Jarno Ristimäki

Android Interface to a Wireless Autonomous Wide-Area Sensor Network

Helsinki Metropolia University of Applied Sciences
Master of Engineering
Information Technology
Thesis
7 May 2012

Metropolia

Abstract

Author(s) Title Number of Pages Date	Jarno Ristimäki Android Interface to a Wireless Autonomous Wide-Area Sensor Network 66 pages 7 May 2012
Degree	Master of Engineering
Degree Programme	Information Technology
Specialisation option	Mobile Programming
Instructor(s)	Jaime Jiménez, Researcher Antti Piironen, Principal Lecturer

This thesis was done for LM Ericsson's research department, NomadicLab. The aim of the thesis was to create an Android application to a wireless autonomous wide-area sensor network. The objective was to be able to monitor and control nodes, sensors and actuators from a mobile device through a graphical user interface. Another objective was to be able to set associations between sensors and actuators with the application.

The application was based on an existing prototype which had a command line interface for monitoring and controlling nodes in a wireless sensor network. It was developed as a native Android application as the HTML5 web application did not meet the requirements in the beginning.

A web service was developed along with the Android application to provide an interface towards the nodes, sensors and actuators in the peer-to-peer network. The application used the HTTP GET method to request data from the web service. HTTP POST was used for sending data. The web service was connected to the P2P network via Java RMI. The data was returned from the web service in the JSON and XML format.

The application was taken into use with the prototype. It was developed further and it was used to demonstrate the prototype solution in various events.

Keywords	M2M, IoT, Android, peer to peer, sensor, HTML5, jQuery, mobile

Metropolia

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Jarno Ristimäki Android-käyttöliittymä langattomaan autonomiseen laaja-aluesensoriverkkoon 66 sivua 7.5.2012
Tutkinto	insinööri (ylempi AMK)
Koulutusohjelma	tietotekniikka
Suuntautuminen	mobiiliohjelmointi
Ohjaajat	tutkija Jaime Jiménez koulutuspäällikkö Antti Piironen

Tämä opinnäytetyö tehtiin LM Ericssonin tutkimusosastolle, NomadicLabille. Työn tarkoituksena oli kehittää Android-sovellus langattomaan autonomiseen laaja-aluesensoriverkkoon. Tavoitteena oli mahdollistaa noodien, sensoreiden ja aktuaattoreiden seuranta ja hallinta graafisen käyttöliittymän avulla käyttäen mobiililaitetta. Toinen tavoite oli pystyä määrittämään sovelluksella assosiaatioita eri sensoreiden ja aktuaattoreiden välille.

Sovellus perustui olemassaolevaan prototyyppiin, jossa oli komentorivikäyttöliittymä langattoman sensoriverkon noodien seurantaan ja hallintaan. Se kehitettiin natiiviksi Android-sovellukseksi, sillä HTML5-verkkosovellus ei täyttänyt alussa vaatimuksia.

Android-sovelluksen ohessa luotiin verkkopalvelu tarjoamaan rajapinta vertaisverkon noodeille, sensoreille ja aktuaattoreille. Sovellus käytti HTTP GET-pyyntöjä tiedon pyytämiseen verkkopalvelulta. HTTP POST-metodia käytettiin tiedon lähettämiseen. Verkkopalvelu yhdistettiin vertaisverkkoon Java RMI:n avulla. Tiedot verkkopalvelusta palautettiin XML- ja JSON-muodossa.

Sovellus otettiin käyttöön prototyypin yhteydessä. Sovellusta kehitettiin edelleen ja sitä käytettiin esittelemään prototyyppiratkaisua eri tapahtumissa.

Keywords	M2M, IoT, Android, vertaisverkko, anturi, HTML5, jQuery, mobiili

Contents

Abbreviations and Terms

Used abbreviations

3G	Third Generation mobile communication system
6LoWPAN	IPv6 over Low-power Wireless Area Networks
ADT	Android Development Tools
Ajax	Asynchronous JavaScript and XML
API	Application Programming Interface
APK	Android's Application Package file
APL	Application Layer
APS	Application Support Sublayer
ASP	Active Server Pages
BEEP	Blocks Extensible Exchange Protocol
CAN	Content Addressable Network
CERP-IoT	Cluster of European Research Projects on the Internet of Things
CLI	Command Line Interface
CoAP	Constrained Application Protocol
CoRE	Constrained RESTful Environments
CSS3	Cascading Style Sheet version 3
DB	Database
DES	Data Encryption Standard
DHT	Distributed Hash Table
DOM	Document Object Model
DTLS	Datagram Transport Layer Security
EEPROM	Electrically Erasable Programmable Read-Only Memory
ETSI	European Telecommunications Standards Institute
FTP	File Transfer Protocol
GUI	Graphical User Interface
GSM	Global System for Mobile Communication
GPRS	General Packet Radio Service
GPS	Global Positioning System
HTML5	HyperText Markup Language version 5

HTTP	Hypertext Transfer Protocol
ICE	Interactive Connectivity Establishment
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
iOS	formerly iPhone Operating System (before June 2010)
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
JSP	JavaServer Pages
LN	Local Node
LR-WPAN	Low-Rate Wireless Personal Network
M2M	Machine-to-Machine
M2MCE	M2M communication enabler
MAC	Medium Access Control
MCN	Monitoring and Controlling Node
MD5	Message-Digest Algorithm
MIB	Management Information Base
MIC	Message Integrity Check
MVC	Mode-View-Controller
NAT	Network Address Translation
NDK	Native Development Kit
NMS	Network Management Station
NWK	Network Layer
OHA	Open Handset Alliance
OID	Object ID
OS	Operating system
P2P	Peer-to-Peer
PDU	Protocol Data Unit
PN	Proxy Node
PNG	Portable Network Graphics
PHP	PHP: Hypertext Preprocessor
PHY	Physical Layer
RAM	Random Access Memory

RELOAD	REsource LOcation And Discovery
REST	Representational State Transfer
RISC	Reduced Instruction Set Computer
RMI	Remote Method Invocation
ROM	Read-Only Memory
RPC	Remote Procedure Call
SNMP	Simple Network Management Protocol
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SIP	Session Initiation Protocol
SKKE	Symmetric-Key Key Establishment
SMI	Structure of Management Information
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSL	Secure Sockets Layer
TC	Trust Center
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol / Internet Protocol
UDDI	Universal Discovery Description Integration
UDP	User Datagram Protocol
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WLAN	Wireless Local Area Network
VM	Virtual Machine
WN	Wide Area Node
WP	Windows Phone, Microsoft's mobile operating system
WPAN	Wireless Personal Network
WS	Workstation
WSDL	Web Services Description Language
WSN	Wireless Sensor Network
WWAN	Wide Wireless Area Network
XML	Extensible Markup Language

XML-RPC	XML Remote Procedure Call
ZDO	ZigBee Device Object

Used terminology

Android

Operating system for mobile devices and tablets developed by Open Handset Alliance (OHA). OHA is led by Google.

jQuery Mobile

Cross-platform and cross-device JavaScript web framework for tablets and smartphones.

Overlay

Peer-to-Peer network behavior where a virtualized network is formed over the physical network by the peer protocols. [1,3]

PhoneGap

PhoneGap is based on HTML5 and JavaScript and is used for bridging web applications to mobile devices' native features.

Prototype

JavaScript framework for developing dynamic web applications.

Robotium

Automated test framework for Android applications.

Sencha Touch

JavaScript framework for developing rich web applications for iOS and Android using HTML5.

WebKit

A web browser engine. WebKit is an open source project developed by multiple companies including Apple, Nokia, RIM, Samsung, Google and others [2].

ZigBee

Wireless technology which is based on the IEEE 802.15.4 specification and is designed for low-cost and low-power wireless sensor networks.

1 Introduction

The Internet of Things (IoT) and machine-to-machine (M2M) communication is becoming more and more everyday life each day. Different machines and sensors talk to each other using the Internet Protocol without having to have any human intervention in between. Measurements and acting according to different rules based on the measurement result is done automatically in an M2M world. Monitoring and controlling the measurements and actions is essential.

Mobile applications can be developed in many ways. As HTML5 is getting wider support from different browser vendors, it becomes a very good candidate for a mobile application platform. It gives the benefit of portability since it works in the same way regardless of the device platform. Native applications are restricted to their own platform and they are guaranteed to work smoothly on the device intended, as web applications can be slow from time to time depending on the browser capabilities.

This thesis is part of a machine-to-machine (M2M) project done in Ericsson Finland's research department called NomadicLab. The goal was to provide a user-friendly access to an existing wireless autonomous wide-area sensor network via a mobile device. The existing prototype has a command line interface shell for monitoring and controlling the nodes in the peer-to-peer network. The command line interface is not very good for promoting the prototype for possible commercial use in the future. The objective was to develop an Android application to fulfill the need for a mobile graphical user interface for controlling and monitoring the nodes in a wireless sensor network.

This thesis is divided into three parts. The first part describes the theory and background behind the existing prototype. The second part is about web and native application design, and the last part describes how the application was implemented and tested and what the plans for future development are.

2 Theoretical Background and the Existing Prototype

2.1 The Internet of Things (IoT)

The Internet of Things is counted as the next big possibility and challenge for the Internet engineering community, technology users and the society as a whole. IoT involves connecting embedded devices such as home appliances, sensors and even toys to network based on Internet Protocol (IP). Microcontrollers, batteries, low-power radios and microelectronic components have evolved into IP-enabled, smart embedded devices and their connectivity to services on the Internet have become a trend in the industry. The Internet services are connected to the physical world by including data from different sensors and control of different devices and machines, often named actuators. This latest area of the Internet that includes wireless low-power embedded devices is called the Wireless Embedded Internet. The cluster of European Research Projects on the Internet of Things (CERP-IoT) has defined the Internet of Things (IoT) as follows. [3,xvii.]

> Internet of Things (IoT) is an integrated part of Future Internet and could be defined as a dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual "things" have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network.
>
> In the IoT, "things" are expected to become active participants in business, information and social processes where they are enabled to interact and communicate among themselves and with the environment by exchanging data and information "sensed" about the environment, while reacting autonomously to the "real/physical world" events and influencing it by running processes that trigger actions and create services with or without direct human intervention. [4,6.]

There are many types of devices connected to the Internet already today ranging from mobile phones, home automation and personal health devices to environmental monitoring systems, smart metering and industrial automation. The Internet today is the Core Internet that includes backbone routers and servers and millions of network devices and the Fringe Internet. Figure 1 illustrates the vision of IoT and how the Internet is expanded from the core Internet. [3,3.]

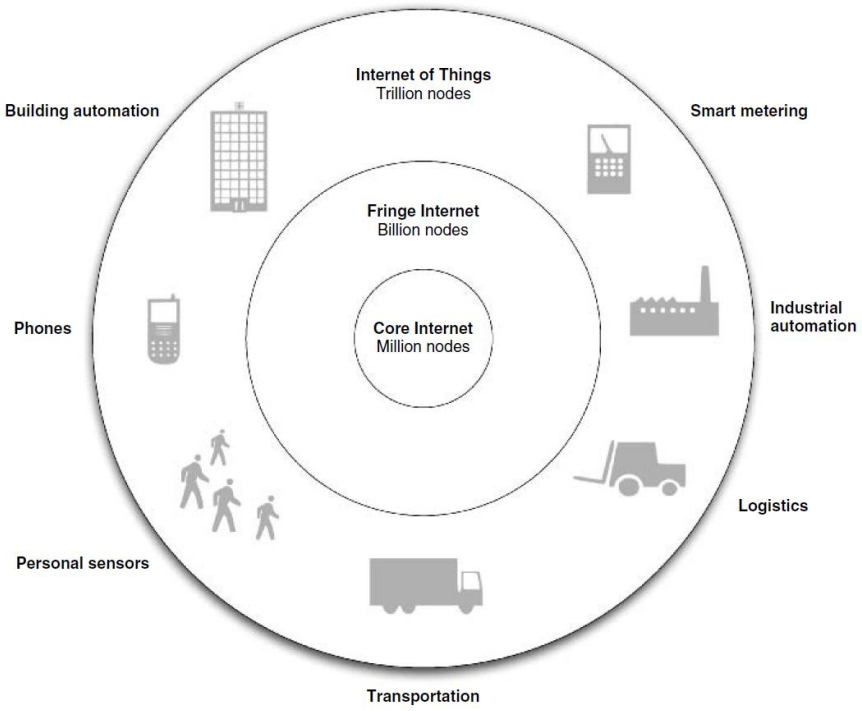


Figure 1. Vision of the Internet of Things. Copied from Shelby and Bormann (2009) [3,4]

As shown in figure 1, the Fringe Internet comprehends all personal computers, laptops and local network infrastructures including billions of devices. The scale of the Internet of Things is estimated with the potential of trillions of IP-enabled devices. When more and more devices can be accessed from the Internet, it makes for example smart homes, better logistics, better healthcare and better environmental monitoring possible. Also as automation is taken further, machine-to-machine (M2M) communication and possibilities are more utilized. The assumption is that the number of devices in the Internet of Things will soon exceed the number of computers in the current Internet and its size will keep increasing at a fast rate. [3,1-3.]

IPv6 over Low-power Wireless Area Networks (6LoWPAN)

The Internet Engineering Task Force (IETF) is the organization that sets some of the standards for the Internet. Among others, it sets the standards for Internet Protocol version 6 (IPv6) over low-power wireless area networks (6LoWPAN). This protocol plays a key role on the Wireless Embedded Internet. IPv6 requires more resources from embedded low-power devices that such devices are intended to provide. These requirements include complex security, web services such as Transmission Control

Protocol (TCP), Hypertext Transfer Protocol (HTTP) or Simple Object Access Protocol (SOAP), management via SNMP and frame size. 6LoWPAN was created to tackle these problems by simplifying the header structure of IPv6. Having a smaller frame size requires less bandwidth, and the hierarchical addressing model is ideal for wireless embedded devices. IETF's description of 6LoWPAN is below. [3,xvii,4-6.]

> 6LoWPAN standards enable the efficient use of IPv6 over low-power, low-rate wireless networks on simple embedded devices through an adaptation layer and the optimization of related protocols. [3,6.]

6LoWPAN will soon be introduced into to the ZigBee protocol, which is widely used in machine-to-machine communication. Applications where 6LoWPAN gives the most advantage are applications where embedded devices need to change information with Internet-based services, low-power networks need to be connected together and scalability is required throughout large network infrastructures with mobility. [5;3,9.]

2.2 Machine-to-Machine (M2M)

Machine-to-machine communication implies wired or wireless communication between different devices. These devices can be for example sensors or meters, which capture different events like rise of temperature or pressure or if the amount of a dangerous gas exceeds a certain limit. Based on that information other devices are notified over the Internet that an event has occurred. M2M means that no human intervention is needed nor should happen. All communication and actions are automated. Some traditional machine-to-machine systems include a cellular modem (M2M module) integrated to an embedded device and an Internet-based back-end system. The device is controlled and monitored by the M2M system and information is communicated to the back-end M2M system over IP. 6LoWPAN can be considered as an extension to machine-to-machine communication and due to native IP 6LoWPAN networks can be connected to M2M services via simple routers. [6;7;3,8.]

There are many large companies involved in M2M business (Vodafone, Ericsson, Tieto, AT&T, O_2, etc.) and the number of services is growing all the time. The European

Telecommunications Standards Institute (ETSI) is also working on a standard for cellular-based machine-to-machine communication. [8;9.]

Mobile broadband technologies such as HSPA and LTE provide many benefits to machine-to-machine communication such as low latency, high reliability and high bandwidth. By using mobile networks as the communication channel also sets new requirements on mobile operators. The amount of transferred data will be tenfold compared to current data rates. The data exchange can happen during the time when the network load is low and the data can first be collected from multiple sources and then be combined into a bigger package, which is then transferred to the M2M service. When more and more devices are connected to the Internet via mobile networks, operators need to be ready to provide high quality of service even though the number of devices will increase dramatically. [10,40.]

2.3 Protocols

2.3.1 Peer-to-Peer (P2P)

In peer-to-peer (P2P) networks' end devices (peers) are generally interconnected to each other without having a server in between. Peers are able to self-organize into different network topologies to share resources like files, audio or video for example. When the network is large and there are millions of users, a server-based network design has some weaknesses, among others the fact that when the servers are down, the data is not accessible. In a P2P network, data is shared among many peers and data is accessible even if some users leave the network. P2P protocols form a virtualized network over the physical network, most commonly the Internet, which is described as an overlay. Well-known P2P networks are those formed for file sharing applications like Napster, KaZaa and various BitTorrent clients. Another well-known application is Voice over P2P, for instance in Skype. Proof of the scalability of these systems is that Skype connects over 30 million users during peak hours at the same time. The properties of most P2P systems include resource sharing, networking, decentralization, symmetry, autonomy, self-organized, scalability and stability. [1,3,5-7;11.]

Peer-to-peer overlays are commonly separated into unstructured and structured overlay types. Beyond these two, there are also hierarchical, federated, service, sensor and semantic overlays and also overlays meant for mobile nodes in IP and ad hoc networks. An unstructured overlay is an overlay where a node relies only on its nearest neighbors when it sends messages to other nodes in the overlay. A structured overlay is an overlay where all the nodes keep the routing information in cooperation with each other on how to get to other nodes in the overlay. [1,10.]

In structured overlays each peer has a local routing table. When a peer joins the overlay, the routing table is initialized by using a specialized bootstrap procedure. Information on changes of the routing tables is periodically exchanged between the peers as part of overlay maintenance. A hierarchical overlay uses multiple nested overlays. Nested overlays are connected in a tree model. A message from peer to another peer in a different overlay goes through the nearest common parent overlay in the hierarchy. In a federated overlay, the overlay is constructed from a set of independent overlays where each has a separate administrative domain. All overlays are autonomous and messaging between different overlays requires peering arrangements. Service overlay definition is used when the overlay is used as a basis for multicasting, voice over IP or content delivery networks. The term "service overlay" is also referred to when service orientation is added to P2P overlays. [1,10-21.]

When an information relationship is stored in the overlay and a routing topology is formed according to semantic associations, the overlay is called semantic overlay. In sensor overlays elements of a grid or a sensor infrastructure are interconnected. The purpose of a sensor overlay is to keep physical layer constraints hidden from applications and separate physical layer routing from data collection. Overlays having mobile devices have four properties that have an effect on overlay communication which differ from desktop computers. These are roaming, multi-homed interfaces, node heterogeneity and energy limitations. Roaming causes change in IP address and in conventional overlays the result is a leave-join sequence because of re-binding the overlay address to the new IP address. Energy limitations also can out the device in stand-by mode and it causes the device to be disconnected from the overlay. A mobile ad hoc network is a wireless ad hoc network where mobile nodes act as routers and

hosts. A network infrastructure is not used when nodes route messages to other nodes. [1,10-21.]

Early large-scale versions of peer-to-peer networks that were designed for sharing files, music and other data between millions of users were based on unstructured overlays. In unstructured overlays nodes are organized into random data structures and therefore behave in an unpredictable fashion, unsuitable for voice and video sharing that require real-time interaction and unsuitable as well as for finding rare data in the overlay. Structured overlays have been developed to tackle both issues. They provide scalable network structures, which are based on a distributed data structure that supports deterministic data lookup behavior. Structured P2P overlays define rules where nodes can be placed in the overlay and gain benefits of improved data lookup efficiency. These rules are called algorithms. The algorithms are designed to assign each file to a node, so that they can be efficiently found by using directed search protocols. By using the algorithms a search can be executed with as little communication as possible, hence saving energy in low-power end nodes. Mostly the algorithms are based on Distributed Hash Tables (DHT), which use a unique key identifier for each file and node in the peer-to-peer network. [1,233-224;12,2.]

Distributed Hash Table (DHT)

A distributed hash table is a self-organizing, robust, scalable and efficient overlay routing infrastructure that can be used in P2P networks with millions of nodes. In DHTs data is mapped to keys that are *m*-bit identifiers taken from the identifier space. Each node that is part of the DHT has also an identifier which is picked from the same identifier space, and each node is responsible for storing a subset of keys in the identifier space. A node also stores a value that is typically paired with a key and can be for example the address of the node storing the data or the actual data. Normally, keys are taken from hashing meta-data and node IDs come from hashed public keys or IP addresses. [1,257-258,260.]

A DHT has a scheme (algorithm) that defines the overlay structure. The scheme also defines how routing between different nodes is handled and how the node state is maintained. Regardless of the scheme, a two-method interface for applications is

provided by a DHT: *insert(k, v)* for inserting key-value pairs into the DHT and *lookup(k)* for fetching the value for the given key. As DHT routing is identifier-based, $O(\log n)$, n being the number of nodes in the network, overlay neighbors are stored by each node in the overlay. To route queries from the requester to the node containing the target key, a deterministic algorithm is used for reaching the target in $O(\log n)$ overlay hops. [1,258.]

There are many different algorithms that can be used in a dynamic hash table. These include Chord, Pastry, Kademlia, Content Addressable Network (CAN), Tapestry and Viceroy. From the thesis point of view only Chord is relevant and will be elaborated more thoroughly. [1,260-265.]

Chord

The Chord algorithm puts nodes and keys into an identifier ring. Chord supports load balancing, which can be added as an extension to the basic scheme and is a simple and flexible DHT. The algorithm has four basic components and operations: key placement, successor links, fingers and key lookup. In the following as m-bit identifiers are used the identifier space is $[0,2^m-1]$, a node with ID i is denoted as N_i and a key with ID j as K_j. [1,260.]

Key placement means that a node N_i stores a key K_j and the node immediately follows k_j in the identifier ring. It means that such a node N_i is chosen where there is no node $N_{i'}$ where $K_j \leq N_{i'} < N_i$. N_i is also called the successor of K_j and is referenced as *successor*(K_j). Key and node IDs can be the same, meaning that the key (K_i) is stored at the node having the same ID (N_i). [1,260.]

In Chord successor links, each node N_i holds a link to its successor N_j which is the neighboring node. This kind of definition of key successors is also valid for nodes. Key's successor is guaranteed to be reached as long as successor links are correct even though the path might be long and requires going around the identifier ring node by node. This is achieved when nodes run periodically a *stabilize()* procedure. In the procedure, when N_j is a newly joined node, node N_i asks its successor N_s for N_s's predecessor N_j. N_i sets N_j as its successor if $N_j \neq N_i$. In assumption that $N_i < K_j < N_s$

and nodes N_i and N_s already exist in the DHT. When N_j joins the overlay the first time, it searches for K_j and receives N_s's address and after that sets it as its successor. As a final step N_s transfers keys in (N_i, K_j) to node N_j. [1,260.]

A finger table is also stored in node N_i. It contains m entries, where m is the length of the identifier. The address of $successor(N_i+2^{Kj-1})$ is maintained by the j-th finger. Figure 2 illustrates an example of Chord fingers with 6-bit identifiers. [1,260.]

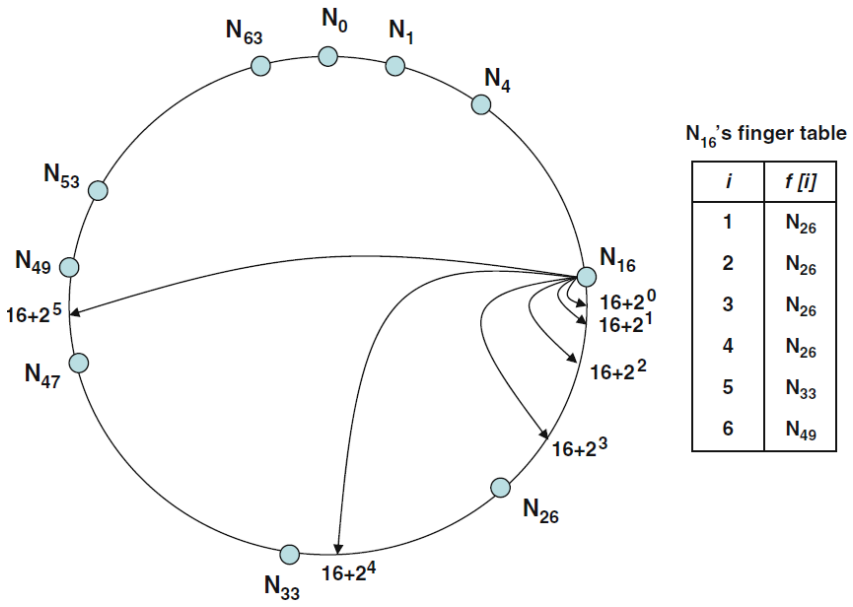


Figure 2. Chord fingers. Copied from Shen (2009) [1,261]

Each node executes periodically a *fix_fingers*() procedure to refresh the entries in the finger table. Each procedure call updates the next finger as the table is iterated in a round-robin way. Figure 2 shows how a node with an identifier of 16 forms its finger table. Finding the successor of the finger in turn completes the update of the finger table. [1,261.]

Key lookup is a simple lookup algorithm that sends a request forward, hop by hop, until the successor of the key is found by using successor links. The length of the lookup path is $O(n)$ hops. The lookup path is considerably shortened, to $O(\log n)$ hops, with the use of fingers. Figure 3 shows an example of how Chord routing is done with the help of fingers. Each lookup hop has a label of the entry in the finger table that is

used for sending the message forward. When the key resides between the successor and the current node, *succ* label is used. [1,260-261.]

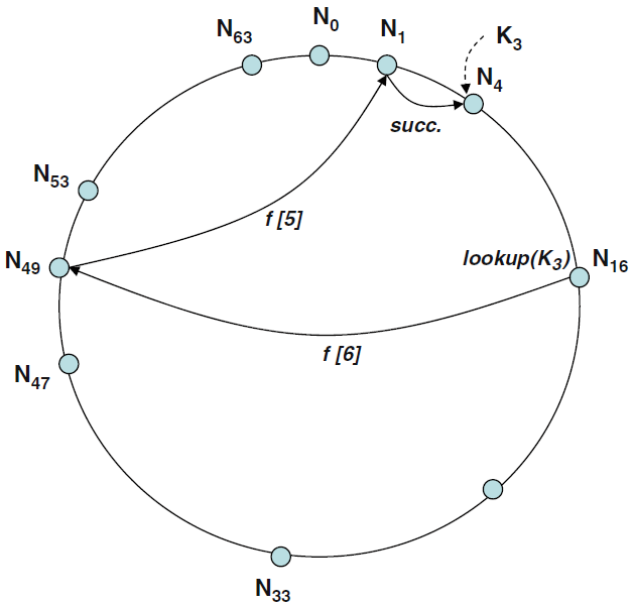


Figure 3. Chord routing with fingers. Copied from Shen (2009) [1,261]

Generally, to find out where key k is located, node i checks if k can be found from (i,*successor*(i)). If k is found, the request is just forwarded to the successor. If k is not in the successor, the request is forwarded to the largest finger N_i+2^{kj-1} that immediately precedes k. The procedure is repeated for each intermediate node until the desired key is found. With the use of fingers, the distance to the key's successor is approximately halved. [1,262.]

There are two main ideas to make Chord more robust, key replication and taking virtual nodes into use. Instead of having the key k only at the *successor*(k) the key is replicated on the r successors of k. The replication guarantees that when K_j=*successor*(k) is out, the *successor*(j) still answers to lookups for k. When r=O(log$n$) is set, it creates robustness against high levels of node dynamics. If the node has enough bandwidth and computing power, it can run multiple virtual nodes, and peers can contribute their resources to make the DHT's quality better. [1,262.]

REsource LOcation and Discovery (RELOAD)

REsource LOcation And Discovery (RELOAD) is a P2P signaling protocol to be used on the Internet. RELOAD provides a self-organized and a generic overlay network service. It allows peer-to-peer nodes to route messages efficiently between each other and to effectively save and fetch data in the overlay. The security framework, usage model, NAT traversal, high performance routing and pluggable overlay algorithms are critical features for an Internet P2P protocol that RELOAD provides. [13.]

To reduce malicious behavior and prevent attacks in a peer-to-peer network, where peers do not trust each other, RELOAD has a central enrollment server for providing credentials to every peer. These credentials are used for authenticating peers and the operations they perform. [13.]

RELOAD is designed to scale for a vast range of various types of applications such as P2P multimedia communications with the Session Initiation Protocol (SIP). It allows new application usages to be defined, each having their own data types. Rules for the use of these new data types can also be defined to RELOAD. With a simple documentation process that describes the details for each application, RELOAD can be used with a variety of new applications. [13.]

One design aspect of RELOAD is to take various network environments into account, as many of the nodes can be behind firewalls or Network Address Translators (NATs). RELOAD can use interactive connectivity establishment (ICE) to create new RELOAD or application protocol connections. This is one of the NAT traversal operations in RELOAD's base design. [13.]

RELOAD holds a simple and lightweight-forwarding header, which minimizes the need of peer resources in a P2P network. The requirement is set by the nature of overlay algorithms where peers route requests on behalf of other peers in a peer-to-peer network. These requests - data packets - and their handling increase the load and require bandwidth and processing power from the peers. High performance routing is a key feature in P2P communication and essential when peers are low-power devices. [13.]

Structured and unstructured overlay algorithms can easily be implemented with RELOAD since it provides an abstract interface towards the overlay layer. To be able to be used in most of the network topologies in P2P overlays, RELOAD provides a generic structure. RELOAD cannot be used by itself but needs to be combined with a specific overlay algorithm that suits the overlay requirements and provides the best efficiency for routing the messages. Implementation of the Chord algorithm is mandatory when using RELOAD. [13.]

RELOAD is designed to fulfill the requirements that SIP sets for a P2P protocol. However, RELOAD can be used with other various applications and is not restricted to be used only with applications using SIP. Along with peers, the RELOAD protocol also supports clients, which do not take part in routing or storing data. [13.]

2.3.2 Simple Network Management Protocol (SNMP)

The Simple Network Management Protocol (SNMP) was introduced in 1988 as a device management protocol for devices on IP networks. SNMP is an Internet-standard protocol and so far there have been three releases: SNMPv1, SNMPv2 and SNMPv3, where v stands for version. Many different types of network devices support SNMP including switches, routers, printers and computers. It provides a simple list of operations to remotely manage devices in the network. [14,ix,1.]

SNMP can be used to monitor and control a network device such as a router for example. It gives the ability to change the state of an interface (on and off), monitor the speed of an interface and send notifications if the temperature of the device is rising to an alarming level for example. Each device has its own properties that can be monitored and controlled. In addition to physical devices, SNMP can be used to monitor software such as databases and web servers as well. [14,1-2.]

There are two types of objects in SNMP: managers and agents. A manager is a server running the network management software and is often referred to as the Network Management Station (NMS). An agent is a piece of software running on the managed device. An NMS polls (sends queries for information) and receives traps from agents. A trap is sent asynchronously to an NMS and it is an indication from the agent to the

NMS that an event has occurred. Traps are not sent as a response to queries made by NMS. Communication between an NMS and an agent is shown in figure 4. [14,3.]

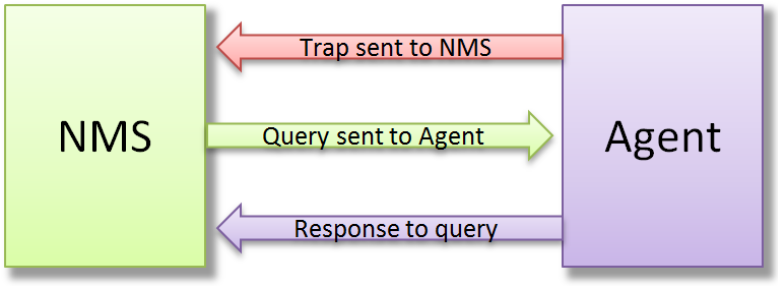


Figure 4. Communication between an NMS and an agent. Copied from Mauro (2005) [14,4]

Traps are usually indications that something unusual or something that should not happen has happened. A trap can also contain a "clear all" message, which is sent after the flawed state has been fixed. Traps and queries can occur at the same time and there are no restrictions on when messages can be sent between an agent and an NMS. [14,4.]

Managed objects and their behavior are defined via the Structure of Management Information (SMI). An agent can track managed objects and it contains a list of all available objects that have been defined by the SMI. The storage for the managed objects is called Management Information Base (MIB). MIB contains all statistical and status information that NMS can access. While the Structure of Management Information is used for creating the managed objects, the Management Information Base is the actual definition of the managed object. [14,4.]

Data between managers and agents is sent using the User Datagram Protocol (UDP) as transport protocol in SNMP. The reason for using UDP instead of TCP for messaging is UDP being connectionless. Connectionless means that packets are sent without having an end-to-end connection between the agent and the NMS. This also makes UDP unreliable since lost packets cannot be noticed at protocol level, but have to be checked in the SNMP application. A timeout is often used for determining lost packets and the number retries can be manually defined. UDP introduces a weakness in SNMP since packets can be lost and in case of traps the successful messaging is essential. Polls can be sent one after another if the reply for the query does not arrive. The

strength of UDP comes in low overhead. This reduces the impact on the performance of the network when more polls and traps are sent. TCP easily floods the network in the case of SNMP when the network is failing. [14,19-20.]

SNMP's management objects are ordered in a treelike model. An object ID (OID) is formed from a set of numbers (integers) separated by a dot. In the object tree the topmost node is called the root and nodes with children are called subtrees. If a node does not have a child, it is called a leaf node. An example of an object tree is shown in figure 5. [14,24.]

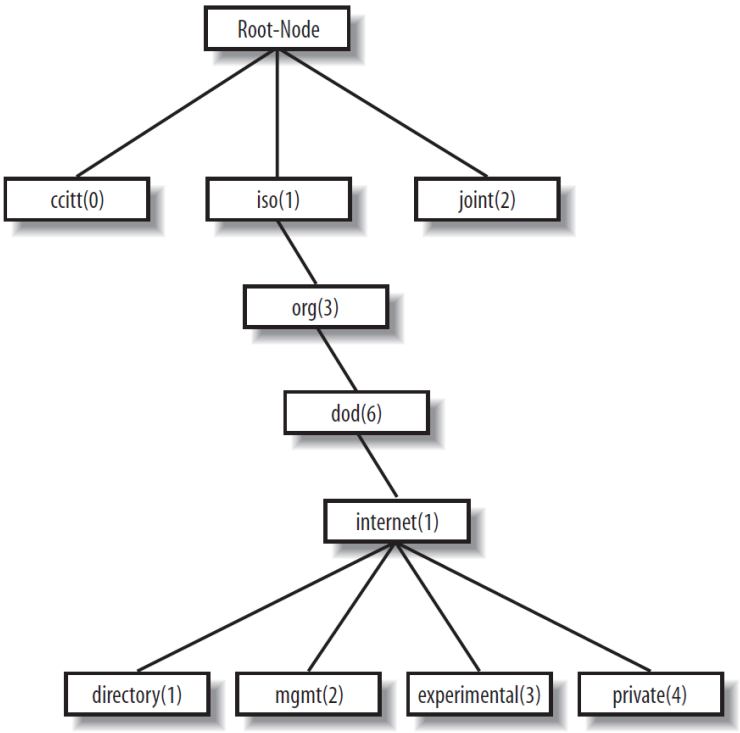


Figure 5. SMI tree model. Copied from Mauro (2005) [14,24]

The objects in a tree view can also be referred to by giving the path to the desired object in logical names separated with dots like in the numerical representation. In figure 5 the root is *Root-Node* and it has a subtree containing *ccitt(0)*, *iso(1)* and *joint(2)*. As *ccitt(0)* and *joint(2)* do not have any children but they are called leafs. To get the information from *mgmt(2)* managed object, the full path is either 1.3.6.1.2 or *iso.org.dod.internet.mgmt*. Managers and agents send and receive messages in Protocol Data Unit (PDU) format. Available SNMP operations are: get, getnext, getbulk, set, getresponse, trap, notification, inform and report. These operations can be used

for retrieving the required information from the target node and for sending information to the manager and the agent. [14,24-25,37.]

SNMP provided very little security until SNMP version 3 was released. Versions 1 and 2 had passwords but they were communicated in plain text, which made them very vulnerable if the network was being listened. In SNMPv3 the notions manager and agent are replaced with the name SNMP entity. Each entity has an SNMP engine and from one to many SNMP applications. SNMPv3 uses Message-Digest Algorithm (MD5) or Secure Hash Algorithm (SHA) for authentication encryption. SNMP messages are encrypted and decrypted with Data Encryption Standard (DES). [14,73-76.]

2.3.3 Constrained Application Protocol (CoAP)

Constrained Application Protocol (CoAP) is a specialized web transfer protocol that is used in M2M applications for constrained networks and nodes. CoAP is being designed by The Constrained RESTful Environments (CoRE) working group. Their aim is to modify the Representational State Transfer (REST) architecture into a more suitable form to support most of the constrained nodes (e.g. 8-bit microcontrollers with limited ROM and RAM memory) and networks (e.g. 6LoWPAN). IPv6 over low-power wireless area networks supports expensive fragmentation of IPv6 packets whereas CoAP is designed to keep the message overhead small and limit the fragmentation. HTTP's client/server interaction model resembles the one used in CoAP but in the CoAP implementation in the M2M network a machine can act as both, a client and a server. CoAP handles the message changes asynchronously over a datagram-oriented transport such as UDP, unlike HTTP. Abstract CoAP protocol layering is introduced in figure 6. [15.]

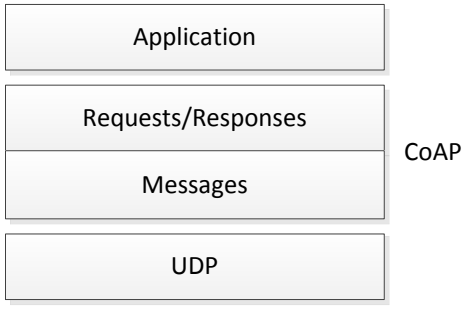


Figure 6. CoAP layering. Copied from IETF (2011) [15]

CoAP is a single protocol and messaging and request/response are just features of the CoAP header. The protocol is optimized for M2M applications and the main goal is to design a generic web protocol for constrained environment keeping in mind for example energy and building automation. [15.]

Messages in CoAP are communicated asynchronously between CoAP end-points. Messages are used to transfer requests and responses of the Constrained Application Protocol. The protocol relies on non-reliable transports such as UDP. This can make the messages go missing, arrive duplicate or arrive out of order. To overcome UDP's weakness, a lightweight reliability mechanism has been implemented in CoAP. It contains a simple stop-and-wait retransmission feature, it can detect duplicates and there is support for multicast. [15.]

There are four message types defined in CoAP: Confirmable, Non-Confirmable, Acknowledgement and Reset. A confirmable message is sent when an acknowledgement is required. A confirmable message must not be empty and it always carries a request or a response. A non-confirmable message is sent when an acknowledgement is not required. This message carries also a request or a response and must not be empty. When a specific confirmable message arrives, an acknowledgement message will be sent. Confirmable messages are identified by their Message ID. An acknowledgement message must echo the confirmable message's Message ID and it must either be empty or contain a response. When a confirmable or a non-confirmable message is received but for some reason the context to process it is missing, a reset message will be sent. The message must be empty and it must echo the Message ID of the confirmable or non-confirmable message. [15.]

2.3.4 ZigBee

ZigBee is a standards-based wireless technology that is designed for low-power, low-cost wireless sensor and control networks. It operates on the 2.4 GHz radio frequency and is based on the 802.15.4 specification made by the Institute of Electrical and Electronic Engineers (IEEE). The 802.15.4 specification is the Wireless Personal Network (WPAN) specification. Currently ZigBee supports over 64000 devices on a single network. [5;16.]

The ZigBee protocol stack consists of four main layers: Application (APL) layer, Network (NWK) layer, Medium Access Control (MAC) layer and Physical (PHY) layer. The protocol stack is introduced in figure 7. [17,2.]

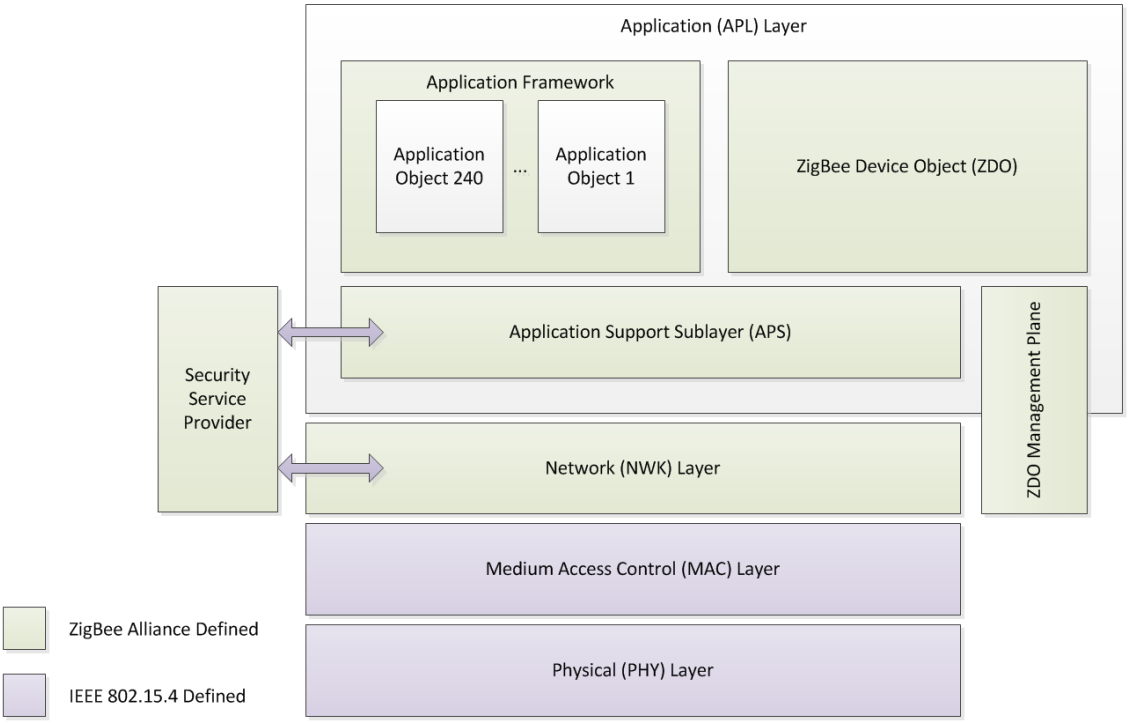


Figure 7. ZigBee protocol stack architecture. Modified from ZigBee Alliance (2012) [17,2]

The application layer consists of Application Framework, ZigBee Device Object (ZDO) with ZDO Management Plane, Application Support Sublayer (APS) and the manufacturer-defined application objects. APS provides an interface between the APL and the network layer. The interface has a general set of services that are used by the application objects and ZDO. The communication between the two layers goes through these services. Application objects are hosted in the application framework environment on ZigBee devices. As many as 240 different application objects can be defined in the framework (from 1 to 240). ZigBee protocol has also a number of reserved application object numbers: 0 for data interface towards ZDO, 241-254 for future use and 255 for broadcasting data to all application objects. ZigBee Device Objects bring configuration attributes into use. They are applications that take application support layer and network layer primitives into use to implement ZigBee End Devices, ZigBee Routers and ZigBee Coordinators. ZDO Management Plane manages the the communication between the Application Support Sublayer and the

Network layer with the ZigBee Device Object. It also allows the ZDO to handle requests for network access and security by using ZigBee Device Profile messages. [17,17-19, 213;18,9.]

The network layer is needed for providing a service interface towards the application layer and for providing functionality to guarantee correct operation of the IEEE 802.15.4-2003 MAC sub-layer. To communicate with the APL the network layer conceptually contains two service entities that provide the mandatory functionality. These services are the management service and the data service. [17,259.]

The Security Service Provider provides security mechanisms for the Application Support Sublayer and Network layer, which use encryption. The Security Service Provider is configured and initialized via ZigBee Device Object and is initialized by ZDO [17,18]. [18,9.]

The Medium Access Control layer provides reliable data transfer between a node and its nearest neighbors. The MAC layer also helps to improve efficiency and avoid collisions of the data packets. Assembling and decomposing of data packets is also a responsibility of the Medium Access Control layer. [18,9.]

The Physical layer includes two layers which operate in two different frequency ranges. The PHY layer also provides the interface towards the physical transmission medium, the radio for example. The lower frequency layer handles frequencies for Europe (868MHz) and for the US and Australia (915MHz). The second, higher, frequency layer (2.4GHz) is used virtually all over the world. [18,9.]

A large variety of products have been developed by hundreds of companies using ZigBee. These products include smart energy products such as In-Home Display, which shows the used electricity and how much it costs, home automation products such as Door Sensor, which logs and sends information if a door has been opened. All sensors and devices work wirelessly over a radio network using ZigBee. [5.]

2.4 Existing Prototype

The research department at LM Ericsson in Finland has designed and developed a wireless autonomous wide-area sensor network prototype. The prototype consists of a local node (LN), a proxy node (PN), a wide area node (WN) and a monitoring and controlling node (MCN). A machine-to-machine connection enabler (M2MCE) is an application which enables the communication between the nodes. Different nodes are introduced after the architecture in more detail. Architecture of the prototype system is introduced in figure 8. [19,26.]

The designed architecture consists of low-rate wireless personal area networks (LR-WPAN) which are formed from a set of sensors or local nodes. LR-WPANs cannot be directly interconnected to communicate with each other since they can be placed many kilometers apart. PNs, WNs and the MCN are connected via 3G and GSM to enable wireless communication and wide area deployment. Simple Network Management Protocol is used for monitoring and controlling the nodes. [19,26-27.]

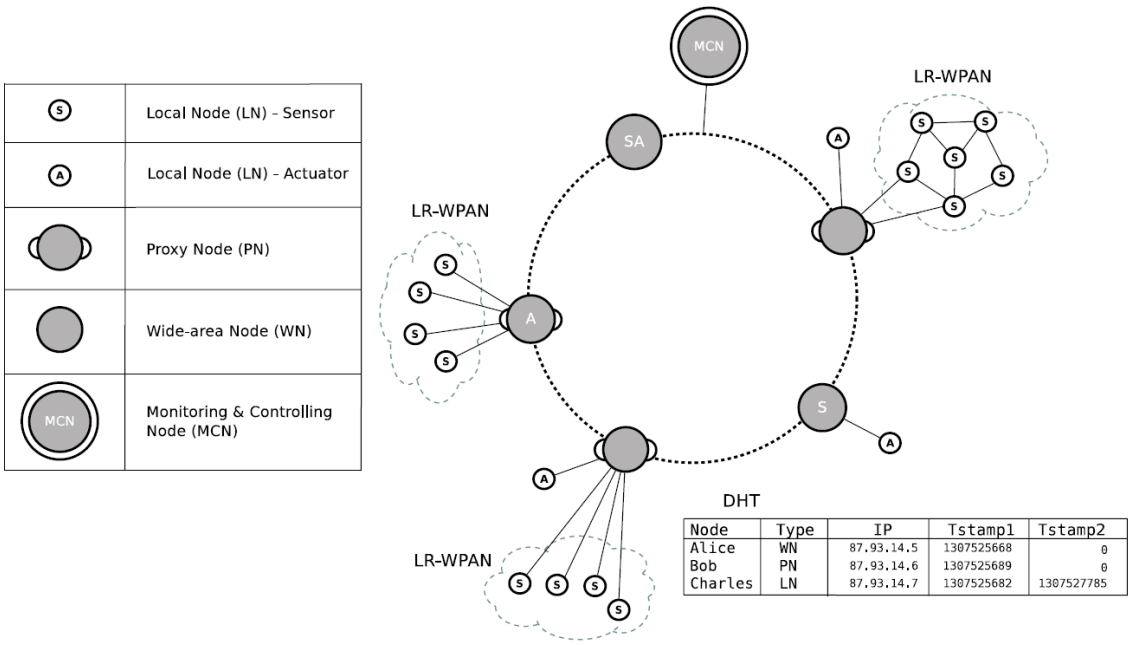


Figure 8. Overall architecture of the system. Copied from Jiménez (2012) [19,27]

A distributed hash table overlay is used to connect the WNs and PNs, so that robustness against failures, scalability and connectivity is guaranteed in the network.

DHT is responsible for storing the long term data of all the sensors. The data can be used afterwards to see what kind of trends for example a temperature sensor gives in a certain location. This has not yet been implemented to the DHT in the prototype. The DHT uses Chord algorithm as it is mandatory to be implemented with RELOAD. From the prototype point of view RELOAD has the main architectural principles that are needed. [19,27.]

The Constrained Application Protocol is used for addressing and retrieving resources in the overlay. CoAP Confirmable messages are used when acknowledgement is required between overlay nodes. PNs and WNs are provided with constant power and they are running DHT, so that they can run a CoAP server without any problems. There are two types of messages used between LNs and PNs: Periodic updates and Direct queries. Periodic updates are sent from LN towards PN when a pre-defined threshold value of a sensor is exceeded. Direct queries are sent to PN and are used for getting the latest value from the LN. Detailed information on how different nodes are connected together, how they communicate with each other and how they connect to the overlay can be found from Jiménez's thesis [19]. [19,27,32.]

Local Node

Local nodes are used for gathering information from their surrounding environment. The gathered data is shared with other nodes via the proxy node that acts as an Internet enabled device and a wireless personal network coordinator. The prototype's local node is shown in figure 9. [19,26.]

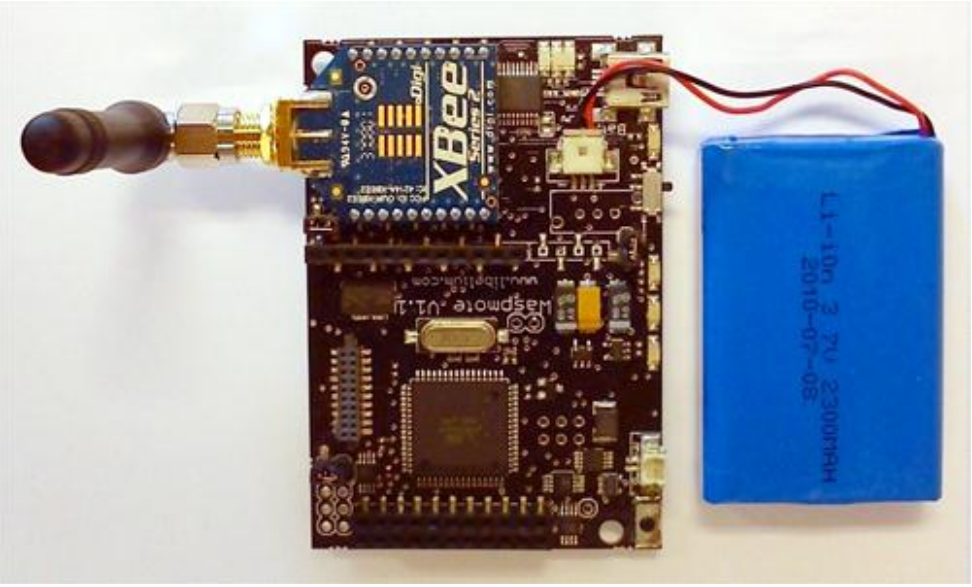


Figure 9. Wireless sensor platform Libelium Waspmote. Copied from Jiménez (2012) [19,50]

The Waspmote contains a ZigBee transceiver (Digi4 XBeeTM ZB5). A number of different extension boards for different applications such as gases, agriculture and event management are also available for Waspmote. The prototype uses the default version with a temperature sensor and an accelerometer. It has a 8MHz 8-bit RISC-based Atmel3 microcontroller, 8KB of RAM, 4KB of EEPROM and 128KB of flash memory. It also supports GPRS and Bluetooth. The power consumption for the Waspmote is 9mA while connected, 62µA on sleep mode and 0,7µA in hibernation. [19,50.]

Proxy Node

A proxy node has two purposes; it acts as an Internet enabled device and a WPAN coordinator. Communication between sensors and a PN is handled with ZigBee protocol. To connect different LR-WPANs together the use of a PN is required. All proxy nodes must be addressable and they need to be able to define associations between each other depending on the usage. The proxy node is introduced in figure 10. [19,26,31.]

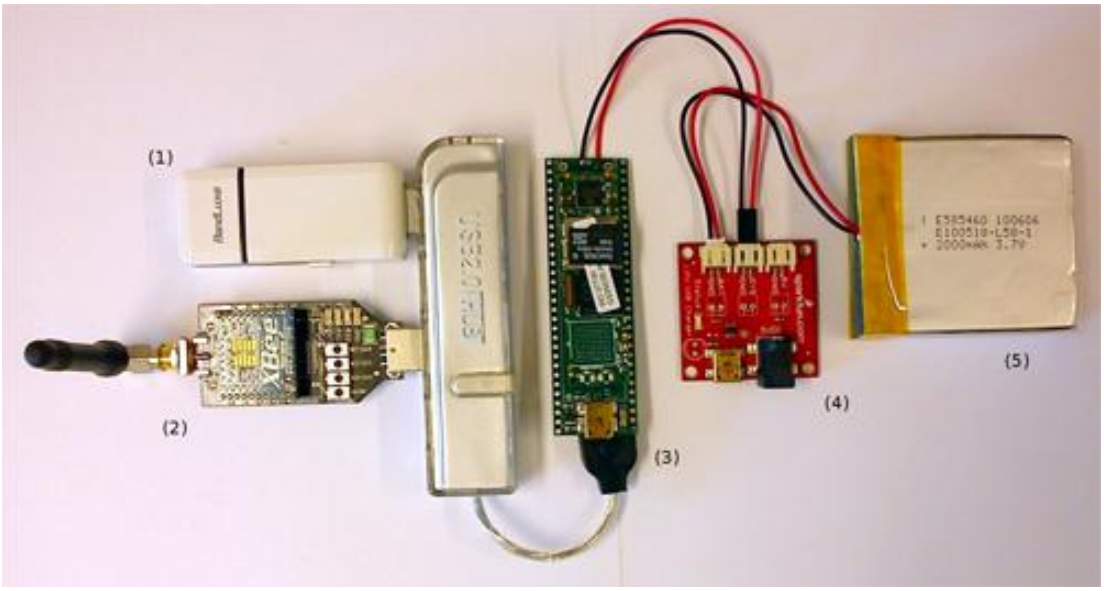


Figure 10. A Proxy Node with 3G USB Modem (1), XBee ZB transceiver (2), Pinto-TH board with Overo Earth module (3), LiPoly Charger (4) and batteries (5). Copied from Jiménez (2012) [19,53]

The proxy node is built on a Gumstix Overo Earth computer-on module with a ARM Cortex A8 CPU. The processor is a OMAP 3503 Application Processor from Texas Instruments and it runs on 600MHz. It has a 4GB micro SD card with a Linux running on it. The Overo Earth module is powered by a Pinto-TH expansion module which has a USB mini-AB port and 5V pins. The proxy node can be charged while powered by using a 6Ah Polymer Lithium Ion Battery and a Polymer Lithium Ion battery charger. Waspmote Gateway (XBee ZB transceiver) is used for communicating with the WPAN. [19,51-53.]

A sensor or an actuator module can also be attached to a proxy node. Proxy nodes are not as constrained as local nodes. Even if they are not so constrained, they consume much less power than for example a desktop or a laptop computer does. [19,26.]

Wide Area Node

Wide area nodes can be either sensors or actuators. WN is a stripped-down version of a proxy node by not having the transceiver for communicating with the WPAN (Waspmote Gateway, a XBee ZB transceiver). It is part of the DHT overlay but it does not behave like a proxy node nor is responsible for any LNs. [19,26,51-53.]

Monitoring and Controlling Node

A monitoring and controlling node monitors LNs and their resources, WNs and PNs. It occasionally connects to the overlay and plays as important role as a wide area node. MCN's main purpose is to create associations between different devices in the network to react depending on the situation. [19,26.]

M2M Communication Enabler

M2MCE is used for interconnecting all the different nodes in the overlay: local nodes, proxy nodes, wide area nodes, monitoring and controlling node and actuators. The M2M communication enabler bridges wide wireless area networks (WWAN), MCN monitoring and WPAN networking. M2MCE allows MCN and other nodes to reach the sensor information. [19,26.]

M2MCE provides a bookkeeping mechanism for storing information of nodes in the overlay and different parameters that are needed. It also gives nodes the permission to make independent decisions based on the information in the network. [19,29.]

2.5 Security

There are some assumptions and limitations made regarding security to the prototype: certificates are not distributed, self-signed certificates are used instead and it is assumed that MCN knows all the public key certificates of each node in the overlay and the public key of the MCN is known to all nodes. The security of the prototype is handled in two levels: WPAN and WWAN. [19,39.]

WPAN security covers the communication between the wireless sensor network (WSN) nodes. Trust Center (TC) manages the security in ZigBee. The WSN coordinator is the repository for security keys and so takes the role of ZigBee security manager. The TC decides which devices can join the network. The decision is forwarded from the proxy node to the MCN. There are three keys that ZigBee uses to manage security and they are Master, Network and Link key. Each node must have a shared key which is identical with other nodes in the network. Devices that communicate with each other

use link keys. When link keys are generated during the Symmetric-Key Key Establishment procedure (SKKE), master keys are used as an initial shared secret. When messages are sent, the payload is encrypted and Message Integrity Code (MIC) is added by integrity protection. MIC is a signature bound to the sender. [19,40.]

WWAN covers security in overlay messaging, on CoAP and SNMP in the application level and security between the DHT nodes (CoAP-IP mapping and bookkeeping security). Each node in the DHT has a certificate list of the nodes it has contacted. X.509 certificates, also specified in RELOAD, are used. Each WWAN node's CoAP name, NodeID in DHT and the overlay ID is used for computing the individual signature. Messages are sent by using the Datagram Transport Layer Security (DTLS), an encrypted transport protocol. Signatures are generated and verified by using standard public and private key cryptography. CoAP security is handled by using DTLS, and SNMP has its own user-based security model and view access control model for covering security. CoAP-IP mapping security is provided by the overlay. Objects stored in the overlay are digitally signed by the creating peer to prevent tampering. Bookkeeping security is achieved by using another overlay layer just for bookkeeping, so MCN can get overlay data with only one request. Node information list entries are protected to guarantee the security. [19,41.]

3 Web and Native Application Design

3.1 Web Services

A web service is any service that can be remotely accessed, usually on the Internet, and uses standardized Extensible Markup Language (XML) for messaging. Web services are not restricted by a single operating system or any particular programming language. Web services are made accessible by defining a Uniform Resource Locator (URL) for each. Applications using a web service send requests to its URL and get responses based on the requests. Web services can be accessed like normal web sites with a web browser but the benefit comes when another application is automatically requesting the information the web service provides. [20,6;21,10.]

Messaging can be done by using various methods. These include using SOAP, XML Remote Procedure Call (XML-RPC) and XML. The methods reside in the XML Messaging layer of the web service protocol stack, which is presented in figure 11. SOAP can be used between any platforms and applications. The main focus is on transporting Remote Procedure Calls (RPCs) via HTTP. SOAP message contains a header describing how the payload should be handled and the actual payload. Plain XML documents can also be sent by using SOAP. The XML-RPC protocol is used for sending RPCs in plain XML. XML-RPC messages are sent via HTTP POST. XML documents can also be used as message containers which are then processed by the application that sent the request. [21,12;20,15-16.]

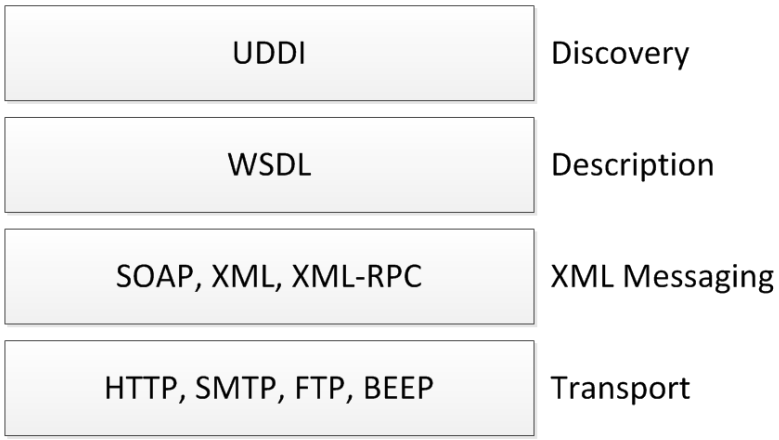


Figure 11. Web service protocol stack. Copied from Cerami (2002) [20,12]

Universal Discovery Description Integration (UDDI) is used to describe how a web service can be found and utilized. The Web Services Description Language (WSDL) is a specification of what method calls the web service responds to, what transport protocols are supported and from where a specific web service can be found. Transport layer protocols are responsible for sending the information from the source to the destination. Messages can be sent via HTTP, Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP) or Blocks Extensible Exchange Protocol (BEEP) that is built on top of TCP. Due to TCP, BEEP has features such as security, authentication and error handling built in. BEEP is a more efficient protocol than HTTP as it requires only 30 bytes of overhead per message whereas HTTP requires from 100 bytes to 300 bytes. Even though not so efficient, HTTP is still the preferred transport protocol. [21,13;20,17-20.]

REpresentational State Transfer (REST) and RESTful Web Services

The Representational State Transfer (REST) term comes from a PhD dissertation written and published by Roy Fielding in 2000. It is not architecture as such but is a set of constraints. When these rules are applied to a system's design, a software architectural style is created. When REST is applied to an application, it becomes RESTful. A RESTful system has to follow these rules: it has to be a client-server system, it has to be stateless, meaning that no sessions should be needed for clients, it has to support a caching system, it has to have a unique address, it must be scalable and code should be provided on demand. The constraints set by REST do not say which technology or methods should be used but defines how to transfer data between different parties. [22,7-8.]

The different components that make for example a web service RESTful are resources, representations, Uniform Resource Identifiers (URIs) and HTTP request types. In this case resources are single items such as temperature values at a certain time, detailed information of a proxy node or they can be lists of items such as a list of nodes in a DHT. Representation is the actual information sent between the client and the server. Depending on the client's needs, the representation can be for example a text file, an image a JSON stream or an XML stream. A web service has to provide different representations through the same URI. A URI is the address to the resource. The HTTP request types that are used are POST for creating new resources, GET for retrieving resources, PUT for updating resources and DELETE for deleting resources. [22,8-11.]

RESTful web services are applications residing on a local intranet or on the Internet. They are used for manipulating resource data. RESTful web services can perform server side functions and logic but they always have to return the resource data in a proper representation form. [22,12.]

Restlet Framework

The Restlet framework makes the creation of RESTful Java web services easy. At the time of this writing, the Restlet framework is in version 2.0.12. It can be used in both client and server applications. The Restlet framework supports the major standards

including JSON, XML, SMTP and HTTP. Development environments such as Java Standard Edition, Java Enterprise Edition and Android are also supported in Restlet. [22,126;23.]

The framework is an extension to Java Servlet. The Restlet web service needs to have *web.xml* file where the Restlet service is configured. URIs to the different services are defined in the main Restlet application class which forwards the requests towards business objects and other resources. The Restlet framework supports all HTTP message types (GET, POST, PUT and DELETE) which are essential to the RESTful web services. [22,126.]

3.2 Application Design

3.2.1 Web Application and Frameworks

3.2.1.1 HTML5 Application

Applications done with HTML5 can be web pages that use the HTML5 markup to display the page. Server side languages such as PHP: hypertext preprocessor (PHP), active server pages (ASP) and JavaServer pages (JSP) can be used as long as the output markup is in HTML5.

By using CSS3 web pages can be defined to dynamically change the look and feel when the browser width changes. This allows the same page to be viewed with multiple resolutions without having to create a separate page for mobile devices for example as they have smaller screen resolution compared to a laptop or a desktop computer. Good examples are *http://m.yle.fi* where the "m" stands for mobile and *http://www.bbc.co.uk/mobile/index.html* which is as well a separate page meant for mobile devices only. The need for these separate pages fades as CSS3 provides support for media queries where the device's screen resolution can be checked and different styles can be applied based on that information.

HTML5 provides new features such as local storage, indexedDB, file handling, offline mode, web workers and web sockets. As HTML5 is still a working draft, all new features are not supported in every browser and it is possible that the features are changed before HTML5 is standardized [24]. These are explained in more detail below.

Taking different JavaScript libraries into use on the web application can enrich the user experience to a new level. Four JavaScript frameworks (prototype, jQuery Mobile, Sencha Touch and PhoneGap) that are widely used are introduced more thoroughly in chapters 3.2.1.2-3.2.1.5.

Local Storage

There are two separate storages defined under local storage in HTML5: localStorage and sessionStorage. The difference between these two is that localStorage is persistent even though the browser is restarted and sessionStorage lives only the browser session. Different browser vendors have defined the local storage file to be at most 5MB in size and it is stored per web site to the user's local hard disk. If there are more browsers in use, each browser has its own local storage, so there is no cross-browser support for a single local storage. [25,48.]

Data can be added, modified and deleted from the storage object. The information is stored in key-value pairs. Values are stored as objects and all data types are type cast to strings, so the data cannot be saved separately as integers or dates as data can be stored in a relational database. Local storage can be very helpful for example in games when the game state and high scores need to be saved. [25,49.]

IndexedDB

IndexedDB is a NoSQL database and follows the same policy as the local storage; it is accessible only from the page it was created at. IndexedDB is supported only in two browsers at the moment: Chrome (from version 11) and Firefox (from version 4). It is an alternative and a better solution for local storage when more structured data storage is required and it resides on the local hard disk as local storage. The native format for data storage is JavaScript object and it does not need to be mapped into a

SQL table structure. The database is browser-specific like local storage and cannot be accessed but from the originating browser, meaning that using the application from many devices and browsers with one data storage still requires server side data storage. [25,59-60.]

Files

Earlier it has been possible only to upload files to the server via HTML forms. With HTML5 comes a form file input which allows JavaScript to access the file data directly. It is also possible to drag and drop files from the user's computer to the web page and upload files in that way. Google uses this method in Gmail when enclosing attachments to an email. [25,67.]

When uploading a file via web form it provides a FileList object that consists of *File* objects. The file objects contain information of the file's name, MIME type, size and Last Modified Date. JavaScript cannot access the full path to the file. The file can be asynchronously fully read with a FileReader object as URL, text, binary string or as an array buffer. [25,69.]

Newer versions of XMLHttpRequest interface allow files to be uploaded to the server by using the FormData interface. XMLHttpRequest interface has two events that are very useful for telling the user about the status of the upload: onprogress and oncomplete. These events can be monitored, and when the file size and current progress is known, the current upload status can be dynamically shown to the user. [25,70.]

Offline Mode

Sometimes when using a web application on a mobile device, the network coverage disappears and with that the connection to the Internet. When the Internet connection is down, normally the web application is as well. HTML5 introduces a manifest file where a list of files can be defined to be downloaded to the device's local disk each time the file changes. After the files have been downloaded, they will be used from the local disk instead of the network. [25,75-76.]

In the manifest file there are three groups under which the files can be listed: CACHE MANIFEST, NETWORK and FALLBACK. All files that need to be loaded locally are listed straight under the CACHE MANIFEST header. Files which should always be accessed from the network are listed under the NETWORK header. The FALLBACK provides the opportunity to define different files to be loaded depending on the network connection. The first file is loaded when the connection is up and the second one when the connection is down. These files are listed on the same line. An example of a manifest file is shown in listing 1. [25,77.]

```
CACHE MANIFEST
# Comment line
/index.html
/js/script.js
/css/style.css
/img/image.png

NETWORK:
/sendMail.php

FALLBACK:
/contacts.html      /offline_contacts.html
```

Listing 1. Example of an HTML5 manifest file.

The cache manifest file is referred to in the <html> tag of the document. The manifest file must use a mime type of "text/cache-manifest" in order for browsers to recognize it, and this must be defined on the web server if it has not been set by default. [25,76.]

Web Workers

JavaScript runs on a single thread and has a queue of all the events that have occurred in the browser. JavaScript takes events from the event queue and runs them in an event loop. Depending on the browser's JavaScript engine, computation speed of the device and the number of events that need to be processed, the response time of the web page can vary. The event handling process is shown in figure 12. [25,85.]

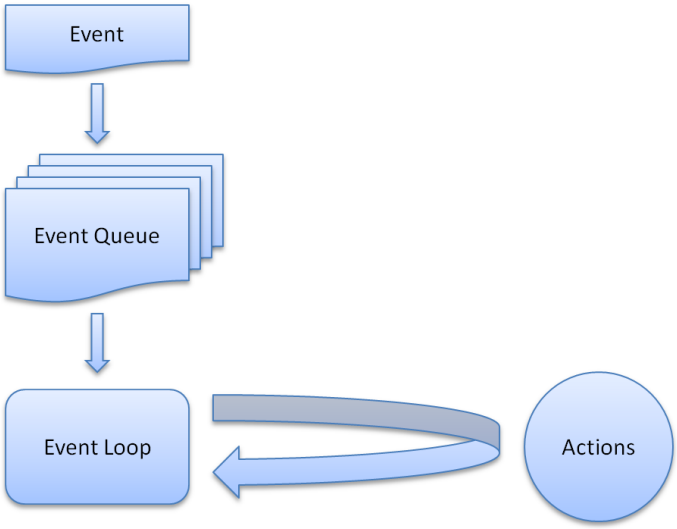


Figure 12. JavaScript event handling. Modified from Kessin (2012) [25,86]

JavaScript events are processed from the event queue when JavaScript runtime is idle. The process works well when the events can be processed quickly and the actions are small enough. If the events require more computation and there are several events in the queue, the user might see the page lagging behind the actions that have been requested. This makes the web page feel sticky and the user experience drops down fast. [25,85.]

With HTML5 comes web workers, separate JavaScript processes that can communicate with each other and with the main process. A web worker can do different kinds of computations and send messages to and receive messages from other workers and the main process. Each web worker has its own event queue and event handling process. If a web worker events take longer to finish, it does not affect the main process and the user does not feel the page getting unresponsive, unlike in the case when the main process has a time consuming event ongoing. [25,86-87.]

There are some restrictions when using web workers. They cannot access the document object model (DOM). Also interfaces such as document object and window object are inaccessible from the web workers. At the moment Internet Explorer and Safari on iOS do not support web workers. [25,88.]

Web Sockets

Previously the hypertext transfer protocol has been used for sending and receiving files (for which it was designed for) and for some extent for changing real time or semi-real time information between a server and a client. This can be done with HTTP but is a cumbersome way to handle it. HTML5 comes to aid with web sockets. Web sockets resemble TCP/IP sockets in the way they work as the web socket can be opened from the browser to the server and kept open as long as needed. The socket can then be explicitly closed when unnecessary. [25,101.]

A socket is a real time data channel that supports bi-directional data transmission whereas HTTP is just a polling protocol for simple requests. With sockets there is no need for sending HTTP headers which reduce the amount of data to be transferred on each message. This becomes an essential feature when many requests and messages are sent in a small period of time and as little data as possible should be sent. [25,102.]

3.2.1.2 Prototype Framework

Prototype is a JavaScript framework that provides easy Document Object Model (DOM) manipulation and broad Ajax support. As of this writing, prototype is in version 1.7.0.0 which was released on November 16th, 2010 and has not been updated since. The framework includes only one JavaScript file that has all the classes and functions. Prototype is used for creating dynamic web pages viewed on desktops. [26.]

The framework does not provide any GUI styles or enhancements such as jQuery Mobile and Sencha Touch does. It provides support for class-style object-oriented programming with inheritance and wider support for event management. Prototype is just for creating dynamic content to web pages with JavaScript. The prototype is meant to be the base library for creating plugins and rich web content libraries and frameworks. Default Android browsers having WebKit engine support Prototype well. [27,11;28,21.]

A test application written for Android with a WebView for showing the web content shows that when only the Prototype JavaScript file is loaded, an HTML5 Canvas animation runs much more smoothly (bigger frame rate) than if jQuery Mobile framework was used.

3.2.1.3 jQuery Mobile Framework

jQuery Mobile is a JavaScript framework based on the jQuery library that is widely used in desktop web development. jQuery is like the prototype framework but has more functionalities and features in it. There is also a broad range of plugins available from different developers for jQuery that extend the basic framework. [29.]

At the time of writing this thesis jQuery Mobile is in version 1.1.0. jQuery Mobile is supported by all big platforms: iOS, Android, Blackberry, Windows Phone, MeeGo, Kindle. Supported desktop browsers are Chrome, Firefox, Internet Explorer and Opera. The framework consists of different jQuery plugins and widgets and it aims to provide a cross-platform API for mobile web application development. [29;30,1.]

Usage of jQuery Mobile requires the jQuery library to be loaded first since jQuery Mobile is an extension to jQuery. It consists of a JavaScript file, a CSS file and some PNG files which contain the default icons. jQuery Mobile provides five different default color themes and new themes can be created on the jQuery web page. The framework has its own styles for buttons, title bars (header and footer), toolbars, navigation bars and form components. It also supports touch-based actions such as swipes but does not support multi-touch actions like pinching. [29.]

3.2.1.4 Sencha Touch Mobile Framework

Sencha Touch is a JavaScript framework for building mobile apps with HTML5 for Android, iOS and BlackBerry. The current version of Sencha Touch is 2.0 and it was released on March 6th, 2012. The framework is much like the jQuery Mobile framework providing rich UI controls, theming, model-view-controller (MVC) support and over 300

built-in icons. There is also an API for creating different types of charts included in the Sencha Touch framework. [31.]

Sencha SDK includes a native packager which allows building of native Android and iOS applications from the Sencha Touch application. The SDK works on Windows and Mac so developing iPhone/iPad apps does not require a Mac. Sencha Touch framework also has native device APIs which allows using and monitoring camera, orientation, native confirmation dialogs and network connectivity. [31.]

Ajax and JavaScript Object Notation (JSON) is also supported in Sencha Touch, making calls to server side and other web services very easy and fast. There is support for full DOM manipulation and a wrapper for geolocation. Geolocation support provides easy utilization of Google Maps. [31.]

3.2.1.5 PhoneGap

PhoneGap is a framework that can be used as a platform to other mobile web application frameworks such as jQuery Mobile and Sencha Touch. PhoneGap provides access to native features such as camera, accelerometer, compass and files. Supported features for different OS vendors are listed in figure 13. [32.]

	iOS iPhone / iPhone 3G	iOS iPhone 3GS and newer	Android	OS 4.6-4.7	OS 5.x	OS 6.0+	WebOS	WP7	Symbian	Bada
ACCELEROMETER	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
CAMERA	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
COMPASS	✗	✓	✓	✗	✗	✗	✗	✓	✗	✓
CONTACTS	✓	✓	✓	✗	✓	✓	✗	✓	✓	✓
FILE	✓	✓	✓	✗	✓	✓	✗	✓	✗	✗
GEOLOCATION	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MEDIA	✓	✓	✓	✗	✗	✗	✗	✓	✗	✗
NETWORK	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NOTIFICATION (ALERT)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NOTIFICATION (SOUND)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NOTIFICATION (VIBRATION)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
STORAGE	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗

Figure 13. PhoneGap feature support. Copied from Adobe (2012) [32]

PhoneGap is supported on seven platforms: iOS, Android, BlackBerry, HP WebOS, Windows Phone 7 (WP7), Symbian and Bada. Only iPhone 3GS and newer, Android and WP7, support all native features that PhoneGap offers. The framework is based on HTML5 and JavaScript. Current version of PhoneGap is 1.5 and it was released on March 6th, 2012. [32.]

The usage of the PhoneGap framework requires the application to be written and built in native environment. To be able to use the device's features through PhoneGap, the framework needs to be imported to the native application. For example in Android the import definition is "*import com.phonegap.*;*" and the application needs to extend *DroidGap* instead of *Activity*. After importing, PhoneGap framework can be utilized. [32.]

3.2.2 Native Application

Native applications are applications which have been developed to be used on a certain operating system, platform or a device. "Native app" term is often used when talking

about mobile apps since they have been created to work on a particular device platform. Android applications are written mostly in Java, iOS applications in Objective-C and Windows Phone applications in C#. The application related to this thesis is developed for Android. Information related to the Android operating system and application development for Android is briefly introduced below. [33.]

Google Android

Android is a free, open source operating system for mobile handsets and tablets developed by Open Handset Alliance (OHA), which was founded by Google. The operating system is built on Linux kernel version 2.6 and is released under the Apache License that is an open source license. [34;35,35.]

The first mobile phone that used Android as an operating system (OS), HTC Dream, was released in October 2008 and used Android version 1.0. After one year in the market there were only about 20 different Android handsets available. Based on a report by Gartner, made in February 2010, the sales by the end of year 2009 meant a 3.9% market share for Android while Symbian was leading with 46.9%. In the end of quarter 3, 2011 (Gartner's report from November 2011) there were over 60 million Android devices sold to end users and Android OS held a market share of 52.5%. The next competitor was Symbian with a market share of 16.9% and iOS held the third place with a 15.0% market share. [36,9;37.]

The Android OS supports a large variety of features including 2D and 3D graphics, multitasking, various animated transitions in applications, multi-touch input and support for many audio, video and image formats. It uses a WebKit engine-based browser that has a vast support for hypertext markup language version 5 (HTML5) and cascading style sheets version 3 (CSS3). Android users can also replace any native, pre-installed application with a third-party application, for example the dialer, the messaging application and home screen. [35,35;36,3.]

Applications developed for Android can utilize the device's hardware (camera, GPS, accelerometer, etc.), Google Maps and geocoding with location-based services and background services. This means that applications do not necessarily have a visible

user interface but instead run in the background and for example send different kind of notifications based on different actions. Android applications can also use a built-in SQLite database for storing custom data. Android provides the means for different applications to interact with each other and use other applications' resources if they are shared. An example would be a database. Widgets and different live background images and folders are also supported in Android. [36,6-8.]

Currently there are three Android OS versions which are the most used (Google's Android Market report from January 2012): 2.1 (Éclair), 2.2 (Froyo) and 2.3.3-2.3.7 (Gingerbread). Android as a mobile phone OS has grown from version 1.0 to 2.3.7. Version 3.x.x is used in tablets only. Android 4.0 (Ice Cream Sandwich) was released on October 19th, 2011 and is again used in all devices, mobile phones and tablets. [38.]

Application Development in Android

Android applications can be developed using the Android software development kit (SDK) and native development kit (NDK) provided by Google. Android tools also integrate into Eclipse IDE with Android Development Tools (ADT) plugin providing a rich development environment. [39.]

NDK provides tools for creating applications and libraries by using C or C++ and is rarely used. The SDK contains tools for creating Java applications, the most common way to develop applications for Android. Later in this thesis when the "native application" term is used, a Java application is referred to. [39.]

Many different types of applications can be developed for Android: native, widgets, services, web apps, combinations of native and web apps and so on. This thesis concentrates on native and web applications and how applicable each approach is for this specific application. [39.]

3.2.3 Combining HTML5 GUI with Native GUI

It is possible to create mobile apps by combining native and web app features. Native apps can contain a web view for showing web content which can be of any size. The GUI can contain native elements such as buttons, text fields, labels, tabs etc. alongside with a web view. In Android it is possible to call Java methods through JavaScript and return data from Java methods to a JavaScript function. An example would be a JavaScript function calling a Java method which shows a native toast on the screen. Another example can be a JavaScript function calling a Java method which returns a String value to another JavaScript function and the value is dynamically inserted into a text field inside the web view. Native apps can be created by having only a full screen web view as the GUI. The content can be loaded from the Internet or by using local files.

There are various possibilities for combining and using native and web content together but all the elements that the web GUI provides can most likely be done with native way as well, so the benefit of combining the two is quite small, if non-existing. Also, depending on the device and browser, web content might not run as smoothly as native content. The combined application also loses its portability since there are parts written in the native language.

3.2.4 Testing Tools

3.2.4.1 Robotium Framework

Robotium framework is an Android framework for automated testing. As of this writing the framework is in version 3.1. It provides the means for easy black box (no source code available) function, system and acceptance test case scenarios. Activities, Dialogs, Menus, Context Menus and Toasts are fully supported by Robotium. Robotium framework is used as JUnit test for Android applications. [40.]

Robotium test cases can be written for an application based on the text, text boxes, buttons, etc. displayed on the screen when the application is running. With the

framework different UI elements can be clicked and touched, text can be inserted into text fields, screen can be scrolled, orientation can be changed and screenshots from any test case point can be taken. To be able to run tests on an application, the application source code does not need to be known. It is also possible to run Robotium tests on a pre-installed application but it requires rooting of the device. The framework works with Maven and Ant and can easily be utilized to be part of continuous integration. [40.]

3.2.4.2 Testdroid Cloud

Testdroid Cloud is a service provided by a company called bitbar and is in beta state at the moment. The cloud service offers the possibility to run an Android application on 81 different Android-based devices. Supported devices are manufactured by HTC, Samsung, Sony Ericsson, ZTE, LG, Huawei, Motorola and Amazon and the OS version ranges from 1.5 to 4.0.3. [41.]

The Android application package file (APK) is uploaded into the cloud, optionally with an existing Robotium test APK. Testdoid Cloud runs the application with the test project on all available devices and shows different types of results: graphs, logs, screenshots and so on. Testdroid Cloud can also fetch screenshots taken within the Robotium tests and show them for each device. Screenshots are stored in the device's SD card and a package containing all the screenshots from all devices can be downloaded from the Testdroid Cloud web page. [41.]

The Testdroid Cloud service is Secure Sockets Layer (SSL) encrypted. All the data that is uploaded to the devices can only be seen by the user who has done the uploading. Test results are for the owner's eyes only as well. All the devices are real physical devices from where all test-related data is erased after each test. Each device is rebooted before the test run. Testdroid Cloud also provides a VPN connection to the Testdroid hosting center if a company would need one. [41.]

4 Application Implementation

4.1 Development and Design

The application was developed by using a simulated environment. The idea in the beginning was to connect to the P2P overlay via Java Remote Method Invocation (RMI) which was already in place in the MCN CLI application. Due to the fact that Java RMI is not supported in Android and could not be used as such, a new solution was needed. To overcome the impediment, a web service approach was taken. The development environment is introduced in figure 14.

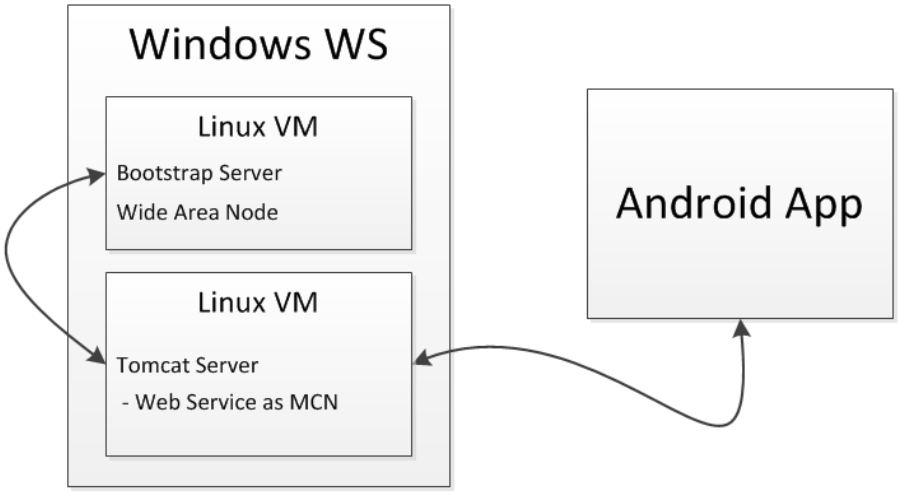


Figure 14. Development environment.

The development was done in a Windows workstation (WS) where two Linux virtual machines (VM) were running. The first VM was running the bootstrap server and a simulated WN. Apache Tomcat web server was running on the second virtual machine. The web service, which is described in more detail later, was running on the Tomcat server. Virtual machines had bridged network adapters originating from the Windows host, so that communication between the machines worked. Android app was run on a real device connected to the same network via a wireless local area network (WLAN).

The prototype environment is not using any virtual machines and the bootstrap server is running on a Linux WS with Apache Tomcat. The Android app is connected to the

DHT overlay via the web service running on Apache Tomcat. The prototype environment is shown in figure 15.

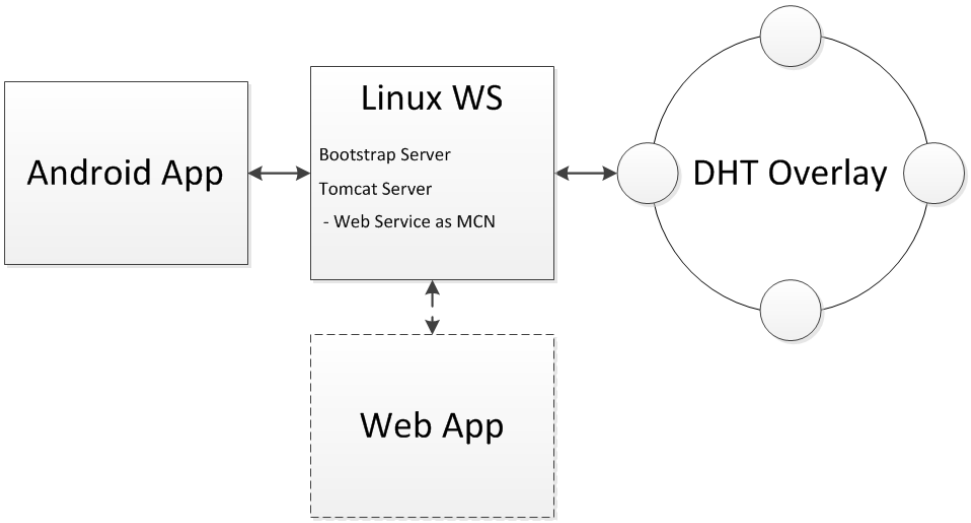


Figure 15. Prototype environment.

Using the web service enables the possibility to create a web application which could also be used to monitor and control nodes in the DHT overlay. The web application was not part of the thesis and was not implemented and is so marked with dashed lines in the figure.

Activities

The application runs on devices having Android 2.1 or later. It starts up to a tab view where the first tab contains a list of all the nodes found in the DHT overlay. Each row in the list contains an icon describing the node type, node name and node location. Figure 16 below shows the node listing.

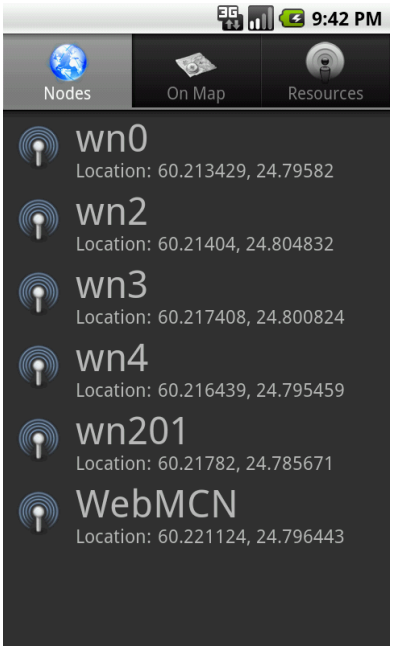
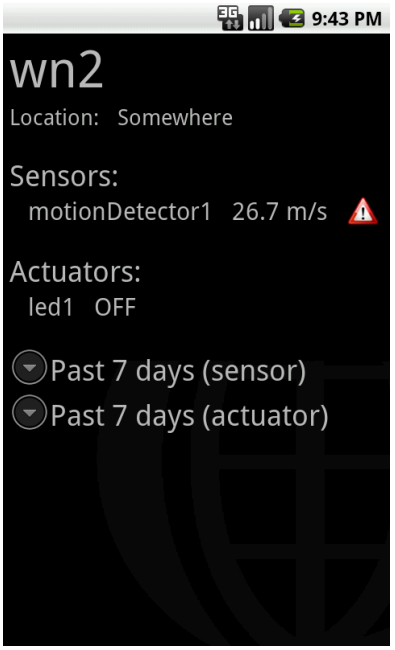


Figure 16. Node listing.

Each row can be clicked or touched to view more detailed information on the node. Detailed information view is its own activity showing what sensors and actuators the node contains and what the current state of the sensors and actuators is. It also shows a chart from the past seven days displaying the highest, lowest and average values. The values are drawn to a line chart that is based on AChartEngine library. The chart type (Avg, Min, Max) can be changed by tapping the corresponding radio button. The sensors can be given certain threshold values (low, high) to indicate when the connected actuator should react. A detailed view is shown in figure 17.a and 17.b.

a) 

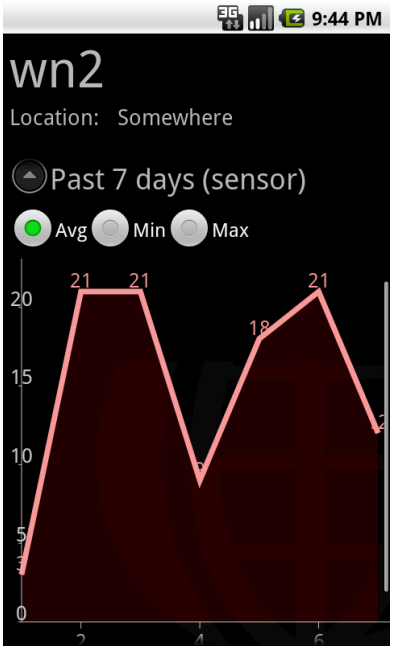
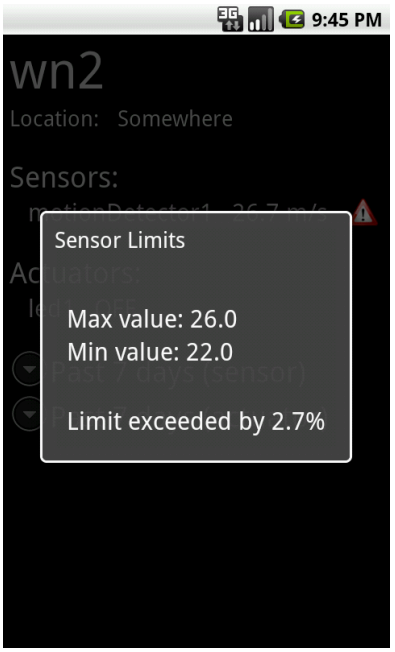
b) 

Figure 17. a) Detailed information. b) Detailed information showing the sensor graph.

If the threshold value is exceeded, a notification image will be shown next to the current value. By clicking on the image, the current limits are shown in a dialog. The thresholds can be changed by clicking on the "Set Limits" menu item in the menu. The sensor limits and setting them is presented in figure 18.a and 18.b.

a) 

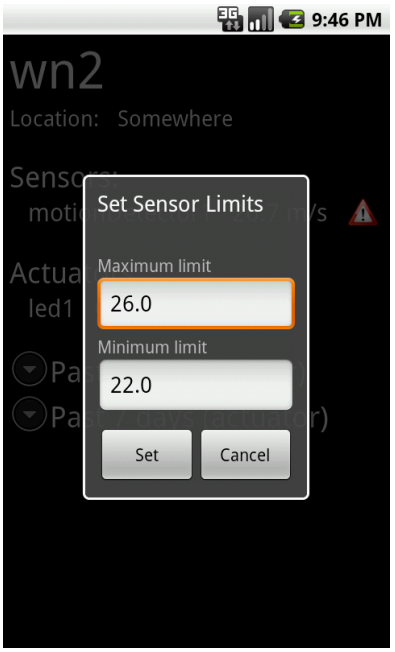
b) 

Figure 18. a) Sensor limits dialog. b) Setting the sensor limits.

Figure 18.a illustrates what the currently set thresholds for this sensor are and how many percentages the current value has exceeded or is below the set thresholds. The dialog for setting the thresholds is shown in figure 18.b. When the maximum and minimum values are left empty, the limits are removed.

The second tab in the main view displays the nodes on a map based on their Global Positioning System (GPS) coordinates. Each node can be clicked or touched to view its detailed information. The map tab activity is shown in figure 19.a. Google Maps API is used in the map activity to be able to display images on top of the map and do actions when a point on the map is touched or clicked.

a) 

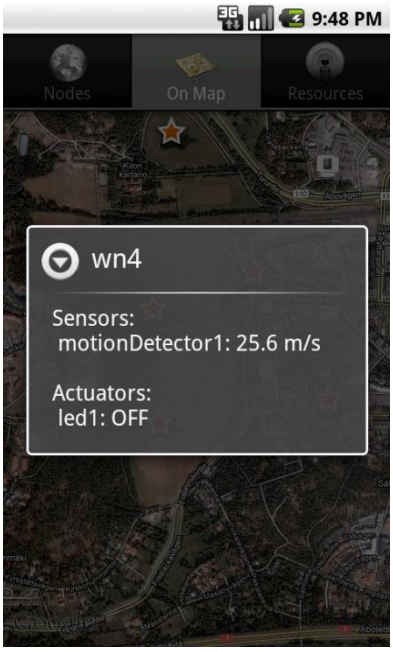
b) 

Figure 19. a) Nodes indicated by stars on the map. b) Current node information.

Detailed node information is shown in a dialog. It contains the name of the node, what sensors it holds and their current values and the actuators with their current values. In figure 19.b the name of the node is "wn4" and it has a motion detector sensor and a led as an actuator.

Resources (sensors and actuators) are listed in the Resources tab shown in figure 20. In the resource list all sensors are listed first followed by the actuators. Each row shows the sensor or the actuator name, the host to whom it belongs and the current value. If the limits are set and the current value is in between the limits or there are no

limits set, the value is shown in green. If the current value is past the thresholds, it is shown in red.

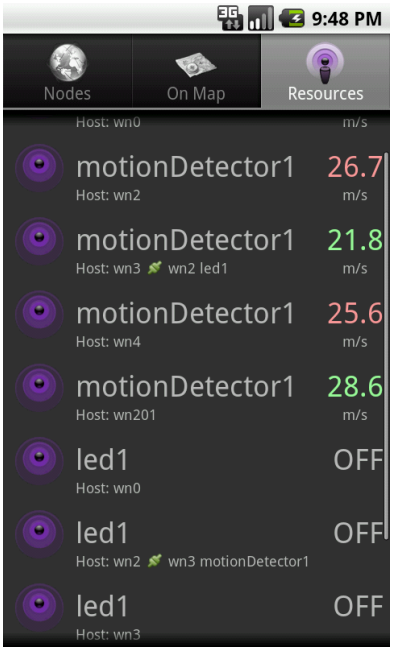


Figure 20. Resources tab.

If the associations have been defined, a small connection icon, the host name and the resource of the associated node is displayed next to the host information. In the current prototype only one association can exist at a time. Associations can be reset (removed) and set (added) by making a long press on a resource. This brings up a context menu with actions for adding and removing associations. The context menu is shown in figure 21.a.

a) b)

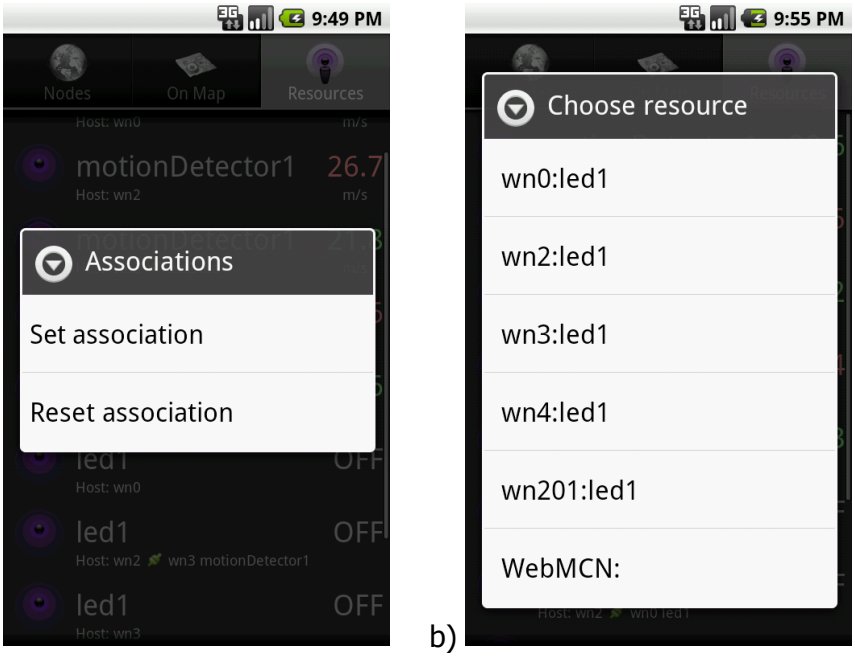


Figure 21. a) Context menu for setting and resetting associations. b) List of resources for setting the association with.

When "Set association" is selected from the context menu, a new list is shown. Figure 21.b illustrates a list of resources to which an association can be made from the selected resource. If the source resource is a sensor, a list of actuators is shown and if the source resource is an actuator, a sensor list is shown. Associations can be made to a resource residing in the same node or in another node, meaning that for example Node1 sensor can be associated with Node2 actuator.

4.2 Accessing Sensor and Actuator Data

Currently the command line interface (CLI) application for MCN uses Java Remote Method Invocation (RMI) to get connected to the overlay and for fetching and sending data. At the moment Android does not support Java RMI and due to this a new solution was needed to access the DHT overlay information. The new solution was the introduction of a web service.

Web Service

Apache Tomcat was chosen to act as the Java web server where the Java-based web service runs. It is easily configured and does not need any tricks to get it up and running. To be able to connect to the overlay, M2MCE needs to be started on the web server. This enables the use of the RMI methods and communications with the DHT.

The web service was created to handle requests from the Android app towards the DHT overlay, to act as a monitoring and controlling node. Requests towards the web service are sent by using HTTP. Web service supports HTTP GET and HTTP POST messages and returns the requested information in either XML or JSON format depending on the HTTP message's Accept header. The Accept header can be *text/xml* or *text/json*. Communication between the Android app and DHT via the web service is shown in figure 22 below.

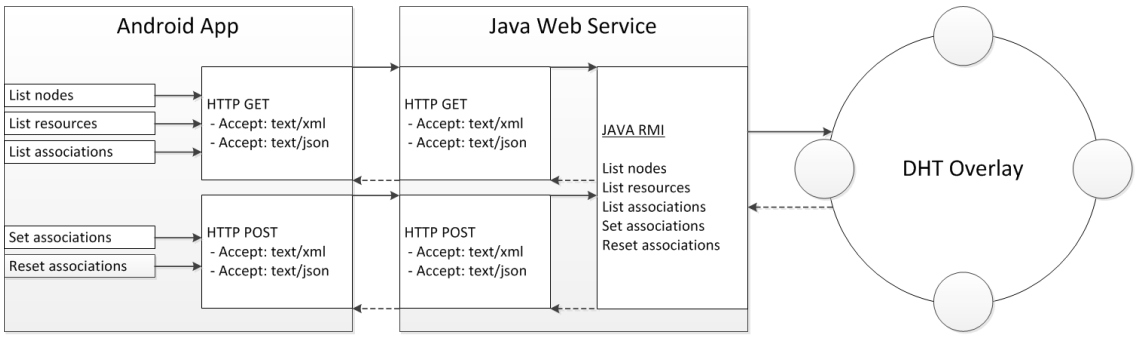


Figure 22. Communication with the DHT overlay via the web service.

As shown in figure 22, requests are sent towards the web service when a list of nodes, a list of resources (sensors and actuators) or a list of associations between sensors and actuators is needed and when setting and resetting associations between a sensor and an actuator. When a request arrives to the web service, an RMI client is created to communicate with the DHT overlay. The RMI client is the same as the existing MCN CLI shell with slight modifications. MCN CLI shell prints all the requested information to the screen and the information cannot be used like that. Instead of printing the information, it is put into variables and returned back to the method which created the RMI client. This way the information can be properly parsed into XML and JSON. When the parsing is done, the requested data is returned from the web service to the

Android app (example of an XML response is shown in listing 2) and is parsed again to be able to use the information within the mobile application.

```
<nodes>
<count>6</count>
<node>
<name>wn2</name>
<type>PN</type>
<location>Somewhere</location>
<sensorname>motionDetector1</sensorname>
<sensorvalue>21.6</sensorvalue>
<sensorunit>m/s</sensorunit>
<actuatorname>led1</actuatorname>
<actuatorvalue>OFF</actuatorvalue>
<sensorhistoryvaluesavg>
          6.0,27.0,18.0,12.0,9.0,21.0,18.0
</sensorhistoryvaluesavg>
<actuatorhistoryvaluesavg>
          0.0,1.0,0.0,0.0,0.0,0.0,1.0
</actuatorhistoryvaluesavg>
<limits>true</limits>
<limitmax>26.0</limitmax>
<limitmin>22.0</limitmin>
<latitude>60.21404</latitude>
<longitude>24.804832</longitude>
<association>
          wn3,motionDetector1,wn2,led1
</association>
<link>/nodes/wn2</link>
</node>
<node>...</node>
</nodes>
```

Listing 2. Example of returned XML request.

The previous example is the response of the *list nodes* request sent to the web service. The XML contains the number of the nodes found, the node's name, type and location,

sensors and their values and the unit for sensor value, actuators and their values, sensor's and actuator's history data for the past seven days and the maximum and minimum limits. The nodes can have a GPS sensor, and for that the latitude and longitude values are needed. Resource associations are also listed in the XML. The XML also contains an element with a direct link to this node's data.

4.3 Testing

Android JUnit test cases were written by using Robotium framework. Robotium test cases were run in Testdroid Cloud to guarantee that the application worked and the look and feel was the same in all devices. Due to the fact that the web server is not in a public network, Testdroid Cloud could be used only when all the data was generated on the application side.

Testing was done in a simulated environment with two Linux virtual machines emulating the nodes and in a real prototype environment with real nodes. In both environments Apache Tomcat was running on a Linux and it was connected to the DHT overlay via the web service. Real devices, HTC Desire and Sony Ericsson Xperia Ray, were used when the application was tested while under development.

5 Discussion

5.1 Portability

As the application was done natively for Android, it is not easily ported to other platforms as such. Porting would require a new platform-independent design including a platform-related look and feel. Introduction of the web service has a great effect on the portability and diversity of the application. Due to the web service, applications can be written in any language and to any platform and they still can access the DHT.

Applications can be web applications running in browsers, desktop applications, mobile HTML5 applications or native mobile applications. In web applications and HTML5 mobile applications Ajax functions can be used for requesting the information from the

web service and for updating the UI dynamically according to the received information. The web service brings the application development possibilities to a whole new level.

5.2 HTML5 App versus Native App

Plain HTML5 web applications have their benefits when the applications are not complex. Web applications can reside on a web server anywhere and can be accessed from any device that has a modern enough browser. The benefit is also in distribution; the application does not have to be put to any marketplace for people to use the app. On the other hand there are no checks either and reviews done by anyone who does the evaluation when distributing the application through a marketplace. This can result in malicious apps that can harm the end user's device and get access to personal information such as contacts and emails.

When the application is done with HTML5 and it resides on the Internet, it also means that it is usable by any modern browsers used in laptops and desktops. Also when using CSS3 it is possible to seamlessly change the layout of the application depending on the screen width and height. This makes the application runnable on any device without having to make any changes to the code based on the type or vendor of the hardware and there is no need to install anything. HTML5 also makes the application run in the same way on all the devices. Releasing an update to the application is also fast and efficient since only the web pages need to be updated and they are available to everyone on the second the pages are updated, whereas in native applications the update procedure goes through the marketplace and a new installation is required.

Depending on the application it is not always clear which approach would be better, native or HTML5. It depends on the complexity of the application, what resources are needed from the device and how portable the application should eventually be. There could be a case where a customer has ordered an application for just one platform due to certain agreements made with the platform owner or for all platforms where full portability would be ideal.

Usability

Native GUI has its benefits like HTML5 GUI does. Native applications can easily use all the features the device offers: sensors, camera and so on. When using HTML5 separate frameworks and libraries are needed for accessing the device features and still all of them cannot be accessed.

From the application provider point of view it is usually so that whatever the platform of the device is, the application should look the same. Companies want to keep their brand and style visible in the application throughout the different platforms. This is very easy to implement using HTML5 since all devices show the GUI in the same way if the browsers support all the features. From the user point of view this might be an issue since the user could want all the applications look the way the platform looks like, meaning that iPhone applications should look like iPhone applications and Android applications should look like Android applications. This is easily done when using the native GUI elements for developing the application. The benefit of an HTML5 application is its portability. Native applications are faster and have full access to all device functionalities. Current HTML5, CSS3 and SVG support in browsers can be seen from table 1.

Table 1. HTML5, CSS3 and SVG support in desktop and mobile browsers. Copied from Deveria (2012) [42].

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Opera Mobile	Android Browser
								10.0: 41%	2.1: 39%
	6.0: 8%	3.6: 53%				3.2: 43%		11.0: 59%	2.2: 44%
	7.0: 12%	9.0: 81%				4.0-4.1: 50%		11.1: 65%	2.3: 46%
	8.0: 22%	10.0: 82%	17.0: 87%	5.0: 65%		4.2-4.3: 57%		11.5: 66%	3.0: 62%
Current	9.0: 48%	11.0: 82%	18.0: 87%	5.1: 74%	11.6: 70%	5.0: 73%	5.0-6.0: 27%	12.0: 74%	4.0: 67%
Near future	10.0: 77%	12.0: 82%	19.0: 88%	5.2: 76%	12.0: 75%				
Farther future		13.0: 82%	20.0: 89%						

Performance

Having created a simple example using HTML5 canvas and testing it with two different phones and a tablet (SonyEricsson's Xpreria Ray, HTC Desire and Samsung Galaxy Tab 10.1) it is clear that HTML5 performance becomes an issue compared to native application. Simple drag and drop operations, page and activity transitions are also

slower and not so smooth in an older phone, which has a great effect on usability and user experience.

5.3 Plans for the Future

As the current environment holds only the Android app, the web service and the P2P network, it is required that a database is introduced to the network. The database can be used for storing the history data for sensors and actuators along with other information. The nodes in the P2P network will be pushing the sensor and actuator data towards the web service. The database can be easily accessed from the web service and the data can be presented in XML or JSON like the other data currently is.

Another plan will be making the application portable by using web technologies. As the web service can be accessed from anywhere, it is convenient to create an HTML5 application, since it can be used with any modern device and the same application can be used via desktop browsers as well.

In the future the MCN should also be updated to support different actions as resources. This will enable creating associations with sensors and actions, meaning that when for example pressure in a reactor rises over a pre-defined limit, a number of control valves are opened to prevent explosion. This would make the MCN more event-driven. Also connections to social networks could be utilized for different kinds of status updates.

6 Conclusions

The future Internet will have a massive variety of different kind of devices connected to it. Already today many different types of devices, such as home appliances, can be connected to the Internet providing information measured and monitored by the device. IP-enabled devices are more and more interconnected into M2M networks where a monitoring device can request an action from another device to normalize an abnormal state. In a large scale automated machine-to-machine network monitoring and controlling the machines is essential.

The purpose of the thesis was to create an Android application providing a graphical user interface for monitoring and controlling nodes in a wireless autonomous wide-area sensor network. The application was intended to give common users a suitable way to monitor and control nodes, sensors and actuators instead of a clumsy command line interface.

Several conflicts occurred during the application design and development. In the beginning the plan was to create a mobile web application with HTML5, which would have been a portable solution. The plan needed to be re-thought since the MCN works with Java RMI. MCN could not be used from a web application implemented with HTML5 since it would have required server-side programming to be used. The solution was to go on with a native application. When MCN connection implementation was in turn, it turned out that Java RMI was not supported in Android. To get around this problem a Java web service was introduced as the solution to act as an interface towards the P2P network. The web service could be reached with the Android application and the web service could connect to the P2P network with Java RMI.

This application will be used with the existing prototype. It will give users a more user-friendly interface for monitoring and controlling nodes in the peer-to-peer sensor network. The application can be used for viewing past sensor and actuator data and to see what the current values of the sensors and actuators are. Users can also see with one look if any of the sensor values are out of the permitted limits.

References

1. Shen, Xuemin. Handbook of Peer-to-Peer Networking. New York: Springer; 2010.

2. Apple, Nokia, RIM, Google, Others. The WebKit Open Source Project. [online] URL: http://www.webkit.org/. Accessed 8 January 2012.

3. Shelby, Zach and Bormann, Carsten. 6LoWPAN: The Wireless Embedded Internet. West Sussex: John Wiley and Sons, Ltd; 2009.

4. Saint-Exupery, Antoine de. Internet of Things - Strategic Research Roadmap. s.l.: Cluster of European Research Projects on the Internet of Things; 2009.

5. ZigBee Alliance. ZigBee Technology. [online] URL: http://www.zigbee.org. Accessed 2 October 2011.

6. Numerex. M2M Communications. [online] URL: http://www.numerex.com/. Accessed 3 March 2012.

7. TechTarget. What is M2M (machine-to-machine)? [online] URL: http://searchnetworking.techtarget.com/definition/M2M. Accessed 3 March 2012.

8. Wireless, Jasper. M2M. [online] URL: http://m2m.com/. Accessed 3 March 2012.

9. ETSI. ETSI M2M. [online] URL: http://www.etsi.org/Website/Technologies/M2M.aspx. Accessed 4 March 2012.

10. M2M Alliance. M2M Journal. 2011;9.

11. Skype. Skype. [online] URL: http://www.skype.com. Accessed 7 March 2012.

12. Seet, Boon-Chong. Mobile peer-to-peer computing for next generation distributed environments: advancing conceptual and algorithmic applications. Hershey: IGI Global; 2009.

13. IETF. REsource LOcation And Discovery (RELOAD) Base Protocol - draft-ietf-p2psip-base-20. [online] URL http://tools.ietf.org/html/draft-ietf-p2psip-base-20. Accessed 11 March 2012.

14. Mauro, Douglas and Schmidt, Kevin. Essential SNMP, 2nd Edition. Sebastopol: O'Reilly; 2005.

15. IETF. Constrained Application Protocol (CoAP) - draft-ietf-core-coap-08. [online] URL: https://datatracker.ietf.org/doc/draft-ietf-core-coap/?include_text=1. Accessed 31 December 2011.

16. Digi International, Inc. ZigBee Wireless Standard. [online]
URL: http://www.digi.com/technology/rf-articles/wireless-zigbee. Accessed 16 February 2012.

17. ZigBee Alliance, Inc. ZigBee Specification. San Ramon: ZigBee Alliance, Inc; 2007.

18. Daintree Networks. Getting Started with ZigBee and IEEE 802.15.4. Mountain View: Daintree Networks; 2008.

19. Jimenéz, Jaime. Master's thesis: Adapting a DHT (Distributed Hash Table) to a Self-Reliant M2M (Machine-to-Machine) Network. Helsinki: Aalto University; 2011.

20. Cerami, Ethan. Web Service Essentials. Sebastopol, California: O'Reilly; 2002.

21. Potts, Stephen and Kopack, Mike. Sams Teach Yourself Web Services in 24 Hours. Indianapolis: Sams Publishing; 2003.

22. Sandoval, Jose. RESTful Java Web Services. Birmingham: Packt Publishing; 2009.

23. Restlet SAS. Restlet. [online]
URL: http://www.restlet.org/. Accessed 15 April 2012.

24. W3C. HTML5. [online]
URL: http://dev.w3.org/html5/spec/Overview.html. Accessed 11 February 2012.

25. Kessin, Zachary. Programming HTML5 Applications. Sebastopol: O'Reilly Media, Inc.; 2011.

26. Prototype Core Team. Prototype JavaScript framework. [online]
URL: http://www.prototypejs.org/. Accessed 20 November 2011.

27. Dupont, Andrew. Practical Prototype and script.aculo.us. New York: Apress; 2008.

28. Porteneuve, Christophe. Prototype and script.aculo.us. Dallas: The Pragmatic Programmers LLC.; 2007.

29. The jQuery Project. jQuery Mobile. [online]
URL: http://jquerymobile.com/. Accessed 20 November 2011.

30. Reid, Jon. jQuery Mobile. Sebastopol, California: O'Reilly Media, Inc.; 2011.

31. Sencha. HTML5 Framework for Desktop and Mobile Devices. [online]
URL: http://www.sencha.com/. Accessed 14 March 2012.

32. Adobe Systems Inc. PhoneGap. [online]
URL: http://www.phonegap.com. Accessed 31 December 31 2011.

33. TechTarget. What is native application. [online]
URL: http://searchsoftwarequality.techtarget.com/definition/native-application-native-app. Accessed 18 March 2012.

34. Google. Android. [online]
URL: http://www.android.com. Accessed 20 November 2011.

35. Allen, Sarah, Graupera, Vidal and Lundrigan, Lee. Pro Smartphone Cross-Platform Development: iPhone, Blackberry, Windows Mobile and Android Development and Distribution. New York: Apress; 2010.

36. Meier, Reto. Professional Android 2 Application Development. Indianapolis: Wiley Publishing, Inc.; 2010.

37. Pettey, Christy. Gartner Says Sales of Mobile Devices Grew 5.6 Percent in Third Quarter of 2011. [online]
URL: http://www.gartner.com/it/page.jsp?id=1848514. Accessed 8 January 2012.

38. Google. Android Platform Versions. [online]
URL: http://developer.android.com/resources/dashboard/platform-versions.html. Accessed 8 January 2012.

39. Google. Android Developers. [online]
URL: http://developer.android.com. Accessed 16 March 2012.

40. Reda, Renas. Robotium. [online]
URL: http://code.google.com/p/robotium/. Accessed 31 March 2012.

41. bitbar. Testdroid cloud. [online]
URL: http://beta.testdroid.com. Accessed 31 March 2012.

42. Deveria, Alexis. When can I use. [online]
URL: http://caniuse.com/. Accessed 4 February 2012.