schlüssel gehören zu jeder Sicherheitsrelation. Für zusätzliche Sicherheitsmassnahmen müssen weitere Informationen wie Authentifizierungsmechanismus, Authentifizierungsschlüssel usw. ausgehandelt werden. Es muss also immer zuerst ein „Handshake" Verfahren zwischen beiden Geräten durchgeführt werden, um einen sicheren und verschlüsselten Kommunikationskanal aufbauen zu können. Dieser Informationsaustausch sollte seinerseits auch wieder sicher und verschlüsselt sein. Heutzutage geschieht dies häufig durch persönlichen Austausch der sensiblen Informationen, sei es mündlich oder schriftlich.

Um sichere Verbindungen aller Art zwischen fremden Geräten aufbauen zu können, werden grundsätzlich Konfigurations- und Sicherheitsparameter benötigt. Die Konfigurationsparameter werden wegen der immer grösser werdenden Vielfalt an verschiedenen und miteinander nicht kompatiblen Technologien und Standards wichtiger denn je. Auch wenn die Geräte zweier Geschäftsleute durchaus die Fähigkeiten hätten, eine gesicherte Verbindung aufzubauen, z.B. mittels Bluetooth oder WLAN, müssen diese gemeinsamen Technologien richtig konfiguriert werden. Zu solchen Konfigurationsparametern zählen bei Bluetooth unter anderem die Adresse oder der Dienst (Bluetooth Profile) und bei WLAN die Netzkennung (Service Set Identifier). Das richtige Einstellen dieser Parameter erlaubt zunächst eine ungesicherte Kommunikation mittels der gewählten Technologie. Um diese zu sichern, müssen Sicherheitsparameter wie die zu verwendenden Schlüssel und Verschlüsselungsalgorithmen ausgehandelt werden. Genau hier treten die grössten Probleme auf. Die Art und Weise des Aushandelns und Verteilens der Sicherheitsparameter bestimmt in der Praxis die effektive Stufe der erreichten Sicherheit. So nützt der beste Verschlüsselungsalgorithmus nichts, wenn die verwendeten Schlüssel für jeden zugänglich sind. Ein kurzer Blick auf den Bildschirm oder die Tastatur des Benutzers, während dieser gerade den Schlüssel eingibt, erlaubt jedem Fremden das Abhören der vermeintlich sicheren Kommunikation. Um genau diesen Austausch von Konfigurations- und Sicherheitsparametern sicher und möglichst benutzerfreundlich durchzuführen, wurde CAHN (Cellular Assisted Heterogeneous Networking) entwickelt.

CAHN ist ein Ansatz für die Organisation von verschiedenen mobilen Geräten in einer heterogenen Umgebung. CAHN trennt die Datenebene von der Signalisierungsebene, um den automatischen Aufbau von sicheren heterogenen Verbindungen zwischen Geräten zu ermöglichen. Dabei wird ein zelluläres Netz als gemeinsamer Kommunikationskanal zwischen den einzelnen Benutzern genutzt. Als Kommunikationskanal für den Transport von Konfigurations- und Sicherheitsparametern wird das zelluläre GSM Netz benutzt, welches sich durch hohe Verfügbarkeit und grosse Abdeckung auszeichnet und durch einen zellulären Provider zur

Verfügung gestellt wird. Die bestehende Vertrauensbeziehung zwischen dem Benutzer und dem Operator eines zellulären Netzes kann als Ausgangspunkt für die Authentifizierung (Subscriber Identity Module) in CAHN Applikationen wieder verwendet werden. So erfolgt dann genau dieser Austausch von Signalisierungsnachrichten zwischen zwei mobilen Geräten, während die spätere Datenkommunikation direkt über das lokal verfügbare Netz, z. B. Bluetooth oder WLAN erfolgt. Mit Hilfe des CAHN Systems ist es möglich, eine sichere Verbindung per Mobiletelefon aufzubauen.

CAHN bezweckt verschiedene mobile Technologien als Transportmechanismus für die Datenebene zu benutzen. Dies gibt den Anwendern die Möglichkeit, Daten und Dienste gemeinsam zu nutzen. Zu diesem Zweck beabsichtigt CAHN die Erweiterung der Datenebene auf eine bereite Auswahl der heutigen Technologien wie Bluetooth und WLAN. Diese Diplomarbeit wird sich mit der Definition eines neuen CAHN Protokolls für WLAN Technology beschäftigen. Die Protokoll Engineering Grundlagen dienen als Fundament für den Aufbau eines Kommunikationsprotokolls. Sie ermöglichen die Analyse und die Verbesserung des vorhandenen CAHN Bluetooth Protokolls und unterstützen somit die Entwicklung von weiteren Protokollen für andere Kommunikationstechnologien wie WLAN.
Das Ziel dieser Diplomarbeit wird die Integration des CAHN Ansatzes und der WLAN Technologie sein. Um diese Integration zu verwirklichen wurde ein Lösungsansatz basierend auf dem Bloom Filter Konzept vorgeschlagen. Dieser Ansatz benutzt das mathematische Konzept des Bloom Filters um CAHN spezifische Information (MSISDN) zu reduzieren und die CAHN Fähigkeit bereitzustellen.

Mit Hilfe solcher kombinierten Lösungen könnten in naher Zukunft die Vorteile völliger Mobilität mit den Vorteilen einer festen Büroinfrastruktur vereint werden. Die verschiedenen Arbeitsumgebungen würden somit zu intelligenten Umgebungen erweitert, von wo aus alle nötigen Ressourcen permanent verfügbar wären.

ACKNOWLEDGEMENTS

First of all, I owe my cordial thanks to my parents, Abbas and Mansoure for their infinite encouragement and invaluable guidance during all these years.
My special thanks go to Prof. Dr. Torsten Braun for coordinating this diploma work. I would like to thank Marc Danzeisen who supervised me with patience and great help.
Furthermore, I want to acknowledge Swisscom AG for the valuable assistance that I had during my diploma work.
I would like to express my sincere gratitude to Simon Winiker for his fruitful support and companionate friendship during the last year.

CONTENTS

GLOSSARY

AP	Access Point
ARP	Address Resolution Protocol
ASN	Abstract Syntax Notation
BSS	Basic Service Set
BSSID	Basic Service Set Identifier
CAHN	Cellular Assisted Heterogeneous Networking
CAHND	Cellular Assisted Heterogeneous Networking Daemon
CCM	CAHN Communication Manager
DHCP	Dynamic Host Configuration Protocol
ETSI	European Telecommunication Standardization Institute
FACCH	Fast Associated Control Channel
FDT	Formal Description Technique
GSM	Global System for Mobile Communication
HLR	Home Location Register
HPLMN	Home Public Land Mobile Network
IP	Internet Protocol
LOTOS	Language for Temporal Ordering Specification
LNP	Local Number Portability
MSC	Main System Controller
MSISDN	Mobile Subscriber ISDN
PAN	Personal Area Network
PDU	Protocol Data Unit
PROMELA	Process Meta Language
PSTN	Public Switched Telephone Network

SACCH	Slow Associated Control Channel
SAP	Service Access Point
SDL	Specification and Description Language
SHA	Secure Hashing Algorithm
SIM	Subscriber Identity Module
SMS	Short Message Service
SPIN	Simple PROMELA Interpreter
SS7	Signalling System Number 7
SSID	Service Set Identity
TA	Terminal Adapter
TE	Terminal Equipment
TCAP	Transaction Capability Application Part
WEP	Wired Equivalent Privacy
WLAN	Wireless LAN
UDCP	USSD Dialogue Control Protocol
USSD	Unstructured Supplementary Service Data
VPLMN	Visitor Public Land Mobile Network
VLR	Visitor Location Register
XBC	Xspin Bar Chart

1 ABOUT THIS DIPLOMA WORK

Cellular assisted heterogeneous networking (CAHN) will engage a wide range of today technologies to a compact and user friendly tool. In the past years, many new technologies like Bluetooth, WLAN, GSM and Infrared have been developed and each technology needs its own specific configuration. While these technologies becoming more and more popular, suitable solutions for an easy and fast handover for configuration are needed. For example, different security and authentication mechanisms require different configuration information. Merging the configuration information of those technologies into one simple configuration is a key motivation for this diploma work. The main concern of this work is the definition of a new communication protocol with respect to CAHN. To be able to use CAHN with different kind of technologies, appropriate protocols have to be designed.

In heterogeneous environments, the setup of a connection between two nodes can be quite complicated. Nowadays, there are many different access technologies and as a result, the setup of the network quickly gets confusing. CAHN builds a framework for heterogeneous networking which covers diverse technologies and helps simplifying the access process. CAHN suggests using a cellular network to assist the users in a heterogeneous network. The cellular network thus serves as a channel which can be used for the signalling parameters. The exchange of the needed parameters for a connection establishment and the negotiation of the needed parameters to create a secured connection can be handled over such a channel. The typically high coverage of a cellular network makes it very valuable for the use as a signalling channel. The operator is able to locate the nodes accurately and easily. The operator will serve as a signalling and configuration provider for heterogeneous networks. He can locate the peers, authenticate them and offers a configuration for the secured connection establishment.

In a possible target scenario, CAHN can enable networking among nodes with different heterogeneous access technologies and it can assist in the establishment of secured connections among the participating nodes. The presence of different technologies and the increase of complexity in heterogeneous environments make the connection establishment among their devices difficult for the users.

There is already an existing implementation of the CAHN concept which adapts Bluetooth technology and establishes a secure link between nodes. The CAHN Bluetooth protocol [5] is the first CAHN protocol which uses the GSM network as the signalling channel to exchange Bluetooth specific parameters. The data transfer occurs over a secure Bluetooth connection.

This diploma work tries to expand the CAHN protocol to use the WLAN technology. The CAHN WLAN protocol should be able to build a secure WLAN connection (ad hoc and access point) between nodes. In analogy with the CAHN Bluetooth protocol, it is necessary to boast two channels to be able to exchange the WLAN specific parameters and data. Consequently, the WLAN is best for the data transfer and the GSM network would provide the signalling channel.

For the purpose of CAHN, the WLAN architecture and services were analyzed. The results have shown that porting the WLAN technology for CAHN can not be done in a way similar to the CAHN Bluetooth protocol. The main problem using the WLAN technology is the lack of an existing service discovery protocol (SDP) as in Bluetooth. SDP is a basic strategy to provide a dynamic environment for service sharing and access among nodes. The main components of a SDP are the service catalogue server, the service server and the client. The service catalogue server contains information about the available services in the network. The service server can register information about the service it provides on the service catalogue server. A client can then query the catalogue server for available services and gather information on how to contact the service server to access the provided service. This means that a client can look for possible services in its environment and choose the desired service.

In WLAN, there is no mechanism to figure out what kind of services a WLAN node offers. Consequently, a client node does not have the possibility to find out which nodes present the CAHN services. Thus, the CAHN WLAN protocol has to provide this requirement and signalizes the CAHN ability to client nodes. Based on this concept, this diploma work will suggest some approaches to solve this problem. The Bloom approach is a very promising solution that will be deeply analyzed in this work.

The following section gives an overview of all subjects treated in this diploma work and explains briefly the topic of each chapter. This diploma thesis contains three main topics that follow these concepts:

1. Protocol engineering of the existing CAHN Bluetooth protocol and the reuse of the existing components of the CAHN Bluetooth protocol for the definition of the CAHN WLAN protocol.

2. Design of the CAHN WLAN protocol based on the analogies to the CAHN Bluetooth protocol and the implementation of the designed protocol.

3. Evaluation and testing of the new protocol and suggestions for refining the protocol performance.

1.2 DOCUMENT STRUCTURE

Chapter 1 gives an overview of this diploma work.

Chapter 2 contains all basics for this diploma work. Chapter 2 presents an overview of protocol engineering and explains the basic idea of the formal description of a protocol which will be the basic for a formal description of the existing CAHN protocol. This chapter also contains an introduction to the USSD technology and discuses the usage of USSD for CAHN as an alternative solution to the SMS. Furthermore, chapter 2 presents an overview of the WLAN technology and the Bloom filter concept.

Chapter 3 gives a formal description of the CAHN Bluetooth protocol written in PROMELA which is based on the latest version of CAHN Bluetooth protocol.

Chapter 4 contains an overview of CAHN USSD module as a transport medium for CAHN messages over the GSM network.

Chapter 5 introduces a first draft of the CAHN WLAN protocol and defines the protocol messages and the corresponding components. Beside the protocol, some serious issues to enable WLAN for CAHN are treated and solutions about how to deal with them are suggested.

Chapter 6 covers the implementation of the CAHN WLAN protocol.

Chapter 7 evaluates the implementation of the CAHN WLAN protocol and presents some performance results.

Chapter 8 suggests future work and concludes this diploma work.

2 RELATED WORK

This Chapter describes all the work related to this diploma thesis. These related works are separated into two main groups. The first part covers all "non CAHN" related work and the second part contains the existing CAHN protocol and components.

The following sections 2.1, 2.2, 2.3 and 2.4 will present all necessary non CAHN work which influenced this diploma work. These sections give an overview of different topics like protocol engineering, PROMELA, USSD, WLAN and the Bloom filter concept.

2.1 Protocol engineering

Communication architectures and their most important part, which are protocols, are larger and more complicated than ever. Communication protocols are the fundament for the realization of todays telecommunication nets. Protocols define rules and principles for communication systems and describe how communication systems interact with each other. To achieve this, protocols need a collection of various functions and rules, like for sending and receiving data, for security and authentication and error handling. For example, layered protocols offer services to the next higher layer and these services are well defined and specified. Protocol and service specifications are used in formal verification and validation activities. From a given protocol and service specification an efficient implementation can be automatically achieved.

Protocol design is related to several subjects such as operating systems, computer networks, and data transmission. However, the designing of a protocol whose correctness can be proved is a challenge and sometimes, a desperate task. Communication software is a special kind of software developed to be used in telecommunication or computer networks to provide a set of well defined services. This software is allocated in different network nodes, creating the need for protocols that make the interaction among the different components easy. Protocols provide their services coordinating and synchronizing distributed components. Although the main protocols used in computer networks have been standardised by international organisations [7]. The software that implements them is difficult to design and to test because of:

- Standards making use of informal text in formal descriptions.
- Formal descriptions are not adequately validated.
- Standards do not include some details that depend on the machine where protocols are implemented.

The application of formal methods and software engineering methodologies to the protocol design has led to a new area called protocol engineering [8]. The domain of protocol engineering includes all of the activities related to the protocol design such as service and protocol specification, validation, verification, implementation and testing. Protocol engineering allows achieving efficient and reliable communication software.

A Formal Description Technique (FDT) is chosen when a protocol has to be specified in a formal way. Although there are many FDTs, five of them are highlighted here, as they are the most popular. These are ESTELLE [9] (Extended State Transition Language), SDL [10] (Specification and Description Language), ASN.1 [11] (Abstract Syntax Notation One), MSC [12] (Message Sequence Charts), based on extended finite state machines and LOTOS [13] (Language for Temporal Ordering Specification), based on process algebra and process calculation.

Every one of mentioned FDTs is an international standard and the selection of one or the other depends on the specific needs of the user or designer. With other words, No FDT satisfactory fulfils all requirements.

For this diploma work, a prosa language similar to PASCAL [14] is chosen for the textual notation of the CAHN Bluetooth protocol. It contains very basic functionality to describe a protocol and its messages. It fulfils all the needed features to express the main structure of a given protocol.

Formal description

There are 3 language levels [14] to specify the whole structure of a protocol. Each level explains the functionality of a certain range in the protocol. The specification of a protocol contains all services and their behaviour which describe the correlation between each protocol part. The three needed specification languages are:

- Service description (Level S)
- Protocol description (Level P)
- Mapping description (Level M)

Each layer performs a well defined function according to a set of well defined rules. This is done by exchanging messages between layers. The messages contain both data and control information. These messages are referred to as Protocol Data Units (PDU) and the set of rules is referred to as a protocol [15]. The interface between each layer is well defined, with the layer above depending on the services provided by the layers below. These layers exchange Abstract Service Primitives (ASP) as a mean of specifying and using the services of the layer below. Therefore each layer communicates a peer layer according to the defined protocol.

Services

Telecommunication systems provide communication services. The services make possible that data and programs can easily be shared and changed among the users. There are two types of services: symmetric and asymmetric. Asymmetric services are based on the client/server principle. The client asks the server about a service and the server accepts or rejects the request. On the other hand symmetric servers provide their services at the same time on two or more “service access points”. Service users may introduce or receive service primitives (SP) on the service access points (SAP).

The level S language describes the protocol services by characterising them on the service access points.

Protocol

The second level language characterizes the communication mechanism of a protocol. This level specifies the features of a given protocol. Protocols are either connection oriented or connectionless. Entities are active objects of the service provider. Entities serve the service access points. A service access point has always just one entity but an entity can serve more than one service access points. If two protocol entities have the same capabilities, the protocol is symmetric, otherwise it is asymmetric. The protocol entities interact through messages with their environment. For example, protocol entities communicate by exchanging protocol data units (PDU). In a protocol specification, PDUs need to be defined exactly. This is because the PDUs are exchanged in heterogeneous environments.

The level P language describes the core of the protocol. The mapping of protocol information into transmitted data must be defined precisely in a protocol specification.

Mapping

Protocol layer information can not be translated directly from one layer into a lower layer. The question is how to make a logic connection between two entities or two protocol layers. The protocol information is translated incrementally through a series of protocol layers. Each layer contains the same amount of information as the layer above but in a form closer to the form that is ready to be sent through the net. This activity of translating from one layer to another is called mapping.

The level M language describes the mapping mechanisms in a protocol.

PROMELA and SPIN

Without suitable tools it may be possible to design a protocol but in most cases it will be impossible to prove its correctness. SPIN (Simple PROMELA interpreter) [17] is an accepted software tool based on the usage of formal validation models written in PROMELA that can be used for the formal verification of distributed software systems and protocols. The language allows dynamic creation of concurrent processes. XSPIN is a graphical program to control SPIN which is written in Tcl/Tk and was used for a better interaction with SPIN in this implementation.

Given a model system specified in PROMELA, SPIN can perform random or interactive simulations of the system's execution or it can generate a C program that performs a verification of the system state space. During simulations and verifications, SPIN checks for the absence of deadlocks, unspecified, receptions, and unexecutable code. The verifier can also be used to prove the correctness of system invariants and it can detect non-progress execution cycles. The verifier is designed to be fast and to use a minimal amount of memory. PROMELA programs consist of processes, message channels and variables. Processes are global objects. Message channels and variables can be declared either globally or locally within a process. Processes specify behaviour, channels and global variables define the environment in which the processes run. The syntax of PROMELA is similar to C.

2.2 Bloom filters

A Bloom filter [6] is a space efficient, probabilistic algorithm that is used to quickly test the membership of an entity in a large set of data. The space efficiency is achieved at the cost of a non zero probability of error that is called "*false positive*" and it means the Bloom filter suggests that an element is in the set of data but it is not [18]. The false positive plays an important role for CAHN and therefore, it is precisely defined and analysed in the chapter 5.

Bloom filters use multiple hash functions to reduce the volume of given information and make it possible that all information match into a single array of bits. The Bloom filter is widely used in many applications which take advantage of its ability to compactly represent a set of data and filter out effectively any element that does not belong to the set of data. All of the hash functions in a Bloom filter are configured in a way that their range matches to the length of the array or bit vector. For many applications, the probability of a false positive can be made sufficiently small and the saved space is significant enough that Bloom filters are useful. Bloom filters use one way hashing to store their data, consequently, it is impossible to reconstruct the list of keys in a filter without doing a complete search of the key space. Therefore, Bloom filters make it possible to share information about existing members without broadcasting a complete list of all members to the other stations. For that reason, they may be especially important in ad hoc and peer to peer applications, where size and privacy are heavily limited.

In fact, Bloom filters have a great deal of potential for distributed protocols where systems need to share information about what data they have and about the amount of members available without having out sensitive information.

Mathematical preliminaries

A Bloom filter is a method for representing a set $\mathrm{A} = \{a_1, a_2, a_3, ..., a_n\}$ of n elements (also called keys) to support membership queries in a m bits array [19]. The idea is to allocate a vector v of m bits, initially all set to 0, and then choose k independent hash functions $h_1, h_2, h_3, \ldots, h_k$, each of them with the range $\{1, \ldots, m\}$. This means that the ranges of all chosen hash functions are smaller than the size of the vector or precisely as big as the size of the allocated array. For this purpose, the natural assumption has been made that these hash functions map each element to a random number uniform over the range $\{1, \ldots, m\}$. For mathematical convenience:

$$h_j : a_i \mapsto \{1, \ldots m\} \quad | \quad 1 \le j \le k, 1 \le i \le n$$

It is important to use a high quality hash function in the Bloom filter to guarantee that the hashed values are equally distributed over all possible values. Relevant to CAHN and the size of the SSID, if the SSID field vector is 256 bits long (depending on product), the hash functions would have to return a value between 1 and 256.

For each element (e.g. key) $a \in A$, the bits at positions $h_1(a),\ h_2(a),\ h_3(a),\dots,h_k(a)$ in v are set to 1. A particular bit might be set to 1 multiple times, but only the first change has an effect on the Bloom filter (OR):

$$\left\{(v_1,v_2,\dots,v_m)\ ,\ h_j(a_i)=s \rightarrow v_s=1 \mid \forall a_i \in A,\ \forall h_j,\ 1\le i\le n,\ 1\le j\le k,\ 1\le s\le m\right\}$$

Formula 1: The Bloom filer theory.

To test if a key x is in A, the bits at positions $h_1(x), h_2(x), h_3(x), \dots, h_k(x)$ have to be checked whether $h_j(x)$ are set to 1 for $1 \le j \le k$. If any of them is 0, then certainly x is not an element of the set A.

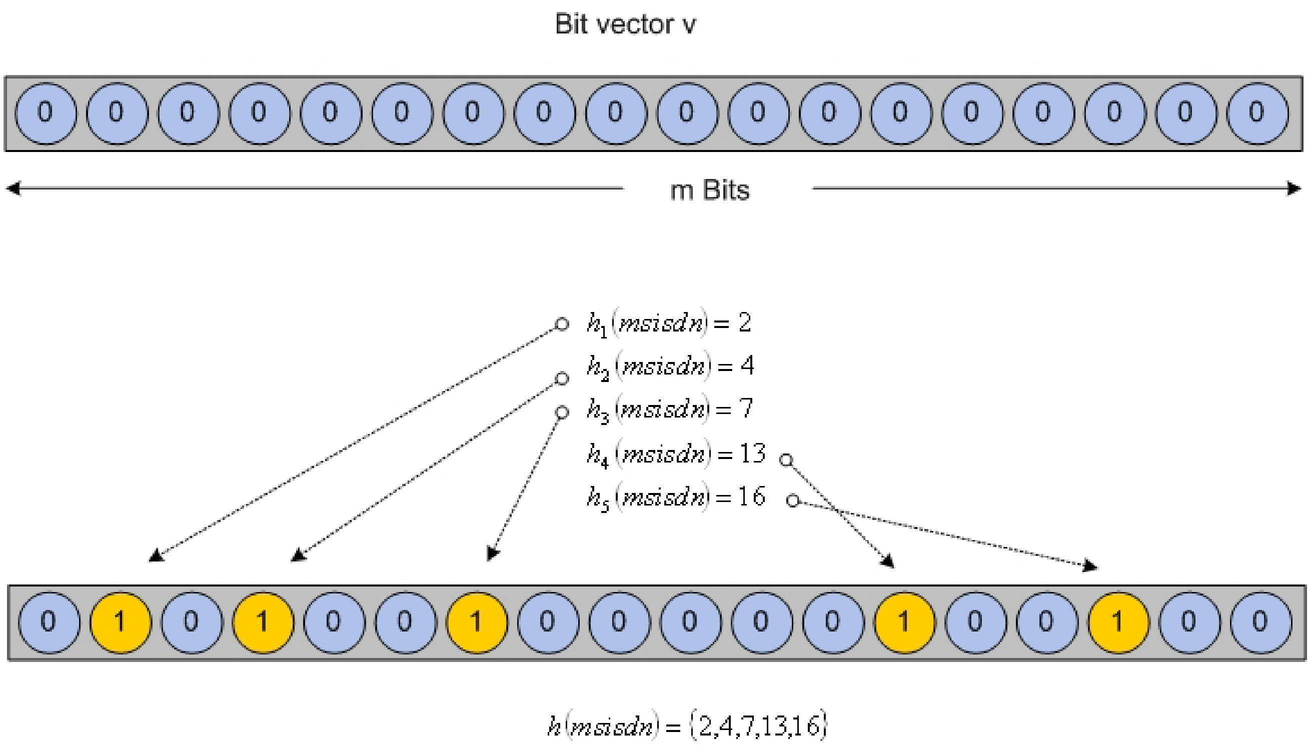


Figure 1: An example of a Bloom filter with 5 hash functions and a vector of 18 bits that is set to 0 as default. The hashed values of the item "msisdn" are set to 1. To check if an item x belongs to vector v, all the bits of the hashed values for x must be checked in the vector v. If all 5 bits are set to 1, the item x is in the vector v or it is a false positive.

Otherwise if all bits $h_j(x)$ are set to 1, it can be assumed that x is in the set A. However, there is a certain probability that this assumption is wrong. This is called a "*false positive*" or for historical reasons, a "*false drop*". The parameters k and m should be chosen such that the probability of a false positive is acceptable.

For many applications, false positives may be acceptable as long as their probability is sufficiently small and controllable. For CAHN this is not a crucial issue, as even if a node is considered accidentally to be a member of a WLAN ad hoc network. The worst consequence consists in doing a useless CAHN setup message exchange and it can not be connected over the WLAN.

There is a trade off between the Bloom parameters and the probability of a false positive. After inserting n keys into an array of bits with size m, the probability of having a particular bit not set is:

$$\left(1-\frac{1}{m}\right)^{kn} \approx e^{-kn/m}$$

And thus, the probability of a false positive with respect to the number of hash functions, the number of elements and the size of the array, is:

$$\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1-e^{-kn/m}\right)^{k}$$

In other words there are just three main parameters which can affect the performance of a Bloom filter:

- Number of hash functions k
- Size of vector m
- Possibility of error $\left(1-e^{-kn/m}\right)^{k}$

By given n and m, the number of hash functions k can be optimized and thus, the false positive rate $\left(1-e^{-kn/m}\right)^{k}$ can be minimized. Using more hash functions increases the chance to find a 0 bit for a key that does not belong to the set A. It means the possibility of distinguishing between more keys is raised by using more hash functions.

But if too many hash functions are used, the Bloom filter will be very dense and it increases the probability of collisions and of course by using fewer hash functions, there will be not enough discrimination between keys. The optimal number of hash functions that minimizes the false positive rate as a function of k can be calculated as following:

$$\alpha = \left(1 - e^{-kn/m}\right)^k = e^{k\ln\left(1-e^{-kn/m}\right)}$$

Formula 2: False positive.

Minimizing the probability of false positive $\left(1 - e^{-kn/m}\right)^k$ is equivalent to optimizing $k\ln\left(1-e^{-kn/m}\right)$ with respect to k:

$$\frac{\partial}{\partial k} k\ln\left(1-e^{-kn/m}\right) = \ln\left(1-e^{-kn/m}\right) + \frac{kn}{m}\frac{e^{\frac{-kn}{m}}}{1-e^{\frac{-kn}{m}}}$$

The function is minimized if the value of the derivate is 0, in this case $k = \frac{m}{n}\ln 2$:

$$\left(1 - e^{-kn/m}\right)^k = \left(1 - e^{-\ln 2}\right)^{\frac{m}{n}\ln 2} = \left(\frac{1}{2}\right)^{\frac{m}{n}\ln 2} = (0.6185)^{m/n}$$

Therefore the false positive function can be optimized when this value is used. In fact, k must be an integer and in practice the k value is less than optimal in order to reduce computational overhead. The appendixes A, B and C present some examples of false positives under variation of m, n and k.

Hashing for Bloom filter

Before choosing appropriate values for k and m (chapter 5), hash functions have to be analysed. The following paragraph shortly explains what a hash function is and which hash method could be used for CAHN.

A hash function h is a transformation that takes a variable input s and returns a fixed size string, which is called the hash value. Hash functions with just this property have a variety of general computational uses, but when used in cryptography the hash functions are usually chosen to have some additional properties. The basic requirements for a cryptographic hash function are:

- The input can be of any length but the output has a fixed length
- $h(x)$ is relatively easy to compute for any given x
- $h(x)$ is one way and $h(x)$ is collision free

A hash function h is a one way hash function if it is impossible to invert, where "impossible to invert" means that given a hash value h, it is computationally infeasible to find some input x such that $h = h(x)$. Examples of well known hash functions are Message Digest 2 (MD2), Message Digest 5 (MD5) and Secure Hash Algorithm (SHA).

For the purpose of CAHN, the choice of an appropriate hash functions is an important task. The hash function is required to deliver a unique hashed value that can be calculated again at any time also by any station, because this value plays a decisive role by recognizing and authenticating CAHN stations. Hence, using variations of classic pseudo random number generators as a hash function for the CAHN is not qualified for this purpose. There are two famous hash functions that are suitable and fulfil the requirements of CAHN. Using the Data Encryption Standard (DES) as a hash made the Bloom filter work at its theoretical efficiency. Another recommendable hash function is the Secure Hash Algorithm (SHA) for it is faster and specifically designed with such requirements in mind. The next section explains what SHA is and how it will be used for the CAHN WLAN concept.

Secure hash algorithm (SHA)

The Secure Hash Algorithm (SHA) algorithm takes an input value of less than 264 bits in length and produces a unique 160 bit message digest. The main advantage of the SHA algorithm is the larger message digest that makes it more secure against brute force collision and inversion attacks. A brute force attack consists of trying every possible key, combination, or password until the right one is found. Statistically a brute force attack will always succeed however brute force attacks against systems with satisfactorily long key sizes may require many years to complete.

Bloom filter features

If the length of a vector v is not a power of two, then the length of vector v can be enlarged virtually to the next larger power of two. The virtual additional bits are all one, but not actually stored. Whenever a bit index falls outside the vector length, the Bloom filter acts as if it would be 1 and ignores to set to those locations.

Another feature is that Bloom filters can easily be halved in size. Supposed the size of the filter is a power of two. The size of the Bloom filter can be halved by "OR" the first and second halves together. When hashing, the high order bit can be masked [20].

The next feature of Bloom is the "oring" of filters. Supposed there are two Bloom filters representing two sets of data with the same number of bits (vector v) and using the same number of hash functions. Then a Bloom filter, that represents the union of these two sets, can be achieved by oring the two bit vectors of the original Bloom filters. With respect to CAHN, this feature of Bloom can be used when not all 265 bits of the SSID field are reserved for the Bloom filter.

Counting Bloom filter

Inserting elements into a Bloom filter is easy. Each element is hashed k times and the corresponding bits are set to 1. But there is no way to delete an element in the Bloom filter. If the corresponding bits of an element are set to 0, it may be setting a position to 0 that is hashed by some other element in the set. In this case, the Bloom filter no longer correctly reflects all keys in the set. With respect to CAHN, this would mean that when a station leaves the CAHN ad hoc network, the SSID of network stays unmodified and the other stations in the network do not realise the absence of this station.

There is another realisation of Bloom filters (counting Bloom filter) [21] that is thought to avoid this problem. Each entry in the Bloom filter is not a single bit but instead a small counter. When an element is inserted, the corresponding counters are incremented and when an element is deleted, the corresponding counters are decremented. To avoid counter overflow, the size of the counter must be chosen correctly. After many deletions this might lead to a situation where the Bloom filter allows a *false negative* (the count becomes 0 when it should not be), but the probability of such a chain of events is so low that it is very unlikely to happen. Analysis reveals that 4 bits per counter should be enough for most applications.

2.3 USSD

Unstructured Supplementary Service Data (USSD) [22] is a GSM service that allows high speed interactive communication between subscribers and applications across the signalling channels of the GSM network. USSD is a technology unique to GSM. The GSM standard includes a wide range of supplementary services (call barring and call forwarding). The services can be managed by entering a specific text string. For example, by entering the text string "***123*1234567# SEND*", all incoming calls, will be rerouted to the number 1234567. However, most mobile station manufacturers provide more user friendly methods to do this. Normally when standard GSM services are managed from the mobile station structured, functional signalling is available. For example, if the user activates the supplementary service "call forwarding", the mobile station recognizes this and invokes a standard signalling procedure to the network. This is because the call forwarding supplementary service is part of the GSM standard [23]. To support old mobiles and operator specific services, the Unstructured Supplementary Service Data mechanism was introduced to the GSM standard. Further standardization allowed the network to send USSD operations to the mobile station, as well as combining mobile and network initiated operations in order to exchange data in a dialogue manner. The content of the messages sent from the network pops up on the mobile station display. Operations as well as messages sent from the mobile station can be routed to an operator provided application in the network. Thus, USSD can be used as a transparent pipe through the GSM network. The most important features of USSD are:

- USSD can be used by operators to provide operator specific services
- USSD is session oriented, unlike SMS, which is a store and forward oriented technology
- Message exchange times for interactive applications are shorter for USSD compared to SMS because of the session based feature of USSD
- USSD commands are transferred over the Fast Associated Control Channel (FACCH), which is faster than the Slow Associated Control Channel (SACCH) that is used for SMS
- It is not possible to bill for USSD directly, but instead, it is possible to bill for the applications which are associated with the use of USSD

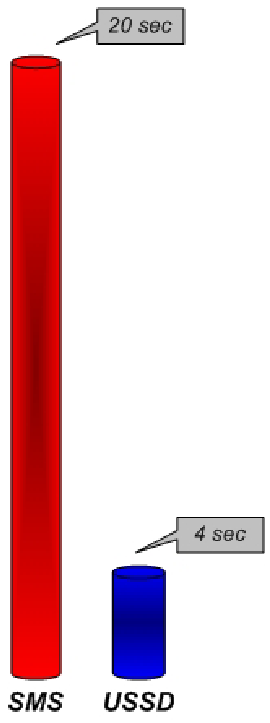


Figure 2: Required time for a payment in a vending machine [24] using both technologies.

- Users do not need to access any particular phone menu to use USSD services. They can enter the unstructured supplementary services data command directly from the mobile phone screen

- USSD commands are routed back to the home mobile network

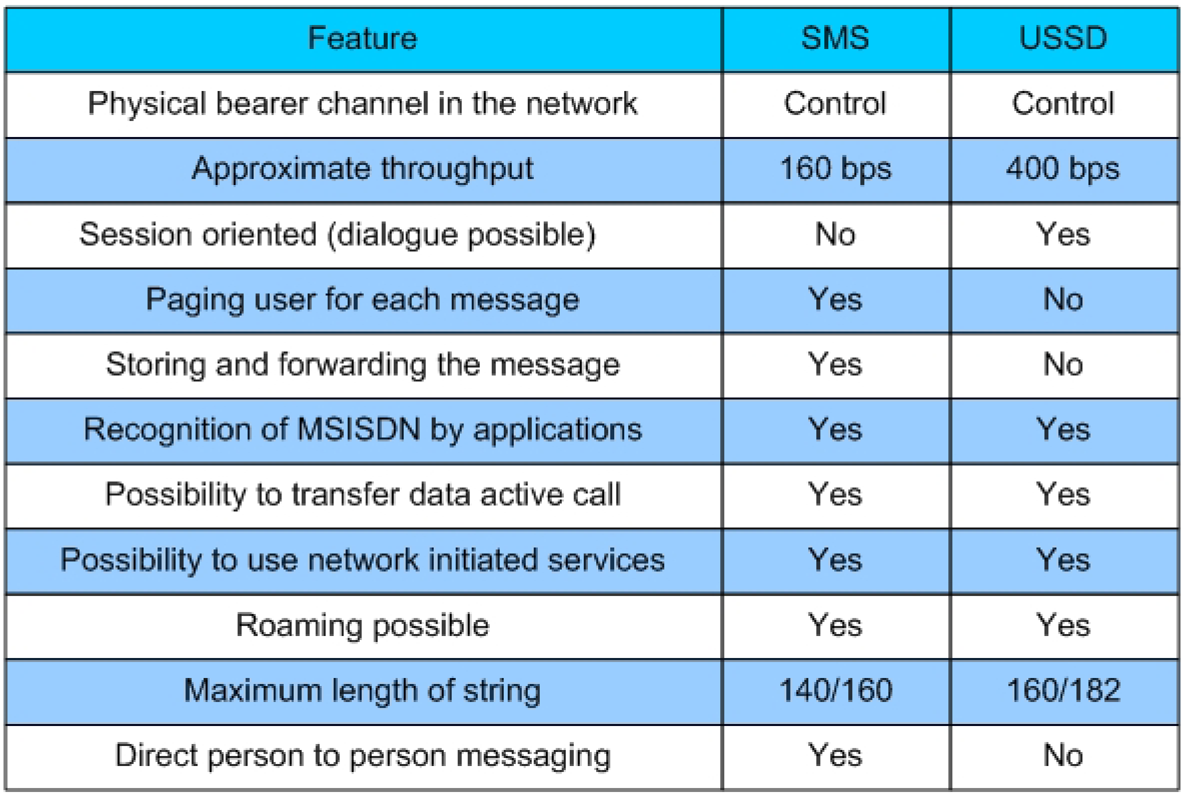
- USSD works on all existing GSM mobile phones (theoretically). Some old mobile phones are only able to send USSD messages

- Both SIM Application Toolkit and the Wireless Application Protocol support USSD

- USSD can be used as a transparent bearer through the GSM network

- Creating and sending an USSD message is as easy as making a call. Indeed, most mobiles allow to store USSD strings under quick dial keys. Creating an USSD message can be easier than creating a SMS

- USSD messages are very flexible in both length and content. USSD makes use of all the digits plus the * and # characters



Feature	SMS	USSD
Physical bearer channel in the network	Control	Control
Approximate throughput	160 bps	400 bps
Session oriented (dialogue possible)	No	Yes
Paging user for each message	Yes	No
Storing and forwarding the message	Yes	No
Recognition of MSISDN by applications	Yes	Yes
Possibility to transfer data active call	Yes	Yes
Possibility to use network initiated services	Yes	Yes
Roaming possible	Yes	Yes
Maximum length of string	140/160	160/182
Direct person to person messaging	Yes	No

Figure 3: USSD vs. SMS.

Unstructured Supplementary Service Data STANDARD

This section contains a brief overview of different standardisations of USSD technology in the past years. In fact, there are three versions of USSD for GSM: USSD in phase 1, USSD in phase 2 and enhanced USSD [25].

Phase 1

In this phase, only mobile initiated operations are supported. Network initiated operations are not supported. This means that the mobile station can send a request to the network and receive a response. There is no dialogue mechanism between the mobile and the GSM network.

Phase 2

In phase 2 of USSD, network initiated operations and dialogues are also supported. An USSD dialogue can be established between the mobile and the network. Multiple subsequent USSD operations can be sent within the dialogue. Phase 2 is the present status of the USSD standard.

Enhanced USSD

The enhanced USSD, also known as USSD phase 2+, is an improved version of USSD phase 2 which extends the usage and efficiency of USSD network functions:

- Support for multiple dialogues
- Adapting the User Data Header concept from SMS

USSD ATTRIBUTES

The following sections describe USSD specific characteristics and parameters. The descriptions are relevant with USSD phase 2. There are two types of USSD dialogues which are used in USSD phase 2: Mobile initiated and network initiated.

Mobile initiated dialogue

The mobile station initiates the communication (dialogue) by raising the "*ProcessUSSDRequest*" operation. The network can respond by either invoking an "*USSDRequest*" operation or release the dialogue by returning the result to the received "*ProcessUSSDRequest*" operation. Both, the mobile station and the network can release the dialogue at any time by sending a "RELEASE COMPLETE" message (figure 4).

Network initiated dialogue

The network initiates a dialogue by invoking the "*USSDRequest*" operation. The mobile station responds by returning the result to the "*USSDRequest*" operation. Both, the mobile station and the network can release the dialogue at any time by sending a "RELEASE COMPLETE" message.

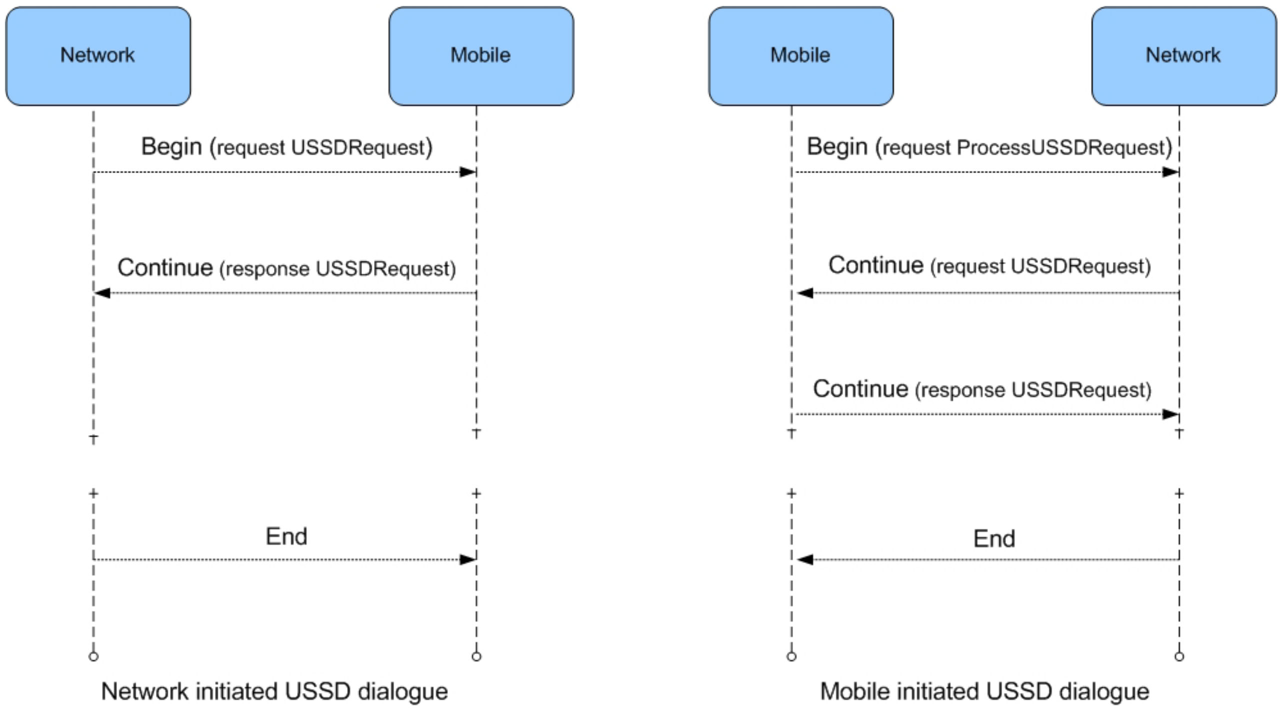

Network
Mobile
Begin (request USSDRequest)
Continue (response USSDRequest)
End
Network initiated USSD dialogue
Mobile
Network
Begin (request ProcessUSSDRequest)
Continue (request USSDRequest)
Continue (response USSDRequest)
End
Mobile initiated USSD dialogue

Figure 4: USSD dialogues.

Service code

Each service in the USSD is recognized by a specific sequence of digits. This number is an unique identifier for an USSD service among all other services. The service code for the initially proposed use of USSD, as a mechanism to manage operator specific supplementary services is: "*#SC*<SI>#"

- A start indicator of between 1~3 characters selected from * and #
- A service code (SC) of 2 or 3 digits between 50 and 150
- A separator character: *
- Supplementary information (SI) of variable length
- # to signal the end of message

The service code is part of the first USSD string sent from the mobile station. It acts as a leader that guides the string to the end node. The end node depends on the value of the service code. The end node can be "Home Location Register" (HLR), "Main System Controller" (MSC) or "Visitor Location Register" (VLR). Once an USSD dialogue is established between the mobile station and the application which runs on the end node, a transparent pipe is opened through the network. Thus, the service code is not needed during the remaining part of the dialogue. The GSM specification recognises two types of service codes for the USSD. "Home Public Land Mobile Network" (HPLMN) service codes always route the USSD string to the HLR. However, "Visitor Public Land Mobile Network" (VPLMN) service codes route the string to the "Visitor Location Register" or to the MSC. To be able to use USSD technology from outside the GSM network, an USSD gateway, which is located inside the HLR, has to relay the USSD messages to an external node. A specific external application provides services to the USSD users which can be located anywhere outside the GSM network. The USSD gateway has to provide the functionality to relay an USSD service code to a certain external node address. Therefore, the USSD gateway has to extract service code from the USSD string and checks whether this service code belongs to an external USSD application.

Multi dialogue

In USSD, there is no possibility to setup a multi dialogue. In the USSD GSM phase 2 specification, only one dialogue between a mobile station and the network is allowed. If the mobile station receives a dialogue request while a dialogue is in progress, the new dialogue request will be rejected with the "USSD Busy" error. Once the dialogue is established between the mobile station and the end node in the GSM network, another dialogue cannot be established.

USSD Addressing

The USSD was designed for dialogues between a mobile station and an USSD application in the location register base. The MSISDN is transported in the dialogue part of the Transaction Capability Application Part (TCAP) message. When a mobile initiated dialogue is established between the mobile node and an application in the HLR, which would be the USSD gateway, the MSISDN and the HLR address is passively included. This is necessary for the external USSD application to be able to respond to the correct mobile node. For a mobile initiated dialogue, the USSD gateway in the HLR is not the end application. The USSD gateway in the HLR will only work as a router and relays USSD operations between the GSM network and the external application. The external application is a provider program that can process the USSD dialogue and sends a response back to the original mobile node. The service code of the USSD dialogue, which is sent to the HLR by the mobile phone, is used to locate and to address the USSD gateway. When the HLR receives a particular USSD service code, the USSD dialogue will be relayed to the USSD gateway and from there to the USSD application somewhere in the net.

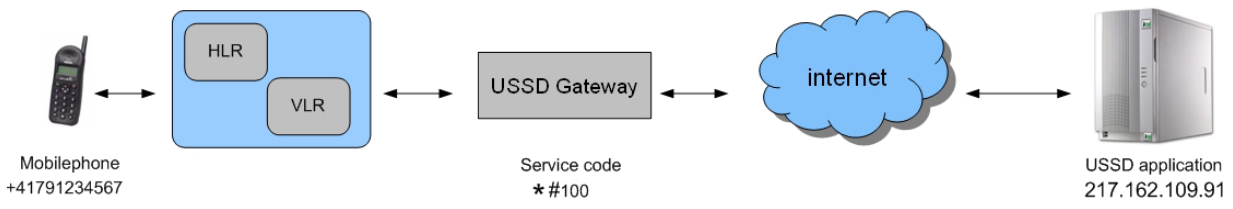


Figure 5: A typical USSD scenario.

USSD routing

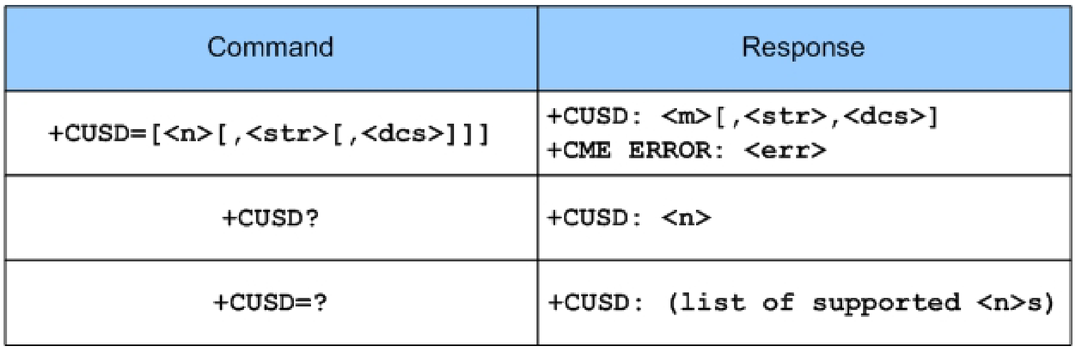
The message exchange through the GSM network is initiated by a mobile station which sends a mobile USSD message. The USSD message is routed to the home location base. The HLR forwards the USSD message to the USSD Gateway. The USSD gateway relays the message to external applications using a protocol like the IP protocol. The external system interprets the message and performs the response indicated by the content of the message. Within a time-out period, the external system acknowledges a successful reception of the message to the mobile via the USSD gateway. The external system can later asynchronously send further information to the mobile phone by SMS.

USSD string

The length of the USSD string in an USSD dialogue is limited to a precise size. According to the USSD GSM specification, the Invoking "*USSDRequest*" and the "*ProcessUSSDRequest*" can have USSD strings of the length of 160 characters. The length of the USSD string is limited to the capabilities of the lower signalling layers (TCAP), which can be configured differently depending on the specific network.

USSD and AT command

The most important AT command (figure 6) for sending and receiving an USSD string, is "*CUSD*". This command allows to control the USSD dialogues according to GSM 02.90 specification that supports both network and mobile initiated dialogues. The following figure shows a brief overview of USSD AT commands in GSM. When the USSD string is given, a mobile initiated USSD string or a response USSD string to a network initiated operation is sent to the network. In case of a successful mobile initiated operation, the response USSD string from the network is returned ahead of the final result code.



Command	Response

`+CUSD=[<n>[,<str>[,<dcs>]]]`	`+CUSD: <m>[,<str>,<dcs>]`
	`+CME ERROR: <err>`

`+CUSD?`	`+CUSD: <n>`
`+CUSD=?`	`+CUSD: (list of supported <n>s)`

Figure 6: AT commands for USSD.

Defined values

- *n*: sets and shows the result code presentation status in the TA (0 disabled and 1 enabled)

- *m*: activates the user interaction. Further user action required by 1

- *str*: USSD string to be sent

- *dcs*: Cell Broadcast Data Coding Scheme in integer format

Signalling system 7

Signalling System Number 7 (SS7) [26] is a protocol for performing out of band signalling in support of call setup, billing, routing and information exchange functions of public switched telephone network (PSTN). Out of band signalling is a signalling mechanism that is not transmitted along the same path as the link transfer channel for data and voice. For this purpose, a separate digital channel called "signalling link" was created. The SS7 messages are exchanged over this signalling link between network elements with up to 64 kilobit per second. The SS7 architecture is designed in a way that any node can exchange signalling information with any other SS7 capable node. The SS7 network and protocol are basically used for:

- Call setup

- Wireless roaming and mobile subscriber authentication
- Local Number Portability (LNP)
- Toll free (800/888) and toll (900) services

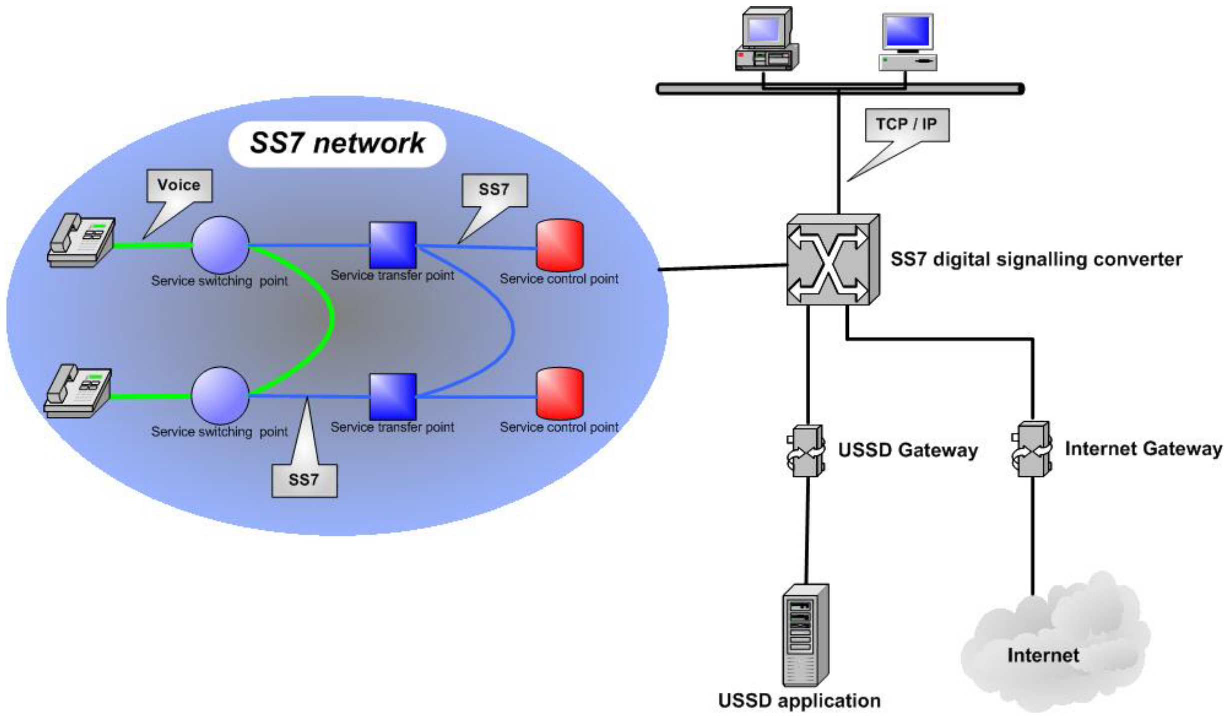


Figure 7: SS7 network.

2.4 WLAN

Wireless networks are showing up everywhere. Corporations and companies are developing and adapting wireless local area networks (WLAN) to allow employees to move freely around the company without being disconnected. Some airports offer wireless access, so that business travellers can continue their work while waiting for a plane. WLAN is an important technology that definitely will be one of the most desired technologies in the future. This chapter defines and describes a WLAN protocol for CAHN. WLAN, as a data channel that providing a high data rate, may be one of the best ways to transfer data between peering nodes in a CAHN environment.

802.11 Standard

The term *wireless* has a different meaning for different people. In general, the term reflects any means of communication that takes place without wires. The 802.11 standard [27] WLAN protocol is analogous to the protocols such as 802.3 (Ethernet) on wired networks. 802.11 is a standard, developed and ratified by the Institute of Electrical and Electronic Engineers (IEEE). The goal of the IEEE 802.11 standard is to describe a WLAN that provides services which are usually only found in wired networks.

WLAN architecture

The architecture of the 802.11 WLAN is designed to support a network where most decisions are made by mobile stations. The architecture is very flexible, easily supporting small and large networks. The WLAN architecture contains several components like the station, the access point, the wireless medium and a basic service set. This architecture allows all of the existing network protocols to run over a WLAN without any special adaptation. This is one of the biggest advantages of using WLAN.

Basic service set

The core of a WLAN network [28] is called a *Basic Service Set* (BSS). A BSS consists of a central Access Point (AP) and several client stations. The AP controls and coordinates all the activities in the BSS. The BSS networks are also called *infrastructure* networks. A BSS is identified with a service set identifier (SSID). This can be thought of as the name of the wireless network. A station that wants to join a BSS network will look for available SSIDs of APs. Some APs send beacons to inform the stations of the SSIDs existence. Without a beacon, a station must know the SSID of the AP in advance to get connected. Once a station has identified the BSS it wants to join, it sends an association request to the AP. The station and the AP commit a handshake process to exchange information about the network parameters as well as any authentication that may be needed in order to join the network. More information about the authentication process will be provided in section "authentication", later in this chapter.

Stations in an *independent* BSS (IBSS) communicate directly with each other and must be within direct communication range. Not every mobile station might be able to communicate with every other mobile station, but they are all part of the same IBSS. There is no relaying and routing mechanism in the IBSS. Therefore, if one mobile station wants to communicate with another station, they must both be in direct communication range or they must both use an existing ad hoc algorithm to relay the packages. The smallest IBSS in 802.11 contains two stations. IBSSs are composed of a small number of stations for a specific purpose and for a short period of time. One common use is to create a short-live network to support a single meeting in a conference room. As soon as the meeting begins, the participants create an IBSS session to exchange and share data. When the meeting ends, the IBSS is dissolved. IBSSs have found a similar use at LAN parties around the world. IBSSs are sometimes referred to as *ad hoc* BSSs or *ad hoc networks*.

CAHN can enable ad hoc networking between the stations by using and adapting the IBSS as the data channel and applying the existing mechanisms (next sections) in IBSS to establish a secure connection among all stations. The cellular network is still the signalling channel that transports all the required information for the establishment of a secure WLAN link.

Scanning

Before using any network, it is obvious that a network can only be joined, if its presence is well known for other stations. Stations must identify a compatible network before joining it. Scanning is a process which helps finding available networks. All stations in an IBSS network need to share the same BSSID (and the same SSID) to be able to communicate with each other. Stations with different BSSIDs will not be able to connect. When an IBSS network is created, one station generates a random BSSID and transmits it to other stations in the local area sharing the same SSID. Once an IBSS station has got a BSSID from another IBSS station, it can establish an ad hoc environment and start the communication with other members of the IBSS. The choice of the value, format and length of the SSID is entirely up to the network administrator or WLAN user. CAHN stations use this WLAN feature to broadcast their MSISDN identities to other CAHN stations. The most important parameters for the scanning and broadcasting process are:

- BSS: scanning can specify whether to look for independent ad hoc networks, infrastructure networks or all other networks

- BSSID: the device can scan for a specific network to join (individual) or for any network that allows joining

- SSID: well known as “network name” is the only parameter that can be manipulated by the user. Most products refer to the SSID as the network name, because it is commonly a “human readable” string

Basic service set identifier (BSSID)

The BSS identifier (BSSID) is a unique identifier for a BSS. Its format is identical to an IEEE 802.11 48 bit address. In an infrastructure BSS, the BSSID is the MAC address of the AP. Using the MAC address of an AP as the BSSID ensures that the BSSID will be unique. In an IBSS, the BSSID is locally administered and is an individual address that is randomly generated by the station that starts the IBSS. The generation of this address from a random number provides some guarantee that the identifier will be unique. But there is a finite probability that the address generated is not unique. In both infrastructure and independent BSSs, the BSSID must be an individual address. If a station is a member of an IBSS, the BSSID of the station is the same as the BSSID of the IBSS.

Service set identifier (SSID)

Some documentation refers [29] to the SSID as the network name because network administrators often assign a string to it as network identity. The SSID is a case sensitive text string. The SSID is a sequence of alphanumeric characters (letters or numbers) and has a maximum length of 32 characters. The CAHN WLAN protocol uses the SSID for CAHN specific information. Every CAHN station announces its identity and CAHN ability by using the SSID field. The CAHN WLAN protocol allows the devices to configure this field before attempting to setup a link by putting as much information in the SSID (figure 8) as possible.

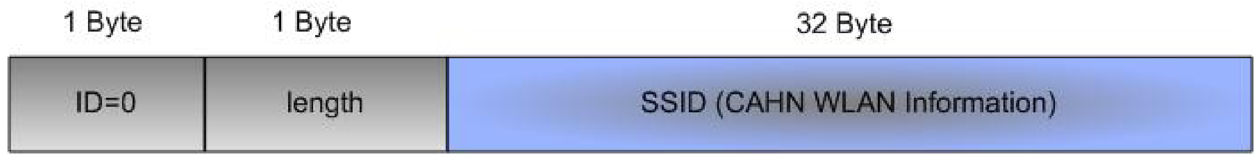


Figure 8: WLAN 802.11 service set identity information element.

Joining

After compiling the scan result, a station can join one of the BSSs by using the SSID of the desired Network. The joining process is a purely local process. There is no indication to the outside world that a station has joined an existing BSS. Joining a BSS requires that all of the mobile station’s Medium Access Control (MAC) and Physical (PHY) parameters are synchronised with the target BSS. The station must adapt its Timer Synchronisation Factor (TSF) timer to the value of the timer from the BSS. Furthermore, the station must synchronise the PHY parameters. This will ensure that the PHY layer is operating on the same channel. The BSSID of the BSS must be adopted as well as the parameters in the capability information field, such as Wired Equivalent Privacy (WEP) and the Dynamic Synchronous Transfer Mode (DTIM). Once this process is completed, the mobile station has joined the BSS and is ready to communicate with any other station in the BSS and the AP.

Authentication

Authentication provides a mechanism for one station to prove its identity to another station or the AP in the WLAN [30]. Each station asserts its an identity and asks other stations about theirs identities. The process of authentication contains messages about identity questions, identity assertions and results. As an example, station A asserting "I am station A" and asking station B "who are you?" At this point the process of authentication varies depending on the algorithm that is used. It may proceed with station B saying "OK, prove me you are station A" and asserting "I am station B". Station A then would offer some proof of its identity and on the other hand request the same kind of proof from station B. If the proofs were acceptable, each station would acknowledge the identity of the other. Beyond that point, communication from the other station is trusted.

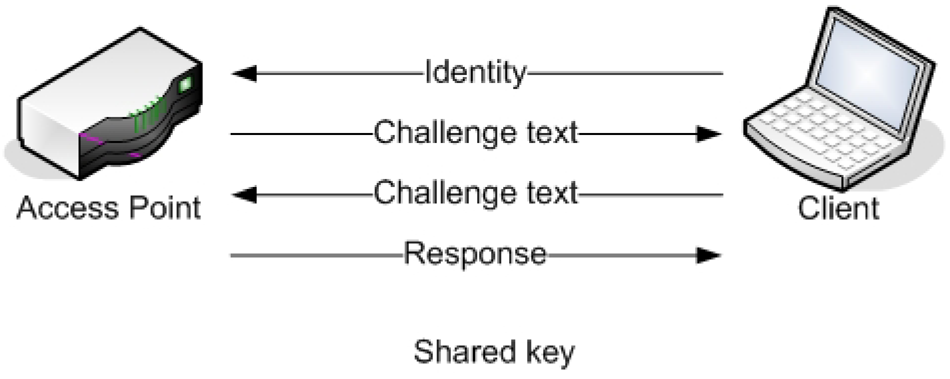


Figure 9: Shared key authentication exchange.

Authentication mechanisms in WLAN can be used between any two stations or stations and APs. Every ad hoc network can use this mechanism to prove its identity. Full proof of identity of a mobile station is necessary if the network is to be protected from unauthorised users. Thus, this important feature of WLAN is used in CAHN for the authentication and verification of devices.

In IEEE 802.11, there are two authentication algorithms available: open system authentication (figure 10) and shared key authentication (figure 9) based on WEP. Open system authentication is not really an authentication algorithm. It is just for those users of WLAN that do not want to use the WEP algorithm. Open system authentication allows the authentication frame exchange protocol to complete with a guaranteed result of "success". In this case, station A asserts its identity to the station B and station B would response with a successful result for the authentication. There is no real verification of the identity on the other side. Open system authentication allows any mobile station to access the network and therefore, it is not really useful for CAHN. It does not provide the security features required by CAHN.

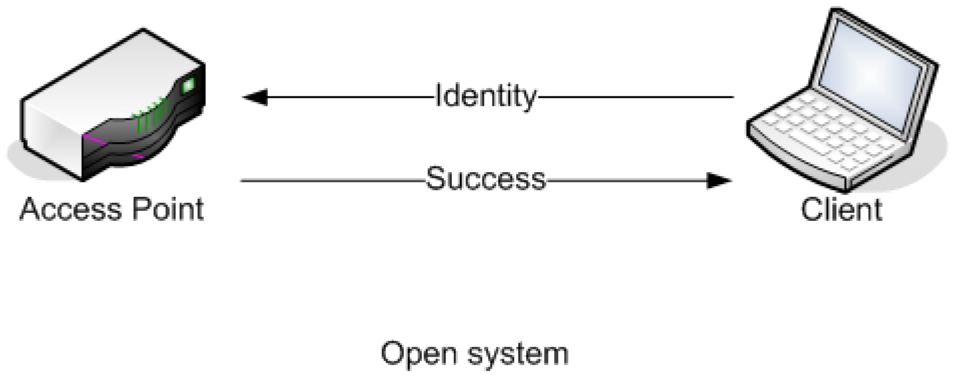


Figure 10: Open system authentication exchange.

The second authentication algorithm is the shared key authentication algorithm. It depends on both stations having a copy of a pre-shared WEP key. This method uses the WEP encryption option to encrypt and decrypt a challenge as proof that the stations share the same key. Shared key algorithm, as its name implies, requires that a shared key is distributed to each station before attempting authentication. Station A sends its identity (assertion) to the station B. responds to the assertion with an assertion of its own. Station B requests the station A.to prove its identity by correctly encrypting a challenge text using the WEP encryption and the shared key and sends the encrypted message back to the station B. Station B then decrypts the frame using the shared key again and returns an authentication message (authentication management frame) to station A with a success or failure notification of the authentication. If authentication was successful, station A is authenticated to station B. CAHN protocol security can choose the shared key authentication algorithm of 802.11 to protect a CAHN link.

Wired equivalent privacy (WEP)

The 802.11 MAC specifications describe an encryption protocol called Wired Equivalent Privacy (WEP). The goal of the WEP is to make WLAN as secure as wired LAN. WEP provides two kinds of wireless security: authentication and confidentiality. WEP uses a shared key mechanism with a symmetric cipher called RC4. The key that a client station is using for authentication, encryption and decryption of the data stream must be the same key that the AP or other clients use. The 802.11 standard specifies a 40 bit key, however most vendors have also implemented a 104 bit key for better security.

2.5 CAHN

CAHN has been integrated with the Bluetooth Service Discovery Protocol [5]. The Bluetooth Service Discovery Protocol builds the basis for the development of an application that integrates CAHN with the Bluetooth technology. Bluetooth will gain an improved authentication method and scalability. On the other hand, CAHN can indicate its features via the Bluetooth Service Discovery Protocol. It means the Bluetooth SDP can be used to learn about the CAHN capabilities that a certain device provides. This related work has shown, that the integration of the two technologies is very promising and that the integration is favourable for both technologies. Therefore, CAHN can offer this enhanced authentication, which is based on the trust relation of the user with the cellular operator. With the help of this authentication, combined with the secured channel that CAHN offers, the PIN can be negotiated dynamically. As a result the Bluetooth link can be secured, using the built-in Bluetooth security mechanisms.

The architecture of CAHN Bluetooth integration contains three different layers which divide the application in three logical parts:

1. The CAHN Communication Manager (waits for and handles requests)

2. The CAHN connector (manages access technology (Bluetooth) related issues)

3. The CAHN adapter (provides an interface to the used cellular network for the CCM)

Undoubtedly, that further work in the topic is necessary to improve the behaviour of the provided application. However, this shows that the implementation itself performs well, and that the idea of integrating CAHN in the Bluetooth SDP is a valuable approach for enabling secured connections in public Bluetooth environments. The following figure 11 illustrates the basic idea of the integration of CAHN with Bluetooth:

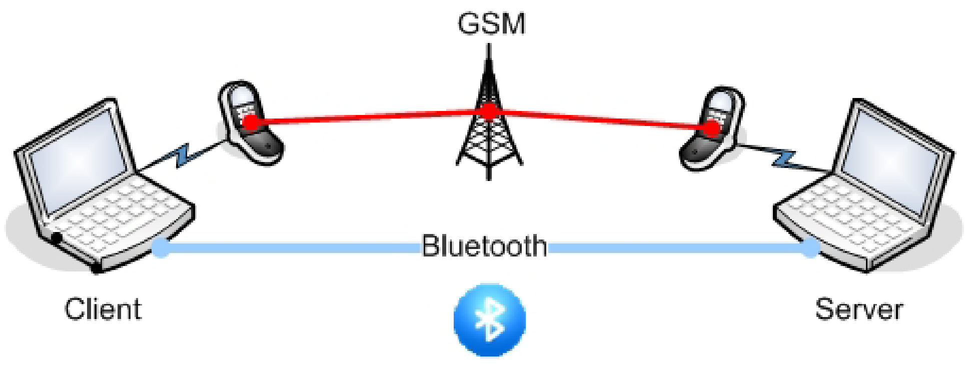


Figure 11: The integration of CAHN with Bluetooth.

Figure 12 shows the CAHN Bluetooth architecture with all corresponding components. The implementation of CAHN is primitive and especially the SMS mechanism does not suit the delivery of CAHN protocol messages over the cellular network. But, the existing components of CAHN can be used to extend its usability to other technologies.

The goal of this diploma work is the extension of the CAHN concept for the WLAN technology. The integration of CAHN with WLAN requires new components which will be added to the existing CAHN architecture. The CAHN adapter has to recognise the CAHN WLAN packets and has to be able to broadcast them over the GSM link. The CCM should be able to handle CAHN WLAN requests and to create the corresponding responses. Today, the CAHN connector contains only the CAHN Bluetooth connector. For WLAN extension, a new CAHN WLAN connector is needed.

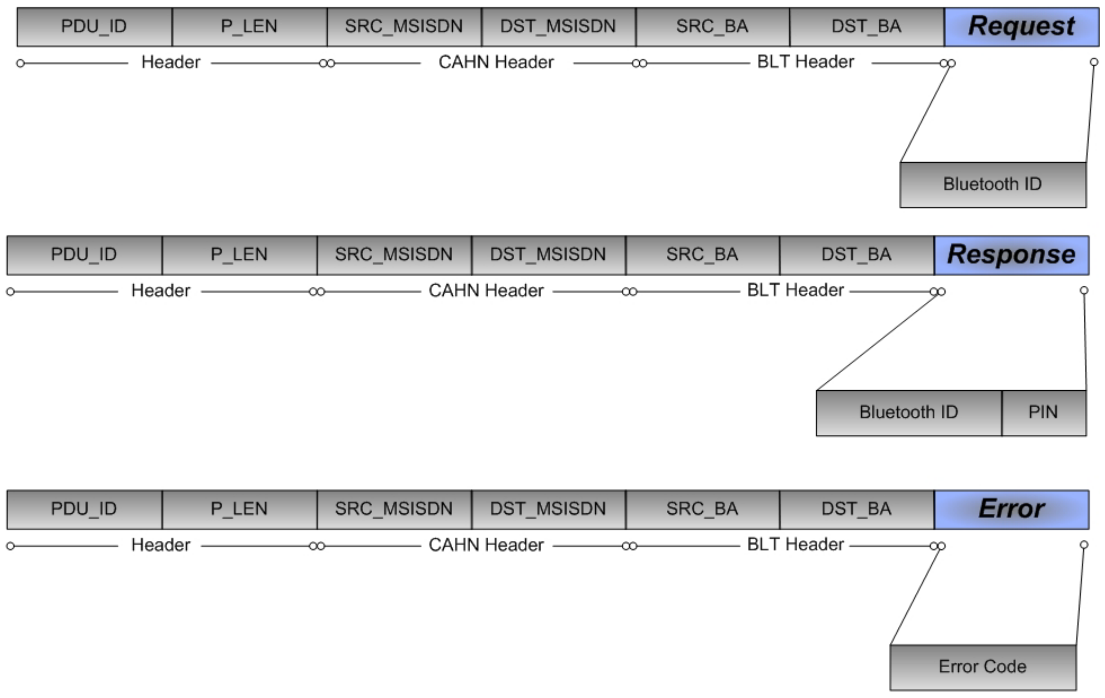


Figure 12: CAHN Bluetooth protocol messages.

3 CAHN BLUETOOTH PROTOCOL

The following chapter contains the formal specification of the CAHN Bluetooth protocol. The CAHN Bluetooth protocol is an asymmetric and hierarchic protocol which uses three types of messages to bootstrap a connection. As lees as two CAHN stations can establish a CAHN Bluetooth link. Station "A" sends a CAHN request which contains its MSISDN and Bluetooth address. Station "B" (Server) handles the request and accepts the connection by sending a CAHN response or rejects it by returning a CAHN error response back to the station B (figure 13).

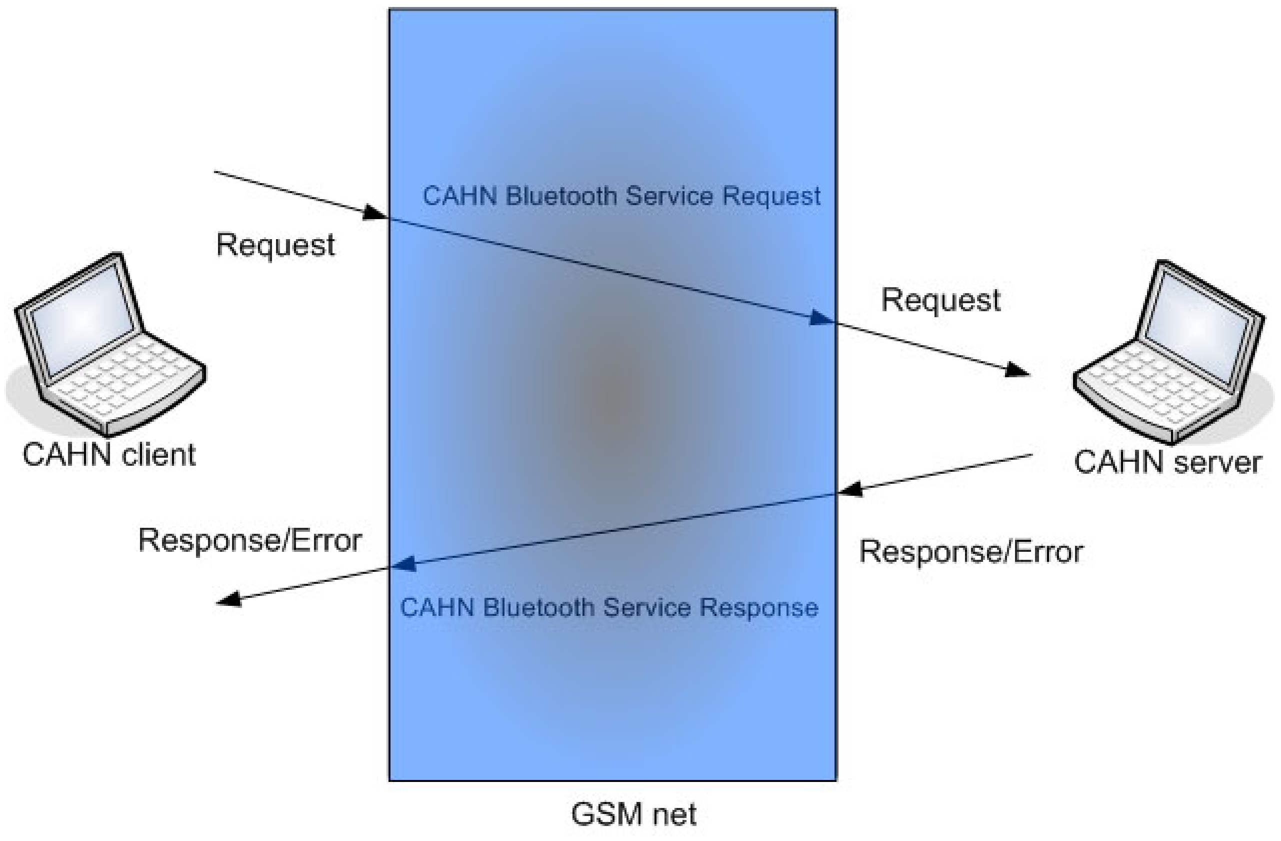


Figure 13: Time sequence diagram of the CAHN Bluetooth messages.

3.1 CAHN Bluetooth state machine

Using a state machine is another way to describe the status of the whole protocol [16]. The local interactions among the different parts of a protocol can be easily explained by utilizing the graph mechanism. A key concept used in many protocols is the finite state machine. With this technique, each protocol machine (i.e. server or client) is always in a specific state at every point in time. Its state consists of all the values of its variables.

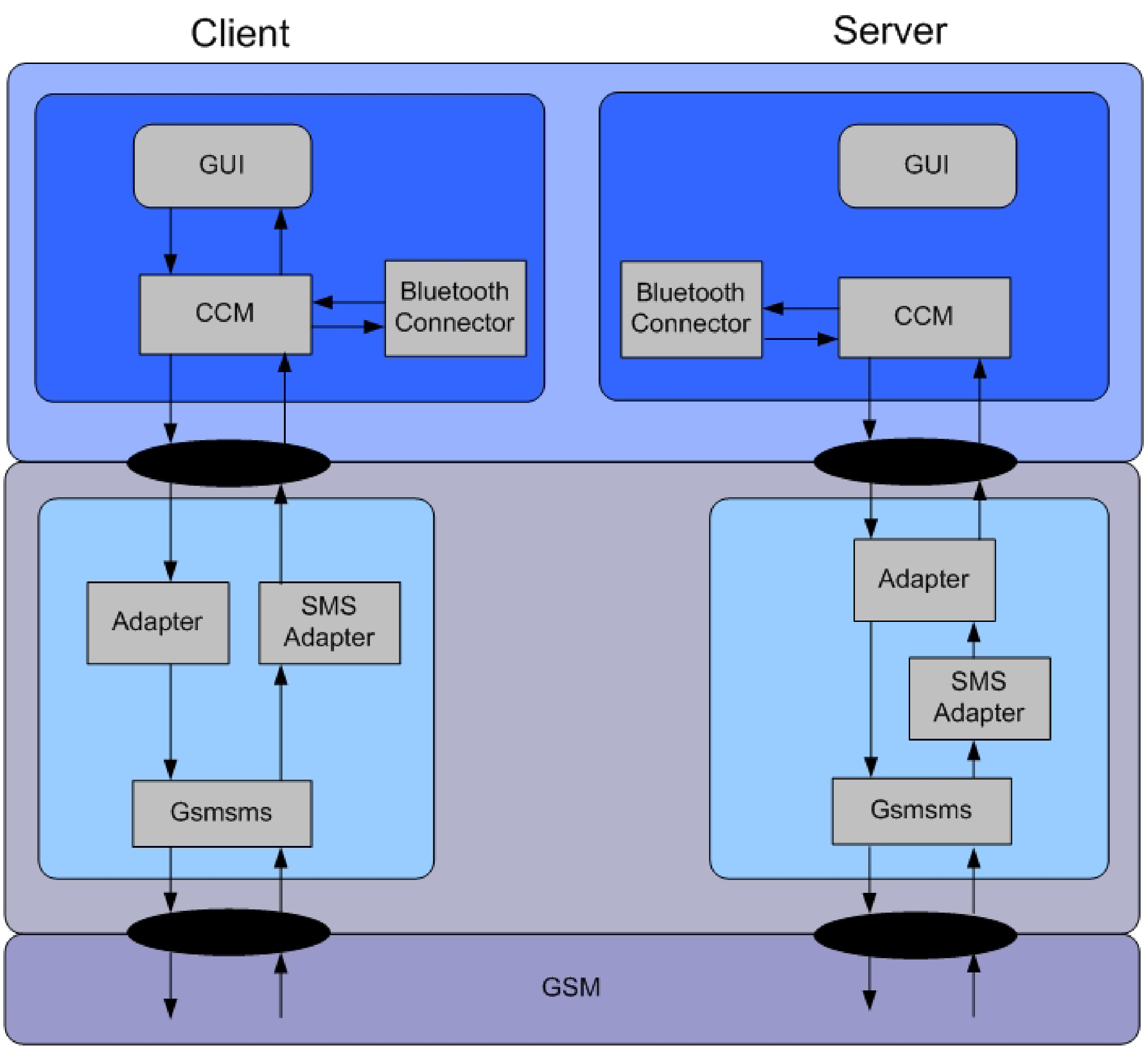


Figure 14: CAHN Bluetooth architecture with SMS transport mechanism.

In most cases, a large number of states can be grouped together for analytical purpose. The state of the complete system is represented by the combination of the states of the two protocol machines. From each state, there are zero or more possible transitions to other states. For a protocol, a transition may happen when a frame or a message is sent, when a packet arrives or when an interrupt (error) occurs. For a complete description of the protocol, it is possible to draw a directed graph showing all the states as nodes and all the transitions as directed edges. The figure 15 shows the CAHN Bluetooth time sequence diagram which contains two separate time sequence diagrams. Both CAHN client and CAHN server diagrams cover three states and the passage of time is top-down.

The following finite state machine diagram in figure 15 shows the interactions in a CAHN Bluetooth client/server system. The CAHN server is always in an "await for incoming request" state. A CAHN request triggers a transition and the server changes its current state to "process the request".

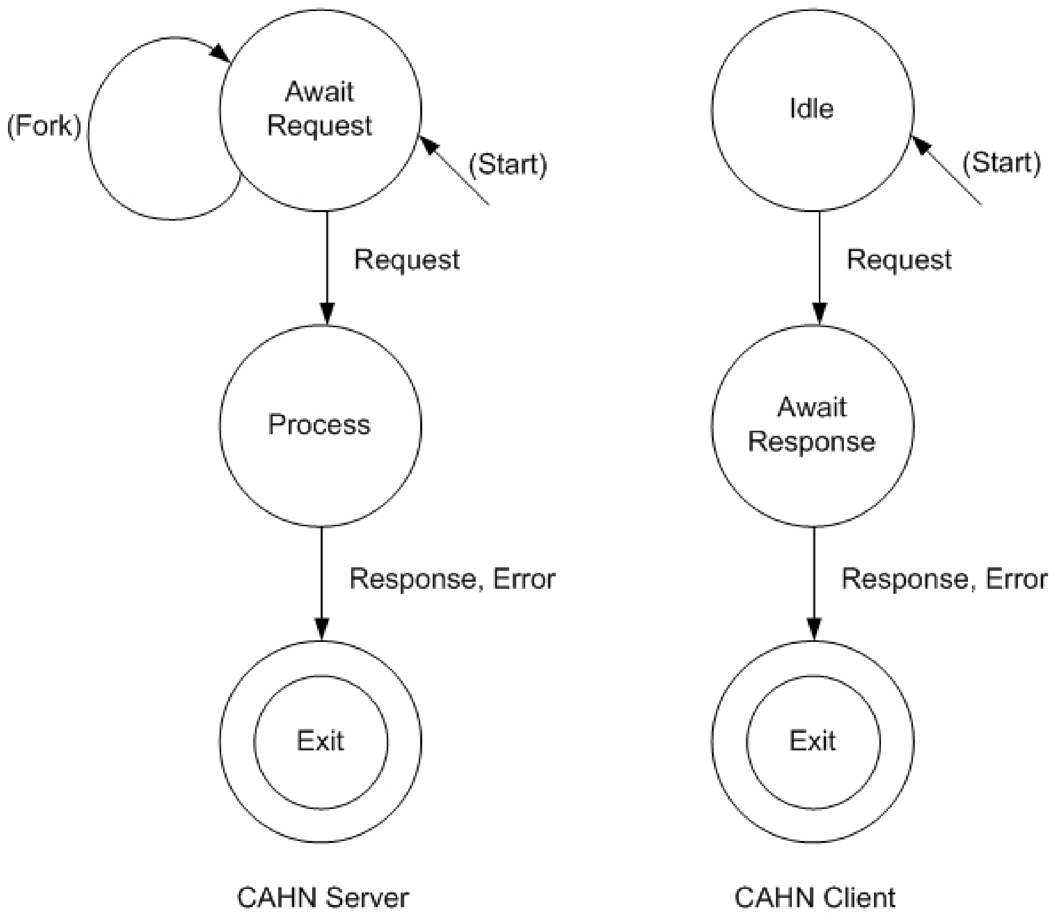


Figure 15: CAHN Bluetooth state machine.

3.2 Formal specification

This section presents some code segments of formal specification of the CAHN Bluetooth protocol. Appendix E contains the entire formal specification of the CAHN Bluetooth protocol according to the given protocol engineering criteria. Figure 16 and figure 17 show two code segments of the CAHN Bluetooth formal specification. The definitions of protocols and services are given in a pseudo specification language. The specification defines the protocol services and messages. Each service in the CAHN Bluetooth protocol is defined precisely in the service specification part of the implementation.

```
Specification CAHN_BLUETOOTH_PROTOCOL
Service
        CAHN_BLUETOOTH_PROTOCOL-CONNECTION_SETUP
            requested cahn_send_data( sock: integer,
                                      buf: char,
                                      size: integer)
                      adapter_send_data( sock: integer,
                                         buf: char,
                                         size: integer,
                                         resp_sock: integer)

            responded cahnd_read_response( sock: integer,
                                           rsp_buf: char,
                                           rsp_buf_len: integer)
                      response_socket_send_data( data: char,
                                                 data_length: integer,
                                                 response_socket: integer)
```

Figure 16: Formal specification of the CAHN Bluetooth protocol.

The implementation extends the formal specification by defining two protocol entities of the client and the server. In this part, services are accessed on the service access points (SAP). Each protocol contains one or more phases which characterize the moment situation in the protocol. "CONNECT" and "DATA TRANSFER" are two major phases of the CAHN Bluetooth protocol

The following code segment shows the formal specification of the CAHN Bluetooth client entity.

```
entity CAHNCLIENTS
        sap client
        var
        begin
          phase CONNECT: cahn_blt_client ‖ sap handler CAHNCLIENT
          phase DATA_TRANSFER
        end// CAHNCLIENT

sap handler CAHNCLIENTS
        service SMS: requested spool SMS
                 requested send SMS
                 responded receive SMS
        loop
            wait event
                 Cahn Bluetooth Service Request: request spool SMS
                                        request send SMS |

                 receive SMS:
                          case code of
                               Cahn Bluetooth Service Response:
                               set event Cahn Bluetooth Service Response |
                               Cahn Bluetooth Error Response:
                               set event Cahn Bluetooth Error Response
                          #case
            #wait
        #loop
#sap
```

Figure 17: CAHN Bluetooth client entity.

3.3 Implementation

In the previous section, the formal specification of the CAHN Bluetooth protocol was introduced. For this purpose and to make a real simulation and validation process, a protocol language simulator and evaluator called "PROMELA" (PROcess Meta LAnguage) [17] was chosen. The following sections present the implementation of the CAHN Bluetooth protocol written in PROMELA. Appendix G contains the entire PROMELA validation and simulation implementation.

The PROMELA implementation of the CAHN protocol contains two main parts: a simulation of the CAHN Bluetooth protocol and a protocol validation. The implementation simulates a CAHN Bluetooth client/server scenario which contains one client and one server. The CAHN client sends a CAHN Bluetooth service request to the CAHN server. The CAHN server analyses the request and returns a CAHN service response massage. The PROMELA simulation contains four simulation modes:

- Message sequence chart panel (figure 18)
- Time sequence panel
- Data value panel
- Execution bar panel

All four simulation modes were integrated in the simulation of the CAHN Bluetooth protocol. The following diagram shows a typical CAHN Bluetooth scenario in which a client sends a request and waits for the response from the server. The numbers on the arrows indicate the message content.

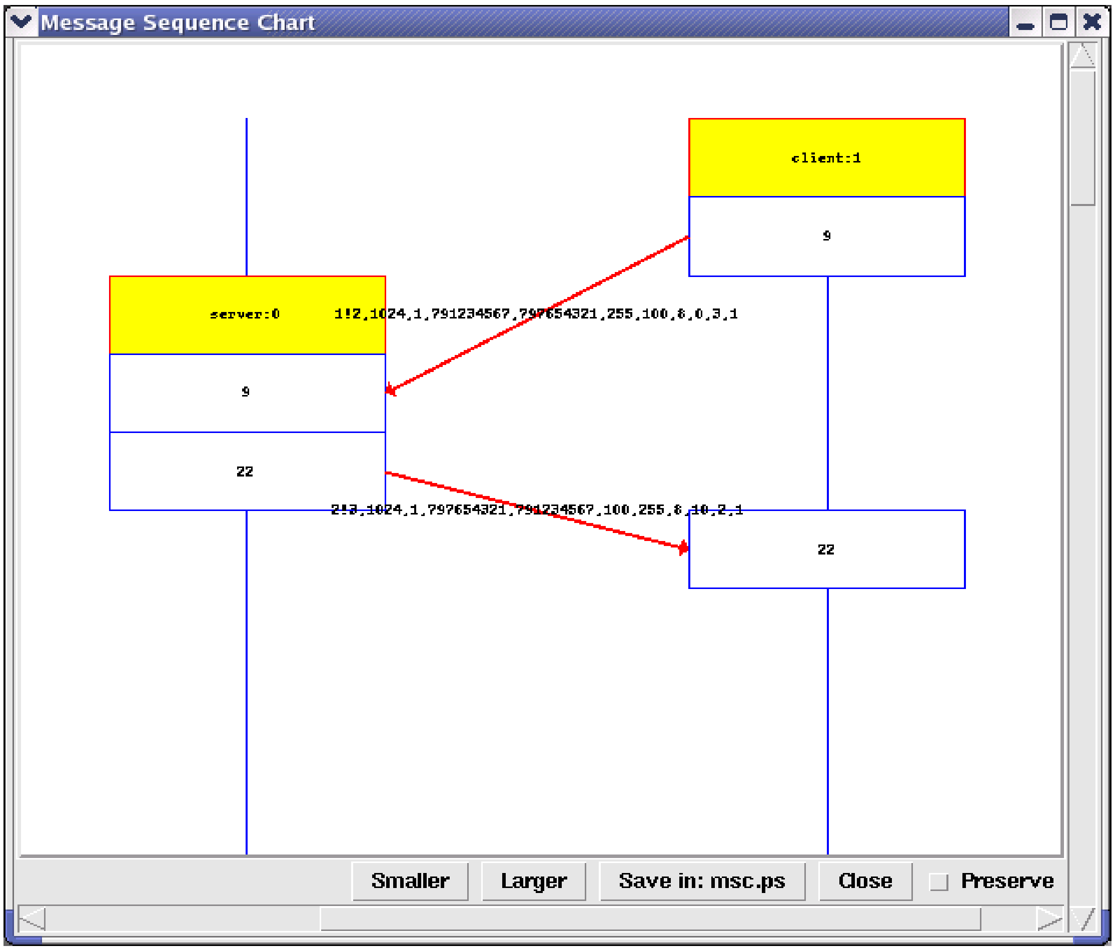


Figure 18: Message sequence chart of the CAHN Bluetooth protocol.

In PROMELA, it is not possible to simulate the socket connection part of the protocol. PROMELA has its own method to simulate the relation between nodes and the network. For this purpose, PROMELA offers a special message exchange mechanism called "channel". Channels represent the concept of message communication between client and server. Each node (client and server) has its own entities and each entity has a transport channel to exchange messages with other nodes:

```
/**************************
 *
 *    Channel
 *
 *************************/

chan cahn_adaptor_client = [0] of {cahn_blt_message}
chan cahn_adaptor_server = [0] of {cahn_blt_message}
```

According to the CAHN Bluetooth state machine, a CAHN node (during connection setup) runs through three different states. For a CAHN client, these three states are idle, awaiting response and exiting. The implementation of the simulation of CAHN is based on the CAHN Bluetooth state machine. The following code simulates a CAHN client entity:

```
/* CAHN client is wating for the response. */
      AWAIT_RESPONSE:
            do
            :: cahn_adaptor_server?
                 pdu_id(p_len, seq_nbr, src_msisdn, dst_msisdn, src_ba,
                 dst_ba, blt_id, data, EOF_response, sms_end) ->
            :: timeout -> break; /* Error message */
            od;

      accept:     IF
                  (pdu_id == CAHN_BLT_SVC_RSP) ->
                        /* Get data from the cahn response message. */
                        /* Mainly the pin code for blt connection setup. */
                        get_data;
                        goto EXIT;
            ENDIF;
            IF
                  (pdu_id == CAHN_ERROR_RSP) ->
                                    get_error;
                                    error_messages;

            ELSE ->
                  assert(false)     /* protocol violation */

            ENDIF;
```

The XSPIN bar chart (XBC) in figure 19 shows the total system steps (processes) by sending (request) and receiving (response) messages during a CAHN Bluetooth connection setup:

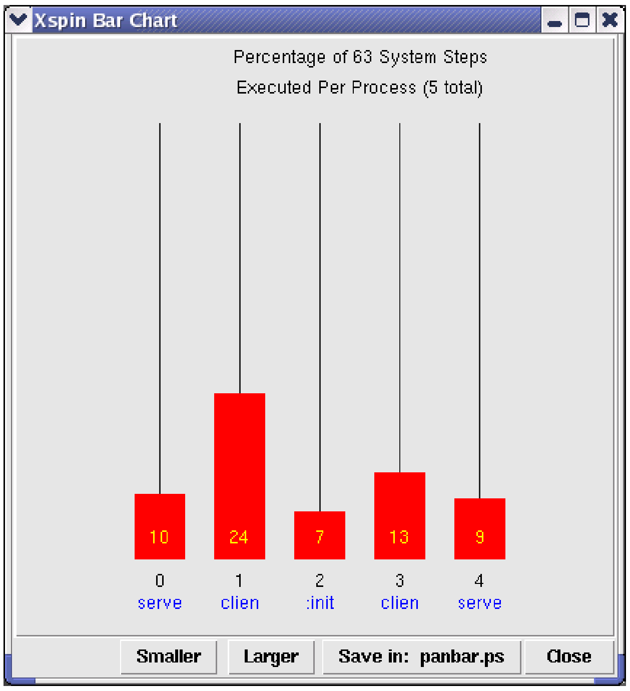
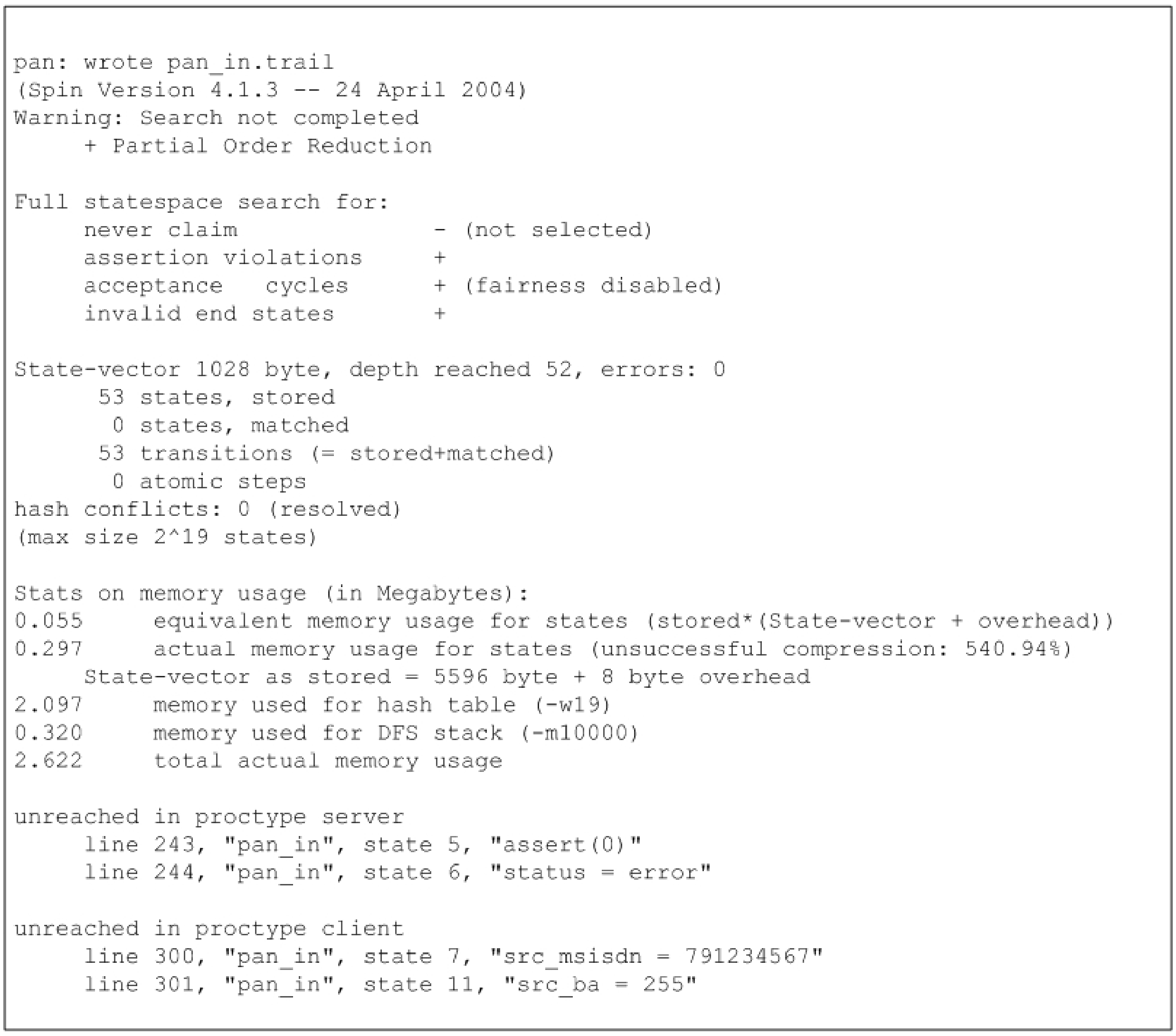


Figure 19: The figure shows the total system steps during a CAHN Bluetooth connection setup. It illustrates the needed number of system processes during connection setup (CAHN request and response).

PROMELA uses two algorithms to evaluate the correctness of the CAHN Bluetooth protocol, the safety and the liveness algorithm. The safety algorithm (state properties) analyses assertions and invalid end states in a protocol. The liveness algorithm checks if there is an acceptance/non-progress cycle or weak fairness. PROMELA also searches and reports unused code.



```
pan: wrote pan_in.trail
(Spin Version 4.1.3 -- 24 April 2004)
Warning: Search not completed
     + Partial Order Reduction

Full statespace search for:
     never claim              - (not selected)
     assertion violations     +
     acceptance   cycles      + (fairness disabled)
     invalid end states       +

State-vector 1028 byte, depth reached 52, errors: 0
      53 states, stored
       0 states, matched
      53 transitions (= stored+matched)
       0 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

Stats on memory usage (in Megabytes):
0.055     equivalent memory usage for states (stored*(State-vector + overhead))
0.297     actual memory usage for states (unsuccessful compression: 540.94%)
     State-vector as stored = 5596 byte + 8 byte overhead
2.097     memory used for hash table (-w19)
0.320     memory used for DFS stack (-m10000)
2.622     total actual memory usage

unreached in proctype server
     line 243, "pan_in", state 5, "assert(0)"
     line 244, "pan_in", state 6, "status = error"

unreached in proctype client
     line 300, "pan_in", state 7, "src_msisdn = 791234567"
     line 301, "pan_in", state 11, "src_ba = 255"
```

Figure 20: PROMELA validation output for CAHN Bluetooth protocol.

3.4 Conclusion

Figure 20 shows a screenshot of the PROMELA simulation and validation results. The PROMELA simulations have shown the message exchange during a CAHN Bluetooth setup connection. The simulations exemplify the total memory usage of a CAHN Bluetooth process.

The above figure shows that the validation of the CAHN Bluetooth protocol covers three main tasks: assertion violations, acceptance cycle and invalid end states. As a result of the PROMELA validation, no validation faults or assertions violation have occurred in the provided application. This means that the complexity of the architecture is surely implementation oriented.

As a result, the components of the CAHN Bluetooth protocol can be used and enhanced for the design of new CAHN protocols based on other interesting technologies.

The PROMELA simulation and validation did not show an existing problem of the CAHN Bluetooth protocol, although it is mentioned in the diploma work [5]. Using SMS for the transport of CAHN messages over the cellular network is not satisfactory. To provide better performing of CAHN applications, other mechanisms than SMS must be applied. USSD [22] mechanism promises to be a better approach for this task.

The next chapter presents message exchange mechanism as an alternative to SMS. Necessary parameter configurations and additional functionality will be identified that are needed in order to use GSM USSD as a bearer of the CAHN protocol.

4 CAHN USSD ADAPTER

CAHN requires a full duplex datagram service from the carrier network. GSM USSD does not provide such a service. Instead, GSM USSD provides a two way alternate interactive service designed to transmit short text strings between the mobile phone and a node in the GSM network. This chapter identifies the necessary parameter configurations and additional functionality that are needed in order to use GSM phase 2 USSD as a transport mechanism of the CAHN protocol (figure 21). As explained in the last chapter, by sending messages over the GSM network, USSD acts faster than SMS, which is an advantage for any USSD based application. Actually, USSD will be an alternative transport medium to the SMS transport mechanism in CAHN. Further, for the end user, USSD dialogues cost less than the short message service. This favors the use of the GSM USSD for the CAHN applications.

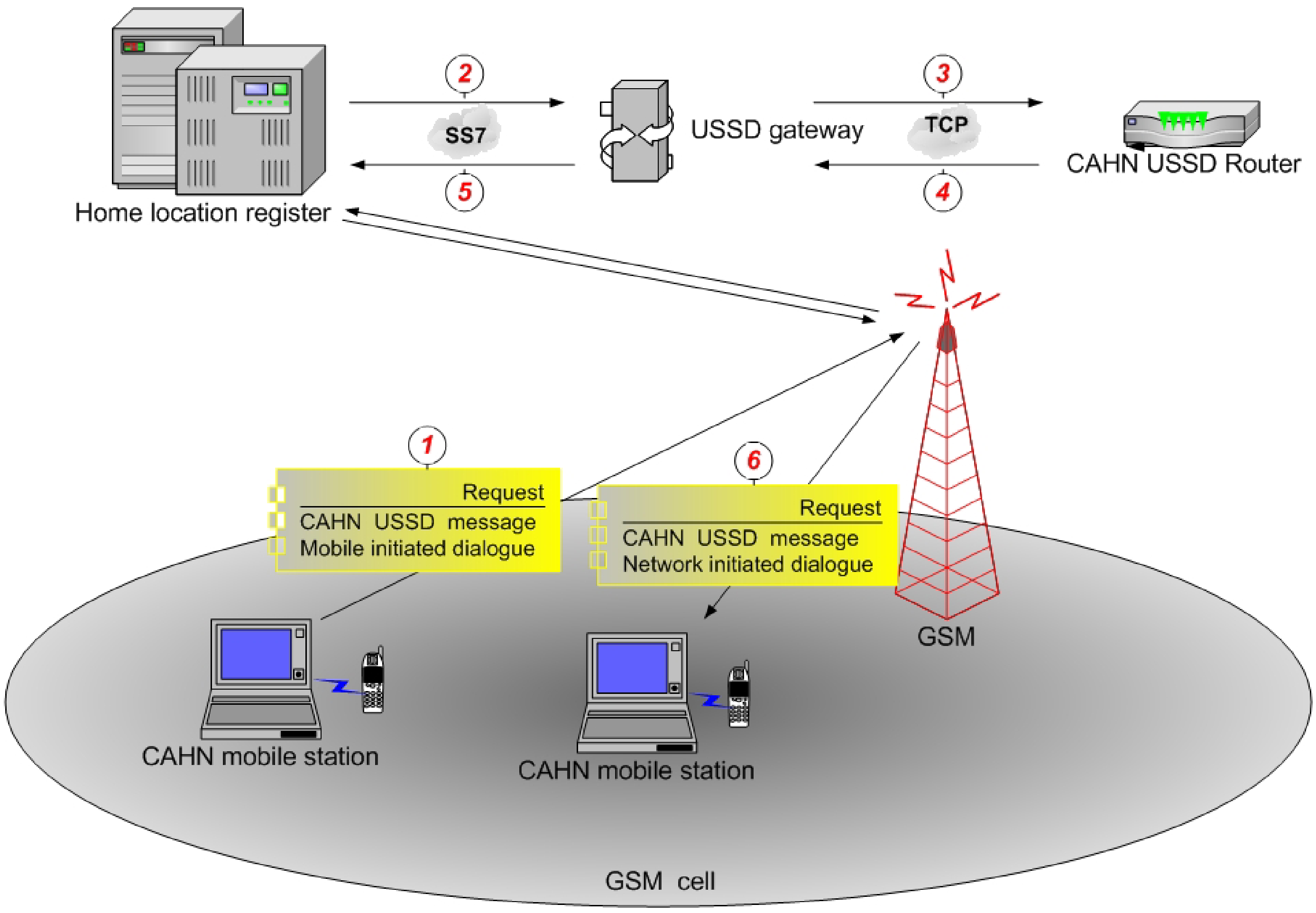


Figure 21: Using USSD as a transport bearer for CAHN. A CAHN mobile station sends a CAHN USSD request (1). The home location register relays this message to the USSD gateway via SS7 link (2). The USSD gateway routs the CAHN USSD message to the CAHN application (3). The CAHN application (CAHN USSD router) analyses this message and relays it back to the corresponding CAHN mobile station (4 and 5). The corresponding CAHN mobile station receives a CAHN USSD request (6).

4.1 CAHN USSD architecture

Once a CAHN service message reaches the USSD module, the USSD adapter takes the message and generates the USSD supplementary information which contains all information of the original CAHN message. The USSD connector initiates a mobile initiated dialogue via GSM network (through the HLR and the provider gateway) to the CAHN USSD router and relays the USSD message. The CAHN USSD router analyses the USSD dialogue by looking at the destination MSISDN in the Data field (supplementary information). The CAHN USSD router initiates a network based dialogue and sends a CAHN message request back to the CAHN node.

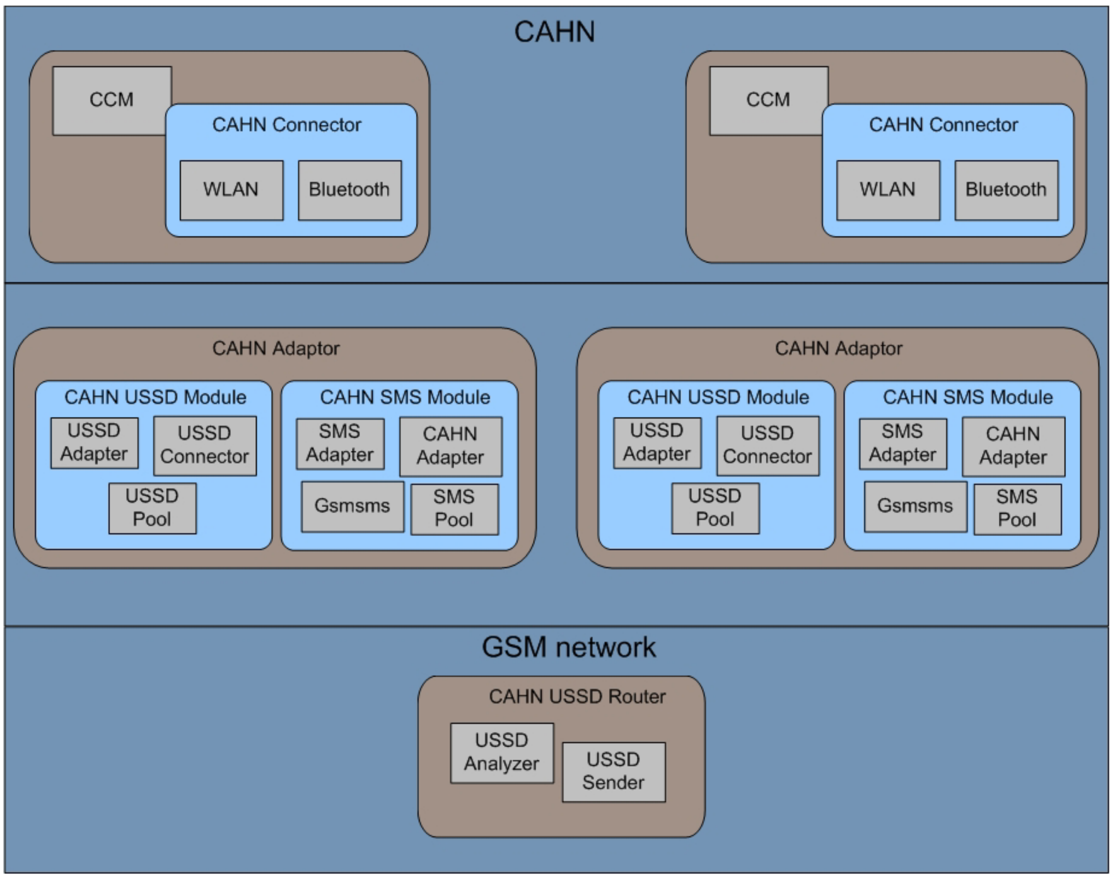


Figure 22: The entire CAHN architecture with USSD adapter.

4.2 CAHN USSD module

The CAHN USSD module is responsible for the transport of all CAHN messages. Figure 22 shows the architecture of the CAHN USSD module. The CAHN USSD module has two components: USSD connector and USSD adapter. It takes a CAHN message (i.e. CAHN service request), converts it into an USSD message and sends it through the GSM network to the other CAHN node. Upon the entrance of a response message, the USSD adapter will transform it to a valid CAHN response message. The CAHN messages that have to be sent and received, are saved in the USSD pool.

4.3 USSD connector

The USSD connector acts as the communication link between CAHN and the GSM network. The USSD connector checks the mobile phone periodically and watches all incoming USSD dialogues (polling method). When an USSD dialogue is established to the mobile station, the CAHN USSD connector looks for CAHN specific information by checking the service code of the dialogue or by analysing the message header. If a CAHN message is formed, the connector will save this dialogue directly in the USSD pool. If no such identifier is present, the connector ignores the USSD dialogue. This does not mean that the USSD dialogue must be ignored by other applications in or out of the phone. The connector is also responsible for initiating an USSD mobile based dialogue. As soon as the connector finds an USSD CAHN message in the USSD pool, the USSD connector will start a mobile initiated USSD dialogue and will send the message to the responsible CAHN end node. In this case, to the CAHN USSD router, which will forward it to the final destination node. Note that the CAHN stations can not directly address other CAHN nodes in the network.

4.4 USSD adapter

The main task of the USSD adapter is to convert CAHN messages into USSD messages and vice versa. After receiving a CAHN message form the CCM, the adapter converts it into an USSD message and saves it in the USSD pool. On the other side, the USSD adapter transforms the USSD messages, which are saved in the pool by the CAHN USSD connector, back into a valid CAHN message and routes them to the CCM component (figure 23).

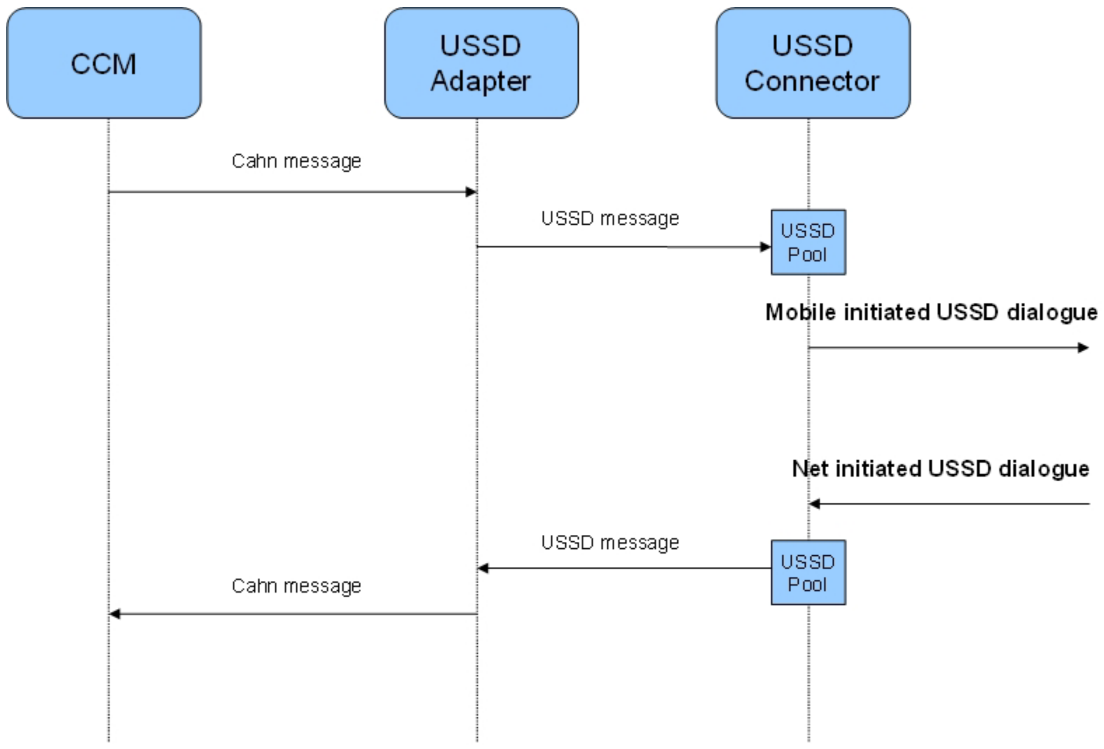


Figure 23: Message exchange between CCM and USSD module.

4.5 CAHN USSD router

Normally, the operators provide services such as USSD applications behind the GSM network (e.g. prepaid or vending machine services), but the CAHN USSD application only functions as a router. This application just relays all received CAHN USSD messages to their target end nodes.

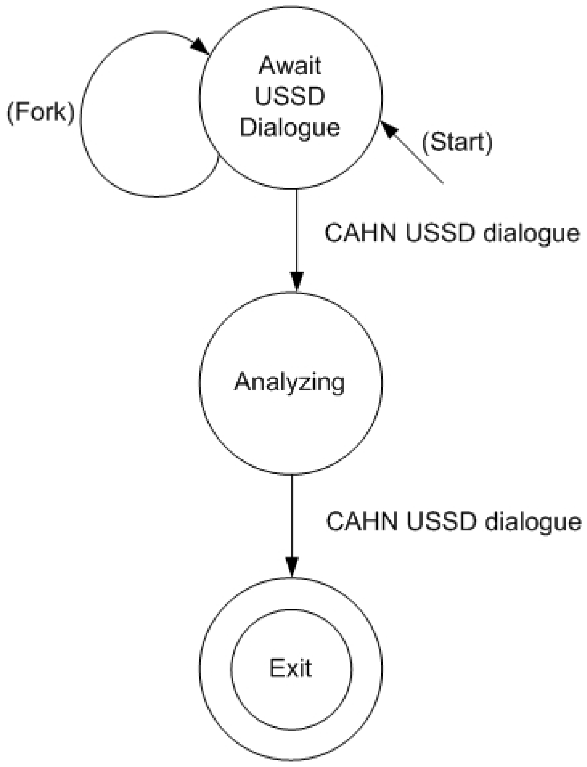


Figure 24: The state machine of the CAHN USSD router.

The CAHN router analyses all incoming CAHN USSD packets from the USSD gateway and relays them to the destination CAHN node through the GSM network. The CAHN USSD router consists of two elements: the USSD analyzer and USSD sender. The USSD analyzer handles the messages by looking for certain information in the USSD dialogue, which is the MSISDN of the destination CAHN node. Once this process has terminated and the MSISDN of the target CAHN node is known, the USSD sender opens a network initiated USSD dialogue via the USSD gateway and sends the original message to the target CAHN station(figure 25).

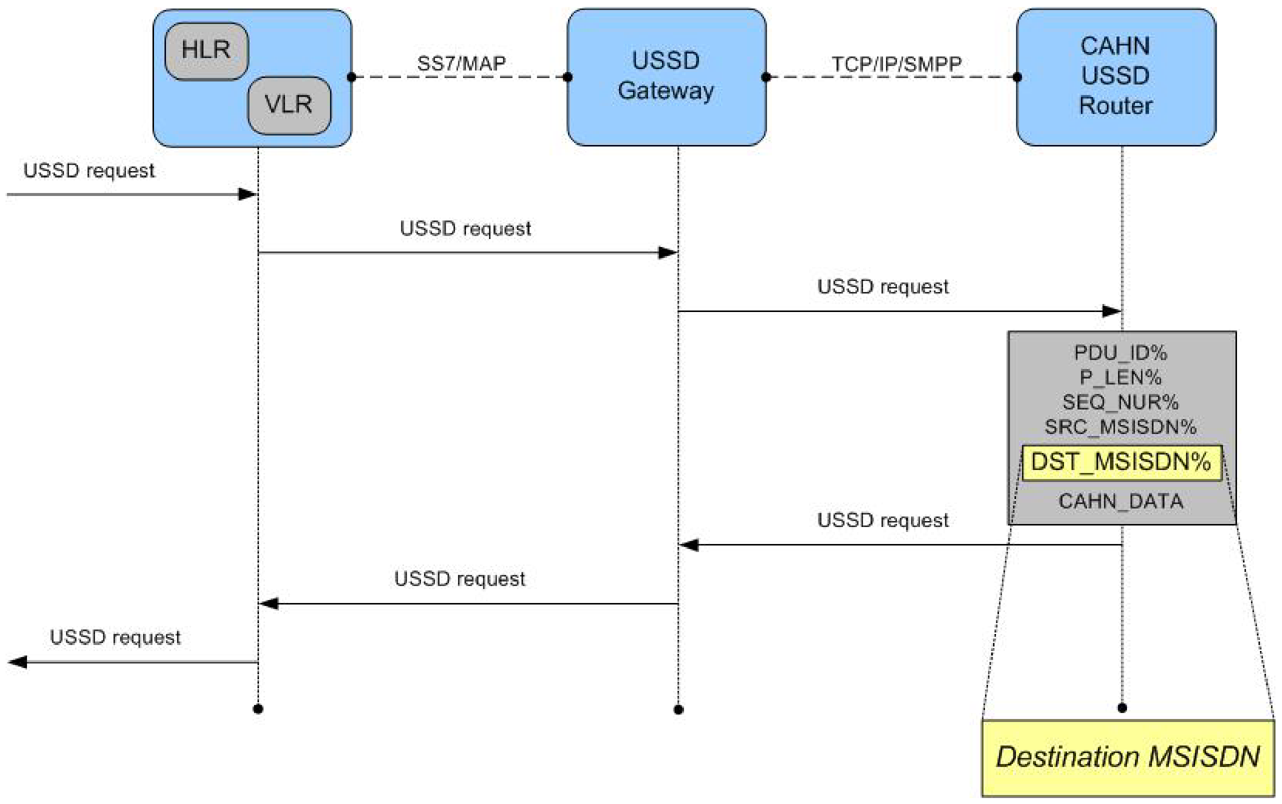


Figure 25: Message flow between CAHN USSD and operator.

The following diagram in figure 26 shows the entire message circulation between the components while sending and receiving CAHN messages.

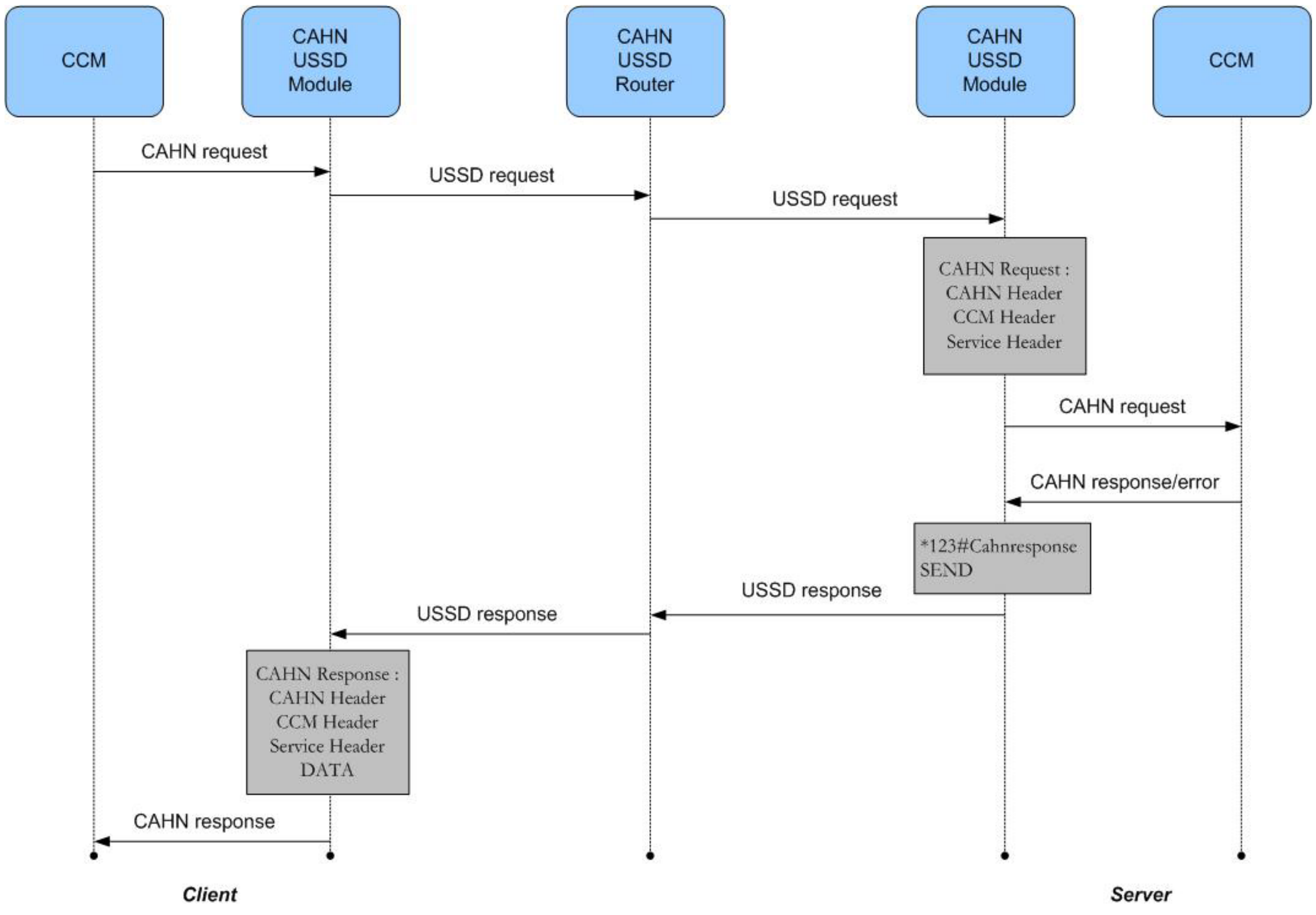


Figure 26: CAHN USSD message flow.

4.6 Formal description of CAHN including the USSD module

Appendix F presents the formal description of the CAHN protocol including the USSD module. The presence of the CAHN USSD router in the GSM network, as a part of the architecture is not visible to either of the CAHN end nodes. They just send and receive the messages via their USSD modules. The router acts as a black box somewhere in the network which makes the messages delivery among the CAHN nodes.

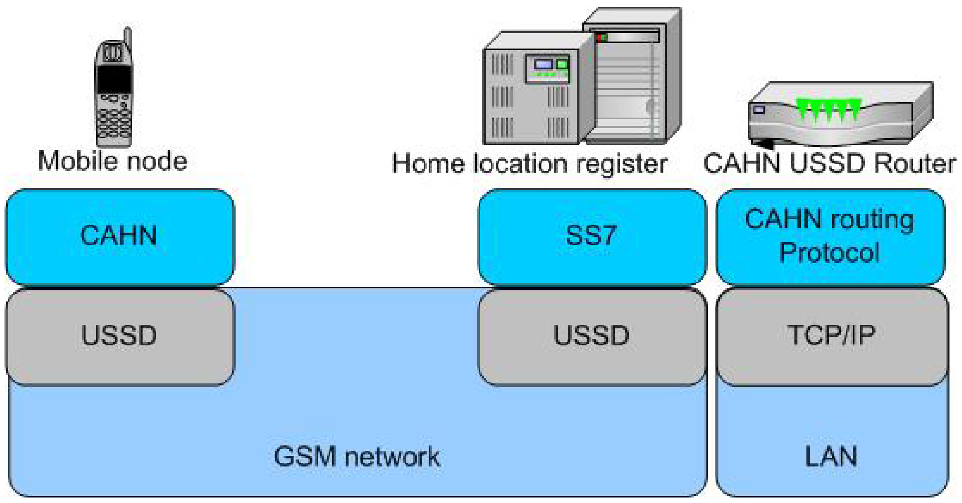


Figure 27: The total architecture of the CAHN USSD module in CAHN.

The architecture of the USSD module is similar to the architecture of the CAHN SMS module. The CAHN messages are saved locally and then exchanged over the GSM network. The main difference to the CAHN SMS architecture is the usage of USSD technology.

The next chapter tries to define a new CAHN protocol for the use with the WLAN [4] technology. The components of the CAHN Bluetooth concept, which can be reused for the CAHN WLAN protocol design will be applied for the enhancement of the CAHN protocol.

5 CAHN WLAN PROTOCOL

In this diploma work, it is assumed that the MSISDN of the destination mobile station is known in advance and only those privileged stations have the chance to open a secure link. Another constraint coming from the 802.11 specification is that all mobile stations in the network must share the same network identity (SSID) to be able to communicate together.

Motivation

The challenge is to offer a mechanism to form a scalable WLAN ad hoc or infrastructure network in a way that a CAHN station can easily join that CAHN WLAN network at any time. A CAHN station must be able to locate the destination station and has to be sure that the found station is really present in the respective network.

In the infrastructure mode, all stations talk directly to an access point. Inside the network, there is no direct link between the stations. In fact, every communication is made through peer-to-peer connections between the AP and a station. Therefore, the link to the AP is essential for any station. Direct connections between nodes are not possible, even if they want to communicate to other nodes. Since an AP and a station have setup a wireless link, the SSID of the entire infrastructure network is equal to the SSID of the AP. When a new station joins the infrastructure network, it must match the SSID of one of the APs.

But in the ad hoc mode, there is no access point (IBSS) and stations communicate directly with each other. With respect to CAHN, the most important feature in the ad hoc network is the scalability of an IBSS in a way that a CAHN ad hoc station can easily join the existing CAHN ad hoc WLAN at any time. The most important point is to determine which stations are present in the ad hoc network and whether an ad hoc network supports CAHN features and which stations are present.

The easiest way to solve this problem may be the concatenation of all MSISDNs in the ad hoc network. This would result in the SSID of the whole ad hoc network. Unfortunately, the size of the SSID is normally limited to 32 bytes. If the SSID is used as concatenation of the MSISDNs of all participating stations, it will cause a scalability problem when expanding the network. Assuming, a MSISDN (integer numbers) requires 7 bytes to be uniquely stored and the SSID containing the concatenation of all MSISDN. As a result, the SSID could represent a maximum number of 4 nodes. The network should be configured to support an ad hoc scenario in which more ad hoc stations are able to communicate. In other words, the problem is the efficient usage of the limited SSID field (figure 8). Therefore, a method has to be found which allows to save a maximum number of MSISDNs in the SSID field.

There are two main solution approaches for this problem: the Bloom filter solution and the random rendezvous solution. The Bloom filter solution is based on the probability theorem of Bloom filters. The random solution assumes that two stations could be at the same place at the same time.

All the mentioned aspects cause a splitting of CAHN wireless architecture into two separate scenarios: an access point scenario and an ad hoc scenario. Both of them adapt the SSID field to broadcast the MSISDN and the capability of being a CAHN device.

The CAHN WLAN protocol is divided in two small protocols: an access point and an ad hoc protocol. Each of these protocols needs its own message definition and architecture. The ad hoc scenario is again split into two sub scenarios with two different solution approaches. Therefore, there are three suggestions to implement the CAHN WLAN concept:

- The access point solution
- The random rendezvous solution
- The Bloom filter solution

5.1 The access point solution

The access point in a CAHN network has a predefined SSID string. A CAHN station, which wants to connect to a CAHN access point, does not need to know any extra information about the CAHN access point. The CAHN station scans its WLAN environment to find the specific CAHN access point SSID. The SSID of the access point contains the MSISDN and the specific CAHN string "*CAHN access point*" (figure 28). The CAHN station analyzes the access point's SSID. It extracts the MSISDN of the access point and sends a CAHN WLAN request. The access point treats the request and returns a CAHN WLAN response. Having received the response, the CAHN station can connect the CAHN access point.

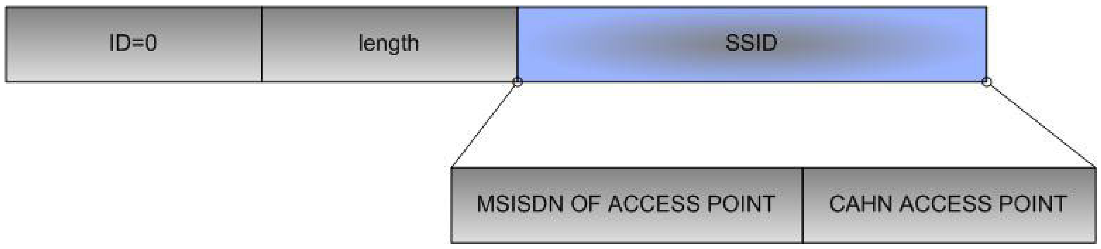


Figure 28: SSID of a CAHN WLAN access point.

Assuming that a CAHN enabled access point has a cellular card which enables it to communicate over the GSM network with other CAHN stations. In other words, the CAHN access point does not bother whether a station is in range or not. It responses to all incoming CAHN requests.

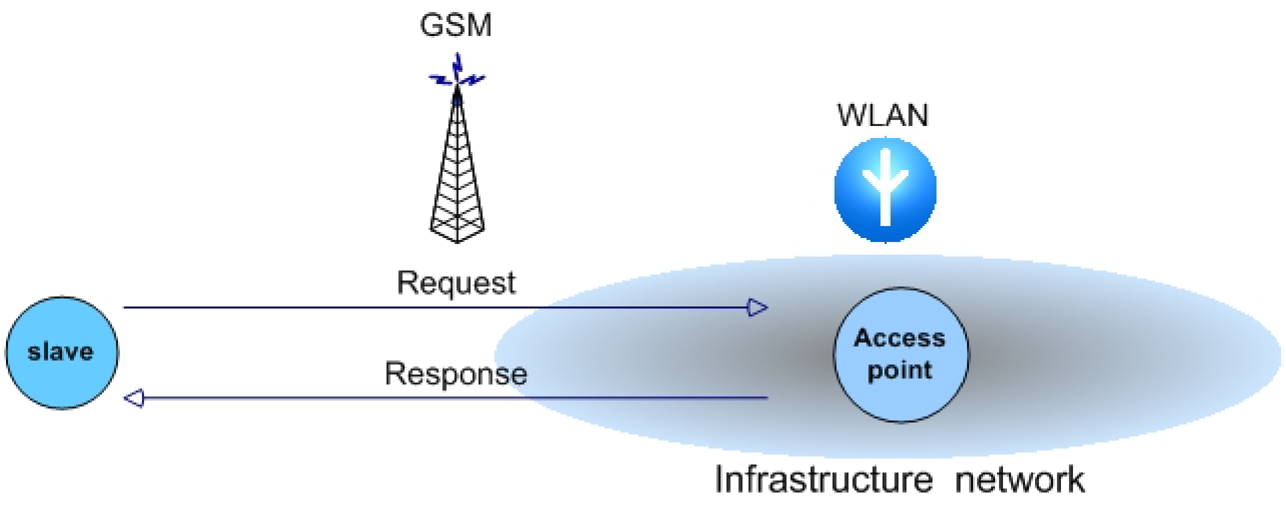


Figure 29: Message exchange in a CAHN WLAN infrastructure net.

CAHN WLAN access point protocol messages

The CAHN WLAN protocol for the access point scenario needs at least three types of messages: a connection request, a connection response and an error message (figure 30). The structures of all these messages are similar to those in the CAHN Bluetooth scenario. All these messages contain a header and a CAHN header. The CAHN WLAN request and the CAHN WLAN response messages have a WLAN header which includes the SSID of each node or the network. The CAHN WLAN response has also a data field which is reserved for the WLAN key. The exact description of each field is defined in the section 5.3.8 later in this chapter. The following figure presents all three CAHN WLAN access point messages. Because of the similarity to the above messages, no further description is necessary to describe these messages:

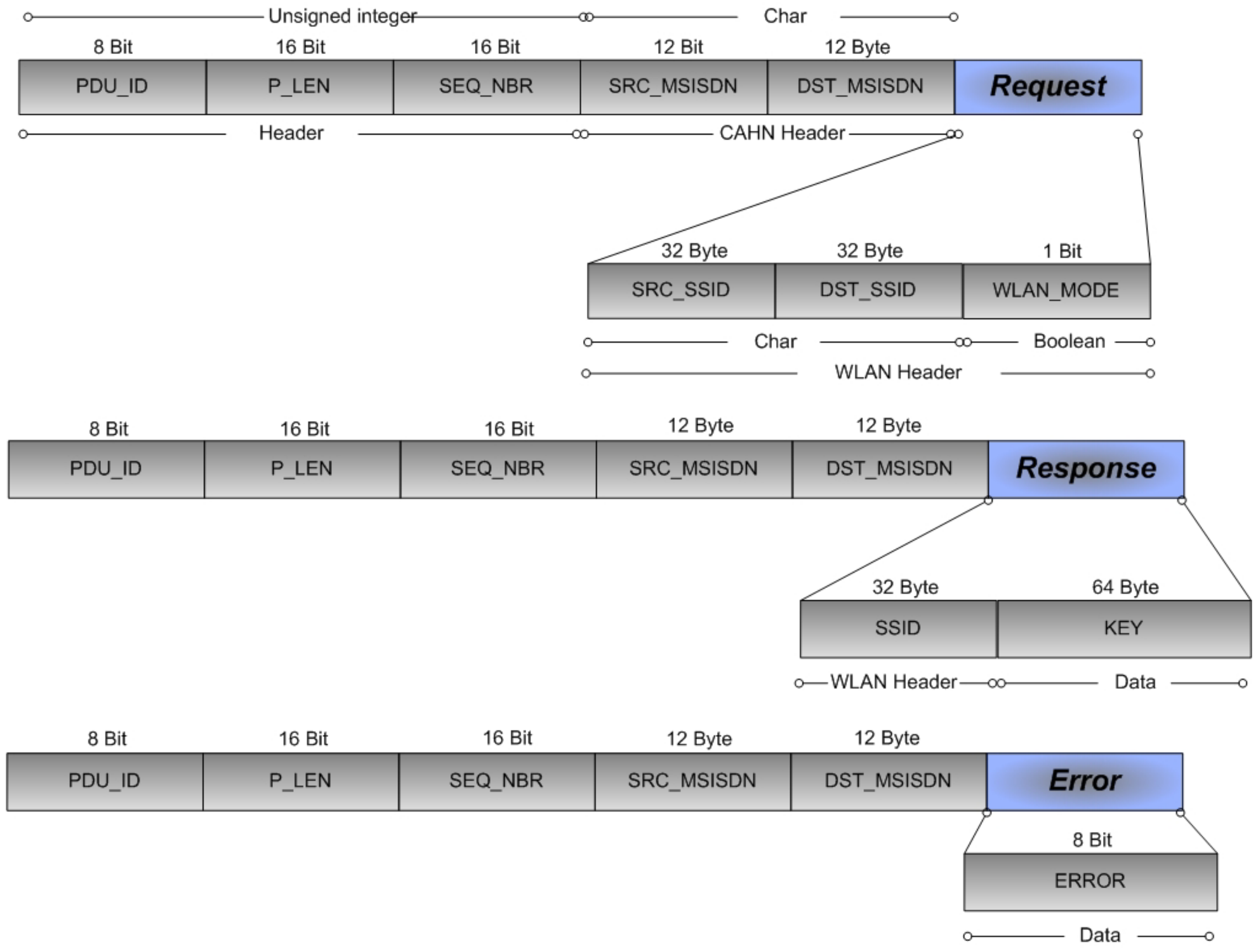


Figure 30: CAHN WLAN messages in access point mode.

The following sections 5.2 and 5.3 will present two approaches to solve the ad hoc CAHN scenario.. An ad hoc network is a local area network with wireless or plug-in connections, where some of the network devices are only temporary part of the network for the duration of a communication session or, in the case of mobile or portable devices, while in some close proximity to the rest of the network.

5.2 The random rendezvous solution

The random rendezvous idea is based on the probability that two stations may be at the same place at the same time. With other words, the random rendezvous solution is based on trial and error. The stations speculate the presence of the desired station in an ad hoc network. To verify, whether the station really is in the network, about the desired station is polled directly. In this case, there is no mechanism deployed to figure out whether a specific station is a member of an ad hoc network. The slave node which knows the MSISDN of the master node, analyzes the identity (SSID) of all existing networks in the direct range. The SSID of the ad hoc network is a fixed and predefined string. If there is any ad hoc network with the supposed SSID text like "*CAHN ad hoc*" (figure 32), the slave node guesses that the desired station probably is also present in that network. The slave node sends a CAHN WLAN request to the master node via the cellular network. The request contains two main values: the MAC address of the slave node and a challenge (WEP) key.

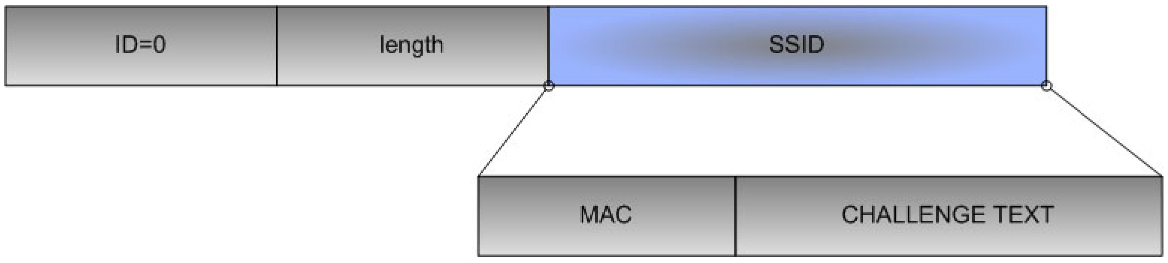


Figure 31: SSID of a CAHN slave node.

After sending the request, the slave node broadcasts its SSID for a certain time interval (timeout). The SSID contains the MAC address of the slave node and a challenge text (figure 31). The master node tries to find the slave node by scanning its environment and analyzing the SSIDs to find out whether the slave node is in direct communication range or not. If the SSID of a station contains the MAC address which matches to that one in the request message, the master node knows that communication is possible. The master node signs the challenge text using the challenge key received within the request message and returns a CAHN WLAN response containing this signed challenge text. The slave node knows both challenge key and the challenge text and can therefore check the signed challenge. If this check is successful, the slave node is definitely sure that the proper master node is in the range. The slave node now uses the challenge key in the CAHN WLAN response message and joins the network.

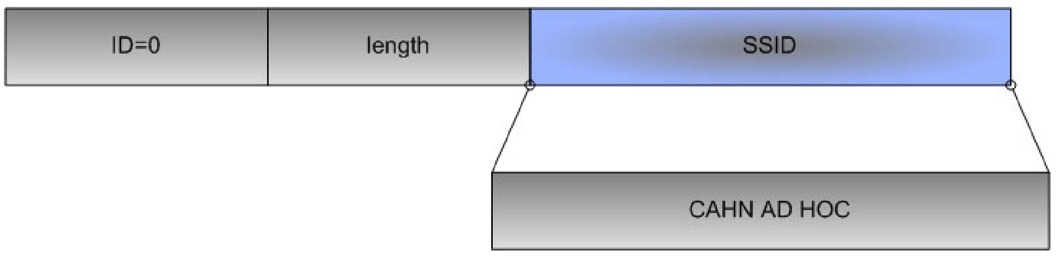


Figure 32: SSID of a CAHN ad hoc net.

This solution needs no more than 3 protocol messages because the SSID is predefined and there is no need to update it after joining the network. The main problem of using this solution is that a slave node can not figure out whether the desired station is in an ad hoc network without asking directly (prior sending a request over the cellular network).

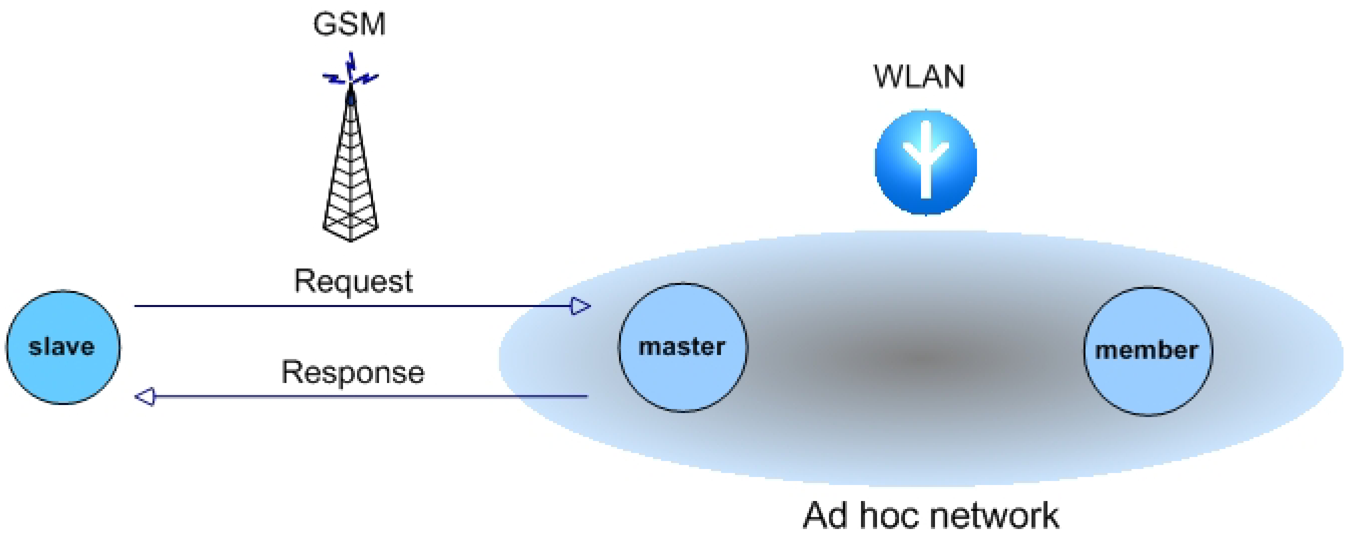


Figure 33: Message exchange in a CAHN WLAN ad hoc network without Bloom filter approach.

CAHN WLAN ad hoc protocol messages

The CAHN WLAN ad hoc protocol must offer at least three types of messages: a connection request message, a connection response message and an error message.

CAHN WLAN ad hoc request message

This cellular message (figure 34) is used to initiate a WLAN connection between a slave and a master node. The CAHN slave node starts a WLAN ad hoc connection by sending a CAHN WLAN request to the master node. Instead of WLAN header, the request message contains a MAC header which is used to prove the identity of slave node. The following figure illustrates a CAHN WLAN ad hoc request message:

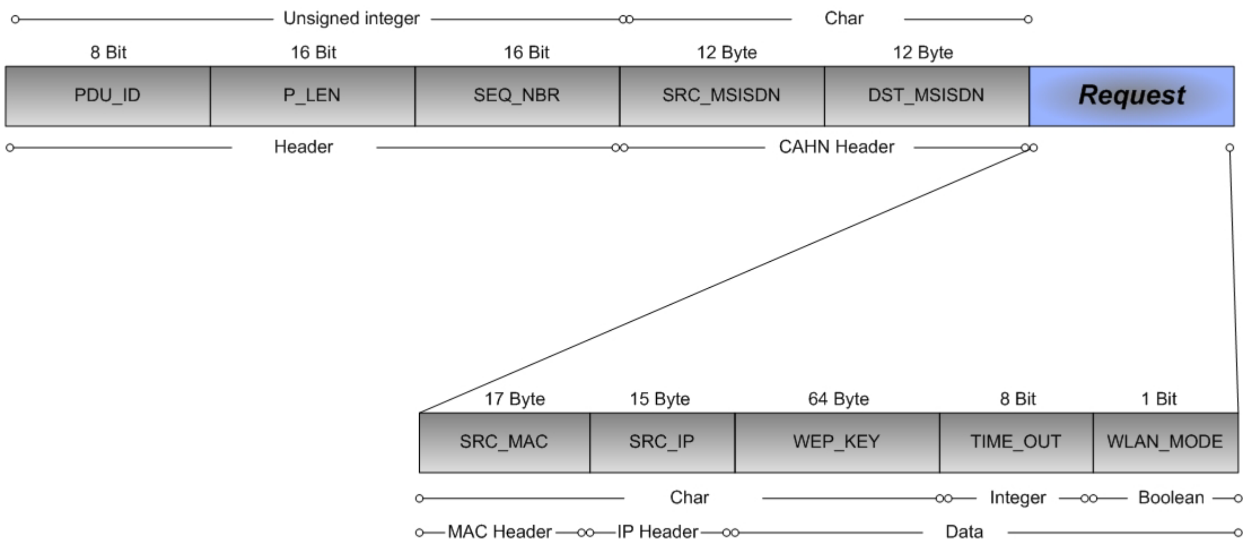


Figure 34: CAHN WLAN ad hoc request message.

- SRC_MAC: source MAC address of sender node (slave node)
- SRC_IP: source IP address of the slave node
- WEP_KEY: a key for encryption of the challenge text (also called CHALLENGE_KEY)
- TIME_OUT: the slave node broadcasts its SSID during this time interval
- WLAN_MODE: ad hoc or infrastructure mode (optional)

CAHN WLAN ad hoc response message

This message (figure 35) is responsible for ad hoc connection setup. The CAHN WLAN ad hoc message has an additional field which contains the signed challenge text. The slave node checks if this field is correctly encrypted:

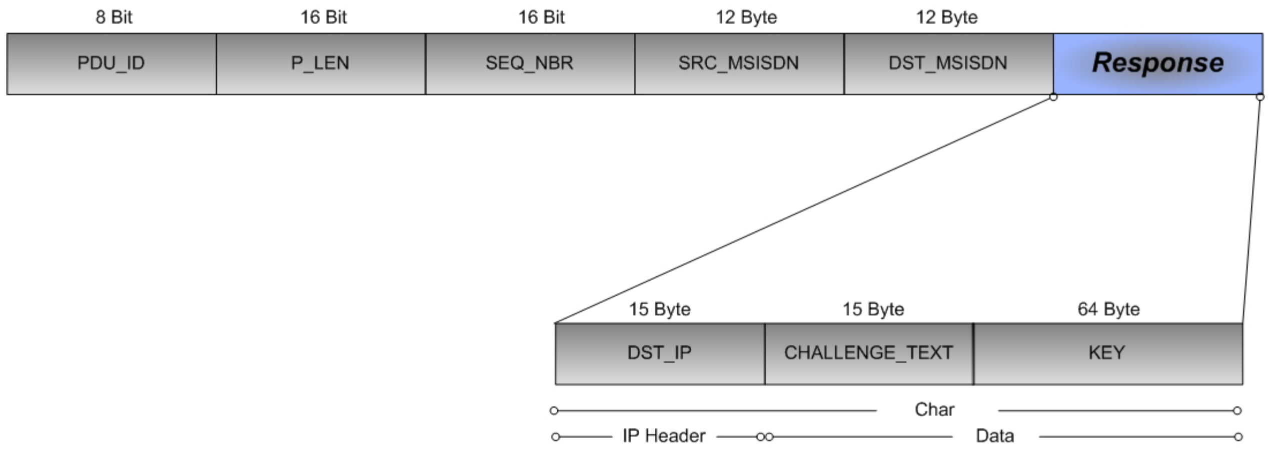


Figure 35: CAHN WLAN ad hoc response message.

- DST_IP: the IP address of the master node

- CHALLENGE_TEXT: encrypted challenge text by using the CHALLENGE_KEY of the slave node

- KEY: a WEP key or a shared key for the 802.11 authentication and encryption

CAHN WLAN error message

In a CAHN WLAN ad hoc scenario, there are two kinds of errors: CAHN specific errors and false positive error. CAHN specific errors are CAHN internal errors like time-outs. A False positive error happens when the master and slave nodes are not in the necessary physical range or when the challenge text is not correctly signed.

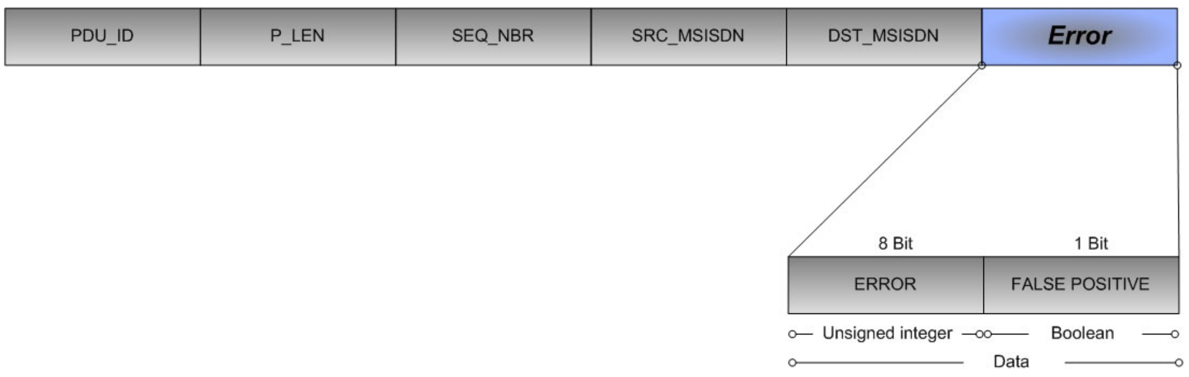


Figure 36: CAHN WLAN error message.

- ERROR: CAHN WLAN errors codes

- FALSE_POSITIVE: the station is not in the ad hoc network

5.3 The Bloom filter solution

In this scenario, it is assumed that the setup of a connection should only be possible if the destination MSISDN is known in advance.

Motivation

The questions at stake are how to know which stations are in an ad hoc network and whether the searched destination node belongs to the network. The idea is, that each station uses a special filter method called Bloom filter to announce its MSISDN within the SSID. A station, which wants to join a CAHN WLAN ad hoc network, sets its SSID by applying the Bloom filter method to its MSISDN. It means the station uses a specific hash value of its MSISDN as the SSID. Then the station looks for other stations with specific MSISDNs by analyzing all scanned SSIDs in its range. In case of success, the station communicates with the desired station or joins the desired network. The SSID of the entire ad hoc network must be adapted after each joining process. The value of the new SSID has to be adapted to the value generated with the Bloom filter by using new MSISDNs.

The goal of the using Bloom filter in CAHN is to reduce the specific CAHN information (MSISDN) and to promote the CAHN ability to other stations. This means that the CAHN stations use the Bloom filter concept to adapt the MSISDNs in a way that they fit the size of a standard SSID. The Bloom filter concept makes it also possible that the CAHN stations can guess whether a specific station belongs to a network or not.

5.3.1 False positive for CAHN

As mentioned in chapter 2, the false positive plays a central role for CAHN applications. The choice of proper parameters k and m for the use of Bloom filter with respect to CAHN, influences the performance of CAHN WLAN applications. For this purpose, the false positive is analyzed under variation of the parameters.

With respect to CAHN, the size of the vector v is usually equal to the size of the SSID. In this case, the size of the vector v is 256 bits long and therefore, the optimal false positive α would be:

$$\ln \alpha = \frac{256}{n} \ln(0.6185) \Rightarrow \alpha \approx e^{-\frac{123}{n}}$$

In a realistic CAHN WLAN scenario, where the numbers of participating nodes is known, it is easy to calculate how many hash functions are really needed. Also the resulting probability of a false positive can be calculated. For example, if $n = 10$ then the number of required hash functions would be $k = 18$ and the probability of false positive is: $\alpha \approx 1.08^{-3}$

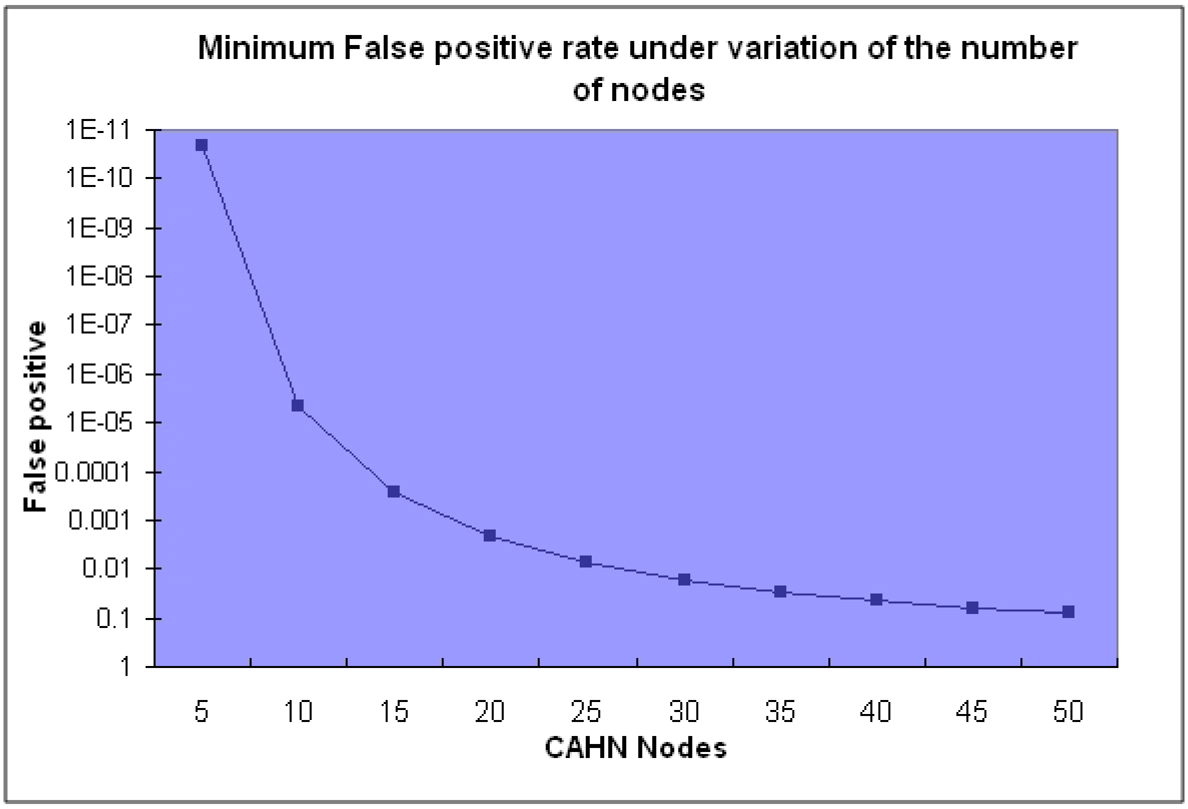


Table 1: Shows the minimum false positive under variation of the number of CAHN WLAN nodes.

When a new station joins the ad hoc network, all stations must choose a new set of hash functions (according to the new number of nodes in the network). Recalculation of the SSID using a various number of hashes is probably not a practical solution, because all stations have to calculate the SSID every time a new station joins the network. To avoid this, a particular number of nodes can be used as a threshold for the CAHN WLAN protocol. If the threshold is outrun, the stations automatically go to the next predefined threshold. As an acceptable threshold for CAHN WLAN, $n = 10$ can be used as a default value to choice the hash functions. As a result, as long as the number of CAHN nodes in an ad hoc network is less than 10, the number of used hashes does not change.

Using Bloom filters with reference to ad hoc networks has also some unwished effect. There is no mechanism in the Bloom filter for turning bits off (set to zero). If an ad hoc station leaves the ad hoc network, other stations in the same ad hoc network do not realize the absence of one specific station because the network identity is still the same as before. Another aspect is that any station can fake the SSID of an ad hoc network by setting all its bits to 1 and thus, preventing to include all of the nodes queried for.

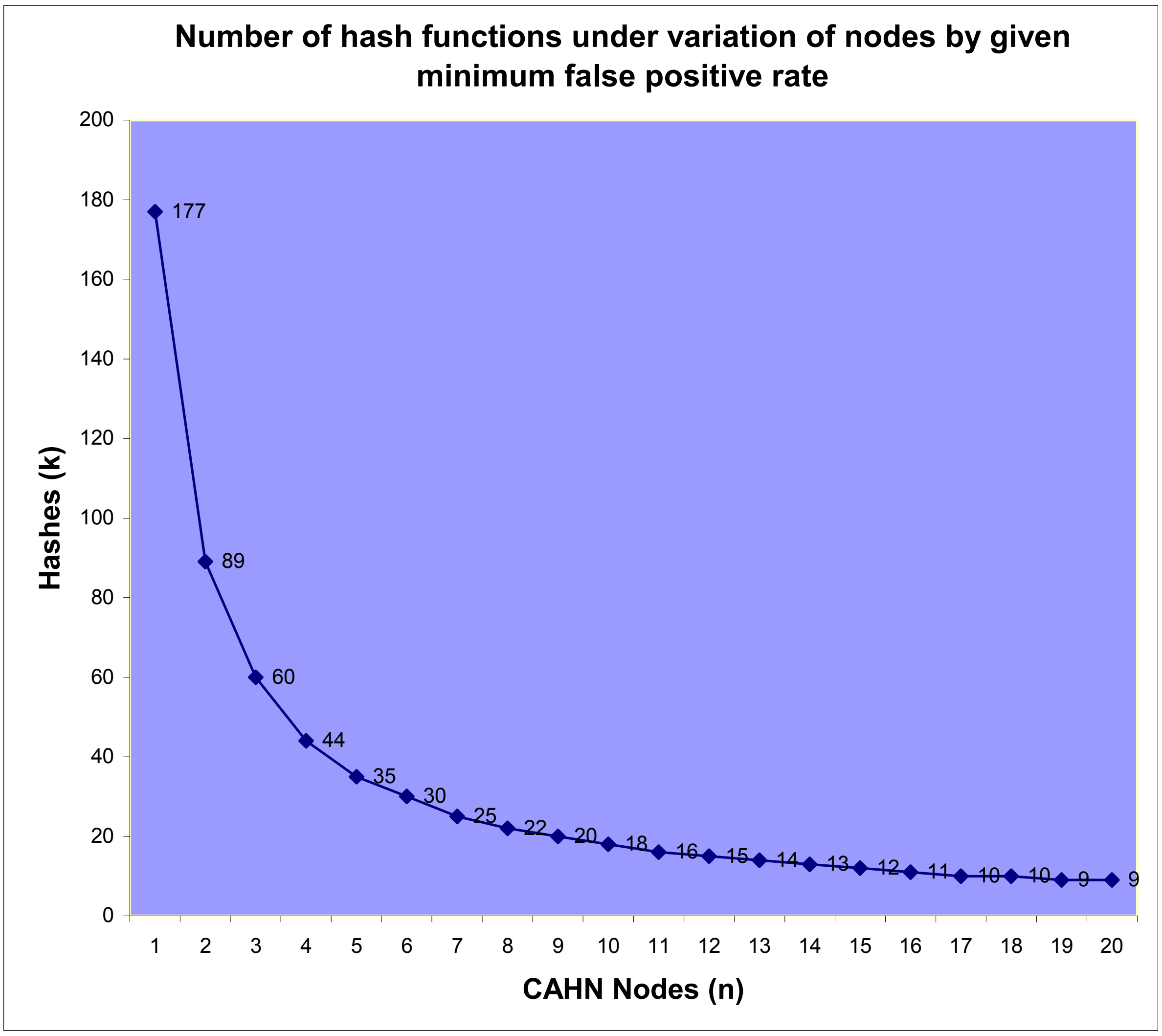


Table 2: Number of hash functions under variation of CAHN nodes by given minimum false positive rate.

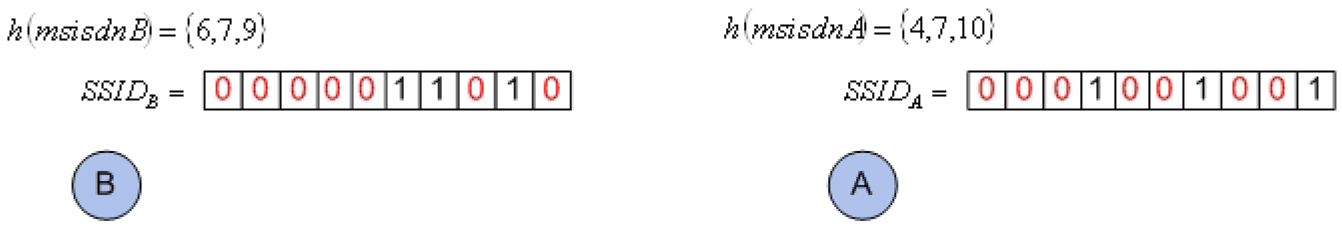
To get a sense of how the error rate and the number of keys affect the memory size of Bloom filters, appendix D lists some sample vector sizes for a variety of capacity and error rate combinations.

As a result of above analysis, for the CAHN application, a counting Bloom filter with 3 bits per counter would be an appropriate choice. The CAHN nodes can calculate their Bloom SSIDs offline by choosing the right number of hash functions. The following table shows three thresholds with the respective number of needed hashes and the approximate value of false positives for CAHN WLAN applications:

Number of nodes (Threshold)	Number of hashes	False positive
10	18	0.00001
15	12	0.0001
20	9	0.001

5.3.2 Using Bloom filters for CAHN

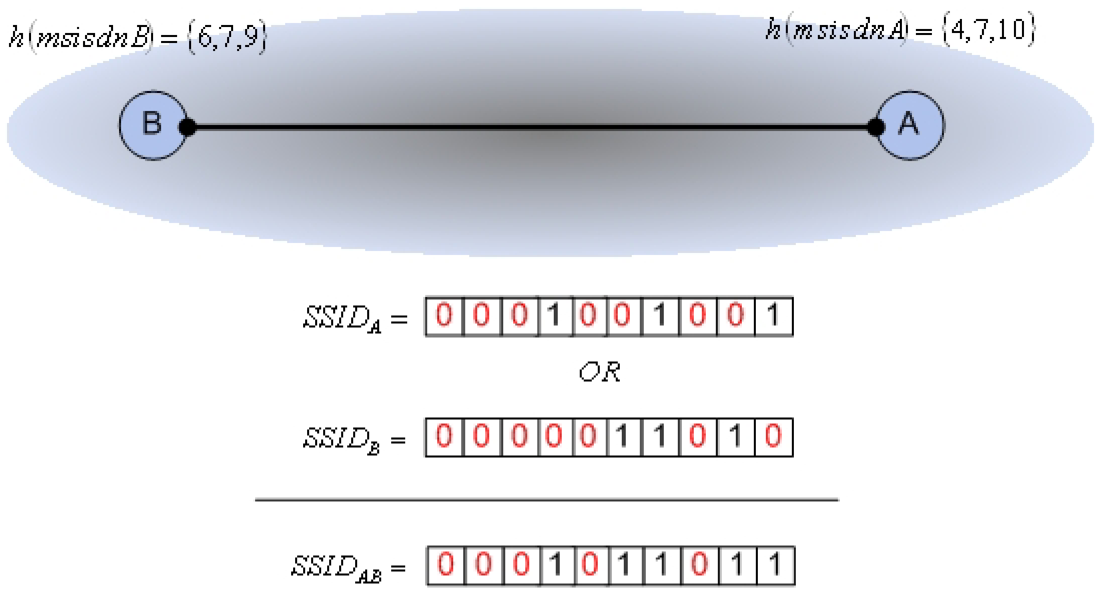
Each CAHN node uses a Bloom filter to represent its MSISDN in the SSID. A CAHN node uses a set of hash functions to calculate the hashed (Bloom) value of its own MSISDN and then sets this value as its SSID. All other CAHN nodes use the same set of hash functions.



$h(msisdnB) = \{6,7,9\}$
$SSID_B = 0\,0\,0\,0\,0\,1\,1\,0\,1\,0$
B
$h(msisdnA) = \{4,7,10\}$
$SSID_A = 0\,0\,0\,1\,0\,0\,1\,0\,0\,1$
A

Figure 37: Stations A and B use the same Bloom filter to hide the MSISDN.

If a CAHN node A (called slave node) wants to join a CAHN ad hoc network or a specific CAHN node B (called master node), it calculates at first the Bloom value of the desired node B using the MSISDN of the node B (assumed that the MSISDN of B is already known). After this process, the CAHN node A scans the location to find out whether the SSID of the node B is available.



$h(msisdnB) = \{6,7,9\}$
$h(msisdnA) = \{4,7,10\}$
B
A
$SSID_A = 0\,0\,0\,1\,0\,0\,1\,0\,0\,1$
OR
$SSID_B = 0\,0\,0\,0\,0\,1\,1\,0\,1\,0$
$SSID_{AB} = 0\,0\,0\,1\,0\,1\,1\,0\,1\,1$

Figure 38: Shows the situation before joining station A. The station A wants to communicate with station B. The station A knows the MSISDN of the station B and sends a CAHN WLAN request to station B.

The node A compares the calculated Bloom value with all scan results. If there is a match between the Bloom values and the SSIDs, the CAHN node A has found the CAHN node B or the CAHN node B is a member node of an existing CAHN ad hoc network. The slave node now is able to send a CAHN WLAN request message through its cellular device.

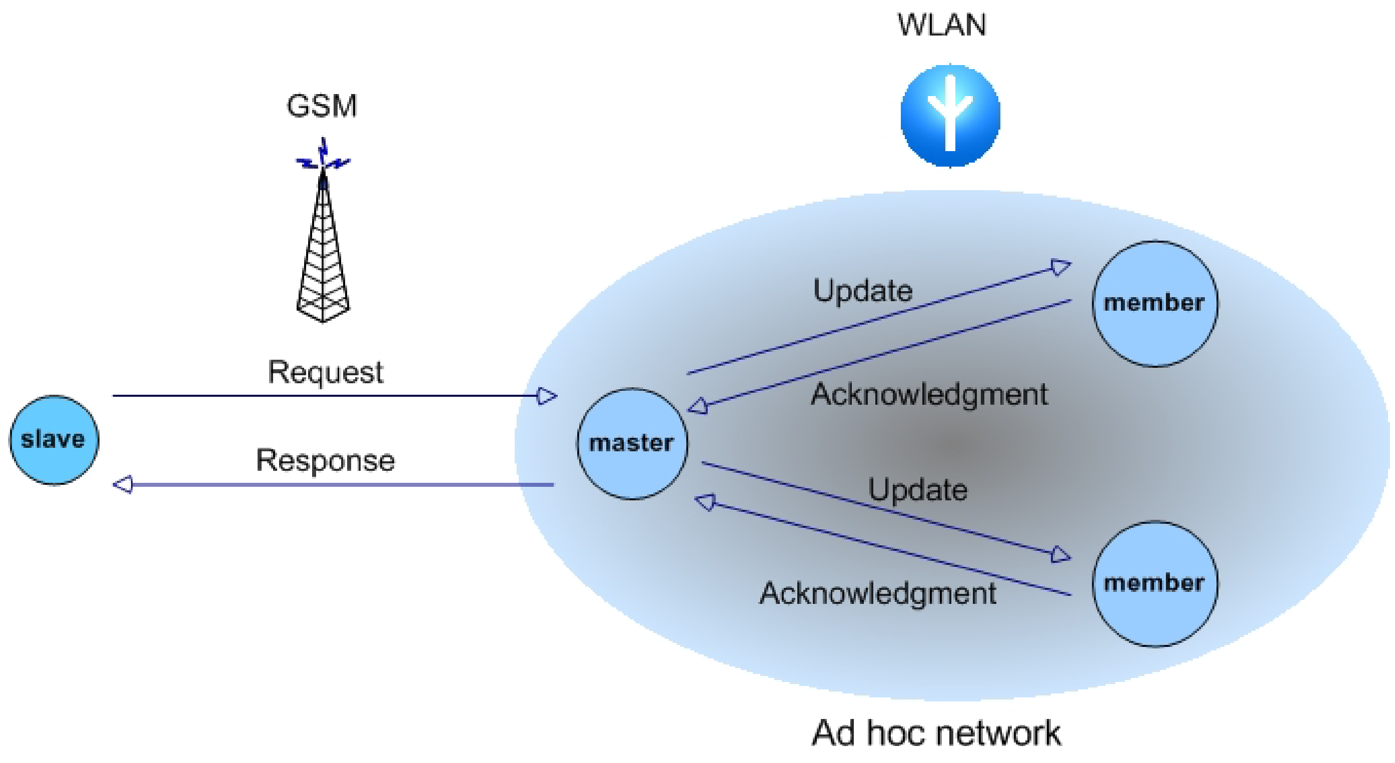


Figure 39: Message exchange in a CAHN WLAN ad hoc network.

The master node B needs to calculate the changed bits of the Bloom Filter and sends (distribution of SSID) them to the other mobile stations in the network (called members). As soon as all member nodes have sent a CAHN WLAN acknowledgment message the master node sends a CAHN WLAN response message to the slave node. After that, the slave node can join the CAHN ad hoc network by setting its SSID and using the WEP key received with the response from the master node.

The following figure shows a CAHN ad hoc network before the joining process. Station A wants to join the network and knows the MSISDN of the station B. Station A calculates the Bloom value of the station B locally and scans its environment to check whether station B is in the range. The station A discovers that station B is in the network and sends a CAHN WLAN ad hoc request to the station B:

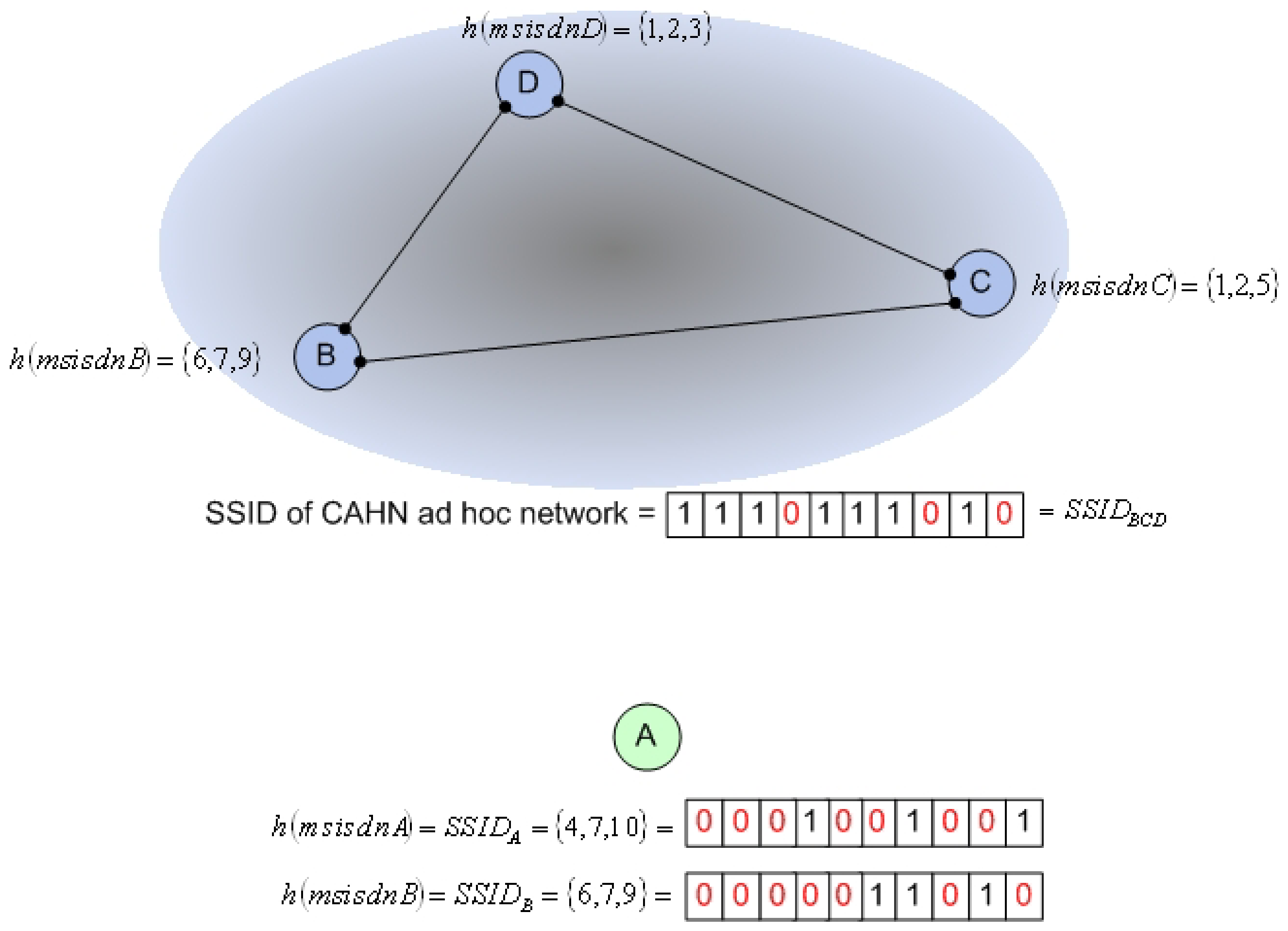

$h(msisdnD) = \{1,2,3\}$
D
$h(msisdnC) = \{1,2,5\}$
C
$h(msisdnB) = \{6,7,9\}$
B
SSID of CAHN ad hoc network = 1 1 1 0 1 1 1 0 1 0 = $SSID_{BCD}$
A
$h(msisdnA) = SSID_A = \{4,7,10\}$ = 0 0 0 1 0 0 1 0 0 1
$h(msisdnB) = SSID_B = \{6,7,9\}$ = 0 0 0 0 0 1 1 0 1 0

Figure 40: CAHN WLAN ad hoc before joining.

Now station A is in the network and adapts its SSID to the SSID of the entire network. All other stations in the network also adapt their SSID according to the new SSID that is sent out by station B through the WLAN link. The following figure shows the ad hoc network after the joining process:

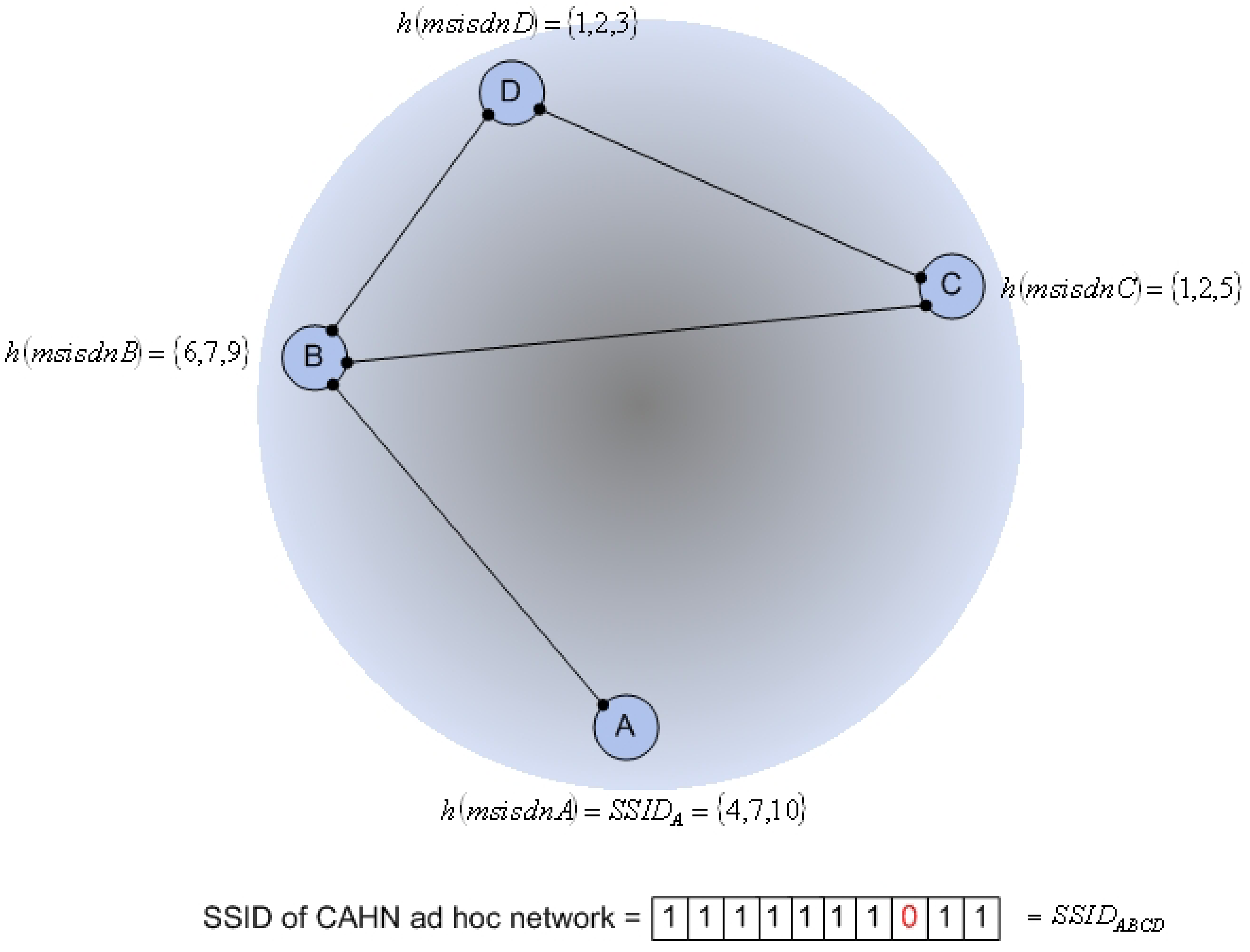

$h(msisdnD) = \{1,2,3\}$
D
C
$h(msisdnC) = \{1,2,5\}$
$h(msisdnB) = \{6,7,9\}$
B
A
$h(msisdnA) = SSID_A = \{4,7,10\}$
SSID of CAHN ad hoc network = 1 1 1 1 1 1 1 0 1 1 = $SSID_{ABCD}$

Figure 41: CAHN WLAN ad hoc after joining.

5.3.3 Adapting the SHA hash algorithm for CAHN

A high quality set of hash functions is needed in order to build a working Bloom filter for the CAHN. A good choice for this purpose is SHA1 [21]. This hash method can be used to create as many hash functions as needed by appending the input with a list of distinct values in the form of integer numbers:

$$sha_i(ix), \quad \forall x \in v, \quad 1 \leq i \leq k$$

To be able to use these hash functions, CAHN had to find a way to control the range of SHA1. SHA1 returns 160 bits of hashed output, useful only in the unlikely case that the v vector is 2^{160} bits long. In case of CAHN, the vector is supposed to be 256 bits long and therefore, the hash functions have to return a value between 1 and 256. Hence, the size of output must be matched to the size of SSID. This can be achieved by a combination of bit chopping and division to scale the output down to a more usable size (module method).

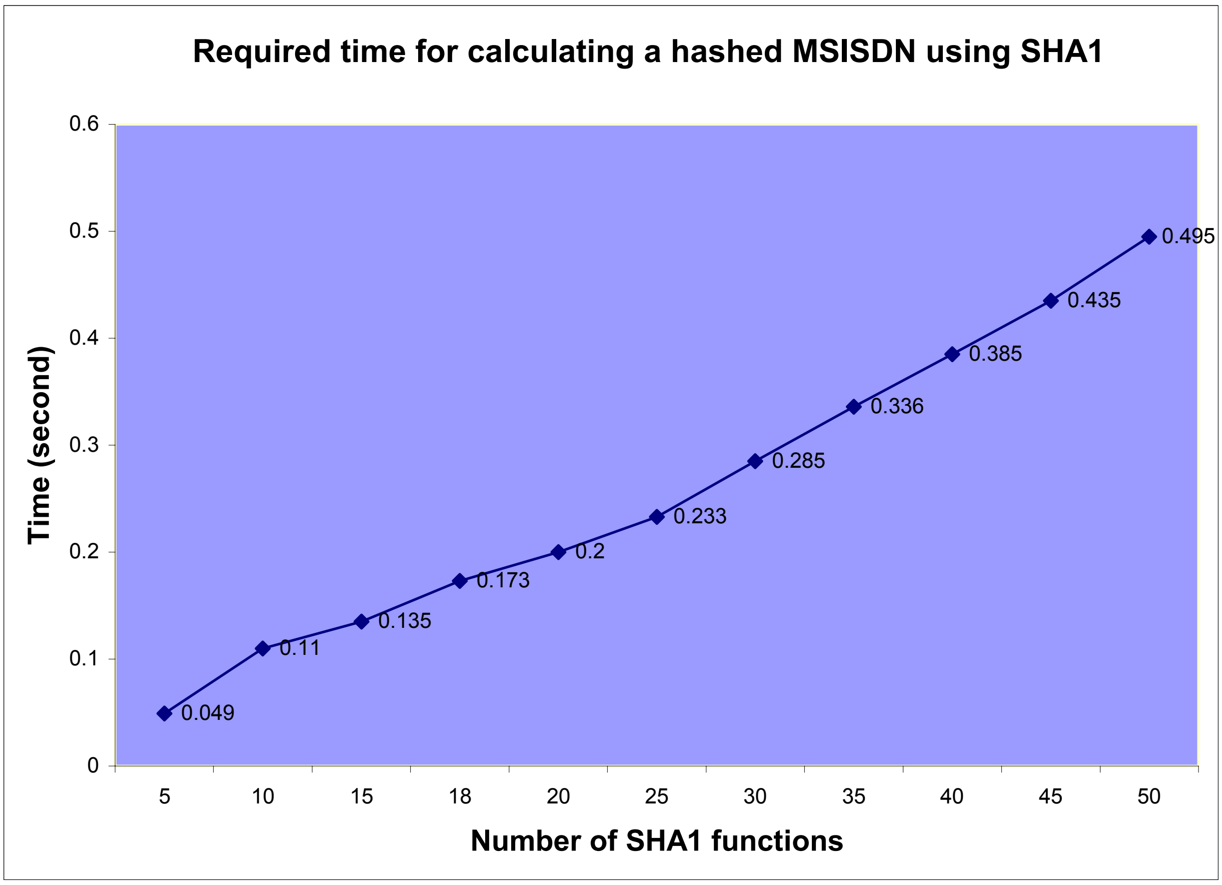


Table 3: Shows that the required time to calculate 18 hash values for a given MSISDN of 9 bytes long averages 0.173 second. As mentioned, 18 is a threshold default value for CAHN WLAN protocol for the number of used hash functions. Test bed: Intel Pentium centrino with 1.3GHz and 128 MByte of RAM under Linux Red Hat 9.0 with kernel 2.6. Appendix H presents the simulation result.

The Chopping or splitting method (figure 42) helps to reduce the size of the function's output. Depending on the size of the vector v, Bloom filters can keep a specific number of output bits and discard the rest of them. In this case, the size of SSID is 256 bits long. The Bloom filter takes the first 9 bits of the output and removes the rest. Thus, it can characterize 2^9 positions in the vector v (table 3). For better security, the hash value can be split into sixteen 10 bits pieces which are combined by the "XOR" function. Like this, all the information of the original hash is preserved.

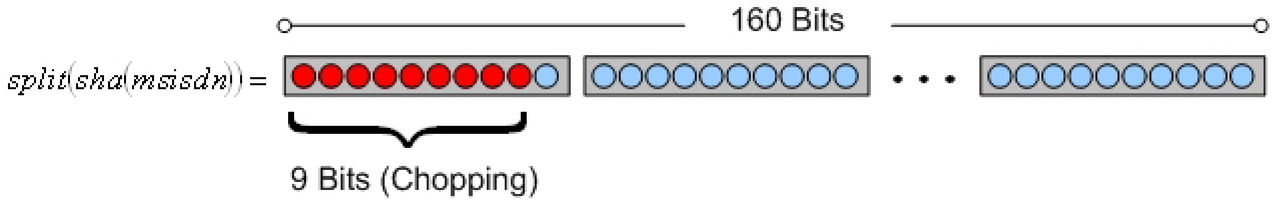


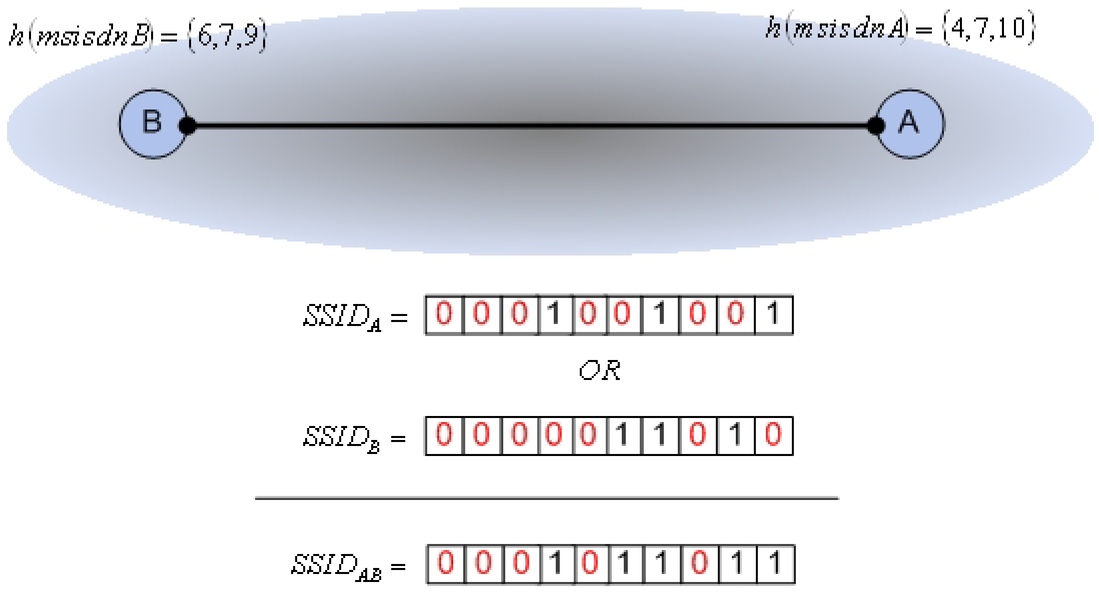
Figure 42: Adapting the SHA for the CAHN.

If the size of the vector v is not a power of two (2^x), then the size of hashes must be scaled down to their real range. The operator "modulo" scales them down to the wished length of 256:

$$(split(sha_i(ix)))\%SCALE, \quad \forall x \in v, \quad 1 \le i \le k, \quad SCALE \in \aleph$$

5.3.4 SSID distribution

In fact, after the joining procedure all, stations in the ad hoc network must have the same SSID and in this case, the same Bloom filter value. Otherwise, the stations can not find each other and the new SSID must be distributed to all existing nodes.



$h(msisdnB) = \{6,7,9\}$
$h(msisdnA) = \{4,7,10\}$
B
A
$SSID_A$ = 0 0 0 1 0 0 1 0 0 1
OR
$SSID_B$ = 0 0 0 0 0 1 1 0 1 0
$SSID_{AB}$ = 0 0 0 1 0 1 1 0 1 1

Figure 43: The SSID of the ad hoc network is equal to the union of all SSID of the stations in the network. This can be achieved by using the mathematical operator "OR". The ad hoc network contains two stations with two different hash values. After joining, both stations have the same SSID.

When a new mobile node (slave node) enters the CAHN ad hoc network, the corresponding Bloom Filter has to be updated by setting the matching hashed bits to one (figure 43). The asked node (master node) needs to calculate the changed bits of the Bloom Filter and sends (using an update message via WLAN) them to the other mobile stations in the network (distribution of SSID). The update message contains the new SSID of the ad hoc network. It is important to keep in mind that when a station leaves the network, the corresponding bits in the Bloom filter can not be simply reset to 0 because other stations may be hashed to some of those bits, which would result in resetting bits of still existing participants. To avoid this problem, a counting Bloom filter has to be used.

Each node in a CAHN ad hoc network has a list of stations. This list contains information about the number of participating CAHN nodes in the network at the moment. Furthermore, the MSISDN of those stations which are only connected to this specific node are stored as well (figure 44). The asked node updates the list by adding the MSISDN of the new node to its list and increments the number of nodes. It is important for all stations in the ad hoc network that they know how many stations take part in the network, because the choice of the number of hash functions depends on the number of nodes. Whenever, a new station joins the ad hoc network, all the present stations increment the number of nodes in their lists and take another set of hash functions, using the Bloom filter formula. As mentioned before a threshold value was defined to avoid too much calculation overhead due to continuous recalculation of the hashes. When this threshold is exceeded, the CAHN WLAN protocol takes the next valid threshold.

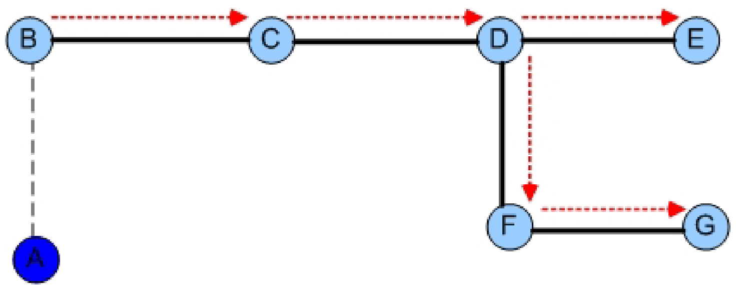


Figure 44: SSID distribution in a CAHN ad hoc network. Station A wants to join the ad hoc network. The master station B sends the new SSID to other nodes in the network. The other nodes also distribute the new SSID.

The asked node sends an update message to all stations that are in the list. The receiving stations immediately update their SSIDs. As a result, the connections between the stations are broken for a specific time period. To keep that outage as small as possible, the stations also return an update message to signalize that they are ready to change their SSID (and set the status bit to 1). If at this time two stations are communicating, they wait until the communication is over and the wireless medium is free. Because both stations are idle, they set the status bit to 1 and return the update message. As soon as all the update messages are back at the corresponding stations, the master node can send a response message to the slave node and setup the updated ad hoc network.

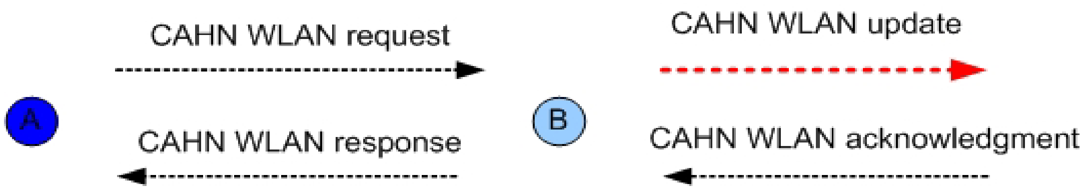


Figure 45: Message flow between CAHN nodes by joining a new station.

5.3.5 Routing

There is no routing mechanism in the 802.11 ad hoc WLAN. Suppose two CAHN stations want to communicate but are not in direct communication range. Thus, the stations can not setup an IP link. In a multiple WLAN scenario where various stations are not in physical range of each other, it is necessary to use an ad hoc routing algorithm that relays the CAHN messages between nodes in an ad hoc network to successfully establish IP links.

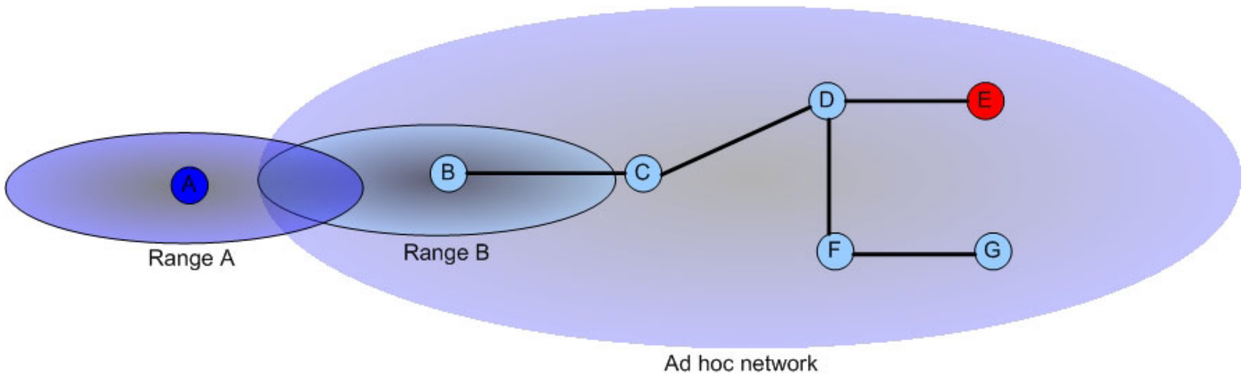


Figure 46: Station A wants to communicate with station E. The scan result of ad hoc network shows that station E is in this network but not in direct range of A. hence, If station A sends a CAHN WLAN request to station E over GSM network, it will receive a CAHN error response.

There are many possible routing algorithms which can be used for this purpose. Even if at this time the routing problem is not of interest for this diploma work, it will certainly be topic of future work.

5.3.6 Security problem with Bloom filters

The physical communication range can cause another problem. If two stations are not in direct communication range, a third station, which is in between these two stations, can invoke an unsuccessful communication setup. A station can sniff the SSID of a slave node or fake the SSID by setting all bits to one. Since any SSID check based on the Bloom filter approach would result in a false positive, the slave sends a CAHN request to the master. The master node checks if it can find the slave. The master finds the attacker node and thinks that it is the slave node, because the attacker node provides the correct SSID (figure 47). The master node sends a response to the slave node but the slave node can not make a connection.

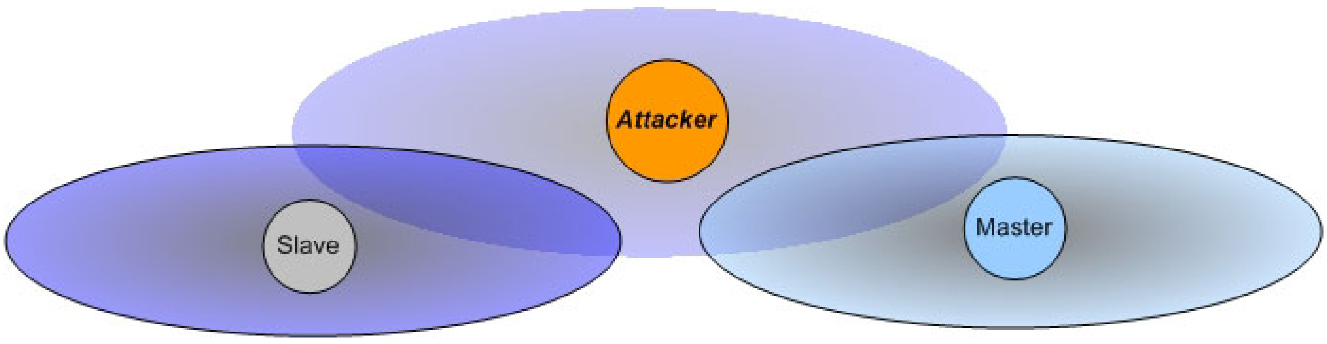


Figure 47: A thinkable attack scenario for CAHN ad hoc network.

5.3.7 CAHN ad hoc scenario with Bloom

As mentioned, a CAHN enabled node has a cellular card that makes it possible to communicate over the GSM network with other CAHN stations. A CAHN WLAN ad hoc network contains three kinds of nodes:

- The master node is a node that receives a CAHN WLAN request
- The slave node is a node that wants to join a CAHN WLAN ad hoc network
- The member node is a node that is already in a CAHN WLAN ad hoc network

A slave node looks for a specific CAHN node to communicate with. The slave node knows the MSISDN of the desired CAHN node in advance. The slave node checks the SSID of all networks in its communication range. It then compares the Bloom value of the desire node with the scan result to find the right network or station. If the Bloom values match, the slave node assume that the desired node is really in communication range. The slave node now sends a CAHN WLAN request to the master node. The master node accepts the request and broadcasts a CAHN update message. The member nodes return a CAHN acknowledgment message to the master node and immediately update their SSIDs. As a consequence of updating the SSID, the connection between member nodes and the master node will be broken, until the new SSID is applied by every node.

When the master node receives the acknowledgment message, it sends a CAHN WLAN response to the slave node and also updates its SSID (figure 48).

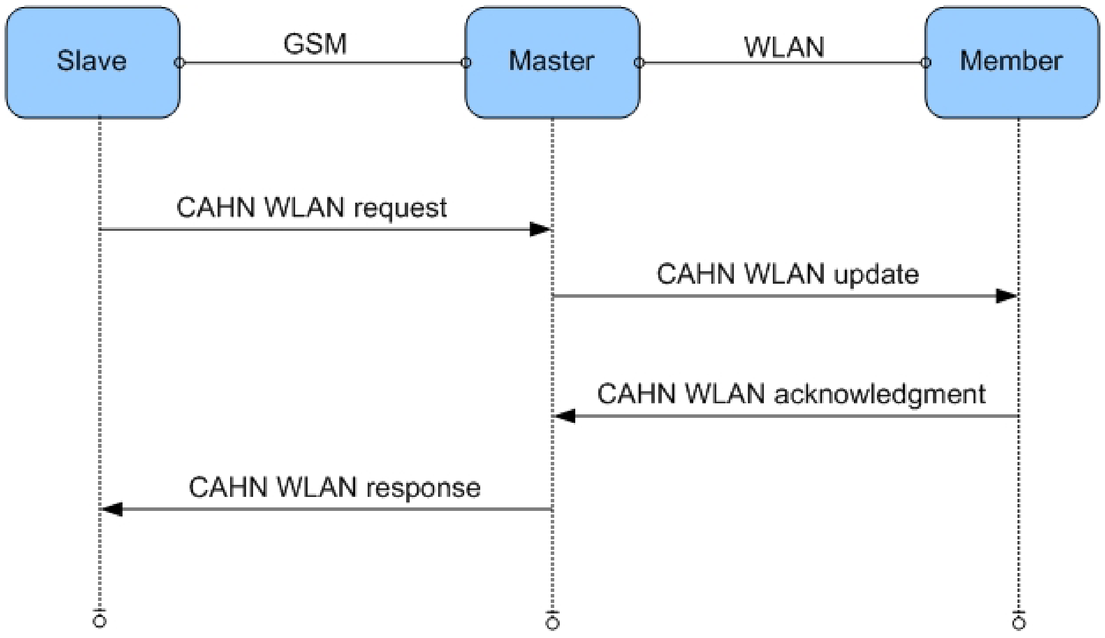


Figure 48: CAHN WLAN ad hoc message flow.

All in all, the order of the processes in a CAHN Bloom ad hoc scenario would be as follow:

- The slave node calculates and sets its SSID offline, using the appropriate number of hash functions. The slave node knows the MSISDN of the desired CAHN master node and it calculates also the hashed value of MSISDN of the master node

- The slave node scans its vicinity to find CAHN enabled devices (SSID of CAHN master node or networks)

- The slave node compares the hashed value of the master node with all discovered SSIDs. If there is any match of the SSID of a network or a node, the slave node speculates that the desired node is located and sends a CAHN WLAN ad hoc request via GSM to this node (master node)

- The master node sends update messages to all other nodes in the network and notifies them about the new node which wants to join the network. The update message contains the new SSID of the network. This process is ended when all member nodes acknowledge the update messages

- The master node also updates its SSID and sends a response message back to the slave node. The response message contains the WEP key for a WLAN connection. As soon as the slave node receives the response message, it can join the network

5.3.8 CAHN protocol messages

The CAHN protocol for wireless LAN (using the Bloom filter) must offer at least five types of messages: a connection request message, a connection response message, an error message, an update message and an acknowledgment message. There are two kinds of message groups: the cellular messages and the WLAN messages. The cellular messages are transported over the cellular network (GSM) and the WLAN messages are transported over the WLAN link.

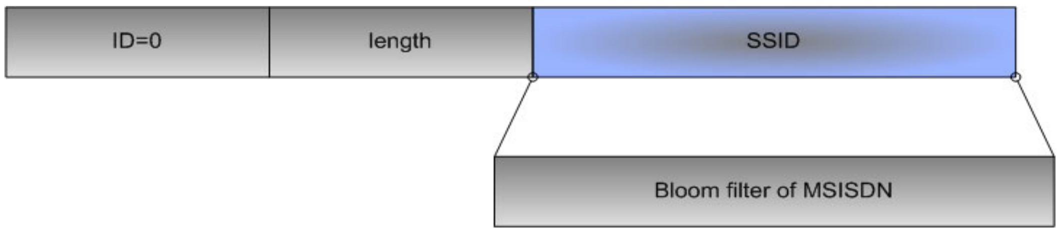


Figure 49: Using the Bloom filter as SSID of the CAHN WLAN.

CAHN WLAN ad hoc request message

This cellular message (figure 50) is used to initiate a connection between a slave and a master node. The CAHN slave node starts a WLAN ad hoc connection by sending a CAHN WLAN request to the master node. The request message contains five parts: a message header, a CAHN header, a WLAN header, an IP header and a data field. The following paragraph explains briefly the meaning of all fields in the defined message:

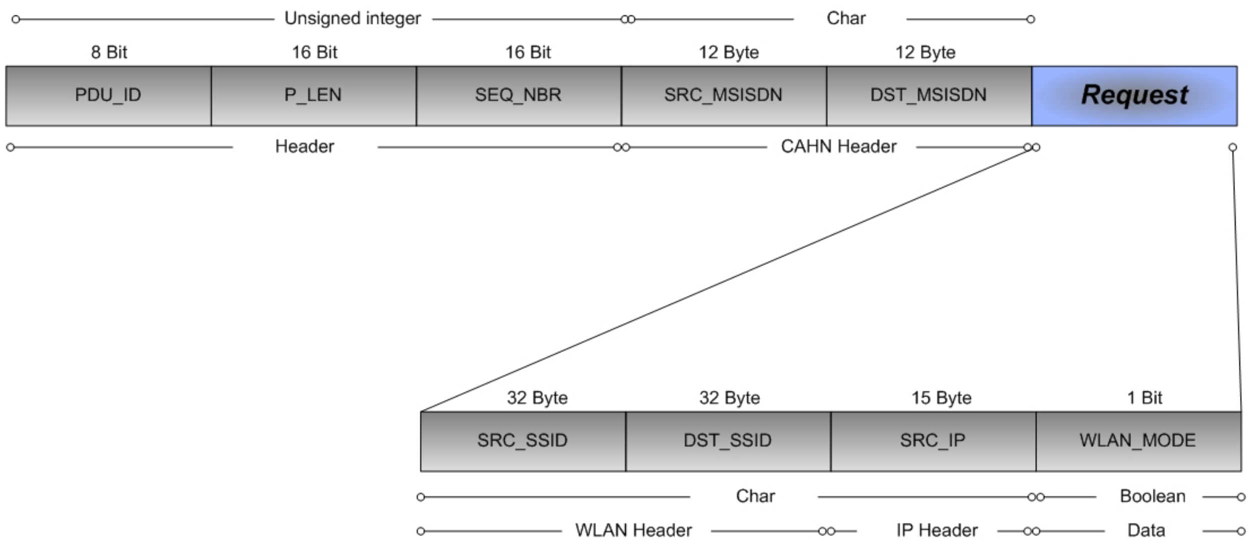


Figure 50: CAHN WLAN ad hoc request message.

- PDU_ID: an unique identifier of the specific CAHN protocol message

- P_LEN: the length of the CAHN protocol packet
- SEQ_NBR: increment to clarify the connection context
- SRC_MSISDN: the MSISDN of the sender
- DST_MSISDN: the MSISDN of the receiver
- SRC_SSID: the network SSID of the sender station
- DST_SSID: the network SSID of the receiver station
- SRC_IP: the IP address of the sender station
- WLAN_MODE: ad hoc or access point mode (optional)

CAHN WLAN ad hoc response message

The response message (figure 51) is also a cellular message. It is responsible for the delivery of necessary information for a WLAN ad hoc connection. A CAHN master responds to a service request message with a service response message. The response message structure is similar to the request message. The WLAN header contains just the SSID of the ad hoc network. The following figure shows the structure of CAHN WLAN response message:

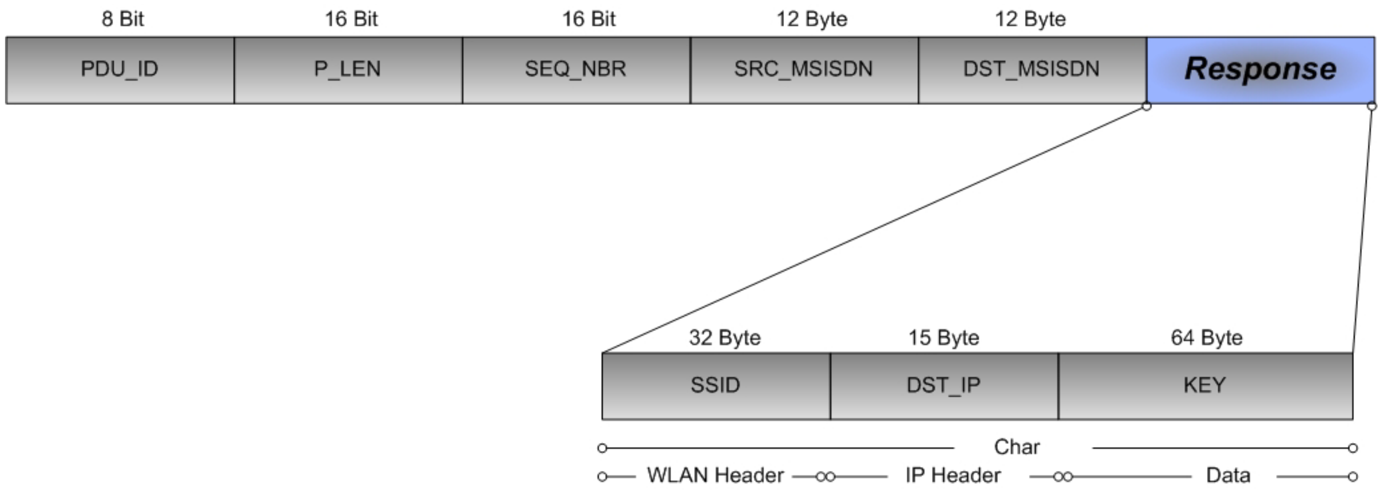


Figure 51: CAHN WLAN ad hoc response message.

- KEY: a (WEP) key or a shared key for the 802.11 authentication and encryption
- DST_IP: destination IP address (master node)

- SSID: the SSID of CAHN WLAN ad hoc network

CAHN WLAN error message

In case of a connection failure or any other reasons that causes a failure, the CAHN WLAN protocol will use the error message (figure 52) to inform the other side. For example, if the master node is not in the supposed ad hoc network (false positive), the master node uses the CAHN WLAN error and returns an error message to the slave node. The error message is also a cellular message. Possible errors in a CAHN ad hoc network would be:

- Slave node is not in direct communication range
- Master node is not in this specific ad hoc network (false positive)
- Stations are busy
- Time-out

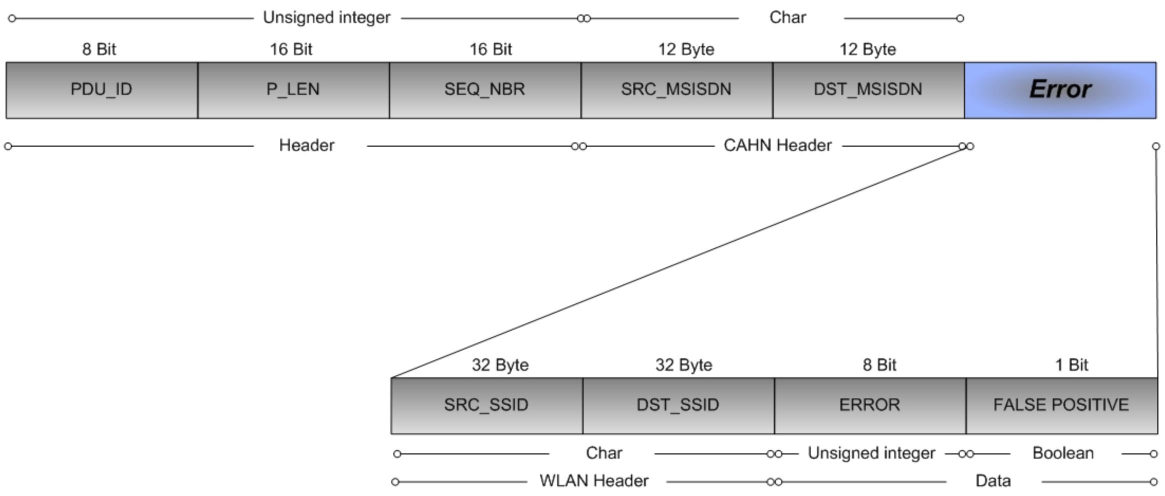


Figure 52: CAHN WLAN error message.

CAHN WLAN ad hoc update message

In the case of a CAHN request, the master node needs to send an update message (figure 53) to its member nodes. The CAHN WLAN update message is sent over WLAN links and therefore it does not depend on the operator. Thus, there is no need for the MSISDN in this message and the CAHN header is obsoletes. The master node informs other member nodes about the new SSID of the network and also the number of actually existing nodes. A brief definition of important fields in a CAHN WLAN update message follows:

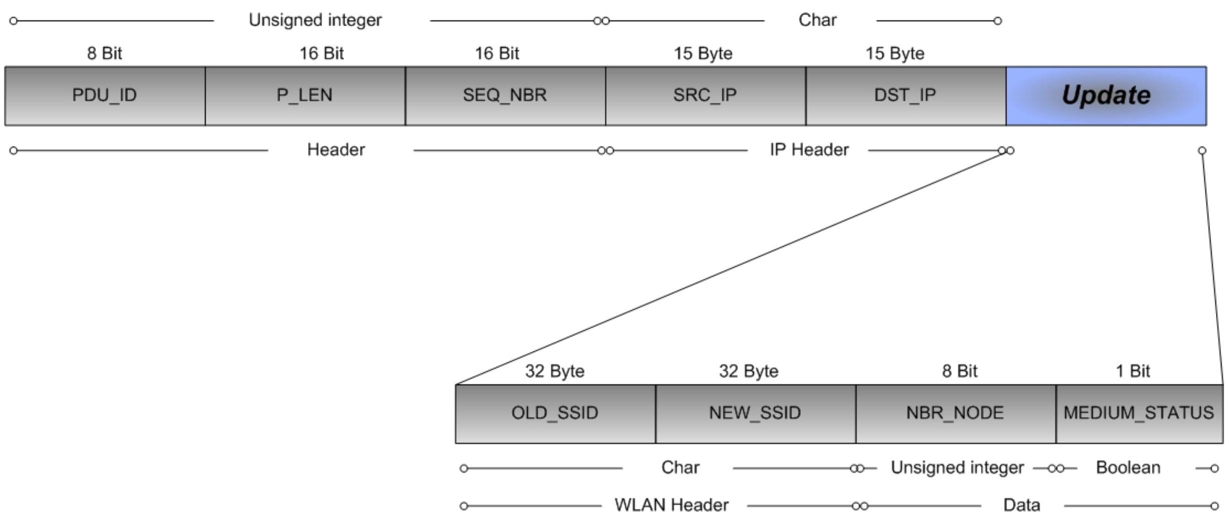


Figure 53: CAHN WLAN ad hoc update message.

- OLD_SSID: the old SSID of the ad hoc network (optional)
- NEW_SSID: the new SSID of the ad hoc network
- NBR_NODE: total number of existing CAHN nodes in this ad hoc network. This number is used to choose the right set of hash functions
- MEDIUM_STATUS: whether the WLAN medium is busy or not. The master node signals that it is ready to break the connection

CAHN WLAN ad hoc acknowledgment message

The stations which got an update message send back an acknowledgment message (figure 54) and signal that they are ready to disconnect and update their SSID. This message is exchanged only between the member stations over the WLAN link. The acknowledgment field shows whether a CAHN member is busy or not.

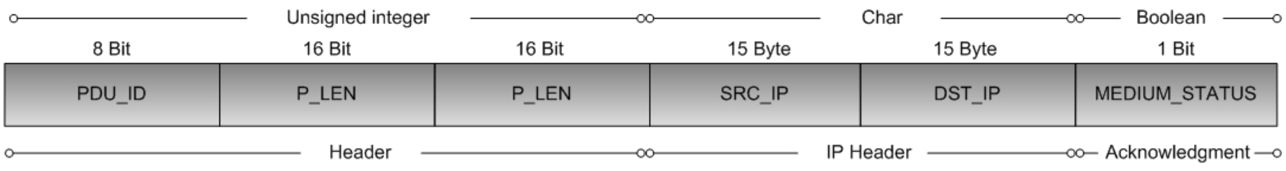


Figure 54: CAHN WLAN ad hoc acknowledgment message.

5.4 CAHN WLAN architecture

The goal of this chapter is to present an architecture that covers the integration of the WLAN technology for cellular assisted heterogeneous networking. The CAHN WLAN architecture extends the usability of the existing CAHN architecture to a wide range of wireless applications including ad hoc and infrastructure based networks. To realize this approach, some new components have been added to the existing architecture. The most demanding requirements of the new architecture are the provision of the basic protocol messages and the corresponding connector, which make CAHN capable of using WLAN technology. This can be achieved by defining appropriate CAHN WLAN protocol messages for information transport among the nodes and the proper WLAN connector. In fact, the connector has to be able to guarantee the communication process between CAHN core and WLAN.

In the following sections, the architecture of the CAHN WLAN protocol and its components will be presented. The architecture has to support both the access point scenario as well as the ad hoc scenario. CAHN WLAN architecture has a layered architecture that is very similar to the CAHN Bluetooth structure. The architecture provides all needed services that CAHN nodes require to establish a WLAN link with a CAHN access point or another CAHN node. Typically, an access point scenario consists of two main roles, which are server (access point) and client. The client/server principle says that a server offers services to other clients and clients use those services by asking the server. For this purpose, the server runs as a background process (daemon) on the system, waiting for incoming service requests. The server analyses the incoming requests from clients and responds to them.

The ad hoc scenario is also similar to the access point scenario. A slave node (client) wants to join an ad hoc network. The slave node contacts a node in the ad hoc network (master node or server) and sends a request.

The core of CAHN WLAN architecture has three main components:

1. The CAHN Communication Manager(CCM)
2. The CAHN WLAN connector
3. The CAHN adaptor

5.4.1 CAHN communication Manager (CCM)

The main job of the CAHN communication manager is to wait for incoming messages. The CCM is responsible for analysing the request messages and the update messages. The CCM creates the response messages as well as the acknowledgment messages. The CCM must decide whether a request message is going to be handled locally or remotely. In the case of a remote request, the request is relayed to a remote node over the cellular network.

In addition, The CCM analyses the update messages, modifies the necessary WLAN parameters and generates the acknowledgment message.

The CCM provides the following features:

- Creation of CAHN WLAN messages (cellular and WLAN messages)
- Offer socket interface to receive CAHN messages

The following state diagram (figure 55) shows the flow of the CCM during a connection setup. The CCM runs as a background process (daemon) and generates a child process (fork) to handle the incoming CAHN request.

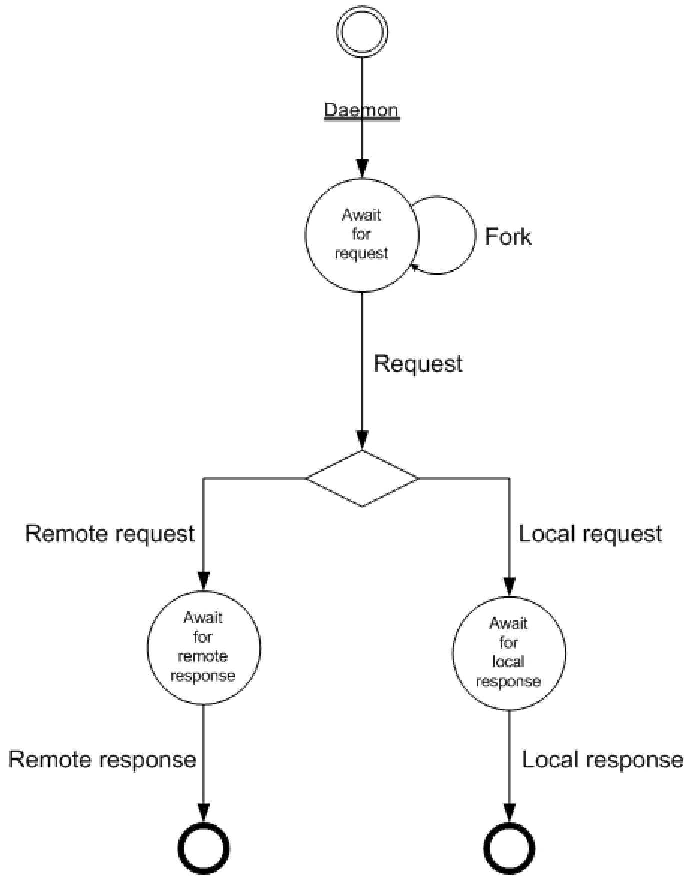


Figure 55: CCM state diagram.

5.4.2 CAHN WLAN connector

The WLAN connector is responsible for handling WLAN requests and update messages. The CAHN WLAN connector is also responsible for generating responses and acknowledgments. The CCM transmits the CAHN WLAN requests and updates to the connector. The WLAN connector processes the incoming messages and returns the answer to the CCM. The connector must be able to carry out the following tasks:

- Providing an interface to the WEP stack of WLAN
- Creation of an unique and valid WEP key
- Interaction with the 802.11 WLAN stack
- Scanning for wireless devices in range
- Supplying an interface to CAHN devices for checking and setting their SSIDs

5.4.3 CAHN adaptor

The CAHN adaptor is responsible for message transformation. The CAHN adaptor takes a CAHN message and transforms it into a transportable form to be sent:

- Over cellular link (The CAHN SMS adapter)
- Over WLAN link (The CAHN WLAN adapter)

At the moment, CAHN uses a SMS interface to convert (remote) messages into a cellular transportable form and sends the SMS over the GSM network to the other CAHN device. Of course, if the CAHN adaptor receives a CAHN SMS, it must be converted into a CAHN valid message. On the other hand, the CAHN adaptor is also responsible for the message exchange over the wireless connection (update and acknowledgment). Here, there is no need of message transformation. The CAHN WLAN messages can be directly sent over the WLAN link and can be saved as a file on the target CAHN node. The CAHN adaptor contains three components: the CAHN adapter, the CAHN SMS adapter and the CAHN WLAN adapter. The CAHN adapter provides UNIX socket interface for the CCM. The CAHN WLAN adapter is responsible for handling all incoming WLAN messages. Furthermore the CAHN SMS adapter is responsible for processing all cellular messages (over GSM).

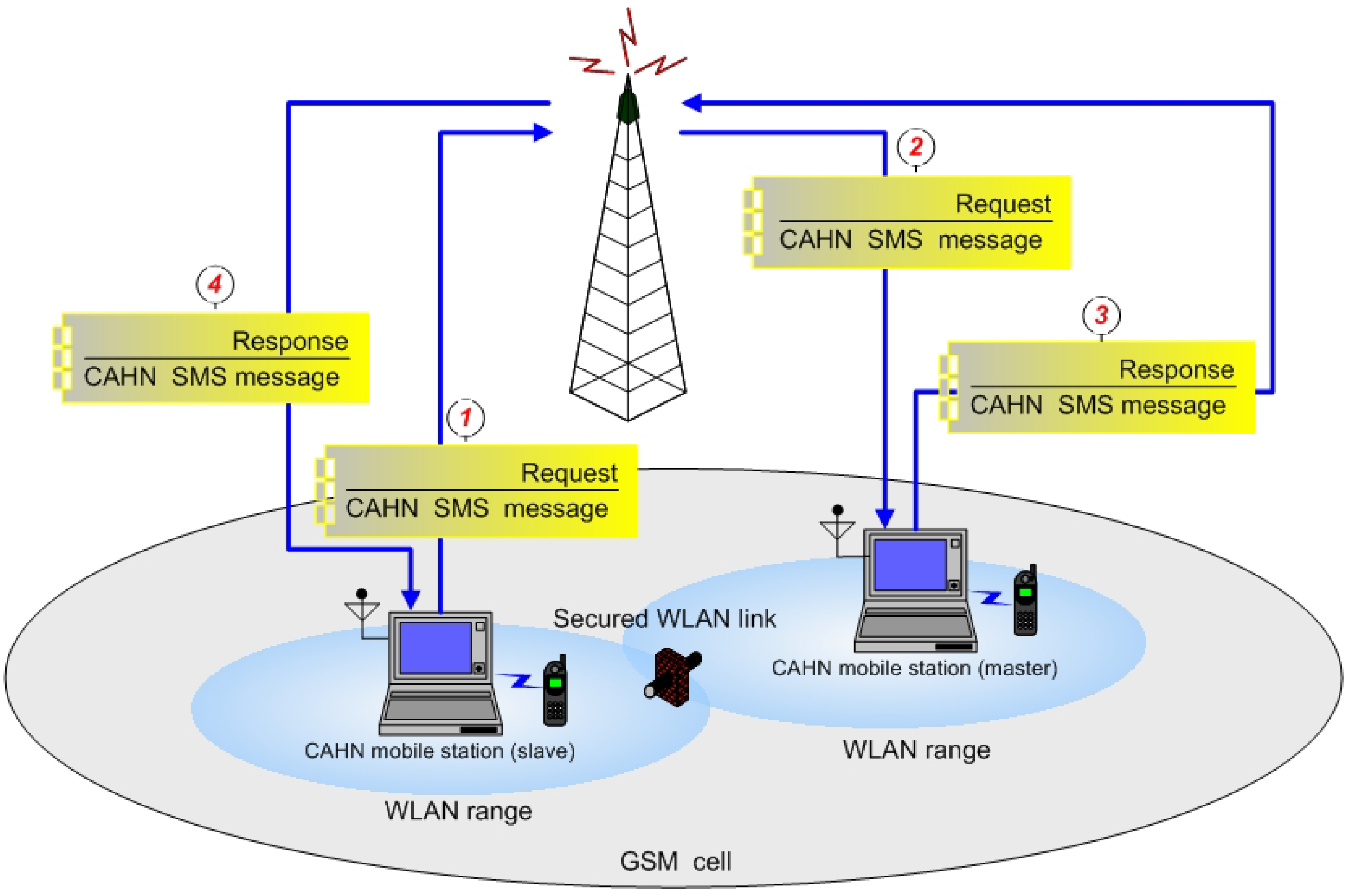


Figure 56: A typical CAHN WLAN scenario.

5.4.4 CAHN SMS adapter

The SMS adapter is directly connected to the cellular device which receives SMSs. Once the cellular device receives a SMS, the CAHN SMS adapter checks whether it is a CAHN SMS or not. If it is not a CAHN SMS, the adapter will put the SMS back on the cellular device. Otherwise, the adapter converts the SMS into a CAHN valid message, depending on the message type. If the message is a request message, the adapter converts it into a CAHN request and relays it to the CAHN adapter. In the case of a response or error message, the adapter sends the message to the CCM (figure 57).

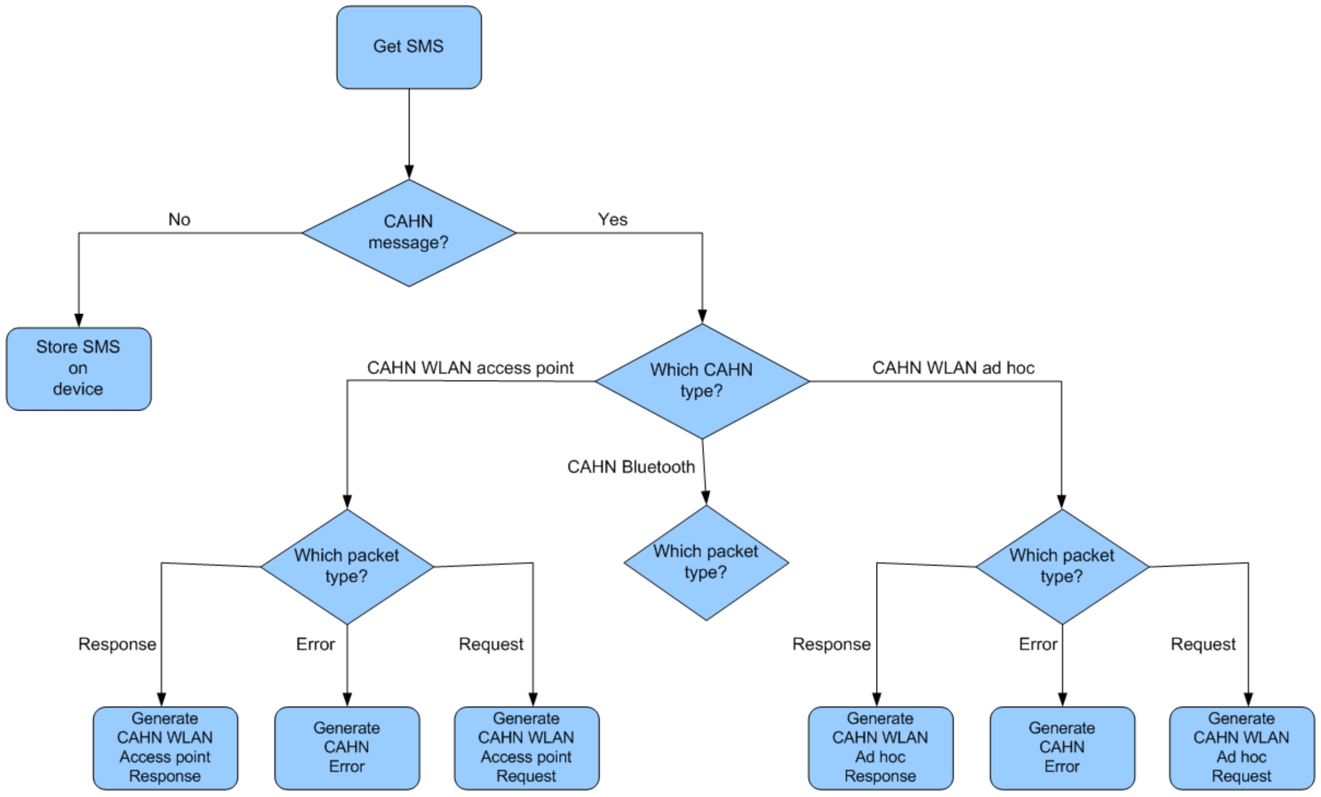


Figure 57: Once a SMS is received, the CAHN SMS adapter checks whether it a CAHN message is or not. As the next step the CAHN SMS adapter identifies the type of the message.

5.4.5 CAHN WLAN adapter

The CAHN WLAN adapter is added to the CAHN architecture for message circulation via wireless LAN links. Beside cellular messages, the Bloom approach causes another set of necessary messages that contain WLAN parameters and Bloom related information. The WLAN adapter is designed for the exchange of update and acknowledgment messages in the ad hoc Bloom filter solution. At the moment, a realisation of the CAHN WLAN adapter does not exist.

5.4.6 GUI

For a convenient usage of the CAHN program, a graphical user interface (GUI) is added to the design. The GUI gives the user the possibility to easily interact with the different components of the architecture. The GUI supports the configuration and access mechanism for both WLAN scenarios.

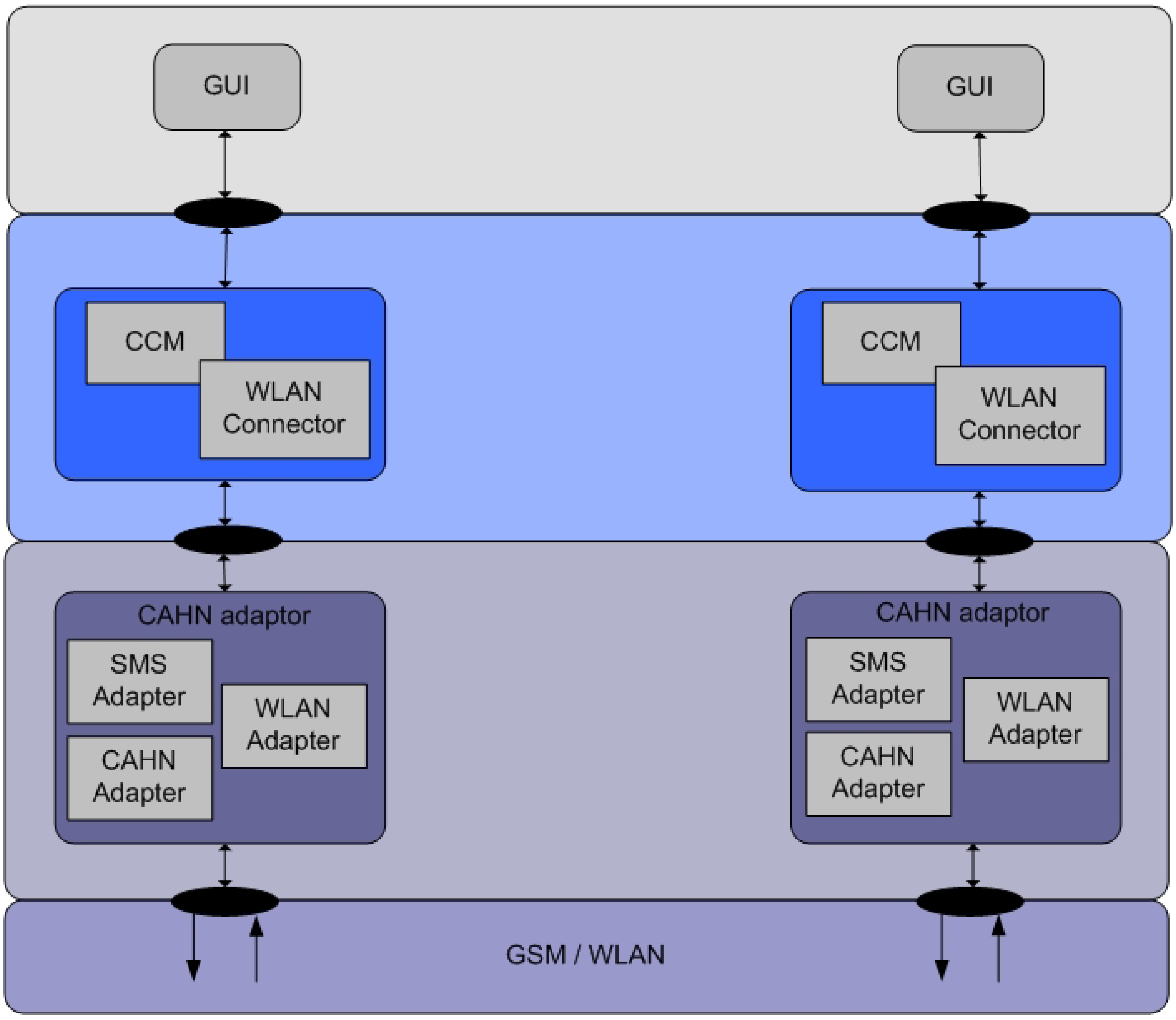


Figure 58: CAHN WLAN architecture with SMS as transport mechanism.

6 IMPLEMENTATION

In the chapter 5, three CAHN scenarios were introduced and discussed. This chapter contains the implementation of two scenarios: the CAHN WLAN access point and the CAHN WLAN ad hoc with random rendezvous solution. As the central reference, the implementation of the existing CAHN Bluetooth protocol [5] was used. The goal of this implementation is to extend the CAHN implementation to provide WLAN support for CAHN protocol.

The platform for the implementation is "GNU/Linux" under the "Red Hat 9.0" operation system. Using Linux as a platform influenced also the decision of using Red Hat WLAN configuration commands in the implementation for better performance. All the CAHN WLAN components are implemented in C, except the GUI. The CAHN GUI is written in C++.

6.1 CAHN communication manager

The CAHN communication manager contains global CAHN packet type definitions. At the moment, there are three different types of CAHN protocol data unit (PDU) IDs: CAHN Bluetooth, CAHN WLAN access point and CAHN WLAN ad hoc (figure 59). The CAHN error messages are also defined in the CCM code.

```
#define CAHN_ERROR_RSP            0x01
[...]
#define CAHN_WLAN_AP_SVC_REQ      0x04
#define CAHN_WLAN_AP_SVC_RSP      0x05
#define CAHN_WLAN_ADHOC_SVC_REQ   0x06
#define CAHN_WLAN_ADHOC_SVC_RSP   0x07
```

Figure 59: CAHN packet identification.

The CCM is also responsible for socket interactions with the CAHN adaptor and the other components. To achieve this purpose, a local UNIX socket interface was chosen. To send and receive requests and response messages, the CCM provides the following methods:

```
inline int cahnd_connect()
inline int cahnd_send_data(int sock, char *buf, int size)
inline int cahnd_read_response(int sock, char *rsp_buf, int rsp_buf_len)
inline int response_socket_send_data(char *data, int data_length, int response_socket)
```

6.2 CAHN WLAN connector

The CAHN WLAN connector completes the definition of the CAHN WLAN messages. The specific data and information of each technology will be defined in the specific connector. Access point and ad hoc related information for CAHN protocol messages are defined in "cahn_wlan_ap.h" and "cahn_wlan_adhoc.h" files. The following "struct" defines the CAHN WLAN access point request:

```
/*
 * CAHN WLAN AP Request Record
 */
typedef struct {
  char      src_msisdn[MAX_SIZE_OF_MSISDN];
  char      dst_msisdn[MAX_SIZE_OF_MSISDN];
} cahn_wlan_ap_svc_req_rec_t;
```

And the CAHN WLAN ad hoc response message would be:

```
/*
 * CAHN WLAN Adhoc Response Record
 */
typedef struct {
  char      src_msisdn[MAX_SIZE_OF_MSISDN];
  char      dst_msisdn[MAX_SIZE_OF_MSISDN];
  char      key[MAX_SIZE_OF_KEY];
[...]
} cahn_wlan_adhoc_svc_rsp_rec_t;
```

The CAHN connector has to provide methods to handle CAHN WLAN requests. These requests are either access point related or ad hoc related. The following methods treat the CAHN WLAN requests:

```
int wlan_ap_process_packet(cahn_wlan_ap_svc_req_rec_t *req, char *rsp);

int wlan_ap_process_response(char *rsp_buf, int rsp_buf_len);

int wlan_ap_parse_rsp_rec(cahn_wlan_ap_svc_rsp_rec_t *rsp, char *rsp_buf, int *rsp_buf_len);

int wlan_adhoc_process_packet(cahn_wlan_adhoc_svc_req_rec_t *req, char *rsp);

int wlan_adhoc_process_response(char *rsp_buf, int rsp_buf_len);

int wlan_adhoc_parse_rsp_rec(cahn_wlan_adhoc_svc_rsp_rec_t *rsp, char *rsp_buf, int *rsp_buf_len);
```

Some requests are not local requests and they must be relayed to a remote node. This is the task of the CAHN adapter.

6.3 CAHN adaptor

The CAHN adaptor converts the CAHN messages into a CAHN SMS and sends them over the GSM network to the remote CAHN node. The CAHN adaptor is always connected to the GSM network and when a CAHN SMS is received by a cellular device, the CAHN adaptor converts it back to a valid CAHN message. The following functions are responsible for the SMS exchange:

```
inline int adapter_connect()
inline int adapter_send_data(int sock, char *buf, int size, int resp_sock)
inline int adapter_read_response(int sock, char *buffer, int buffer_length)
```

As discussed in the last chapter, the CAHN adaptor contains three components. The WLAN adapter also belongs to the CAHN adaptor and it supposed to be responsible for the CAHN Bloom message transfer which is used for the CAHN Bloom approach. Because of the time limitation, this component of designed architecture and the corresponding elements have not been implemented yet.

To access the GSM network, CAHN nodes use a cellular mobile phone that is connected to CAHN (the implementation) via a Bluetooth link. The CAHN SMS adapter provides all functionality the needed for the message exchange via the GSM network.

6.4 CAHN SMS adapter

The CAHN SMS adapter uses a C library to access the GSM mobile phone. The “gsmlib” provides two main services:

- reading and writing of SMS messages from/to the mobile phone
- sending and receiving SMS messages

As soon as a CAHN SMS arrives, the gsmlib informs the SMS adapter which converts the SMS back into a CAHN message.

6.5 GUI

As mentioned in the chapter 5 about the design of CAHN WLAN, the GUI is used to control the entire CAHN implementation and to make it user-friendly. The CAHN GUI is written in C++ using the Kdevelop software.

The service window of the CAHN GUI contains two buttons which represent the two kind of defined CAHN services: Bluetooth and WLAN. The user can choose between these two technologies. The

Bluetooth part is already implemented. By pressing the WLAN button the WLAN service window will open:

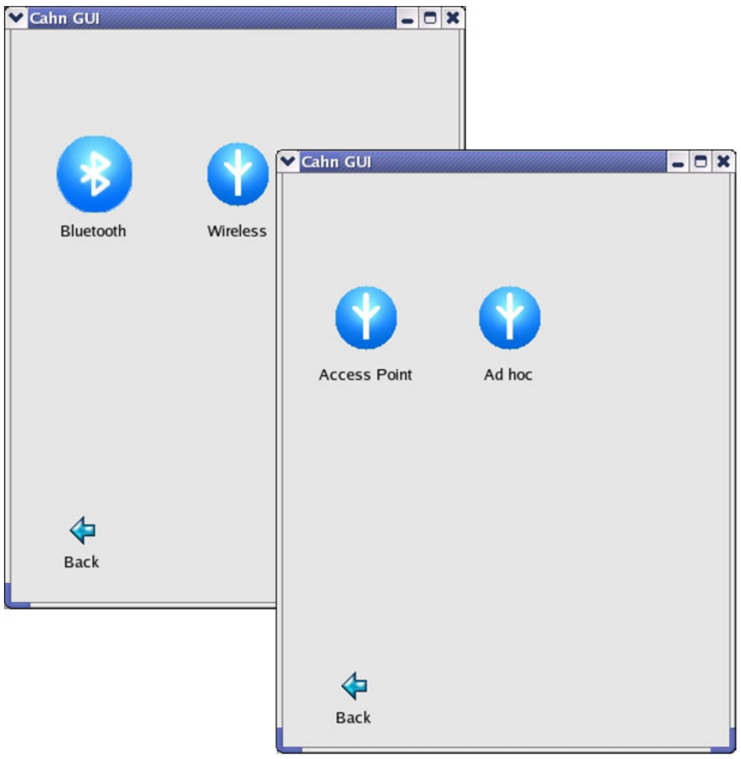


Figure 60: CAHN WLAN services window.

The WLAN service window contains the ad hoc and the access point service (figure 60). Depending on the user choice, a new window emerges that gives the user the possibility to configure or to access the WLAN technology. The user is now able to configure his WLAN device and to establish a connection (figure 61).

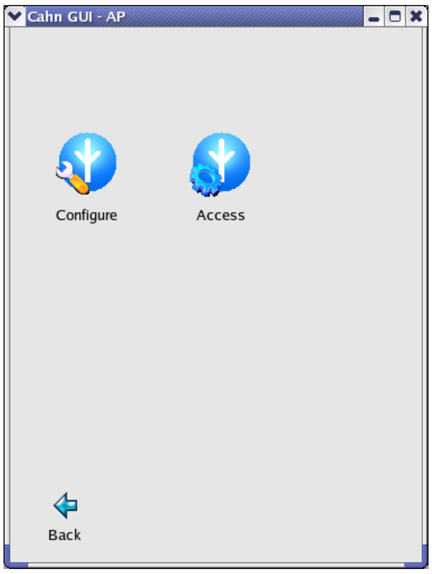
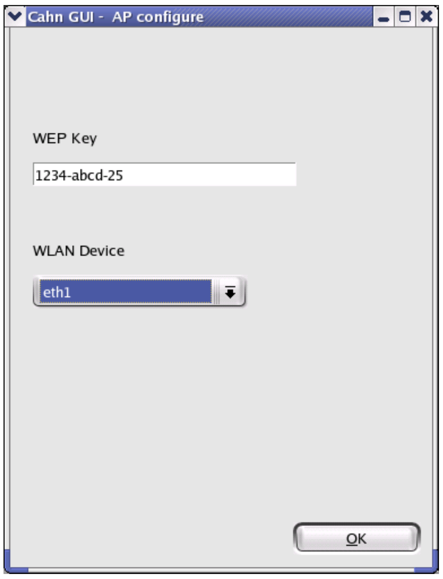


Figure 61: CAHN access point window.

6.5.1 *Configuration*

By pressing the "Configure" button, a new window will appear. The user can configure the CAHN access point by choosing the WLAN device. On the server side, the administrator can also configure the WEP key for the access point. By pressing the "Ok" button, the chosen configurations are enabled. It means the WLAN stack now sets the WLAN device, the USSD and the WEP key for the WLAN encryption (figure 62).



Cahn GUI - AP configure

WEP Key

1234-abcd-25

WLAN Device

eth1

Figure 62: CAHN WLAN access point configure window.

On the other side, the ad hoc configure window (figure 63) contains three configuration fields. Aside from the above possibilities, the user can enter the MSISDN of the desired CAHN ad hoc node. The

CAHN ad hoc configuration window for the master and the slave node, have a similar design. The master node sets the correct WEP key in the WEP key field and provides this key to the slave nodes.

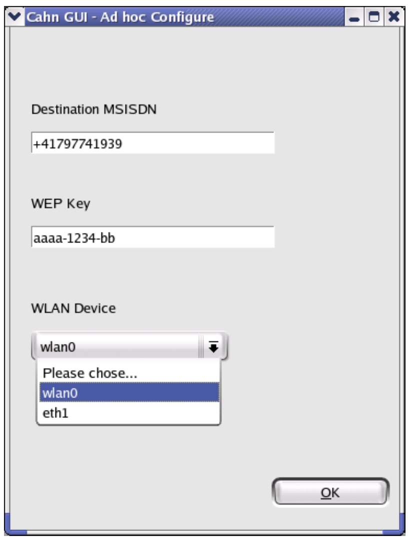
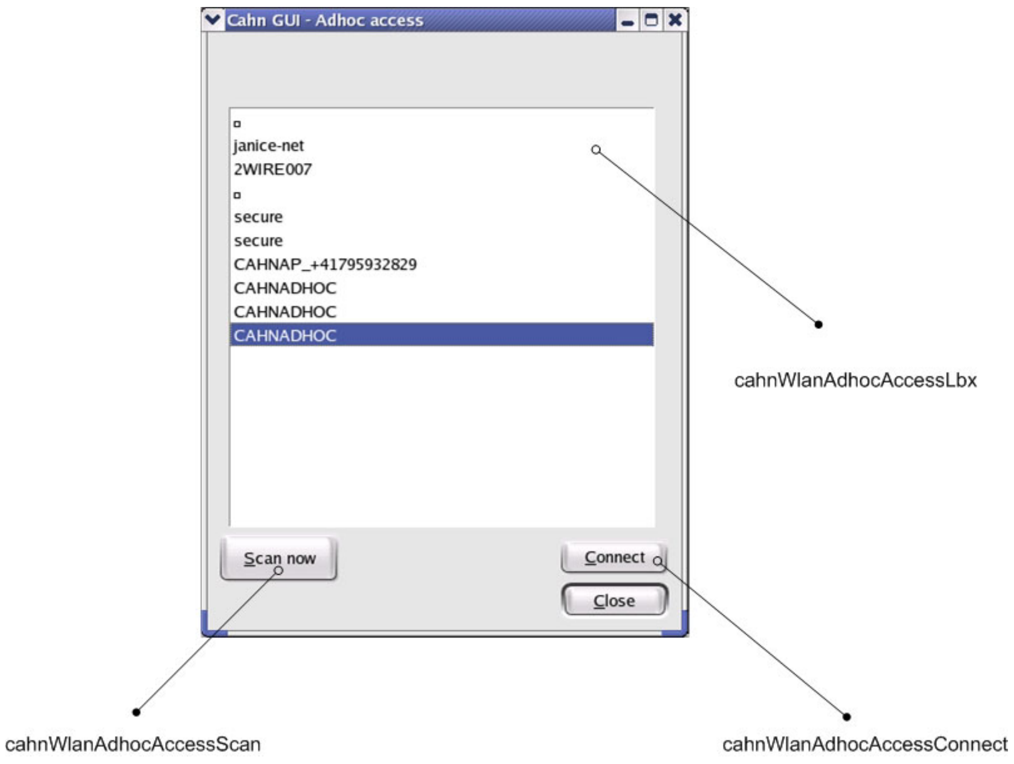


Figure 63: CAHN WLAN ad hoc configure window.

6.5.2 *Access*

The user can access the ad hoc or access point services by pressing the “Access” button (figure 61). The CAHN GUI provides an access window for this aspect. The access window offers two functions for using services. At first, a scan mechanism to find the CAHN enabled devices that can be activated by pressing the “Scan now” button is provided. After this process, all WLAN devices in the area are visible and are listed in the list box. Now, the user can choose the right CAHN node by pressing the “Connect” button (figure 64). The CAHN will take care of all necessary work, including sending the request, handling the response and setting up the WLAN device. Once the “Connect” process is in progress, the user (client or slave) has to wait until the response message or the proper error message appears in the GUI. If the connection setup was successful, the GUI shows the connection acknowledgement in the access window.



¤
janice-net
2WIRE007
¤
secure
secure
CAHNAP_+41795932829
CAHNADHOC
CAHNADHOC
CAHNADHOC

Figure 64: CAHN WLAN ad hoc access window.

The "Close" button terminates the connection setup process and closes the access window. The CAHN WLAN GUI contains 8 sub windows in total. Each scenario requires 3 sub windows providing the related services.

Figure 65 shows the code segment of the scan function of the ad hoc scenario which is integrated into the GUI.

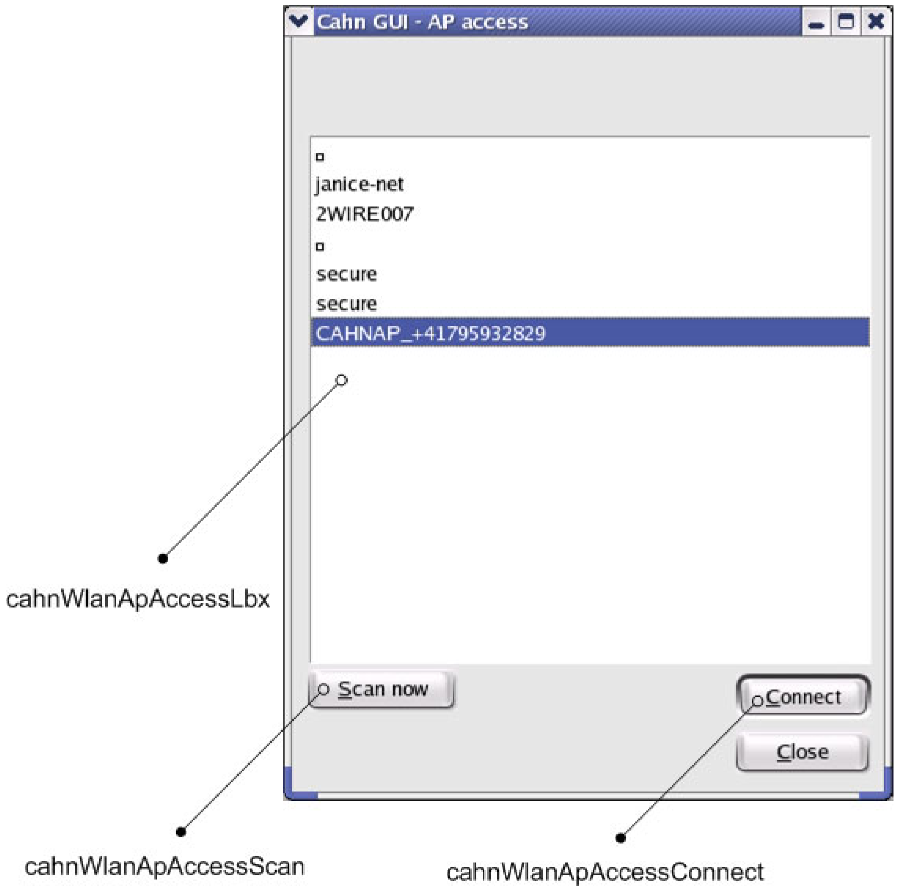
```
void Cahn_wlan_adhoc_access::cahnWlanAdhocAccessScan(){
  // Remove all current Items
  CahnWlanAdhocAccessLbx->clear();

  // Disable Buttons
  CahnWlanAdhocAccessBtn_scan->setEnabled(false);
  CahnWlanAdhocAccessBtn_connect->setEnabled(false);
  CahnWlanAdhocAccessBtn_close->setEnabled(false);
  CahnWlanAdhocAccessBtn_scan->repaint();
  CahnWlanAdhocAccessBtn_connect->repaint();
  CahnWlanAdhocAccessBtn_close->repaint();

  // Change cursor to wait state
  QApplication::setOverrideCursor(QCursor(Qt::WaitCursor));

  // Do scan
  char line[80];

  FILE *iwlist_scan = popen("iwlist scan | grep ESSID", "r");

  while(fgets(line, 80, iwlist_scan)) {
    strtok(line, ":\"");
    CahnWlanAdhocAccessLbx->insertItem(strtok(NULL, "\""));
  }
  pclose(iwlist_scan);

  // Enable Buttons
  CahnWlanAdhocAccessBtn_scan->setEnabled(true);
  CahnWlanAdhocAccessBtn_connect->setEnabled(true);
  CahnWlanAdhocAccessBtn_close->setEnabled(true);
  CahnWlanAdhocAccessBtn_scan->repaint();
  CahnWlanAdhocAccessBtn_connect->repaint();
  CahnWlanAdhocAccessBtn_close->repaint();

  // Change cursor to normal state
  QApplication::restoreOverrideCursor();
}
```

Figure 65: CAHN WLAN ad hoc scan method.

In the access point mode, there is a similar functionality that allows the user to scan for CAHN access points and to see the result of the scan process in the list box. Consequently, the CAHN access window for access points provides the method to establish a connection with the chosen CAHN access point. These two functions achieve the scan and connect process:

```
virtual void cahnWlanApAccessScan();
virtual void cahnWlanApAccessConnect();
```

Figure 66 shows the access window of the CAHN access point scenario after ending the scan process. The tagged line in the box list ("cahnWlanApAccessLbx") presents a CAHN access point with the number "+41795932829" as the MSISDN of the cellular device.



□
janice-net
2WIRE007
□
secure
secure
CAHNAP_+41795932829

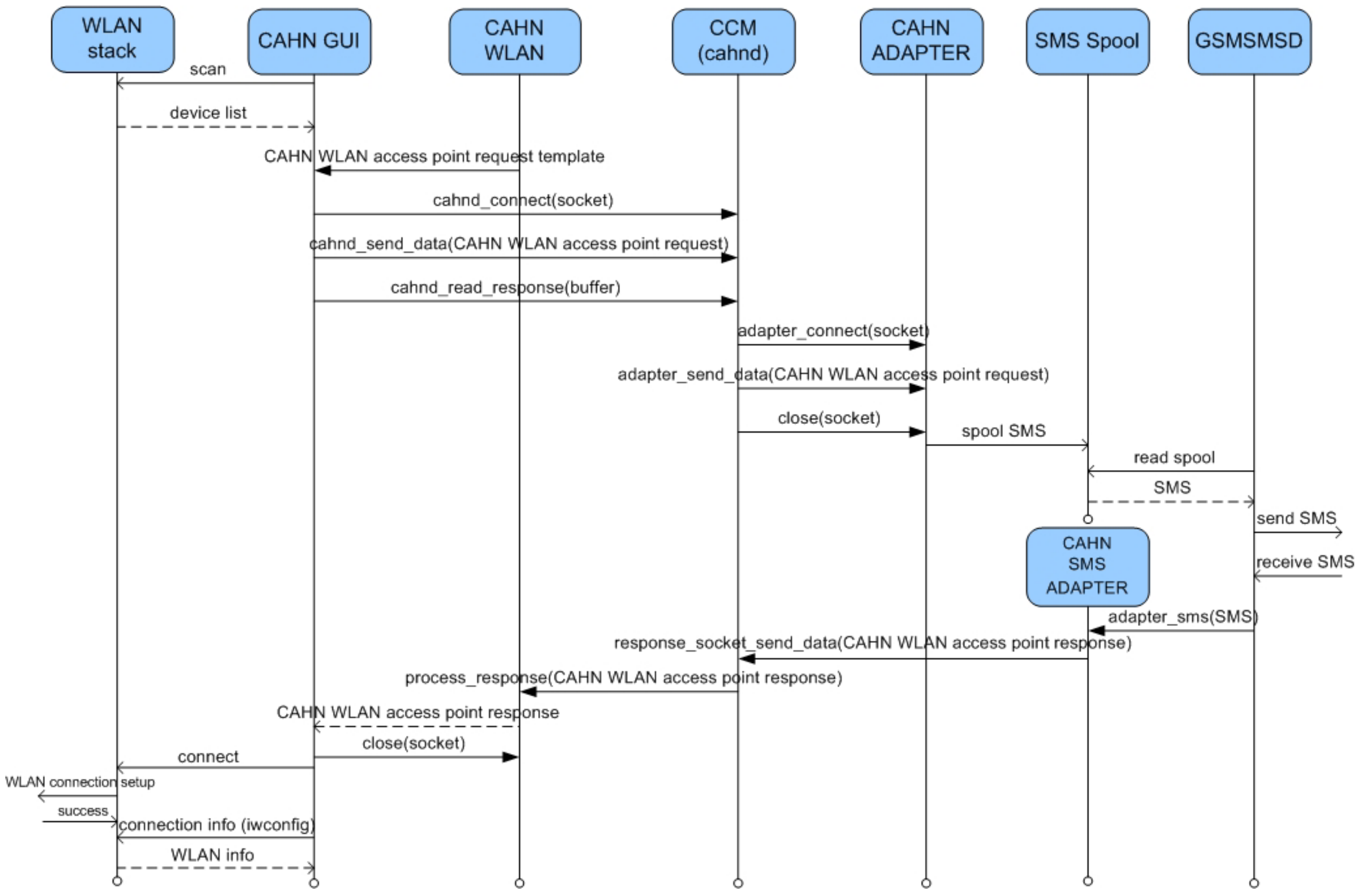
Figure 66: CAHN WLAN access point access window.

6.6 Message flow

The following message flow diagrams show the entire message exchange between the components in a CAHN WLAN scenario. Each scenario has two separate message flows: one for the client/slave side and one for the server/master side. To avoid the overhead, just one message flow from each scenario was chosen.

6.6.1 CAHN WLAN access point

The next diagram (figure 67) shows a CAHN WLAN client which sends a CAHN WLAN access point request to a CAHN access point.



WLAN stack
CAHN GUI
CAHN WLAN
CCM (cahnd)
CAHN ADAPTER
SMS Spool
GSMSMSD
scan
device list
CAHN WLAN access point request template
cahnd_connect(socket)
cahnd_send_data(CAHN WLAN access point request)
cahnd_read_response(buffer)
adapter_connect(socket)
adapter_send_data(CAHN WLAN access point request)
close(socket)
spool SMS
read spool
SMS
send SMS
receive SMS
CAHN SMS ADAPTER
adapter_sms(SMS)
response_socket_send_data(CAHN WLAN access point response)
process_response(CAHN WLAN access point response)
CAHN WLAN access point response
close(socket)
connect
WLAN connection setup
success
connection info (iwconfig)
WLAN info

Figure 67: Message flow on client part of a CAHN WLAN access point.

6.6.2 CAHN WLAN ad hoc

This message flow diagram (figure 68) shows a CAHN WLAN master node which receives a CAHN WLAN ad hoc request service message.

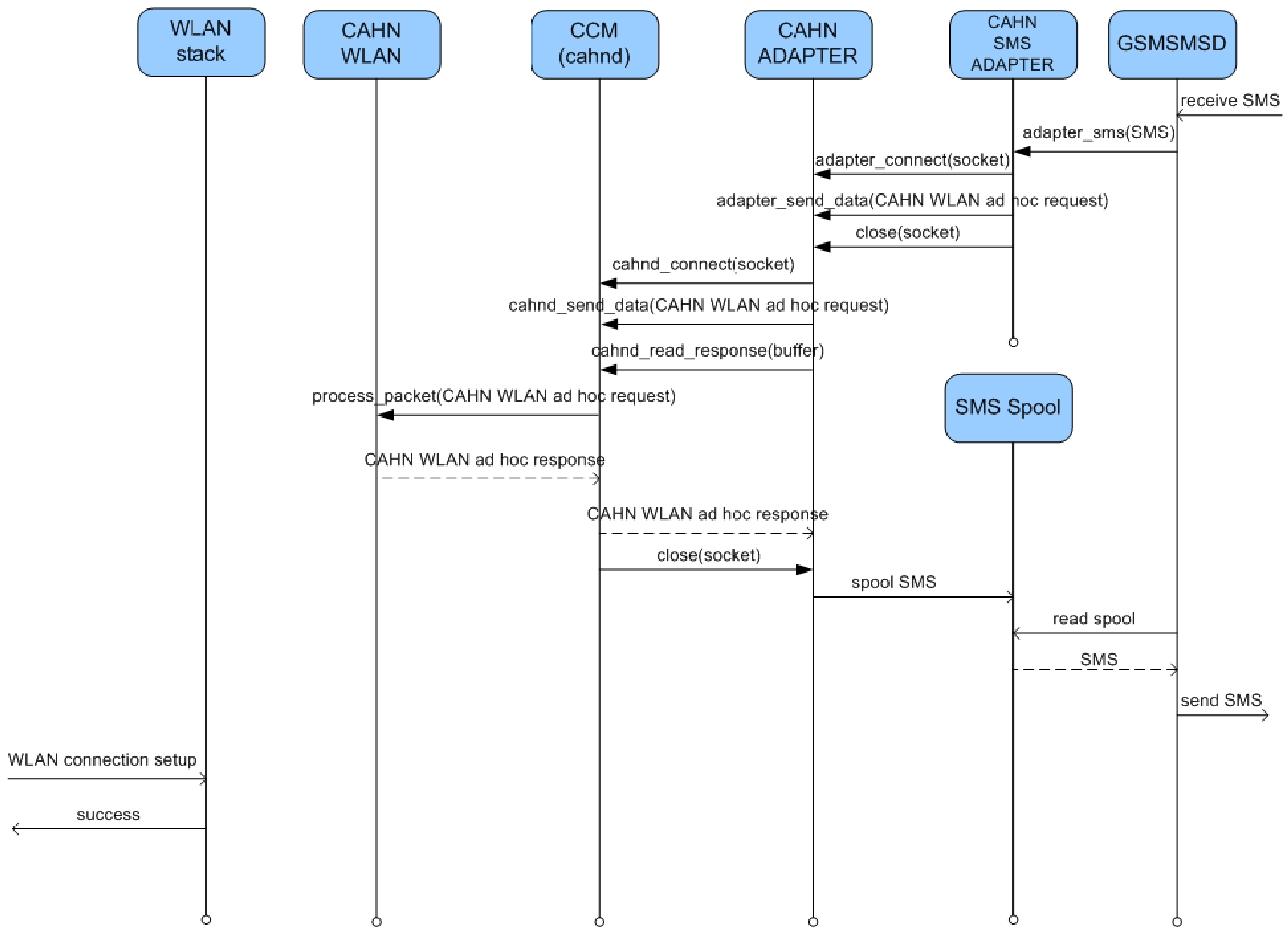


Figure 68: Message flow on master side of a CAHN WLAN ad hoc network.

6.7 IP distribution

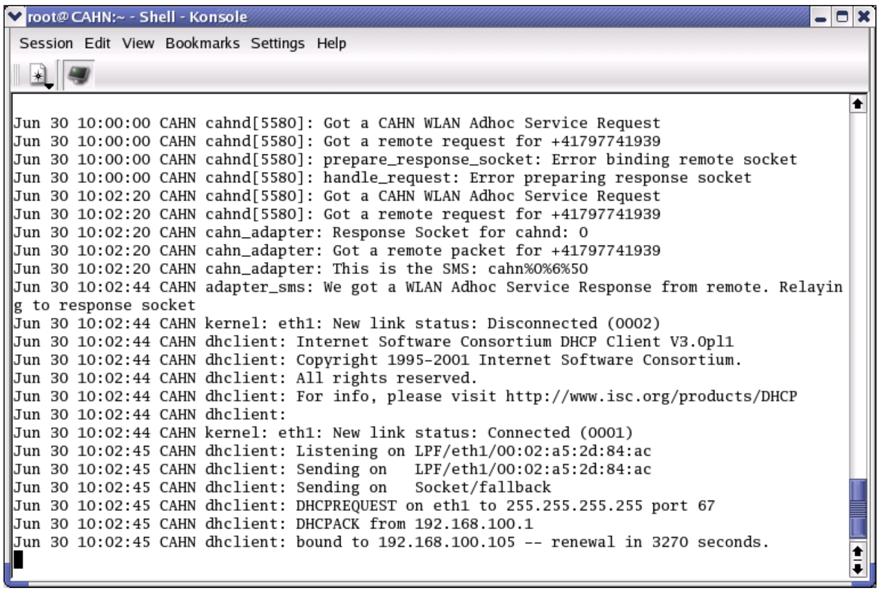
After the WLAN connection setup, an IP address needs to be given to each CAHN station. For this approach, the CAHN WLAN can use one of these solutions:

- IPv6: Each node has its unique IPv6 address. Nodes exchange the IP addresses over CAHN request and response messages. The IP distribution process happens during the connection setup (in ad hoc scenario)

- Auto IP with ARP (Address resolution protocol): After the CAHN WLAN connection setup, a CAHN node broadcasts a packet onto the Ethernet asking "who has IP address *x*?" The broadcast will arrive at every CAHN nodes in the subnet of *x*. The nodes check their IP addresses and if any node owns the IP address *x*, it will respond to the ARP request. The polling node now learns which node has which IP address. When the polling node gets no response, it assumes that the IP address *x* is free. The protocol for asking this question and getting the reply is called ARP. The IP addresses are all in a certain predefined IP range (class A) and in a fixed subnet

- DHCP server (Dynamic Host Configuration Protocol): a CAHN node acts as a DCHP server in a CAHN WLAN network and distributes IP addresses

The DCHP solution approach (figure 69) was chosen for a quick implementation of the CAHN WLAN. In the access point scenario, the CAHN server is configured as a DHCP server which gives IP address to other CAHN nodes:



root@CAHN:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

```
Jun 30 10:00:00 CAHN cahnd[5580]: Got a CAHN WLAN Adhoc Service Request
Jun 30 10:00:00 CAHN cahnd[5580]: Got a remote request for +41797741939
Jun 30 10:00:00 CAHN cahnd[5580]: prepare_response_socket: Error binding remote socket
Jun 30 10:00:00 CAHN cahnd[5580]: handle_request: Error preparing response socket
Jun 30 10:02:20 CAHN cahnd[5580]: Got a CAHN WLAN Adhoc Service Request
Jun 30 10:02:20 CAHN cahnd[5580]: Got a remote request for +41797741939
Jun 30 10:02:20 CAHN cahn_adapter: Response Socket for cahnd: 0
Jun 30 10:02:20 CAHN cahn_adapter: Got a remote packet for +41797741939
Jun 30 10:02:20 CAHN cahn_adapter: This is the SMS: cahn%0%6%50
Jun 30 10:02:44 CAHN adapter_sms: We got a WLAN Adhoc Service Response from remote. Relayin
g to response socket
Jun 30 10:02:44 CAHN kernel: eth1: New link status: Disconnected (0002)
Jun 30 10:02:44 CAHN dhclient: Internet Software Consortium DHCP Client V3.0pl1
Jun 30 10:02:44 CAHN dhclient: Copyright 1995-2001 Internet Software Consortium.
Jun 30 10:02:44 CAHN dhclient: All rights reserved.
Jun 30 10:02:44 CAHN dhclient: For info, please visit http://www.isc.org/products/DHCP
Jun 30 10:02:44 CAHN dhclient:
Jun 30 10:02:44 CAHN kernel: eth1: New link status: Connected (0001)
Jun 30 10:02:45 CAHN dhclient: Listening on LPF/eth1/00:02:a5:2d:84:ac
Jun 30 10:02:45 CAHN dhclient: Sending on   LPF/eth1/00:02:a5:2d:84:ac
Jun 30 10:02:45 CAHN dhclient: Sending on   Socket/fallback
Jun 30 10:02:45 CAHN dhclient: DHCPREQUEST on eth1 to 255.255.255.255 port 67
Jun 30 10:02:45 CAHN dhclient: DHCPACK from 192.168.100.1
Jun 30 10:02:45 CAHN dhclient: bound to 192.168.100.105 -- renewal in 3270 seconds.
```

Figure 69: IP distribution in a CAHN ad hoc scenario. The master node is configured as a DCHP server that arranges IPs to other nodes.

7 EVALUATION

The following section presents some time measurements for both, the access point (section 5.1) and the ad hoc (section 5.2) scenario. The measurements show how the CAHN implementation behaves in a real environment.

7.1 Test bed and equipments

Both scenarios were tested under the same conditions. All test computers were running Linux "Red hat" 9.0 kernels 2.4.2.

For the client side of the access point and the ad hoc scenario, a "Toshiba Portégé P4010" laptop with Intel processor 733 MHz and 128 MB of RAM was chosen. The laptop has a built-in WLAN/Bluetooth adapter. An "Ericsson T68i" mobile phone was connected via Bluetooth to the laptop.

On the sever side of both scenarios a "DELL Latitude D500" with Intel Pentium Centrino 1.3 GHz and 128 MB of RAM was used. The laptop has a built-in Intel WLAN/ Bluetooth adapter and was also connected to an "Ericsson T68i" mobile phone.

All equipments were located in the same room (laboratory) and there were a lot of other WLAN devices placed in this room. The distance between the CAHN server and client was less than one meter:

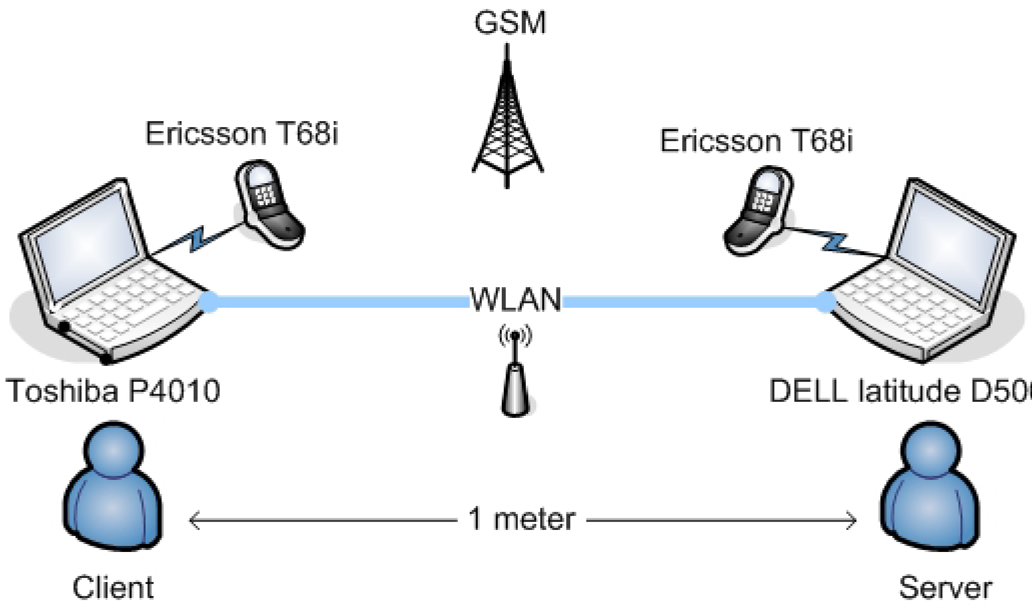


Figure 70: Structure of a CAHN WLAN scenario.

7.2 CAHN WLAN connection time

At the beginning, both client and server were configured and made ready for a connection setup. The time measurement started when the "Connect" button was pressed in the GUI and it stopped as soon as the GUI acknowledged the establishment of a WLAN connection. During this interval, the time was measured.

For both CAHN scenarios, 20 time measurements were made. The following diagrams present the time measurements. The blue graph represents the CAHN WLAN access point performance and the red one represents the CAHN WLAN ad hoc performance time.

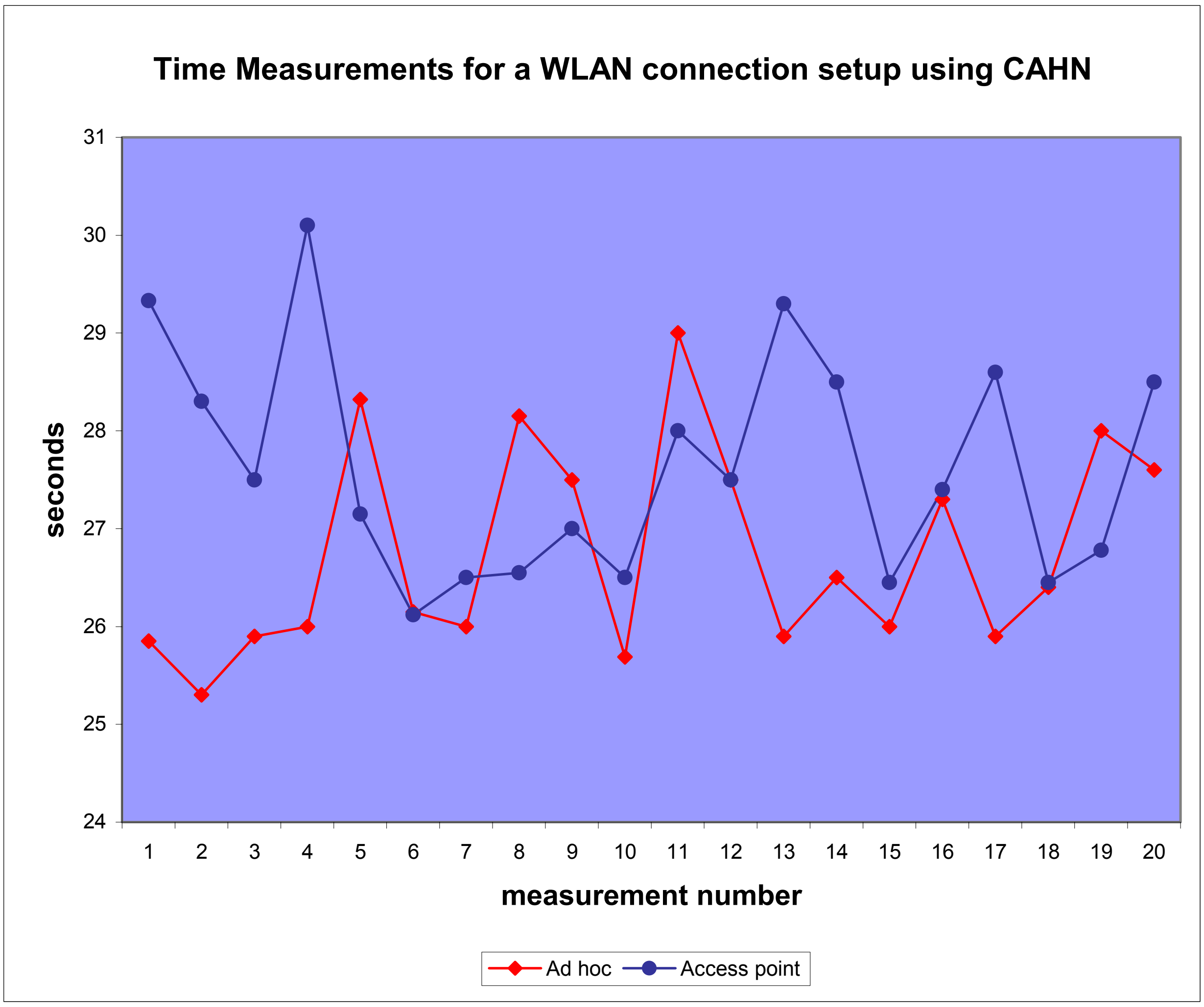


Table 4: CAHN WLAN connection setup takes maximum 30 seconds.

Table 4 illustrates the result of the time measurements of the CAHN WLAN implementation. In fact, both scenarios needed approximately the equal amount of time. The average connection setup time on a CAHN WLAN ad hoc was 26.5 seconds, compared to 27.5 seconds in the access point scenario is. There are three critical factors which affect the time quantity of CAHN WLAN performance:

- SMS exchange delay
- Gsmlib spooling delay
- IP distribution

Many SMS time measurements have shown that two main factors influence the performance of SMS distribution. The first factor is the size of the SMS. Indeed, the SMS time cycle increases when the number of characters in the SMS message grows. The following table shows the time needed for sending a SMS with three different sizes. Both mobile phones used for this measurement were “Sony Ericsson T68i”.

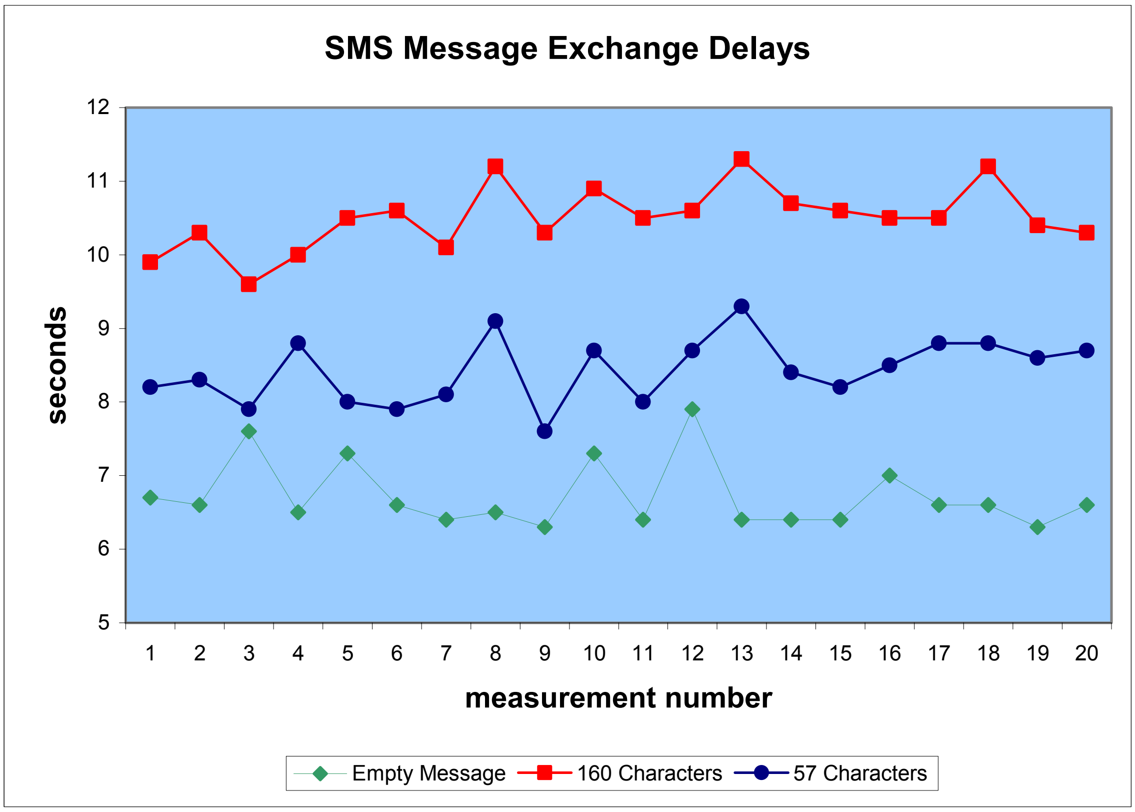


Table 5: SMS exchange delay.

The second factor is the influence of hardware of cellular mobile phone. The analysis of the experiment results have pointed out that the SMS time cycle depends directly on the choice of the type of the cellular mobile phone (table 6). The following table shows the result of two SMS time measurements. The blue graph shows a situation where both sides (sender and receiver) use the same “Sony Ericsson T68i” mobile phone. The red graph shows another situation where sender and receiver do not use the same mobile phones. The sender mobile phone is a “NOKIA 7650” and the receiver uses a “SIEMENS A55”. The size of SMS in both cases is 20 characters (size of CAHN WLAN access point request/response).

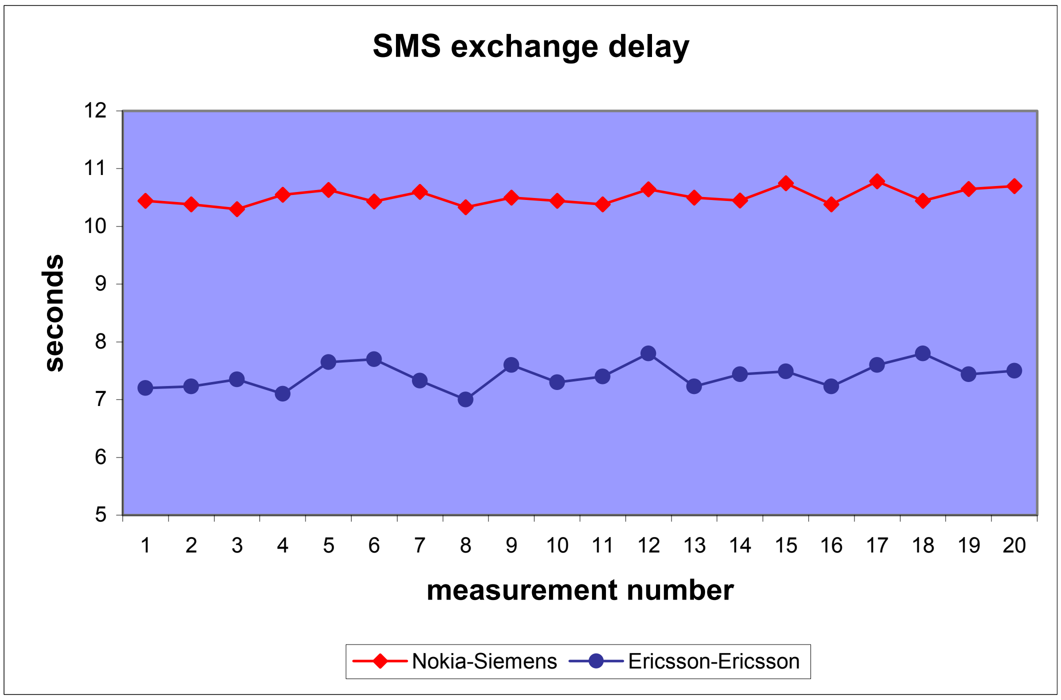


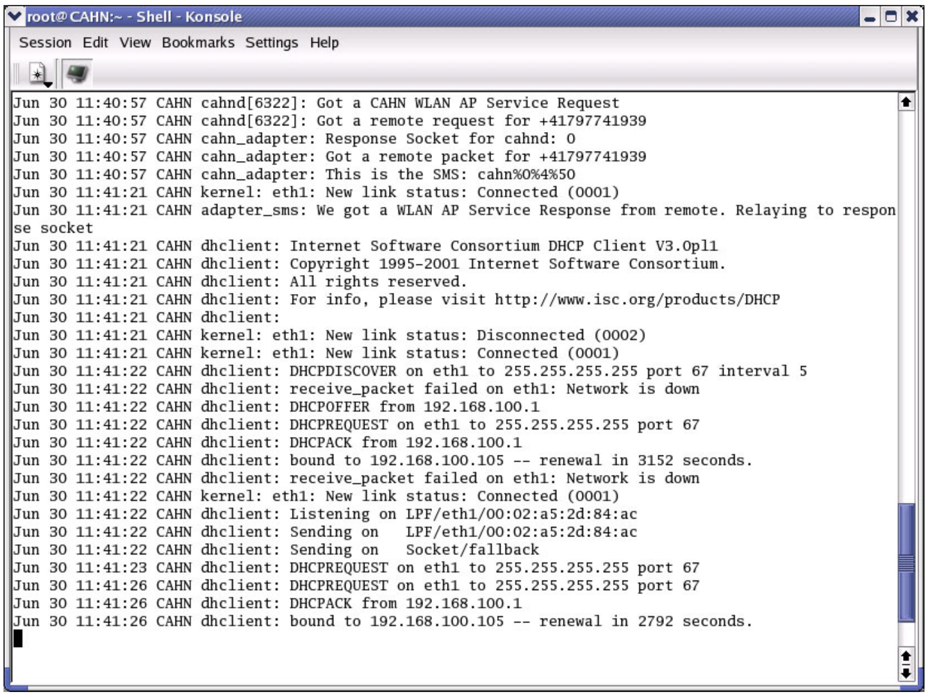
Table 6: SMS exchange delay. The size of SMS is in both measurements equal 20 characters.

Anyway, for this diploma work the best time measurement (with Sony Ericsson mobile phones) was chosen and the following evaluations are also based on this choice.

The next critical factor for the CAHN concept is the usage of Gsmlib library. The “gsmsmsd” spooling mechanism checks periodically (every 2.5 seconds) the spool directory for outbound SMSs. It means that by sending a request, the gsmsmsd needs 2.5 seconds to detect the CAHN message. On the other side, by sending a response the gsmsmsd needs another 2.5 seconds to spool the message. In total, the spooling process costs the implementation about 5 seconds.

The last critical factor is the IP distribution. Depending on which mechanism is used to distribute IP addresses, the time factor can be manipulated. As mentioned in the implementation chapter, for both CAHN scenarios a DCHP solution was applied. The DHCP server delivers the IP address in a certain predefined IP range (class A) and fixed subnet. The problem using this solution approach is the uncertainly of what happens if the DHCP server is down or leaves the net. In an access point scenario, it is imaginable that there is always one unique access point. But in an ad hoc scenario, it can cause conflicts if there are more than one DCHP server in the network. On the other side, to avoid the absence of an IP server in a CAHN network, the DHCP server must inform other nodes (potation DHCP servers) before leaving the network.

Anyway, using this IP distribution solution influences the performance of CAHN. It takes about 3~5 seconds until a CAHN node gets an IP from the CAHN server or master (figure 71). The following picture shows a screen-shot of the log file during a CAHN connection setup:


root@CAHN:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

```
Jun 30 11:40:57 CAHN cahnd[6322]: Got a CAHN WLAN AP Service Request
Jun 30 11:40:57 CAHN cahnd[6322]: Got a remote request for +41797741939
Jun 30 11:40:57 CAHN cahn_adapter: Response Socket for cahnd: 0
Jun 30 11:40:57 CAHN cahn_adapter: Got a remote packet for +41797741939
Jun 30 11:40:57 CAHN cahn_adapter: This is the SMS: cahn%0%4%50
Jun 30 11:41:21 CAHN kernel: eth1: New link status: Connected (0001)
Jun 30 11:41:21 CAHN adapter_sms: We got a WLAN AP Service Response from remote. Relaying to respon
se socket
Jun 30 11:41:21 CAHN dhclient: Internet Software Consortium DHCP Client V3.0pl1
Jun 30 11:41:21 CAHN dhclient: Copyright 1995-2001 Internet Software Consortium.
Jun 30 11:41:21 CAHN dhclient: All rights reserved.
Jun 30 11:41:21 CAHN dhclient: For info, please visit http://www.isc.org/products/DHCP
Jun 30 11:41:21 CAHN dhclient:
Jun 30 11:41:21 CAHN kernel: eth1: New link status: Disconnected (0002)
Jun 30 11:41:21 CAHN kernel: eth1: New link status: Connected (0001)
Jun 30 11:41:22 CAHN dhclient: DHCPDISCOVER on eth1 to 255.255.255.255 port 67 interval 5
Jun 30 11:41:22 CAHN dhclient: receive_packet failed on eth1: Network is down
Jun 30 11:41:22 CAHN dhclient: DHCPOFFER from 192.168.100.1
Jun 30 11:41:22 CAHN dhclient: DHCPREQUEST on eth1 to 255.255.255.255 port 67
Jun 30 11:41:22 CAHN dhclient: DHCPACK from 192.168.100.1
Jun 30 11:41:22 CAHN dhclient: bound to 192.168.100.105 -- renewal in 3152 seconds.
Jun 30 11:41:22 CAHN dhclient: receive_packet failed on eth1: Network is down
Jun 30 11:41:22 CAHN kernel: eth1: New link status: Connected (0001)
Jun 30 11:41:22 CAHN dhclient: Listening on LPF/eth1/00:02:a5:2d:84:ac
Jun 30 11:41:22 CAHN dhclient: Sending on   LPF/eth1/00:02:a5:2d:84:ac
Jun 30 11:41:22 CAHN dhclient: Sending on   Socket/fallback
Jun 30 11:41:23 CAHN dhclient: DHCPREQUEST on eth1 to 255.255.255.255 port 67
Jun 30 11:41:26 CAHN dhclient: DHCPREQUEST on eth1 to 255.255.255.255 port 67
Jun 30 11:41:26 CAHN dhclient: DHCPACK from 192.168.100.1
Jun 30 11:41:26 CAHN dhclient: bound to 192.168.100.105 -- renewal in 2792 seconds.
```

Figure 71: CAHN WLAN access point IP distribution.

7.3 CAHN WLAN ad hoc performance

The total time needed for an ad hoc connection setup using CAHN WLAN averaged 26.5 seconds. This quantity consists of four main factors, which three are not implementation related:

- SMS exchange: 15 seconds
- SMS spooling: 5 seconds
- IP distribution: 4 seconds
- Implementation: 2.5 seconds

To increase readability, a circle diagram is chosen to show the performance aspect of the implementation. The following circle diagram shows how these four factors exactly influence the time performance of the CAHN WLAN ad hoc scenario:

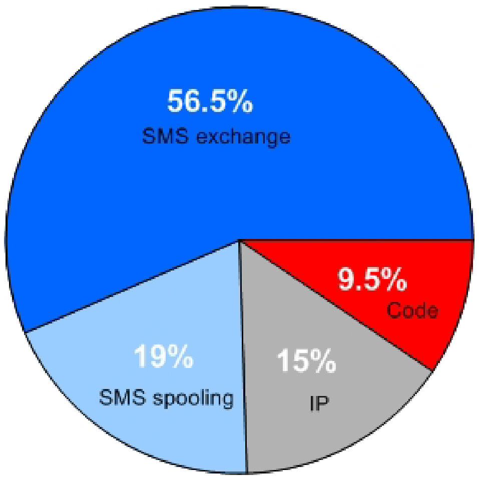


Figure 72: CAHN WLAN ad hoc performance.

7.4 CAHN WLAN access point performance

The total time needed for an access point connection setup using CAHN WLAN averaged 27.5 seconds. The four factors which cause this time dimension are:

- SMS exchange: 15 seconds
- SMS spooling: 5 seconds
- IP distribution: 4 seconds
- Implementation: 3.5 seconds

The following circle diagram shows how these four factors exactly influence the time performance of the CAHN WLAN access point scenario:

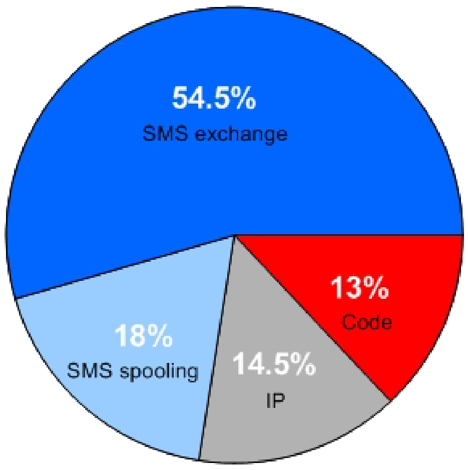


Figure 73: CAHN WLAN access point performance.

Table 7 shows the total connection time for both, CAHN WLAN scenarios and CAHN Bluetooth scenario:

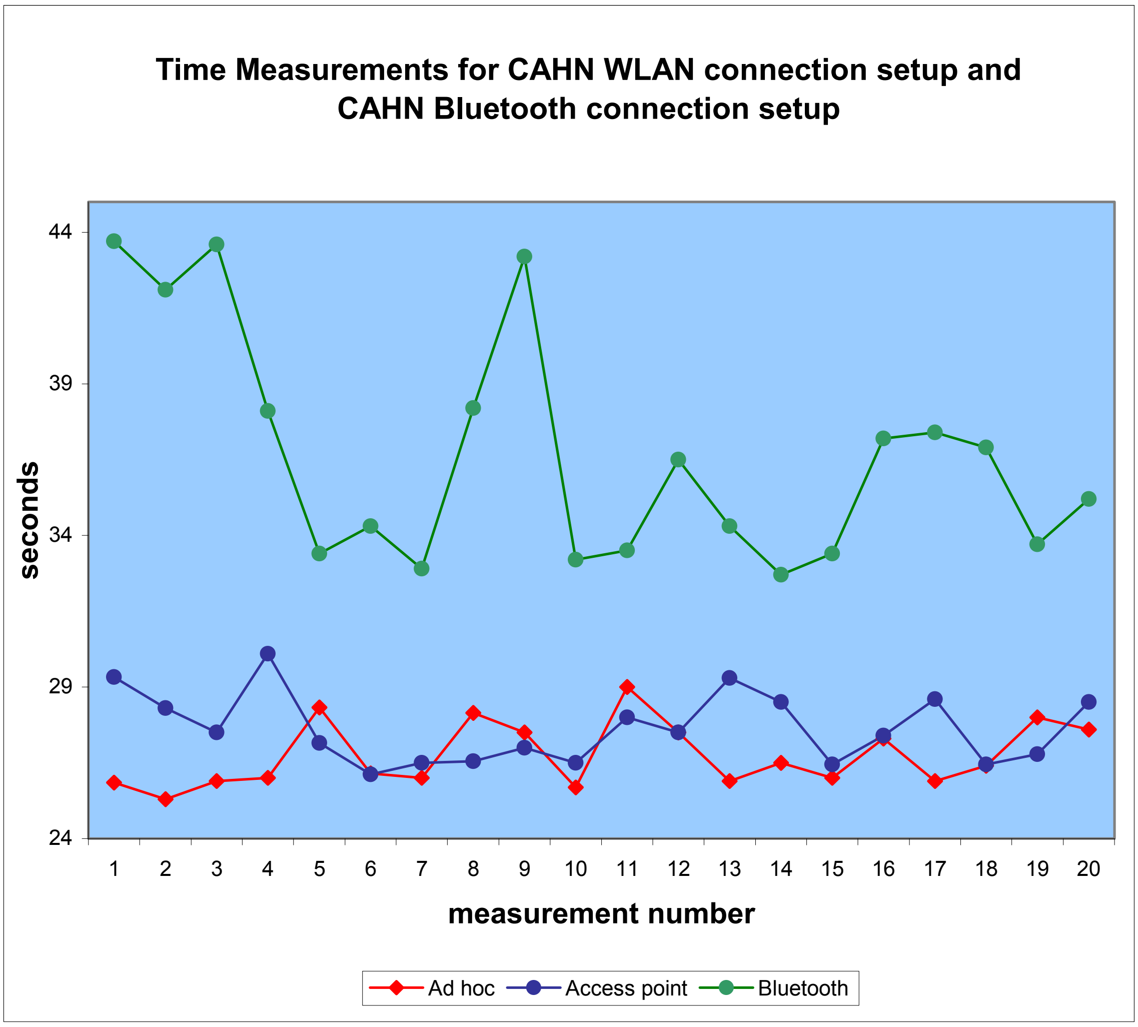


Table 7: Shows the time difference between CAHN Bluetooth and CAHN WLAN.

The PAN connection establishment and the server inquiry cause the main time delay by CAHN Bluetooth protocol [5]:

- PAN connection setup: 6.5 seconds
- Server inquiry: 5 seconds

The WLAN connection setup needs less than 0.2 second to establish a WLAN link. Figure 74 shows the CAHN WLAN message exchange with the related time measurement:

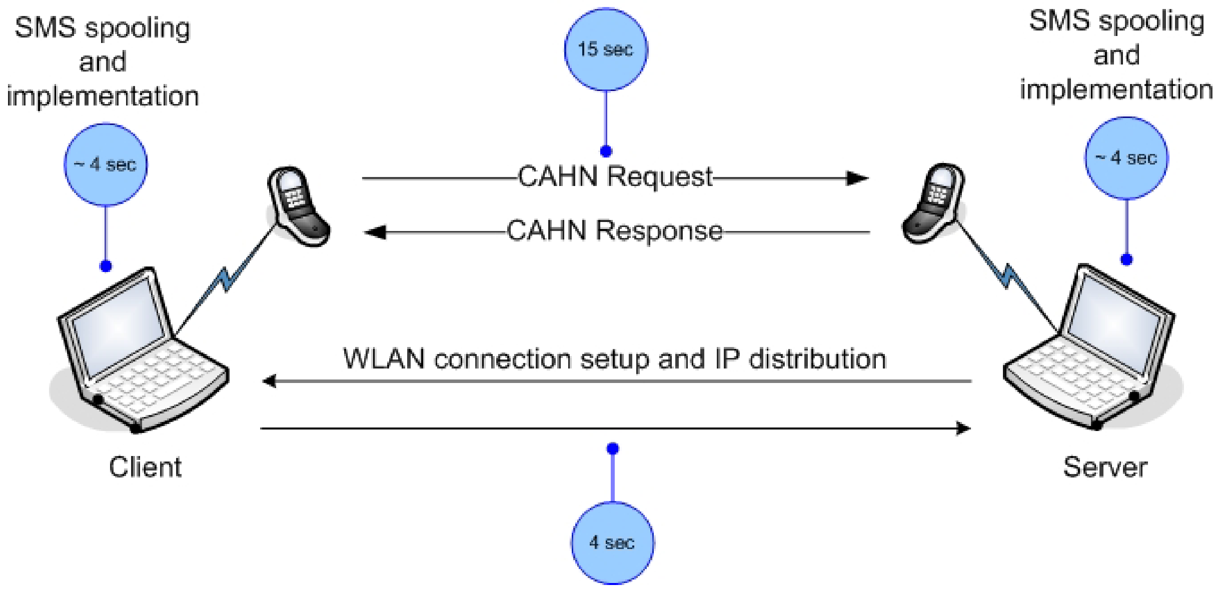

SMS spooling and implementation
~ 4 sec
15 sec
CAHN Request
CAHN Response
SMS spooling and implementation
~ 4 sec
WLAN connection setup and IP distribution
Client
Server
4 sec

Figure 74: The total time measurement of CAHN WLAN scenarios.

8 CONCLUSION

The main goal of this diploma work was to design and implement WLAN support for cellular assisted heterogeneous networking with respect to the existing Bluetooth implementation.

Using the Bloom filter to efficiently use the SSID of WLAN to promote information about the participating CAHN nodes, showed to be a good approach. Yet, a non-zero probability of false positives makes this solution approach susceptible. However, there is a trade off between the number of elements, the size of the vector and the quantity of used hash functions that gives CAHN the possibility to control errors in the system. The other aspect of using the Bloom approach is the message overhead. When a new node joins the network, the SSID of the entire CAHN ad hoc network must be adapted and this causes a complex updating process.

A further issue of development is the scalability and increase of a CAHN ad hoc network. Using the CAHN WLAN protocol in a conference room where all stations reside in the supposed physical range, might be sufficient powerful for all connection requests. But as soon as the dimension of the ad hoc network grows to an amount that two stations are not necessary in their direct range, a separate spontaneous routing algorithm is needed to relay all the IP packets in the network to the intended destinations.

The random rendezvous solution can reduce the message overhead and improves the performance of CAHN. However, this approach does not provide any mechanism to find out whether a certain node is located in a certain ad hoc network. The only way to figure this out is to query the node directly by using its MSISDN.

Over 70% of the performance drawbacks of the CAHN implementation in both scenarios comes from using SMS as transport mechanism. It causes a critical overhead that influences the stability of the implementation, e.g. the lack of the exchange transparency. There is no mechanism in the SMS technology that guarantees the delivery of SMS messages. Actually, a key aspect to improve the performance of CAHN WLAN protocol is to apply an alternative message channel like USSD.

Future work

During this diploma work, no USSD channel could be prepared for the realization and development of the CAHN USSD module. Surely, implementations of the CAHN USSD module will be follow up work of this diploma work. The creation of a billing strategy based on the carried data is also a considerable aspect for the future development and simulations efforts. Integration of the USSD in the CAHN concept will improve the performance and stability of the architecture.

Another possible future work for the CAHN is certainly the implementation of the CAHN WLAN Bloom ad hoc scenario. As discussed in the chapter 5 about Bloom filter, this scenario needs more protocol messages (WLAN and GSM messages) and components than used for the other CAHN scenarios. Consequently, the exchange of WLAN messages during a CAHN ad hoc scenario confirms the need of a proper spontaneous routing algorithm for the Bloom based approach. The IP distribution solution must also be further analyzed.

CAHN still needs much more research work and evaluation to enhance the concept of mobile heterogeneous networking. Future work will continue to explore the CAHN concept but also focus on simulations [31] and security considerations.

REFERENCES

[1] Marc Danzeisen, Daniel Rodellar, Torsten Braun, Simon Winiker, "Heterogeneous networking establishment assisted by cellular operators", The Fifth IFIP TC6 international conference on mobile wireless communications (MWCN 2003), Singapore, October 2003

[2] ETSI standard, "European digital cellular telecommunications system (Phase 2)", March 1999

[3] Bluetooth SIG, "Specification of the Bluetooth System", Version 1.2, November 2003

[4] Wireless LAN specifications, IEEE 802.11 standard, Institute of Electrical and Electronics Engineers, August 1999.

[5] Simon Winiker, "Integration of cellular assisted heterogeneous networking and Bluetooth service discovery protocol", Diploma work at university of Berne, 2004

[6] Burton Bloom, "Space/time tradeoffs in hash coding with allowable error", Communications of the ACM, 1970

[7] Fuchun Joseph Lin, Ming T. Liu, "The Rise of Protocol Engineering", IEEE Software, 1992

[8] Gerhard j. Holzmann, "Desigen and validation of computer protocols", Prentice Hall, 1991

[9] ITU-T, Recommendation Z.100, "Estelle- A Formal Description Technique based on an Extended State, 1997

[10] ITU-T, Recommendation Z.100, "Specification and Description Language ", 1992

[11] ITU-T, Recommendation X.680, "Abstract Syntax Notation One", 2002

[12] ITU-T, Recommendation Z.120, "Message Sequence Charts", 1999

[13] ISO-OSI 8807, "a Formal Description Technique based on the Temporal Ordering of Observational Behaviour", 1989

[14] Hartmut König, "Protocol Engineering", Teubner, 2004

[15] Andrew s. Tanenbaum, "Computer Networks", Prentice Hall, 1996

[16] Behçet Sarikaya, "Principles of Protocol Engineering and Conformance Testing", Ellis Horwood, 1993

[17] "PROMELA and SPIN", http://spinroot.com/

[18] "Bloom filter", http://www.cs.wisc.edu/~cao/papers/summary-cache/

[19] Andrei Broder, Michael Mitzenmacher, "Network applications of Bloom filters: A survey", 40th Annual Allerton Conference on Communication Control and Computing, 2002

[20] L. Fan, P. Cao, J. Almeida, A. Broder, "Summary cache: A scalable wide-area Web cache sharing protocol", ACM SIGCOMM Conference, Vancouver Canada, August 31 - September 4, 1998

[21] Maciej Ceglowski, "Using Bloom filter", http://www.perl.com/pub/a/2004/04/08/bloom_filters.html, April 2004

[22] GSM Recommendation 02.90, "Unstructured Supplementary Services Data"

[23] "Unstructured Supplementary Services Data", http://www.mobileussd.com/

[24] Swisscom mobile, "Mobile vending solution", white paper, http://www.sicap.com/content/pdf_docs/Swisscom_Customer_Case_13920040414152116.pdf, 2000

[25] "WAP over GSM USSD", http://www.wmlclub.com/docs/especwap1.2/SPEC-WAPOverGSMUSSD-19990715.pdf, July 1999

[26] "Signalling system number 7", http://www.ss7.com/

[27] Bob O'hara, al Petrick, "IEEE 802.11 handbook", Standards Information Network, 1999

[28] Matthew s. Gast, "802.11 Wireless networking", O'REILLY, 2002

[29] Jörg Rech, "Wireless LANs", Heise Heinz, 2004

[30] Bruce Potter, Bob fleck, "802.11 Security", O'REILLY, 2002

[31] Isabel Steiner, "Simulations on CAHN", Diploma work at university of Berne, 2005

Appendix

APPENDIX A

False positive rate under various m/n and k combinations:

m/n	*k*	*k*=1	*k*=2	*k*=3	*k*=4	*k*=5	*k*=6	*k*=7	*k*=8
2	1.39	0.393	0.400						
3	2.08	0.283	0.237	0.253					
4	2.77	0.221	0.155	0.147	0.160				
5	3.46	0.181	0.109	0.092	0.092	0.101			
6	4.16	0.154	0.0804	0.0609	0.0561	0.0578	0.0638		
7	4.85	0.133	0.0618	0.0423	0.0359	0.0347	0.0364		
8	5.55	0.118	0.0489	0.0306	0.024	0.0217	0.0216	0.0229	
9	6.24	0.105	0.0397	0.0228	0.0166	0.0141	0.0133	0.0135	0.0145
10	6.93	0.0952	0.0329	0.0174	0.0118	0.00943	0.00844	0.00819	0.00846
11	7.62	0.0869	0.0276	0.0136	0.00864	0.0065	0.00552	0.00513	0.00509
12	8.32	0.08	0.0236	0.0108	0.00646	0.00459	0.00371	0.00329	0.00314
13	9.01	0.074	0.0203	0.00875	0.00492	0.00332	0.00255	0.00217	0.00199
14	9.7	0.0689	0.0177	0.00718	0.00381	0.00244	0.00179	0.00146	0.00129
15	10.4	0.0645	0.0156	0.00596	0.003	0.00183	0.00128	0.001	0.000852
16	11.1	0.0606	0.0138	0.005	0.00239	0.00139	0.000935	0.000702	0.000574
17	11.8	0.0571	0.0123	0.00423	0.00193	0.00107	0.000692	0.000499	0.000394
18	12.5	0.054	0.0111	0.00362	0.00158	0.000839	0.000519	0.00036	0.000275
19	13.2	0.0513	0.00998	0.00312	0.0013	0.000663	0.000394	0.000264	0.000194
20	13.9	0.0488	0.00906	0.0027	0.00108	0.00053	0.000303	0.000196	0.00014
21	14.6	0.0465	0.00825	0.00236	0.000905	0.000427	0.000236	0.000147	0.000101
22	15.2	0.0444	0.00755	0.00207	0.000764	0.000347	0.000185	0.000112	7.46e-05
23	15.9	0.0425	0.00694	0.00183	0.000649	0.000285	0.000147	8.56e-05	5.55e-05
24	16.6	0.0408	0.00639	0.00162	0.000555	0.000235	0.000117	6.63e-05	4.17e-05
25	17.3	0.0392	0.00591	0.00145	0.000478	0.000196	9.44e-05	5.18e-05	3.16e-05
26	18	0.0377	0.00548	0.00129	0.000413	0.000164	7.66e-05	4.08e-05	2.42e-05
27	18.7	0.0364	0.0051	0.00116	0.000359	0.000138	6.26e-05	3.24e-05	1.87e-05
28	19.4	0.0351	0.00475	0.00105	0.000314	0.000117	5.15e-05	2.59e-05	1.46e-05
29	20.1	0.0339	0.00444	0.000949	0.000276	9.96e-05	4.26e-05	2.09e-05	1.14e-05
30	20.8	0.0328	0.00416	0.000862	0.000243	8.53e-05	3.55e-05	1.69e-05	9.01e-06
31	21.5	0.0317	0.0039	0.000785	0.000215	7.33e-05	2.97e-05	1.38e-05	7.16e-06

APPENDIX B

False positive rate under various m/n and k combinations:

m/n	*k*	*k*=9	*k*=10	*k*=11	*k*=12	*k*=13	*k*=14	*k*=15	*k*=16
11	7.62	0.00531							
12	8.32	0.00317	0.00334						
13	9.01	0.00194	0.00198	0.0021					
14	9.7	0.00121	0.0012	0.00124					
15	10.4	0.000775	0.000744	0.000747	0.000778				
16	11.1	0.000505	0.00047	0.000459	0.000466	0.000488			
17	11.8	0.000335	0.000302	0.000287	0.000284	0.000291			
18	12.5	0.000226	0.000198	0.000183	0.000176	0.000176	0.000182		
19	13.2	0.000155	0.000132	0.000118	0.000111	0.000109	0.00011	0.000114	
20	13.9	0.000108	8.89e-05	7.77e-05	7.12e-05	6.79e-05	6.71e-05	6.84e-05	
21	14.6	7.59e-05	6.09e-05	5.18e-05	4.63e-05	4.31e-05	4.17e-05	4.16e-05	4.27e-05
22	15.2	5.42e-05	4.23e-05	3.5e-05	3.05e-05	2.78e-05	2.63e-05	2.57e-05	2.59e-05
23	15.9	3.92e-05	2.97e-05	2.4e-05	2.04e-05	1.81e-05	1.68e-05	1.61e-05	1.59e-05
24	16.6	2.86e-05	2.11e-05	1.66e-05	1.38e-05	1.2e-05	1.08e-05	1.02e-05	9.87e-06
25	17.3	2.11e-05	1.52e-05	1.16e-05	9.42e-06	8.01e-06	7.1e-06	6.54e-06	6.22e-06
26	18	1.57e-05	1.1e-05	8.23e-06	6.52e-06	5.42e-06	4.7e-06	4.24e-06	3.96e-06
27	18.7	1.18e-05	8.07e-06	5.89e-06	4.56e-06	3.7e-06	3.15e-06	2.79e-06	2.55e-06
28	19.4	8.96e-06	5.97e-06	4.25e-06	3.22e-06	2.56e-06	2.13e-06	1.85e-06	1.66e-06
29	20.1	6.85e-06	4.45e-06	3.1e-06	2.29e-06	1.79e-06	1.46e-06	1.24e-06	1.09e-06
30	20.8	5.28e-06	3.35e-06	2.28e-06	1.65e-06	1.26e-06	1.01e-06	8.39e-06	7.26e-06
31	21.5	4.1e-06	2.54e-06	1.69e-06	1.2e-06	8.93e-07	7e-07	5.73e-07	4.87e-07
32	22.2	3.2e-06	1.94e-06	1.26e-06	8.74e-07	6.4e-07	4.92e-07	3.95e-07	3.3e-07

APPENDIX C

False positive rate under various m/n and k combinations:

m/*n*	*k*	*k*=17	*k*=18	*k*=19	*k*=20	*k*=21	*k*=22	*k*=23	*k*=24
22	15.2	2.67e-05							
23	15.9	1.61e-05							
24	16.6	9.84e-06	1e-05						
25	17.3	6.08e-06	6.11e-06	6.27e-06					
26	18	3.81e-06	3.76e-06	3.8e-06	3.92e-06				
27	18.7	2.41e-06	2.34e-06	2.33e-06	2.37e-06				
28	19.4	1.54e-06	1.47e-06	1.44e-06	1.44e-06	1.48e-06			
29	20.1	9.96e-07	9.35e-07	9.01e-07	8.89e-07	8.96e-07	9.21e-07		
30	20.8	6.5e-07	6e-07	5.69e-07	5.54e-07	5.5e-07	5.58e-07		
31	21.5	4.29e-07	3.89e-07	3.63e-07	3.48e-07	3.41e-07	3.41e-07	3.48e-07	
32	22.2	2.85e-07	2.55e-07	2.34e-07	2.21e-07	2.13e-07	2.1e-07	2.12e-07	2.17e-07

APPENDIX D

Memory usage of the Bloom filter:

False positive	Keys	Required size	Bytes/Key
1%	1K	1.87 K	1.9
0.1%	1K	2.80 K	2.9
0.01%	1K	3.74 K	3.7
0.01%	10K	37.4 K	3.7
0.01%	100K	374 K	3.7
0.01%	1M	3.74 M	3.7
0.001%	1M	4.68 M	4.7
0.0001%	1M	5.61 M	5.7

APPENDIX E

```
Specification CAHN_BLUETOOTH_PROTOCOL
Service
        CAHN_BLUETOOTH_PROTOCOL-CONNECTION_SETUP
            Requested cahn_send_data( sock: integer,
                                      buf: char,
                                      size: integer
                                     )
                    adapter_send_data( sock: integer,
                                       buf: char,
                                       size: integer,
                                       resp_sock: integer
                                         )


            responded cahnd_read_response( sock: integer,
                                            rsp_buf: char,
                                            rsp_buf_len: integer
                                           )
                    response_socket_send_data( data: char,
                                                data_length: integer,
                                                response_socket: integer
                                                )

        sap Client
            signal connected, error
            phase CONNECT, DATA_TRANSFER

            phase CONNECT:
                cahn_send_data( sock, buf, size)
                adapter_send_data( sock, buf, size, resp_sock) : server. cahn_send_data
                wait event
                    connected: respond response_socket_send_data( data,
                                                data_length,response_socket)
                    error: respond response_socket_send_data( data,
                                                data_length,response_socket)
                #wait
                blt_process_packet(Cahn_blt_ser_response , response_buf)
                cahnd_read_response( sock, rsp_buf, rsp_buf_len)
                set DATA_TRANSFER

            #phase // CONNECT

            phase DATA_TRANSFER

                // Bluez Stack …

            #phase
        #sap //Client

        sap server
            signal
            phase CONNECT, DATA_TRANSFER
```

```
                phase CONNECT:
                    client.adapter_send_data:      respond cahn_send_data( sock, buf, size)
                                                   blt_process_packet()


                #phase              // CONNECT

                phase DATA_TRANSFER

                    // Bluez stack

                #phase
            #sap
#service //CAHN_BLUETOOTH_PROTOCOL-CONNECTION_SETUP
message
            Cahn Bluetooth Service Request: record(PDU_ID: integer
                                                   P_LEN: integer
                                                   SEQ_ NBR: integer
                                                   SRC_MSISDN: char
                                                   DST_MSISDN: char
                                                   SRC_BA: char
                                                   DST_BA: char
                                                   Bluetooth Service ID: integer
                                                   )
            Cahn Bluetooth Service Response: record(PDU_ID: integer
                                                    P_LEN: integer
                                                    SEQ_ NBR: integer
                                                    SRC_MSISDN: char
                                                    DST_MSISDN: char
                                                    SRC_BA: char
                                                    DST_BA: char
                                                    Bluetooth Service ID: integer
                                                    PIN: integer
                                                    )
            Cahn Error Response: record(PDU_ID: integer
                                        P_LEN: integer
                                        SEQ_ NBR: integer
                                        SRC_MSISDN: char
                                        DST_MSISDN: char
                                        SRC_BA: char
                                        DST_BA: char
                                        Error Code: integer
                                        )


protocol
            CONNECT: CAHNCLIENT. cahn_blt_client ↔ CAHNSERVER. cahn_blt_server
            DATA_TRANSFER


entity CAHNCLIENTS
            sap client
            var
            begin
```

```
          phase CONNECT: cahn_blt_client ‖ sap handler CAHNCLIENT
          phase DATA_TRANSFER
        end// CAHNCLIENT

sap handler CAHNCLIENTS
        service SMS: requested spool SMS
                     requested send SMS
                     responded receive SMS
        loop
            wait event
                     Cahn Bluetooth Service Request: request spool SMS
                                                     request send SMS  |

                     receive SMS:
                             case code of
                                  Cahn Bluetooth Service Response:
                                  set event Cahn Bluetooth Service Response  |
                                  Cahn Bluetooth Error Response:
                                  set event Cahn Bluetooth Error Response
                             #case
            #wait
        #loop
#sap


entity CAHNSERVER
        sap server
        var
        begin
          phase CONNECT: cahn_blt_server ‖ sap handler CAHNSERVER
          phase DATA_TRANSFER
        end //CAHNSERVER

sap handler CAHNSERVER
        service SMS: requested spool SMS
                     requested send SMS
                     responded receive SMS
        loop
            wait event
                     Cahn Bluetooth Service Response, Cahn Bluetooth Error Response:
                                                     request spool SMS
                                                     request send SMS  |
                     receive SMS:
                             Cahn Bluetooth Service Request:
                                     set event Cahn Bluetooth Service Request
            #wait
        #loop
#sap


protocol part cahn_blt_client (peer server)
        timer t: 0..? ms
          begin
        make_cahn_bluetooth_service_request(SRC_MSISDN,DST_MSISDN,
                                           SRC_BA,DST_BA)
            cahn bluetooth service request → server
            start t
            wait event
                cahn bluetooth service response ← server
                    reset t
                    respond respond_socket_send_data(cahn bluetooth service response)
                    set DATA_TRANSFER
                timeout t: respond respond_socket_send_data()
            #wait
          end //cahn_blt_client

protocol part cahn_blt_server (peer client)
          begin
            wait event
                cahn bluetooth service request ← client
                    respond adapter_send_data(cahn bluetooth service request)
                    respond cahnd_send_data(cahn bluetooth service request)
                    process_packet(cahn bluetooth service request)
                    cahn bluetooth service response → client
            #wait
          end //cahn_blt_server

#spec
```

APPENDIX F

```
specification CAHN_PROTOCOL
service
        CAHN_PROTOCOL-CONNECTION_SETUP
            requested cahnd_send_data(
                                    sock: integer,
                                    buf: char,
                                    size: integer
                                    )
                    adapter_send_data(
                                     sock: integer,
                                     buf: char,
                                     size: integer,
                                     resp_sock: integer
                                         )


            responded cahnd_read_response(
                                        sock: integer,
                                        rsp_buf: char,
                                        rsp_buf_len: integer
                                        )
                    response_socket_send_data(
                                             data: char,
                                             data_length: integer,
                                             response_socket: integer
                                             )

        sap Client
            signal connected, error
            phase CONNECT, DATA_TRANSFER

            phase CONNECT:
               cahn_send_data( sock, buf, size)
               adapter_send_data( sock, buf, size, resp_sock) : server. cahn_send_data
               wait event
                    connected: respond response_socket_send_data( data,
                                             data_length,response_socket)
                    error: respond response_socket_send_data( data,
                                             data_length,response_socket)
               #wait
               cahn_process_packet(Cahn_ser_response , response_buf)
               cahnd_read_response( sock, rsp_buf, rsp_buf_len)
               set DATA_TRANSFER

            #phase // CONNECT

            phase DATA_TRANSFER

               // depends on technology e. g. Bluetooth or WLAN …

            #phase
        #sap //Client
        sap server
            signal
            phase CONNECT, DATA_TRANSFER

            phase CONNECT:
                client.adapter_send_data:      respond cahn_send_data( sock, buf, size)
                                               cahn_process_packet()


            #phase          // CONNECT

            phase DATA_TRANSFER

                        // depends on technology e. g. Bluetooth or WLAN …

            #phase
        #sap
#service //CAHN_PROTOCOL-CONNECTION_SETUP
message
        Cahn Service Request: record(PDU_ID: integer
                                     P_LEN: integer
                                     SEQ_ NBR: integer
                                     SRC_MSISDN: char
                                     DST_MSISDN: char
                                     SRC_ADR: char
                                     DST_ADR: char
                                     )
        Cahn Service Response: record(PDU_ID: integer
                                      P_LEN: integer
                                      SEQ_ NBR: integer
                                      SRC_MSISDN: char
                                      DST_MSISDN: char
                                      SRC_ADR: char
                                      DST_ADR: char
                                      DATA: integer
                                      )
        Cahn Error Response: record(PDU_ID: integer
                             P_LEN: integer
                             SEQ_ NBR: integer
                             SRC_MSISDN: char
                             DST_MSISDN: char
                             SRC_ADR: char
                             DST_ADR: char
                             Error Code: integer
                             )

protocol
        CONNECT: CAHNCLIENT. cahn_client ↔ CAHNSERVER. cahn_server
        DATA_TRANSFER


entity CAHNCLIENTS
        sap client
```

```
          var
          begin
             phase CONNECT: cahn_client ‖ sap handler CAHNCLIENT
             phase DATA_TRANSFER
          end// CAHNCLIENT

sap handler CAHNCLIENTS
          service USSD: requested spool USSD
                        requested send USSD
                        responded receive USSD
          loop
               wait event
                     Cahn Service Request:  request spool USSD
                                            request send USSD |

                     receive USSD:
                               case code of
                                     Cahn Service Response:
                                     set event Cahn Service Response |
                                     Cahn Error Response:
                                     set event Cahn Error Response
                               #case
               #wait
          #loop
#sap


entity CAHNSERVER
          sap server
          var
          begin
             phase CONNECT: cahn_server ‖ sap handler CAHNSERVER
             phase DATA_TRANSFER
          end //CAHNSERVER

sap handler CAHNSERVER
          service USSD: requested spool USSD
                        requested send USSD
                        responded receive USSD
          loop
               wait event
                     Cahn Service Response, Cahn Error Response:
                                                    request spool USSD
                                                    request send USSD |
                     receive USSD:
                               Cahn Service Request:
                                          set event Cahn Service Request
               #wait
          #loop
#sap


protocol part cahn_client (peer server)
          timer t: 0..? ms
             begin
```

```
                    make_cahn_service_request(SRC_MSISDN,DST_MSISDN,SRC_ADR,DST_ADR)
                    cahn service request → server
                    **start** t
                    **wait event**
                        cahn service response ← server
                            **reset** t
                            **respond** respond_socket_send_data(cahn service response)
                            **set** DATA_TRANSFER
                        **timeout** t: **respond** respond_socket_send_data()
                    **#wait**
                **end** //cahn_client

protocol part cahn_server (peer client)
                **begin**
                    **wait event**
                        cahn service request ← client
                            **respond** adapter_send_data(cahn service request)
                            **respond** cahnd_send_data(cahn service request)
                            process_packet(cahn service request)
                            cahn service response → client
                    **#wait**
                **end** //cahn_server

#spec
```

APPENDIX G

```
/***************************************************************************
 *                         -  description
 *                            -------------------
 *   begin                : Mon April 26 2004
 *   copyright            : (C)
 *   email                :
 *
 *                       PROMELA Validation and Simulation Model
 *                       CAHN bluetooth protocol
 *                       by Ehsan Maghsoodi
 *
 *

***************************************************************************/

/*
 * These are some general CONT definitions.
 *
 */
#define CAHN_RESPONSE_TIMEOUT 20
#define CAHN_REQ_BUFFER_SIZE  1024
#define CAHN_RSP_BUFFER_SIZE  1024
#define CAHN_PDU_CHUNK_SIZE   512
#define MAX_SIZE_OF_MSISDN    25
#define SIZE_OF_BLUETOOTH_ADR 2     /* the size of the bluetooth address*/
#define     SOCKET            1
#define BLT_CH          8

#define IF              if ::
#define ENDIF           fi
#define ELSE            ::  else
#define FI              :: else fi
#define  true           1
#define  false          0
#define  or       ||
#define  and            &&

#define CAHN_ERROR_RSP        1
#define CAHN_BLT_SVC_REQ      2
#define CAHN_BLT_SVC_RSP      3


/*
 * (Internal) Error codes
 * May be i need them too.
 */
#define CAHN_UNKNOWN_PDU_ID                        1
#define CAHN_INVALID_PDU_SIZE                      2
#define CAHN_CONNECT_ERROR                         3
#define CAHN_SEND_ERROR                            4
#define CAHN_READ_ERROR                            5
#define CAHN_INTERNAL_SERVER_ERROR                 6
#define CAHN_DEVICE_NOT_IN_RANGE_ERROR             7
#define CAHN_DEVICE_ALREADY_REGISTERED_ERROR       8
#define CAHN_ADAPTER_SEND_ERROR                    9
#define CAHN_ADAPTER_READ_ERROR                    10
#define CAHN_PROCESS_RSP_ERROR                     11
#define CAHN_PREPARE_RESPONSE_SOCKET_ERROR         12

/* Error sub routine */
#define error_messages        IF    \
                              (data == 1) -> printf("Remote did not 
understand this packet type\n");    \
                        ENDIF;      \
                        IF    \
                              (data == 2) -> printf("Invalid PDU size\n");
      \
                        ENDIF;      \
                        IF    \
                              (data == 3) -> printf("Not possible to 
connect\n");      \
                        ENDIF;      \
                        IF    \
                              (data == 4) -> printf("Not possible to send to 
cahnd\n");  \
                        ENDIF;      \
                        IF    \
                              (data == 5) -> printf("Not possible to read 
from cahnd\n");   \
                        ENDIF;      \
                        IF    \
                              (data == 6) -> printf("Internal server 
error\n");  \
                        ENDIF;      \
                        IF    \
                              (data == 7) -> printf("Device not in 
range\n");  \
                        ENDIF;      \
                        IF    \
                              (data == 8) -> printf("Already registered\n");
      \
                        ENDIF;      \
                        IF    \
                              (data == 9) -> printf("Not possible to send to 
adapter\n");      \
                        ENDIF;      \
                        IF    \
                              (data == 10) -> printf("Not possible to read 
form adapter\n"); \
                        ENDIF;      \
                        IF    \
                              (data == 11) -> printf("Not possible to 
process the answer\n"); \
                        ENDIF;      \
                        IF    \
                              (data == 12) -> printf("Socket error\n");\
                        ENDIF;      \


/*****************************************
      Type definition

*****************************************/

/*
 * CAHN PDU
 */

typedef cahn_pdu_header{
      int pdu_id;
      int p_len;
      int seq_nbr
};

/*
 * CAHN MSISDN HEADER
 */

typedef cahn_msisdn_header{
      int src_msisdn;
      int dst_msisdn
};
/* Cahn msisdn header sub "routine" */
#define cahn_msisdn_header   atomic{src_msisdn = 791234567; dst_msisdn =
797654321}
#define cahn_msisdn_init(src, dst) = {src_msisdn = src; dst_msisdn = dst}

/*
 * CAHN BLUETOOTH HEADER
 */

typedef cahn_bdasddr{
      byte src_ba[SIZE_OF_BLUETOOTH_ADR];
      byte dst_ba[SIZE_OF_BLUETOOTH_ADR];
      int blt_id
};

#define cahn_bluetooth_header      atomic {src_ba[SIZE_OF_BLUETOOTH_ADR] =
255;  \
                              dst_ba[SIZE_OF_BLUETOOTH_ADR] = 100}
#define cahn_blt_header       atomic {src_ba = 255;   \
                              dst_ba = 100;     \
                              blt_id = BLT_CH}


/*
 * CAHN BLUETOOTH DATA
 */

typedef cahn_blt_data{
      int data
};
/* Cahn data sub "routine" */
#define pin_init atomic{data = 10}
#define pin_generate(pin) atomic{data = pin}
#define get_pin   data
#define get_error data
#define error_code_init(errorCode)  atomic{data = errorCode}

typedef cahn_blt_message{
      int pdu_id;
      int p_len;
      int seq_nbr;
      int src_msisdn;
      int dst_msisdn;
      byte src_ba[SIZE_OF_BLUETOOTH_ADR];
      byte dst_ba[SIZE_OF_BLUETOOTH_ADR];
      int blt_id;
      int data

};

/*
 * Global variable.
 * Do not model opening connection, closing socket and
 * syncing of connections at this level of abstraction.
 * Channels simulate the all those things.
 */

mtype = {response,error, request, await_response, send_request,
send_response, EOF_request, EOF_response, sms_end}
```

```
cahn_blt_message in_cahn_client, in_cahn_server;
mtype status, type;


/*************************
 *
 * Channel:
 *
 * There are two channels
 *
 ************************/

chan cahn_adaptor_client = [0] of {cahn_blt_message}
chan cahn_adaptor_server = [0] of {cahn_blt_message}


/******************************
      Process and methods

******************************/

/* Chan header response sub "routine" */
#define response_header_init  atomic {pdu_id = CAHN_BLT_SVC_RSP;  \
                              seq_nbr = SOCKET; \
                              p_len = CAHN_RSP_BUFFER_SIZE}

#define get_data  atomic {int src_msisdn_tmp = src_msisdn;  \
                  int dst_msisdn_tmp = dst_msisdn;    \
                  int src_ba_tmp = src_ba;      \
                  int dst_ba_tmp = dst_ba;      \
                  \
                  dst_msisdn = src_msisdn_tmp;  \
                  src_msisdn = dst_msisdn_tmp;  \
                  dst_ba = src_ba_tmp;    \
                  src_ba = dst_ba_tmp}


#define response_socket_init(socket_nbr)  atomic {pdu_id = CAHN_BLT_SVC_RSP;
      \
                                          seq_nbr = socket_nbr;   \
                                          p_len = CAHN_RSP_BUFFER_SIZE}

active proctype server(){

      mtype messageType;      /* Message type  */
      byte pdu_id;            /* ID of message */
      int p_len;        /* The length of the message     */
      int seq_nbr;            /* Response socket*/
      int dst_msisdn;
      int src_msisdn;
                  /* int dst_ba_array[SIZE_OF_BLUETOOTH_ADR]; */
                  /* int src_ba_array[SIZE_OF_BLUETOOTH_ADR]; */
      int dst_ba;
      int src_ba;
      int blt_id;
      int data;         /* Pin code or error code*/



      AWAIT_REQUEST:
            do
            :: cahn_adaptor_client? pdu_id(p_len, seq_nbr, src_msisdn,
dst_msisdn, src_ba, dst_ba, blt_id, 0, EOF_request, sms_end) ->
            accept:     IF
                  (pdu_id == CAHN_BLT_SVC_REQ)  ->
```

```
                              goto PROCESS
                  ELSE
                        assert(false);    /* protocol violation */
                        status = error;

                  ENDIF;
            od;

      PROCESS:
            /* CAHN process packet...*/
            /* Check whether the device has already registered for the
service in the DB */
            /* Check whether the device is in range. */
            /* Generate a unique PIN for the device */
            /* Generate the response */

            /* get_data; */
             pin_init;
             response_socket_init(seq_nbr);/* response_header_init; */
             cahn_adaptor_server ! pdu_id(p_len, seq_nbr, dst_msisdn,
src_msisdn, dst_ba, src_ba, blt_id, data, EOF_response, sms_end);


      EXIT:
            end:
            skip;
            /*    Done.
             *    Here suppose to be data transfer actions whicht are
Bluetooth
             *    spesific informations.
             */
}

/* Chan header request sub "routine" */
#define request_header_init   atomic {pdu_id = CAHN_BLT_SVC_REQ;  \
                              seq_nbr = SOCKET; \
                              p_len = CAHN_REQ_BUFFER_SIZE}




active proctype client(){


      mtype messageType;
      byte pdu_id;
      int p_len;
      int seq_nbr;
      int src_msisdn;
      int dst_msisdn;
                  /* represent bluetooth addresses in ASCII format */
                  /* int dst_ba_array[SIZE_OF_BLUETOOTH_ADR]; */
                  /* int src_ba_array[SIZE_OF_BLUETOOTH_ADR]; */
      int dst_ba;
      int src_ba;
      int blt_id;
      int data;

      /* generate a cahn bluetooth request */
      /* sends the request */

      request_header_init;
      cahn_msisdn_header;
      cahn_blt_header;
      cahn_adaptor_client!pdu_id(p_len, seq_nbr, src_msisdn, dst_msisdn,
src_ba, dst_ba, blt_id, 0, EOF_request, sms_end);
      /* The both EOF_request and sms_end are not part of the CAHN request.
*/
      /* They just simulating end of a message. */

      /* Awaiting  for CAHN request from user */
      goto AWAIT_RESPONSE;

      /* CAHN client is wating for the response. */
      AWAIT_RESPONSE:
            do
            :: cahn_adaptor_server?
                 pdu_id(p_len, seq_nbr, src_msisdn, dst_msisdn, src_ba,
dst_ba, blt_id, data, EOF_response, sms_end) ->
            :: timeout -> break; /* Error message */
            od;

      accept:     IF
                  (pdu_id == CAHN_BLT_SVC_RSP) ->
                        /* Get data from the cahn response message. */
                        /* Mainly the pin code for blt connection setup. */
                        get_data;
                        goto EXIT;
            ENDIF;
            IF
                  (pdu_id == CAHN_ERROR_RSP) ->
                                    get_error;
                                    error_messages;

            ELSE ->
                  assert(false)     /* protocol violation */

            ENDIF;


      EXIT:
            get_pin;
            /* Seting up a bluetooth connection. */
            printf("CAHN is connected.\n");
            end:
            skip
            /* Done.
             * Here suppose to be data transfer action.
             */

}

/**************************
      Main

**************************/

init{

        do
        :: run client();
        :: run server();
        :: break;
        od;

}
```

APPENDIX H

```
5 Hashes
real 0m0.048s
real 0m0.049s
real 0m0.050s
10 Hashes
real 0m0.111s
real 0m0.092s
real 0m0.112s
15 Hashes
real 0m0.135s
real 0m0.135s
real 0m0.135s
18 Hashes
real 0m0.183s
real 0m0.161s
real 0m0.163s
20 Hashes
real 0m0.199s
real 0m0.201s
real 0m0.202s
25 Hashes
real 0m0.223s
real 0m0.243s
real 0m0.222s
30 Hashes
real 0m0.297s
real 0m0.273s
real 0m0.270s
35 Hashes
real 0m0.332s
real 0m0.336s
real 0m0.337s
40 Hashes
real 0m0.380s
real 0m0.404s
real 0m0.379s
45 Hashes
real 0m0.423s
real 0m0.425s
real 0m0.447s
50 Hashes
real 0m0.465s
real 0m0.467s
real 0m0.493s
######################################################################
```

Example for 50 Hashes (MSISDN=791234567):

```
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn1
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn2
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn3
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn4
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn5
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn6
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn7
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn8
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn9
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn10
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn11
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn12
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn13
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn14
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn15
```

```
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn16
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn17
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn18
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn19
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn20
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn21
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn22
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn23
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn24
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn25
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn26
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn27
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn28
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn29
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn30
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn31
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn32
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn33
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn34
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn35
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn36
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn37
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn38
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn39
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn40
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn41
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn42
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn43
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn44
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn45
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn46
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn47
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn48
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn49
/usr/bin/openssl dgst -sha1 -hex /root/SHA1-test/msisdn50
[root@CAHN SHA1-test]# time ./SHA1-50test-hex
SHA1(/root/SHA1-test/msisdn1)=
095941b63cd12333d5d86ff267d789d6c9035c90
SHA1(/root/SHA1-test/msisdn2)=
7ebed142a1a05634ccc88c3c30ca628ab7ddff94
SHA1(/root/SHA1-test/msisdn3)=
19b2fa63a356eb779ab65b8da9112c2ab23a9c4d
SHA1(/root/SHA1-test/msisdn4)=
143d578438e789ae2816a2053e5bf76e16359d85
SHA1(/root/SHA1-test/msisdn5)=
724d44dd155e4a5cad96278d767586086f47842c
SHA1(/root/SHA1-test/msisdn6)=
8faa06090333adf68452cd196eee914eb8ec2f98
SHA1(/root/SHA1-test/msisdn7)=
02ea7103da3e68a979c0f8b86381f3e48de3e826
SHA1(/root/SHA1-test/msisdn8)=
29101edc31b3c32addf3607c6094a0beef43d67a
SHA1(/root/SHA1-test/msisdn9)=
8bf7d84b6c06719e6b1d4ea4e039179017e6b4bd
SHA1(/root/SHA1-test/msisdn10)=
d367a21c9e2c4259daab2ab5b8ac9833a03c6538
SHA1(/root/SHA1-test/msisdn11)=
e994f92db46cab84be9061accaafbf1136811481
SHA1(/root/SHA1-test/msisdn12)=
c89197b2f2b253a9374cb331b59284e20dcf62db
SHA1(/root/SHA1-test/msisdn13)=
bf80a9228b9b9ec45962f38df09f0376887568a7
SHA1(/root/SHA1-test/msisdn14)=
8586569b2c3b5f1af2ca211d3d6bb1f3b3a6d6e0
```

```
SHA1(/root/SHA1-test/msisdn15)=
f53529ec317d5cfc1c45a5d858ac2133eba3ae6a
SHA1(/root/SHA1-test/msisdn16)=
8da529db545ed0914deb3ee2d6eb928575475101
SHA1(/root/SHA1-test/msisdn17)=
fc5eb4b7ff353f0b500c31877258fdf2b2183e9c
SHA1(/root/SHA1-test/msisdn18)=
92450de7390176e78967e82ca991be2afa860be2
SHA1(/root/SHA1-test/msisdn19)=
56e18f3c106ef8fd4cfdc735167cad8e2778f44e
SHA1(/root/SHA1-test/msisdn20)=
9ebf204ae3b9af5b94b58e481ccf7cfb6ea99008
SHA1(/root/SHA1-test/msisdn21)=
a0034577f5947dcc9285d3835fa33571ce2e5ac7
SHA1(/root/SHA1-test/msisdn22)=
640cb21762774c713eaf9d69869e2efa2ac07b64
SHA1(/root/SHA1-test/msisdn23)=
afa296aee8b9de39f7b1bb2e7612f95591cc2d83
SHA1(/root/SHA1-test/msisdn24)=
ca40bfa5417309365a8ddbd41f7136d136f3a602
SHA1(/root/SHA1-test/msisdn25)=
00e22cbea301ec87afe53b6430b70cd6d66e80c4
SHA1(/root/SHA1-test/msisdn26)=
7908c96fe6315e967698139958f53f1e2802a558
SHA1(/root/SHA1-test/msisdn27)=
5b54e08589aab75f0d0b1f207c85ccfefde9c379
SHA1(/root/SHA1-test/msisdn28)=
37bea2191c4d9d42a80f5230fd5edab8b8c0282e
SHA1(/root/SHA1-test/msisdn29)=
6f0418ed21e7f31d6bd9295077982e0502c44df0
SHA1(/root/SHA1-test/msisdn30)=
a0518a5b45cf8026c599da552c6dd9bdeb0a368f
SHA1(/root/SHA1-test/msisdn31)=
fbb081a4d8fd80cf88b3eadf6a9127ec1b6f2ed1
SHA1(/root/SHA1-test/msisdn32)=
3888406c522c4dc84edc4948a05cd20c0f52e9e5
SHA1(/root/SHA1-test/msisdn33)=
c53ad398279307d2075a4a83143fe04fe37bd041
SHA1(/root/SHA1-test/msisdn34)=
43eebf47c8bd88823b91d8b193a7f27fc2cbe2e3
SHA1(/root/SHA1-test/msisdn35)=
58bb0c00753eb2c008f1313f6af9f511ddd2b89d
SHA1(/root/SHA1-test/msisdn36)=
85ea9ffa528bc81caae23c75ec1e54490af5cff5
SHA1(/root/SHA1-test/msisdn37)=
ba2896e86cff7813f7c940e8338362aa2b801709
SHA1(/root/SHA1-test/msisdn38)=
1d669c9a2a13eda8ce4696be9b64ae2d16c021fe
SHA1(/root/SHA1-test/msisdn39)=
2a155601ed2a9ff32f22a83108f5dd6c61fd5bd5
SHA1(/root/SHA1-test/msisdn40)=
cde2d7aac2ce6795b49fbb1ef545aa9e15304a6e
SHA1(/root/SHA1-test/msisdn41)=
8cfd3c32d5df9a6904207502d8eca6bfbb3f5963
SHA1(/root/SHA1-test/msisdn42)=
1d99e192dc197f6f505cf3790f9e65340019af57
SHA1(/root/SHA1-test/msisdn43)=
cacd8b6c299b2672893963cfe10f147b1c6bbb58
SHA1(/root/SHA1-test/msisdn44)=
091bdb29209cb3b38f1c5645b5aa469be481ec04
SHA1(/root/SHA1-test/msisdn45)=
9a25a3cbe4c2cb9a99de309e96e3bc332088ea9d
SHA1(/root/SHA1-test/msisdn46)=
87eb5a048cb0814f284e09e26490fe383aff812a
```

```
SHA1(/root/SHA1-test/msisdn47)=
b6ad287e6430b7acace72621db2443465fe5e744
SHA1(/root/SHA1-test/msisdn48)=
62e1c492b90cb7ee5cde809c16c35ed3e82cf307
SHA1(/root/SHA1-test/msisdn49)=
fa708253268c03187d4e74b37d9850b7f087e926
SHA1(/root/SHA1-test/msisdn50)=
d068d04d637eb0bb80c8143fd3433cf2448e6145
real 0m0.493s
user 0m0.380s
sys 0m0.060s
##############################################################################
```