Cloud Based Processing of Real Time Sensor-Data Streams

Paul Lapok

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
Master of Science

18th Aug 2014

School of Computing

Edinburgh Napier University

Authorship Declaration

I, Paul Lapok, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained **informed consent** from all people I have involved in the work in this dissertation following the School's ethical guidelines

Signed:

Date: 18.08.2014

Matriculation no: 40 132 336

Data Protection Declaration

Under the 1998 Data Protection Act, The University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

The University may make this dissertation, with indicative grade, available to others.

Signed:

Abstract

The aim of the project is to design an architecture for real time sensor data streaming, management and live visualisation over the web to contribute to existing research in the field of the Web of Things. The project will investigate the infrastructure between streaming hardware and website. Typical hardware would be programmable, equipped with sensors or the ability to connect up sensors and a possibility to communicate over the internet, e.g. an Arduino board, Raspberry Pi or also a smartphone.

This project focuses on the universality and manageability of the system to allow many users to deploy the sensing data over a web portal and consume the data on their own websites or allow others to integrate the data into third party websites. The requirements for a universal and manageable streaming service will be developed by investigation of existing systems and analysis of use cases in different application areas.

The operational capability of the architecture was investigated by implementing key features and experiments. The project identified the strength and weaknesses of the architecture and investigated the feasibility of the concept. A basic prototype was developed and tested. The feasibility of several parts of the system was proved by implementations and tests. Visualisation possibilities, security and processes were investigated. System requirements for different application areas were defined. System limits were investigated such as the relation between the streamed amount of sensor values and visualisation update rate, the accuracy of the system with high visualisation update rates and the limits of creating streaming instances as a bottleneck. The experiments and tests defined the limits and gave statements about the system performance, security and feasibility.

Keywords: *Sensor Streaming, Real Time, Web Service, Cloud Computing, Node.js, HTML5, WebSockets, Internet of Things, Web of Things*

List of Content

i. List of Figures

ii. List of Tables

iii. List of Source Code

iv. List of Abbreviations

API
Application Programming Interface 16
ATAM
Architecture Trade-off Analysis Method 25
CPU
Central Processing Unit 53
ECG
Electrocardiography 29
GPRS
General Packet Radio Service 16
GSM
Global System for Mobile Communication 16
GUI
Graphical User Interface 31
HTML
Hypertext Markup Language 22
HTML5
Hypertext Markup Language Version 5 22
IoT
Internet of Things 15
IP
Internet Protocol 16
IPv6
Internet Protocol Version 6 21
Json
JavaScript Object Notation 42
LAN
Local Area Network 30
LTE
Long Term Evolution 27
m-Health
Mobile Health 16
P2P
Peer to Peer 21
QoS
Quality of Service 24
RAM
Random-Access Memory 55
RFID
Radio-Frequency Identification 15
SMS
Short Message Service 16

SSH
Sensor Streaming Hardware....17
TCP
Transmission Control Protocol....16
TLS
Transport Layer Security....36
UDP
User Datagram Protocol....41
WebGL
Web-based Graphics Library....23
WISP
Wireless Identification and Sensing Platform....18
WoO
Web of Objects....20
WoT
Web of Things....15
WSN
Wireless Sensor Networks....15
XMPP
Extensible Messaging and Presence Protocol....21

v. Acknowledgements

I would like to thank Alistair Lawson, my research supervisor, for his patient guidance, support and useful critiques throughout this research work.

1. Introduction

The term “Internet of Things” has become ubiquitous in recent years, and involves the connection between objects and communication between those objects at increasingly high-speed. The internet is becoming more and more dynamic in respect to timely visualisation of data. Higher bandwidth and powerful servers allows the transfer of a large amount of data in milliseconds. This increase in resources creates opportunities for new ideas and technologies. The development of an infrastructure that meets the demand of this high speed data transfer is investigated in this work.

The aim of the project is to design an architecture for real time sensor data streaming, and for management and live visualisation of that data over the web to contribute to the existing research field of the Web of Things. This project will investigate the infrastructure between streaming hardware and website. The hardware used for this kind of streaming is typically programmable, equipped with sensors or the ability to connect up sensors and a possibility to communicate over the internet, e.g. an Arduino board, Raspberry Pi or also a smartphone.

The project focuses on the universality and manageability of the system to allow many users to deploy sensing data over a web portal and consume the data on own websites or allow others to use the data on their websites. The requirements for a universal and manageable streaming service will be developed by investigation of existing systems and analysis of use cases in different application areas.

The operational capability of the architecture will be investigated by implementations and experiments. The project will identify the strength and weaknesses of the architecture, and investigate the feasibility of the concept. A basic prototype will be developed and tested. The feasibility of several parts of the system will be proved by implementations and tests. Visualisation possibilities, security, and processes will be investigated. The experiments and tests will define the limits of the system and give statements about the system performance.

In order to meet the above aim, the dissertation is set out as follows. In chapter 2, the literature will be reviewed. This is followed in chapter 3 by an investigation of use cases and an analysis of system requirements. The system architecture will be proposed in chapter 4. Chapter 5 treats implementations and experiments to prove the feasibility of several parts of the proposed system. In chapter 6 the work will be evaluated, followed by the conclusion and future work in chapter 7.

2. Literature Review

This chapter will investigate applications and services where sensing data is involved. The area of sensor streaming and sensor technologies, including Wireless Sensor Networks (WSN) and Radio-Frequency Identification (RFID) will be investigated to give an overview about technical possibilities in the field of sensing. The vision of “Internet of Things” (IoT) and the relations to sensor data streaming will be analysed. The transition to the “Web of Things” (WoT) as an evolutionary step of the IoT will be made to explain the term of virtualisation and infrastructure of sensor streaming. Then web service architectures and sensor streaming management systems will be treated and finally actual web technologies will be investigated.

2.1. Sensor Streaming

Sensors measure a physical quantity and convert that measurement to a signal, which can be presented as a single value over time. The sampling rate of a sensor may vary from milliseconds to hours depending on the tasks and requirements of a system. A large range of Sensors have been used in different applications, to enumerate all of them would be beyond the scope of this work. The focus will be on sensors that can be involved in web applications for visualisation.

Application Areas

Data from sensors have been recorded and streamed for over 30 years, mostly concentrated on environment monitoring. This has included systems monitoring drought (Deng et al., 2013), seismology (IRIS, 2014), tsunamis (Australia, 2014), landslides (Teja et al., 2014) or floods (Y. Zhang, Li, Li, & Guo, 2009) for the purpose of alerting to potentially dangerous situations. Such systems send their data at intervals to their base stations or to a centralized computer (Ha, Lee, Vu, Jung, & Ryu, 2012), sometimes directly over the web where it will be stored in databases for use in longer term research. The update rate of the sensor data in a browser or database may vary from between a few seconds up to hours according to the requirements of most of the systems reviewed, e.g. the systems previously

mentioned. A high update rate compared to the sensor sampling rate or real-time computing, like in industrial applications, e.g. machine control, is not targeted by the systems.

There are existing systems in the area of object tracking over the web as well. Tracking systems consisting of hardware and server using the Global System for Mobile Communication (GSM), General Packet Radio Service (GPRS) and often the Google API to visualize the tracked objects as discussed in (Chadil, Russameesawang, & Keeratiwintakorn, 2008), an open source tracking system using commodity hardware as described in (El-Medany, Al-Omary, Al-Hakim, Al-Irhayim, & Nusaif, 2010), a system using ASP.NET and TCP/IP sockets sending IP data packets over the connection with a time interval of 1 minute. Other systems using the Short Message Service (SMS) to transfer the data with an interval of 12 seconds (Rao, Izadi, Tellis, Ekanayake, & Pathirana, 2009). A fire emergency system was introduced and implemented in Java (Shamszaman, Ara, Chong, & Jeong, 2014). The system has a user interface which presents sensing data but there was no focus on performance of the system especially not on a fast transmission and actualisation rate. Most of these systems store the data in databases which can be accessed over the web.

In 2010, Val Jones established a discussion on two existing solutions for Mobile Health (m-Health) systems (Jones, Gay, & Leijdekkers, 2010). Patients wear sensor systems, measuring electrical activity of the heart (Electrocardiography), electrical potential generated by muscle cells (Electromyography), weight, temperature, respiration, body position, blood pressure, blood glucose and oxygen saturation. The sensor systems transmit their data to a handheld which acts like a communication gateway to a remote healthcare location. The philosophy of both solutions is to design a generic mobile system which can be adapted to different clinical applications (Jones et al., 2010).

The present work will concentrate on high speed data transmission and visualisation. The aim is to design a system which is able to stream sensing data with a high

transmission speed and visual update rate in a range of milliseconds. Data will not be stored on the way between the Sensor Streaming Hardware (SSH) and the website to avoid slowing down the data flow and increase the privacy and security of data.

The system will be kept as universal as possible to allow operational capability in many different application areas. For that reason system attributes should be configurable for different requirements.

2.2. Sensor Technologies

Different kind of sensor technologies can be used for sensor streaming. The following section is an investigation into Wireless Sensor Networks (WSN) and RFID technologies, demonstrating how they feature as a rising technology, and pointing out the possibilities of sensor streaming now and in the future.

Wireless Sensor Networks (WSN)

WSN are used in many application areas including environment monitoring, healthcare, home automation and traffic control. WSN are networks of autonomous devices monitoring physical and environmental conditions at different locations, like temperature, vibration or pressure with sensors (Ha et al., 2012). Moreover they are used in m-Health applications, where wearable body sensor networks monitor the physical condition of patients (Jones et al., 2010). There are different architectures of sensor networks. One scenario can be of sensors which send their data to a sink node and the sink node redirects the data to the web.

Radio-Frequency Identification (RFID)

RFID allows unique identification of objects and tracking them by stationary RFID readers. Systems based on RFID technologies are focused on energy consumption caused by the limitation of energy transmission over the air. Many application areas involving RFID based systems are described e.g. in transportation and logistics domain, healthcare domain, smart environment domain, and personal and social domain (Atzori, Iera, & Morabito, 2010), as well as in Traffic Monitoring, Tracking, Smart Environments, and local and global sensing (Singh, Tripathi, & Jara, 2014).

They all include a base station RFID reader and objects with RFID capacity. RFID can be categorised depending on how it is powered. Active RFID is battery powered and nearly similar to WSN, passive RFID is powered over the air by the RFID reader and hybrid systems like WISP (Wireless Identification and Sensing Platform) are powered by the RFID reader, but have also a small solar cell to allow data logging beyond the range of the reader, which can be up to 10 meter. A future aim is to improve the sampling rate of the sensors and the range of this system (Sample, Braun, Parks, & Smith, 2011). The reader combined with a streaming hardware acts like an interface between local RFID processing and the sensor streaming web service.

The present work will not focus on low level sensor architecture and local communication between sensors. The focus of investigation will be on the infrastructure between the SSH, which could be a sensor node or sink, and the website used to visualise data in real time.

2.3. Sensor Streaming Area

Internet of Things (IoT)

The IoT is a large area with the vision of unique identification of real world objects and interaction with and between different objects and services. Depending on different technologies or application areas the definition of IoT may vary in a number of ways. Some of these definitions focus on uniquely addressable interconnected objects by using RFID technologies, others focus on communication and interaction between objects or investigation in infrastructures and others focus on user centric approaches (Gubbi, Buyya, Marusic, & Palaniswami, 2013). The data transmission between objects is the core statement of these definitions.

Due to the interdisciplinary nature of the subject, three paradigms for realising IoT were defined. The internet – oriented (middleware), the things oriented (sensors) and the semantic oriented (knowledge). The IoT is the interception of these three paradigms (Atzori et al., 2010).

In 2013, Jayavardhana Gubbi defines the IoT as an *"Interconnection of sensing and actuating devices providing the ability to share information across platforms through a unified framework, developing a common operating picture for enabling innovative applications. This is achieved by seamless large scale sensing, data analytics and information representation using cutting edge ubiquitous sensing and cloud computing."* (Gubbi et al., 2013) This definition is also an approach for this work.

The present system will allow deploying and summarizing collected data from single sensors, sensor networks, RFID systems or smartphones and making them accessible over the web for visualisation purposes or allowing communication with other objects.

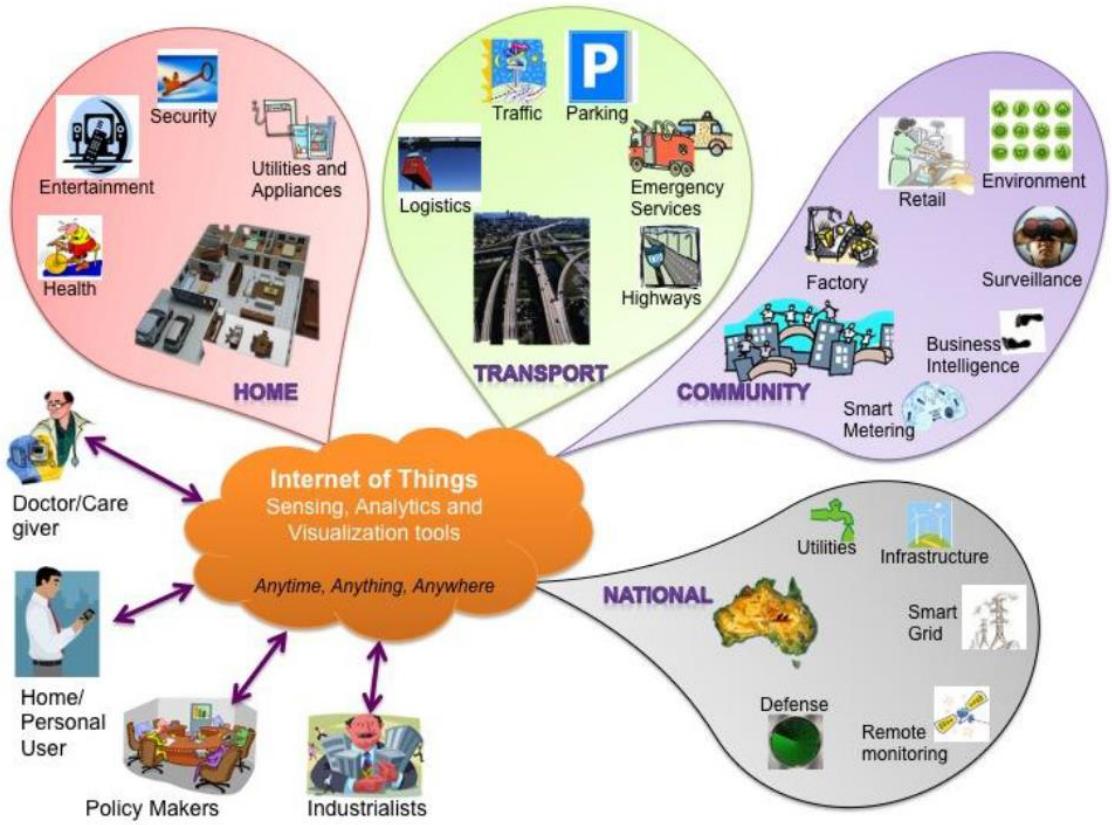

Figure 1: Internet of Things Schematic showing the end users and application areas based on data (Gubbi et al., 2013)

Web of Things (WoT)

The WoT or Web of Objects (WoO) is a higher level of the IoT. It focuses on infrastructure and addresses virtualisation of real world objects. It has been defined by Zia Ush Shamszaman as: *"The goal of the WoO is to deliver a service infrastructure that simplifies the management of the smart service environments able to provide a service that integrates various technologies like cloud computing and social networking. Hence, WoO allows the reuse of existing web technologies to build new applications and services by attaching smart objects to the Web. In this way, smart objects are abstracted as web services and seamlessly integrated into the living world of the Web, while services are discovered, composed and executed as needed."* (Shamszaman et al., 2014)

One output of the present work will be an infrastructure that allows easily deploying and managing sensing data. This infrastructure will be extendable and reusable in specific context, which matches the goals of the WoT. The proposed system does not abstract objects as web services, but their attributes are projected to the web in form of sensing data.

2.4. Architecture

Data Centric Systems

Data Centric systems are the state of the art. The information exchange is running through a central server. Most web services have this centric architecture. Data centric systems can handle massive connections and support different communication protocols, depending on their performance and bandwidth. Scenarios in the domain of smart cities, connected cars, crisis management and health care require user-centric services (Bendel et al., 2013). Data centric systems allow keeping manageability of data on one place. Most of introduced systems in chapter 2.1 and 2.2 are data centric systems.

Peer to Peer networks

In Peer to Peer (P2P) networks all computers have the same status. P2P is a decentralized network architecture where every computer (Peer) is acting like a client and a server at the same time. One approach to connect objects in a P2P way, is to equip every object with hardware and tiny web servers and make them accessible over the web with a standard web browser by a TCP connection (Duquennoy, Lifl, & Grimaud, 2009). In RFID a similar approach can be found by making RFID equipped objects reachable from a network by addressing them over IPv6 address (Atzori et al., 2010). This is done by embedded webservers which make communication between objects easier. An example for near real-time device to device communication with P2P connection over Extensible Messaging and Presence Protocol (XMPP) is done by Bendel in 2013. The setup consists of a Wii Remote controlling a robot over a server with a delay of 112 milliseconds (Bendel et al., 2013). In 2013, Forsström created a P2P architecture which allow real time communication between entities and had the idea of a sensor sharing platform (Forsström & Kanter, 2013). P2P approaches may increase communication speed but making it difficult to ensure security and manageability of data. Also the scalability is an issue in case if one device needs to handle a mass of connections.

Sensing Data Management

Several projects for sensing data management exist, such as introduced by Cia, Liang and Wang in 2011, a monitoring systems, collecting sensor data like temperature or humidity and sending them to a sink node using the ZigBee communication protocol (Cai, Liang, & Wang, 2011). The data is send from the sink over a Network Bridge to a control centre where the data is stored and can be viewed. The data can be accessed over the web. Another project is presented by Zhang in 2013, which has the goal to maximize sharing and utility of available sensor data sources and data services. Zhang also compares different existing commercial sensor data platforms, like TempoDB, COSM and Sensor Cloud. TempoDB (TempoDB, 2014) is a database as a service. It has built-in visualisation tools and focus on time series data sets and on individual users, using own data (J. Zhang et

al., 2013). COSM (LogMeIn Inc., 2014) (renamed to Xively) is a commercial platform for deployment of sensor data and visualisation in nearly real-time for industrial software solutions. It allows sharing of sensing data between users, supports certain commercial devices and focus on sensors with discrete readings (J. Zhang et al., 2013). Sensor Cloud (MicroStrain, 2014) focus on individual users, using own data and allows remote device management. It has a math engine and built-in visualisation (J. Zhang et al., 2013).

All these systems store sensor data to databases in contrast to the idea of present work, where data will be redirected and direct consumed by client applications, e.g. visualisation of data. The decision is to store data on client side. The work will investigate fast processing, high update rates for smooth visualisation and how to deliver a high grade of universality.

The Sensible Things Platform is a web service for deployment of sensors and enabling IoT based applications (Forsström, Kardeby, Österberg, & Jennehag, 2014). It allows connecting different sensors and actuators by registering them to the platform. The platform is an open source architecture which allows developers to build their application based on it. The platform uses P2P connections and has a response time of a few milliseconds. The system does not store any data in a database and is open to use for everyone.

It is nearly similar to the idea of present work with the difference that Forsströms system is a real P2P system without any central server for management of the sensor streams. It is easy to use but coding skills in java are required by the user which requires a much experienced developer.

2.5. Web Technologies

HTML5 and WebGL

The Hypertext Mark-up Language (HTML) is a core technology of the internet. HTML5 is the latest HTML standard for structuring and presenting World Wide Web content. It allows the inclusion of multimedia and graphical content without the use of

plugins (Chen, 2011). The Web-based Graphics Library (WebGL) allows 3D accelerated graphics in web browsers. It uses JavaScript and can be used with HTML5 without any browser plugins (Chen, 2011). It is important for the visualisation part of this work because it allows the actualisation of web content in real time.

WebSockets

WebSockets allow bi-directional communication over TCP between web browsers and web servers or other client or server applications. The usual way before WebSockets exist was to ask the server for a resource, getting the response and closing the connection (Chen, 2011). In one project WebSockets were used to transfer intensive rendering visualisation data in real time. The data was rendered on server side and transferred to a client application to minimize client side processing (Wessels, Purvis, Jackson, & Rahman, 2011). An architecture for a virtual world web client was developed and evaluated consisting of WebGL and WebSockets technology (Dahl, Koskela, Hickey, & Vatjus-Anttila, 2013). WebLab-FPGA is an remote laboratory project connecting WebGL, WebSockets with hardware (Orduña & Angulo, 2014). The hardware can be remotely programmed and control a virtual pump. WebGL and HTML5 was studied in Anderson and Johansson in 2012 with focus on cross-device and cross-browser ability (Andersson & Johansson, 2012). WebSockets are integrated in the HTML5 standard. In this project WebSockets were used for the connection and data transfer between sensor streaming hardware and server and the connection between server and website.

Node.js

Node.js is a lightweight server technology for data intensive real time applications. It uses asynchronous non-blocking input and output processing and is used and established by many companies like Yahoo, eBay, Microsoft and PayPal. Node.js is built on Chrome´s Java Script runtime and is written in C++ and Java Script programming language. Programs for Node.js must be written in Java Script (Joyent Inc., 2014). Alternatives like SignalR from Microsoft exist. Node.js was chosen because of the high acceptance of the community and a larger support. In this project

it is used as the junction between sensor streaming hardware and website. It manages the incoming data stream and broadcast it to the websites.

ASP.NET

ASP.Net is a Microsoft technology for creating web applications. It is a client and server-side technology. ASP.NET is based on the .Net Framework architecture and allows the creation, deployment and execution of web applications and web services. It makes the development process much easier because it provides a lot of development tools and advantages like libraries from the .Net Framework (Microsoft, 2014). Alternative technologies are PHP5 (Hypertext Pre-processor) or JAVA Server Pages. ASP.NET was chosen because of the possibility to use a Microsoft server technology.

2.6. Conclusion

Different technologies have been reviewed. RFID has some problems such as missing standards, mobility support, naming, transport protocols, traffic characterisation and Quality of Service (QoS) support, authentication, data integrity, privacy, and digital forgetting (Atzori et al., 2010). Several different sensor management systems were investigated with different focus, philosophies, architectures and possibilities. There are many sensor streaming services available. Most systems are focused on one specific task. The data refreshing intervals are often higher than one second. The architectures consist mostly of data centric systems with databases and are concentrating on long term research, monitoring or big data analysis. Some sensor platforms use P2P architectures. Web technologies for creating real time systems were investigated such as WebSockets, HTML5 and Node.js.

The present work aims to solve existing issues and provide a means to deploy and manage sensor data in a user friendly way. To do this, it will mainly focus on streaming data from SSH to a website and visualisation of data in real time.

3. Requirement Analysis

This chapter is an investigation of requirements for the sensor streaming web service. The first part is a requirement definition which identifies requirements and specifications for the architecture concept. The second part analyses representative test cases in different application areas and sets them in relation to the previous definition of requirement. The last part lists all identified requirements based on the analysis.

3.1. Requirement definition

The system will be used in many different applications. The focus of the system will be on universal usability and configurability. Some requirements may change for different use cases others not. The requirements were taken form (J. Zhang et al., 2013), which is a nearly similar system handling a big amount of data and connections. Zhangs requirements arose from an Architecture Trade-off Analysis Method (ATAM) and are as follows:

- Scalability
- Reliability
- Interoperability (Static requirement)
- Security
- Integrity (Static requirement)
- Data Freshness (Static requirement)
- Privacy
- Performance
- Reusability (Static requirement)
- Extensibility (Static requirement)

The requirements were divided into changing and static requirements. Static requirements are not changing in different for use cases. Changing requirements

may vary for different use cases. The following changing requirements were rated to distinguish the different use cases.

- Scalability
- Visualisation update rate (Performance)
- Importance of mobility
- Importance of reliability
- Importance of security
- Importance of privacy

For each requirement limits are defined (high, medium or low) which are shown in **Table 1**. The limits are explained and will be used to distinguish the application cases in the next chapter.

	Evaluation of Cases				
Area	Healtcare	Engineering	Transport	Private Sector	HCI
Measuring	ECG	Manufacturing Tolerances	Vehicle Telemetry	Smartphone Sensors	Position/ Acceleration sensors
Visualisation Technology	2D-Graph	Values, 2D-Model	Values, 2D-Graph/Map	2D-Interaction	3D-Interaction
Scalability	High	Low	Low	Low	Medium
Visualisation Rate	Low	Low	Medium	High	High
Mobility	High	Low	High	Medium	Medium
Reliability	Medium	High	Medium	Medium	Medium
Security	High	High	Medium	Low	Low
Privacy	High	Medium	Medium	Low	Low

Table 1: Valuation Basis

The six requirements were identified. They may have different importance and definition for different applications. The complete table can be found in the appendix (A.1).

The first requirement is the scalability. Different applications use a different amount of sensors streamed over a SSH. Some applications may stream many sensors at once and others applications may send a set of sensor data recorded in an interval. Especially in applications with an extremely high sensor sampling rate the data can

be sent after a time interval in a set. It was assumed that most tasks will transmit between 10 and 100 values at once. A “high” amount was defined for sending more than 100 values at once, a “medium” amount for fewer than 100 values and “low” amount for fewer than 10 values sent at once.

The visualisation update rate is the second requirement. It defines the speed of actualising the data in the browser. Some applications may actualize their data in an interval of a couple seconds, like weather data, where a fast update rate is not important because the sensor data changes slowly. Other applications need timely visual feedback, such as a human computer interface for controlling web content, which needs an update rate in the range of milliseconds to produce a smooth visual feedback for the user. Based on the assumption that for most applications an interval of one second will be enough, as discussed in other systems in the literature review in section 2.1, a visualisation rate of 0.1 seconds was defined as “high”, fewer than 1 second as “medium” and more than 1 second as “low”. Smooth visualisations can be produced with a rate faster than 16 frames per second (Neumeyer & Brown, 2014), which is a visualisation rate of 0.0625 seconds. Section 5.6 investigates how close the system can get to a smooth visualisation.

Mobility is the third requirement. It is different between different application areas. Some applications will be used whilst stationary and others whilst mobile which may have an influence on data transmission speed and connection stability. Mobility was divided into “high”, which means a high flexibility nearly everywhere by using mobile internet like 3G. Long Term Evolution (LTE) was not considered since it is still not as widespread. “Medium” means location based flexibility by using Wi-Fi and “low” is for fixed wire internet connections.

Reliability is the fourth requirement. Mechanisms for monitoring the system are needed, to allow exception handling and to redeploy or scale a broken or overloaded server instance. The importance is “high” if there is no tolerance for connection problems because the data is timely needed. “Medium”, if the connection may break but a timely arrival time of data is not important. In this case data can be buffered and

send at once if the connection is rebuild. And “low” if transmitted data can be left unnoticed.

The fifth requirement is the security. The streaming data needs to be secured against unauthorised access of third parties. “High” security is a complete encryption of data with encryption certificates. “Medium” security is a secure transport between destinations. And “low” security is the transmission or raw data, which can be done in case of sharing the data.

Privacy is the sixth requirement. “High” importance of privacy was defined if the streamed data can be abused by a third party to harm an individual person or an organisation. A “medium” importance is defined if the data is privacy relevant but cannot harm an individual person or organisation. And “low”, if privacy is not important, e.g. in case of data sharing.

3.2. Review of Example Application Areas

Five different application areas were investigated to keep the web service as universal as possible. Each area includes a case, which focus on high speed streaming and direct visualisation over the web. A table which summarizes the evaluation of the cases can be found in the appendix (A.1).

3.2.1. Health

Electrocardiography (ECG) is used to measure the electrical activity of the heart and usually visualizing the values in a 2-Dimensional graph. The transmission of an ECG signal was chosen as an example because it is a complex and time critical task. An average human resting heart rate ranges from 60-80 beats per minute (American Heart Association, 2014). The ECG records also values between heartbeats, which creates a more complex graph. An ECG signal was investigated to find out how many values are recorded during a second. PhysioNet (Goldberger et al., 2000) is a database which keeps records of medical data also, among other, ECG signal data. The selected ECG signal (Association for the Advancement of Medical Instrumentation, 2002) has a sampling rate of 720Hz, which could be transmitted in two ways. The first possibility is to send every single value to the website. In that case the transmission and visualisation need to be faster than 1.4ms. The second way is to send a set of 720 values each second, which is more efficient regarding transmission speed. The systems targets flexible users with high mobility thus not critical patients, e.g. for sports, study or diagnosis purpose. Accordingly, it should be able to transfer the data over a mobile connection. In this case, connection interruptions may be caused by poor mobile connectivity. The system needs to be secure against attacks. Data privacy is very important and the data needs to be encrypted, because it has a direct relation to a person.

3.2.2. Engineering

An engineering example is a production line producing a high number of products e.g. one second per product. These products were monitored by e.g. five laser

sensors which were used for detecting manufacturing tolerances. The streaming system is used to send the measured values to a company internal website to increase the flexibility of responsible employees. The data is accessible from anywhere with any device. Tolerances can be visualized as blank values, but also as 2-Dimensional models. The system needs to transmit the five measured values immediately. The visualisation update rate depends on the speed of production which is in this case one update per second. The data can be transferred over a LAN connection, because the production line is stationary. The reliability of the system needs to be high because it is responsible for production quality and the data is kept as a record for the company. The system needs to be safe against attacks, but data do not need to be encrypted. The privacy of data is in that case not important because it has no relation to a person.

3.2.3. Transport

Vehicle telemetry is sent from a vehicle to a website to visualize energy consumption, position and acceleration of the vehicle, or CO2 emission. The data can be visualized as blank values, a 2-Dimensional graph or on a map. In this specific case are not many sensor values involved. The visualisation rate should be faster than a second to allow a smooth visualisation. The system needs to work over a mobile connection, which could cause connection interrupts. The connection between vehicle and website needs to be secured. The data privacy is important but is not in a direct relationship to a person.

3.2.4. Private Sector

A smartphone is used as an input device for interaction with web content. Sensors, like gyroscope, accelerometer and touchscreen are streamed to the website and their values can directly influence the content. There are not many sensors involved, but the visualisation update rate needs to be high to allow a smooth interaction with a minimum of delay. Wi-Fi is used for data transmission, because the interaction takes place in front of a screen. Connection loss may happen, however it is not a critical

application. The connection between device and website needs to be secure. Privacy is not of importance in this area.

3.2.5. Human Computer Interaction

Sensors position and acceleration sensors in gloves are used as an input device to interact with a 3-Dimensional environment directly inside the browser. Three Sensors, gyroscope, accelerometer and magnetometer are needed like in (YEI Corporation, 2014), which is a sensor suit for motion capture. Each sensor generates three values to represent one rigid object. In this case, two fingers are equipped with one sensor system consisting of gyroscope, accelerometer and magnetometer on each finger, which allows many different interactions with a virtual environment. The visualisation update rate needs to be very high to allow a smooth interaction with the environment. The interaction happens in front of a screen, so the sensors could communicate over Wi-Fi. The reliability of the system is in that case not very important. The importance of security and privacy is similar to the usual interaction with websites.

3.3. List of Requirements

This section lists the defined requirements of the system based on the analysis of the application areas and evaluation in the previous section of this chapter.

i. Scalability

The service can be used in different application areas. It can transmit any kind of text based data and different amount of values at once. The amount of values can be configured by a web based Graphical User Interface (GUI).

ii. Visualisation Update Rate

The system can be used with different visualisation update rates. The interval of data transmission needs to be configurable by a web based GUI.

iii. Configurability

The system needs to provide a web based GUI which allows configuration of the system. The user should make only a minimum of configurations on hardware side. The hardware should also provide a GUI for the configuration.

iv. Sensor Data Sharing

The system needs to provide a web based GUI which allows deploying, sharing and publishing the sensor stream.

v. Frontend Development

The user should only need frontend development skills in HTML and Java Script for visualizing sensor data.

vi. Visualising Data

The service should allow visualisation of sensor data as single values, 2D graphs or maps and also interaction with 3D objects without significant delay.

vii. Database Adaption

The user should be able to adapt a database and redirecting data from client side to the backend by using Java Script and storing the data to the database.

viii. Security

The service needs to provide user authentication for the management system (GUI). The user can choose between different kind of security, like encrypted, transport secured and raw data transmission. Potential threats need to be analysed and avoided.

ix. Privacy

The system should hide private user data or trace back of user connections.

x. Reliability

The system needs to be reliable and handle connection loss, overloads and server break downs by monitoring the system and solving problems.

xi. System Limits

System limits should be identified and stored to give the user feedback on configuration and avoid an impossible setup, e.g. extremely high visual update rate with extremely big amount of values to transmit at the same time.

4. Proposed System Architecture

This chapter proposes a system architecture design. It starts with a general overview and description of the system and goes in more detail by investigation of a single connection. The processing sequence from the user's point of view and the internal server processes will be described. The last part will discuss the relation to the requirements and a concept of monitoring the system.

4.1. Overview

The architecture concept will provide a platform for sensor data distribution and sharing. **Figure 2** shows the overall system.

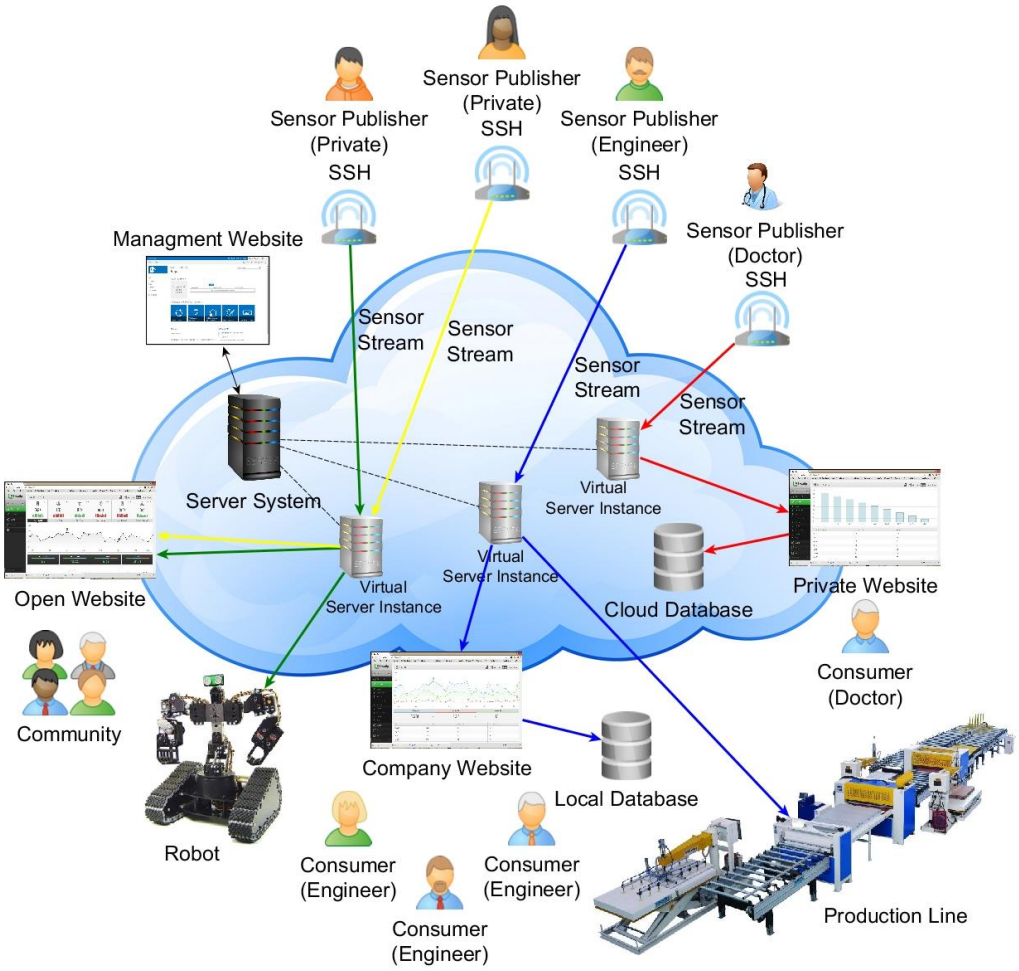

Figure 2: Architecture - Overall System

Different users can use the platform at once. They streaming sensor data over Wi-Fi, but it could also be a mobile connection or LAN to a virtual server instance, created by a management server. The virtual server instance is a Node.js instance and will be configured by a management website on the server by the sensor owner. It will allow configuration of security, update rate, amount of values and usage permissions for other users. The Node.js instance works as a distributor and can be consumed by many websites at once. Websites can connect to the instance and use the stream for visualisation or can redirect the data to a local or cloud-based database. The data can be visualised due to standard web technologies in many ways, e.g. graphs and 3-Dimensional models. The sensor data can be also used by other hardware, like robots or machines. It will be also possible to combine two nodes on one website or merge two sensors of different users on one server node.

4.2. Management User Interface

The management web service is an ASP.NET MVC project which provides a website with a user interface. A user can create an account, authenticate and register a sensor. The following connection properties can be configured by the interface.

Sensor Identification number

The sensor identification number will be generated automatically from the server after registration. The user needs to copy the number together with the username to the streaming hardware. It will be used to identify the sensor stream when it connects the first time to the service.

Amount of Values

The amount of values to stream in one interval to the server will be configured by the user. One sensor has usually one value. It may happen that a sensor has more values, e.g. acceleration in three dimensions, or a user connects ten different sensors to a SSH and wants to send all values to a website at once. Another possibility is to use this attribute for sensors with an extremely high sampling rate, e.g. 720Hz. This would comply with 720 updates on a website during one second, which is not practical and not effective. In that case the user can configure to stream

a set of, e.g. 360, 720 or 1440 values in one interval. The amount of values has an influence on the possible update rate of the system. As more values are streamed at once as larger the package size will be, as slower the data transmission and also the visualisation. The influence of different amount of values will be investigated in a later chapter.

Update Rate

The update rate is the interval in which the SSH send the data to the server and the server actualize the content of the website. Depending on the use case, the user can choose from hours up to millisecond. The influence of different update rates and limits will be investigated in a later chapter.

Data Hold

The user can decide to choose the “Data Hold” option to minimize traffic between SSH and server. The sensor data will be only streamed to the server if the measured value changes. This option is interesting for sensors which measuring slow changing physical quantities, like e.g. room temperature, light or trigger. This function should be integrated between the server and the website as a standard to lower the traffic on client side.

Sensor Publishing

The user can choose to publish the sensor data with the community or make it accessible to specific users. Other users can consume the data on their websites.

Security

The user needs to configure the security between SSH and server. Three options are available for the connection. Standard is the Transport Layer Security (TLS). In addition, the data can be full encrypted by exchange of encryption certificates. But also a transmission of raw data is possible.

4.3. System Tasks

After the registration and configuration, the management server creates a Node.js instance with the configured setup. **Figure 3** shows a single connection between SSH, Node.js instance and website. Also the management website connected to the management server is illustrated. Each element of the architecture has individual tasks.

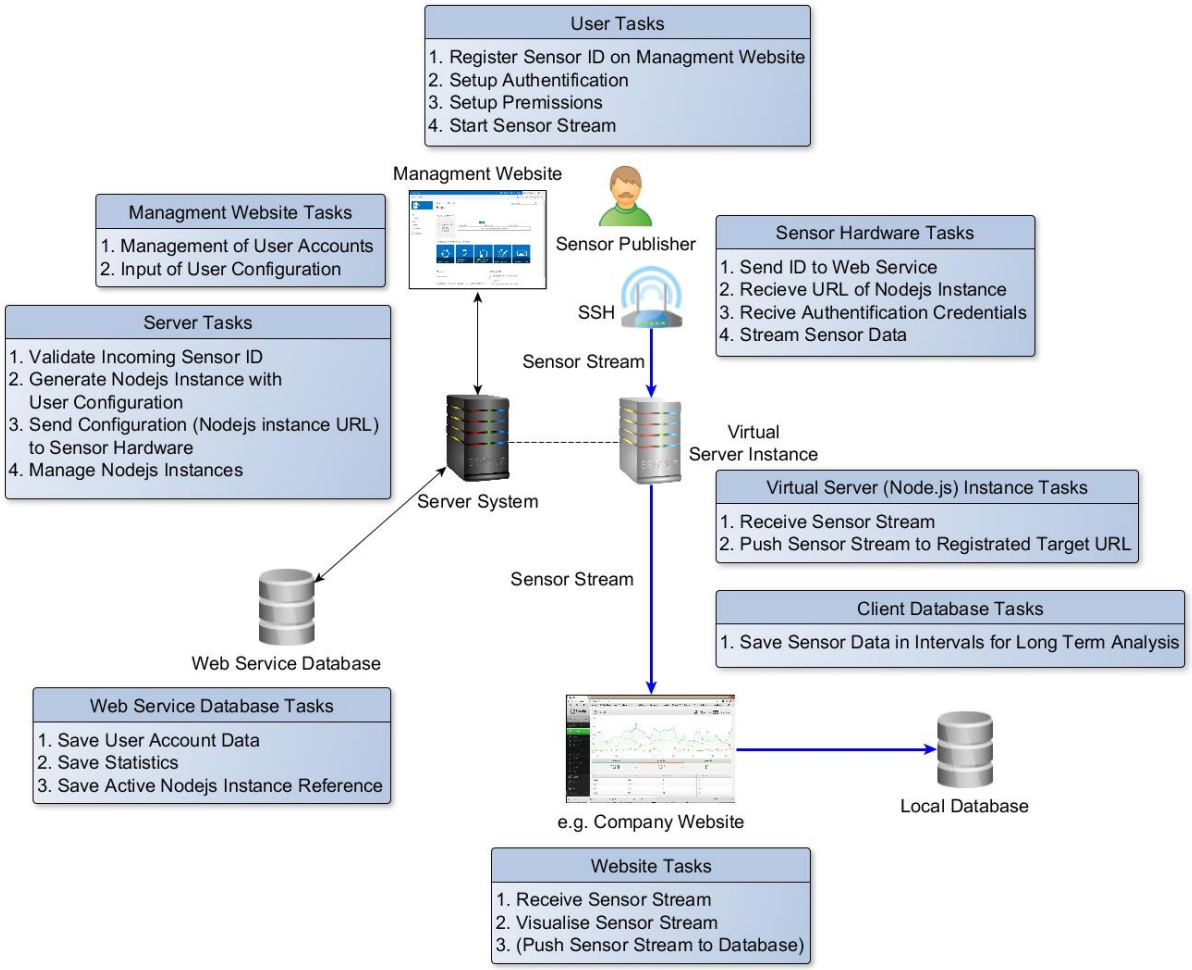

Figure 3: System Process Sequence

The management website provides the user interface for registration and configuration of the hardware. After creating a sensor on the website, the management server generates the Node.js server instance based on the configuration. Like mentioned in 4.2, the user has the task to configure the SSH. The

SSH provide a GUI which can be accessed through the local network by the IP address of the SSH. After turning the SSH on, it connects to a registration service, where the device will be validated based on the stored configuration on the web service. The SSH receives the configuration from the service, connects to the Node.js instance over a WebSocket connection and streams sensor data to the server instance. The Node.js instance is the broker between SSH and visualisation website. It has a WebSocket connection to both ends, one to the SSH and another to the website, which could be also used bidirectional. The visualisation website can have a backend with the possibility to connect to a database and store the sensor data in intervals.

4.4. System Processing Sequence

A detailed model of the service with all processing steps and interactions is illustrated below in **Figure 4**.

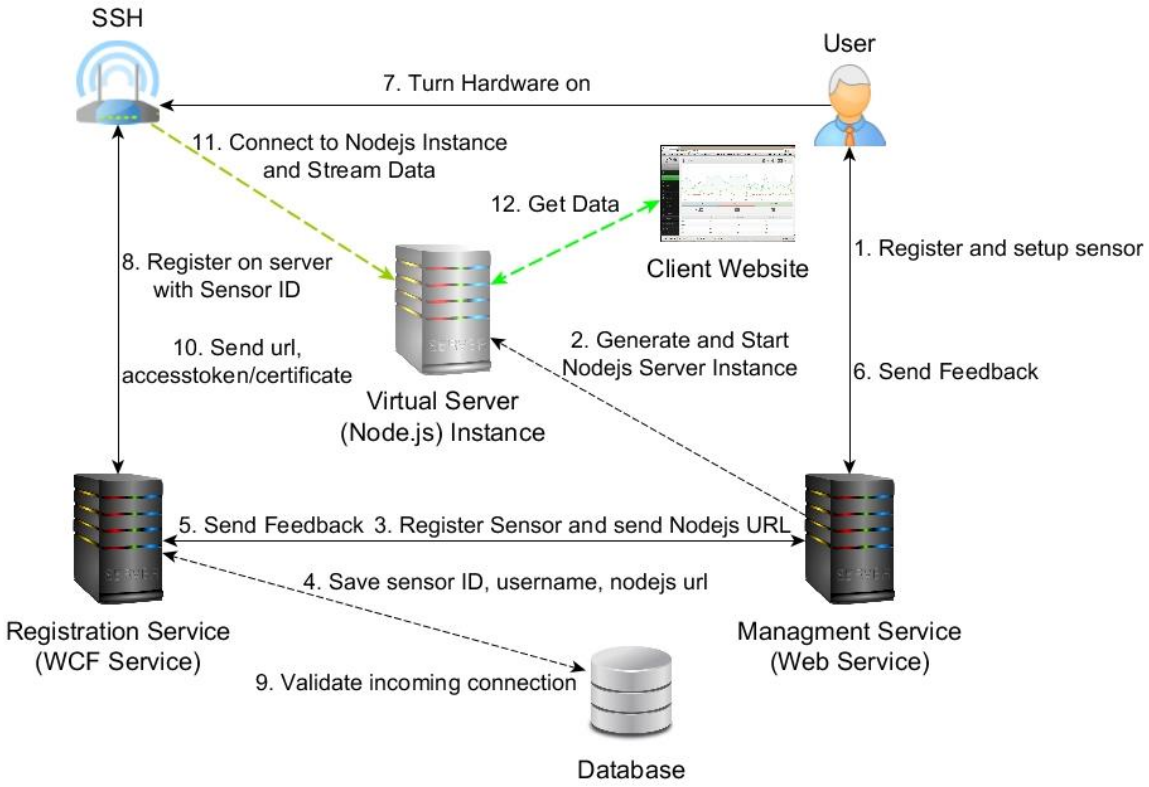

Figure 4: Processing Steps

The first step is the interaction of the user with the management service. The user registers and configures the sensor and data stream. The management service creates a Node.js Server instance with the user configurations and sends the configuration data and Node.js instance address to the Registration Service, which could be hosted on the same server. The Registration Service is responsible for storing configuration data, registration and first configuration of the SSH. It saves the configuration data sent by the Management Service to a database and confirms it. The Management Service confirms to the user that the setup was successful with a sensor ID and the address of the Node.js instance for connecting a website. The user configures the sensor ID and username on the SSH and turns the hardware on. The SSH will automatically connect to the Registration Service with the configured ID and username. The Registration Service compares the ID with the database records and after validation, it sends the users configurations and security certificates to the SSH. Now the SSH knows the address of the Node.js instance and has also the correct security certificates. It connects to it and starts to stream the data immediately. The user can create a website and use the Node.js instance address to fetch the sensor data from the SSH. The user has also the possibility to publish the sensor stream. In that case it appears on the management website in a shared location for a specific user. Permitted users can see the Node.js instance address, can download the certificate and connect to the instance with their websites and fetch the data.

4.5. Monitoring

The system is monitored by a separate web service. The monitoring system is illustrated in **Figure 5**. It increases the reliability of the system. The web service has an own administration web site for statistics and provides an overview over running processes and system failures. The web service fetch a list of all running sensor stream Node.js instances from the registration web service and ping each server of this list. If a Node.js server instance is unobtainable it sends the server information to the management server, which reloads the Node.js server instance. This will guarantee that all servers are running and registered sensors can be streamed.

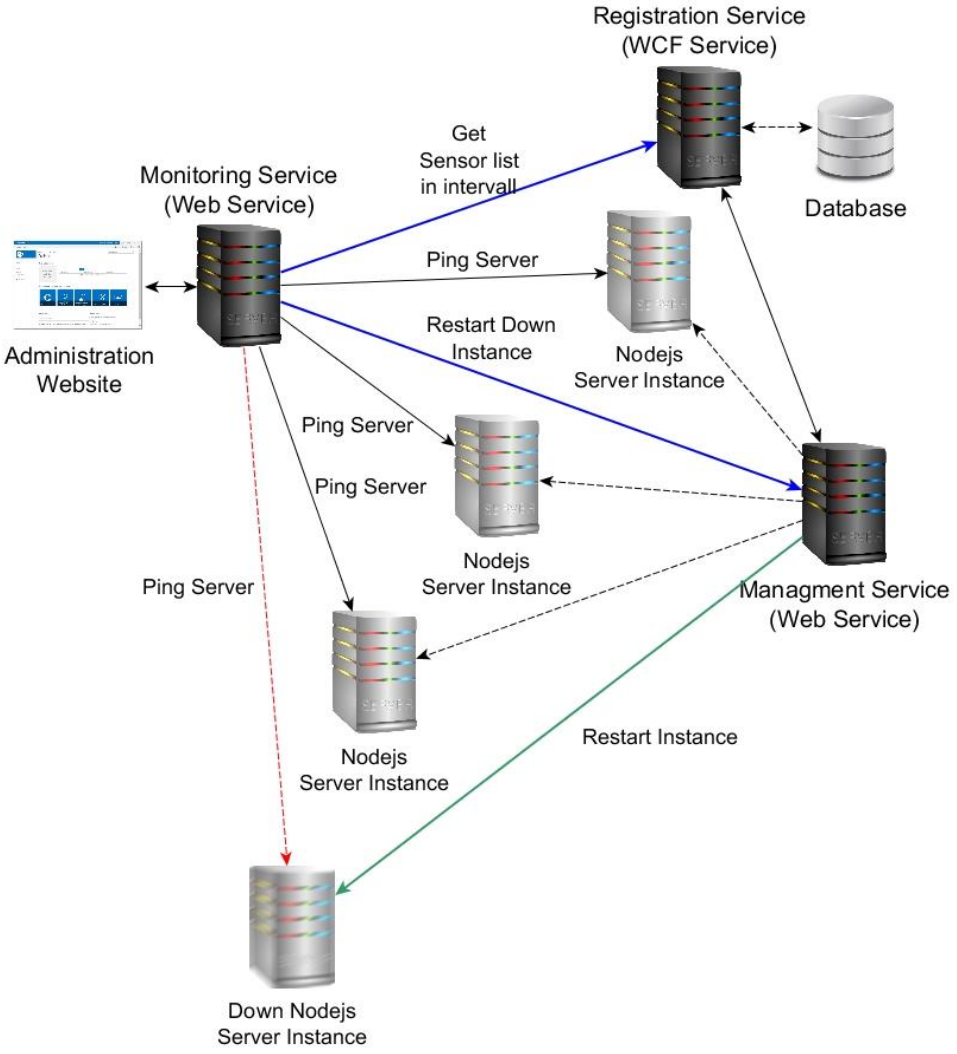

Figure 5: Monitoring Process

5. Implementation and Experiments

This chapter explains the implementation of the software and experiments. Different parts of the architecture concept were implemented and tested to evaluate the operational capability of the concept. The software refers to different points in the concept. Experiments will evaluate the abilities of the architecture and also reveal the limits of the system.

5.1. Node.js Server instance

A Node.js server was implemented in two versions, one version with TLS security configuration and one without. The code can be found in the appendix (A.2). The secure version uses security certificates for transport security, which means the data is encrypted between the endpoints. Both versions can be merged in one server script at a further step. It is possible to instantiate a Node.js server instance with variable ports by adding the port numbers to the execution command.

The server works as a broker between SSH and client websites. A User Datagram Protocol (UDP) connection is used for data transmission between SSH and Node.js server. An UDP connection is fast because after a package is send, there is no confirmation about the arrival of the package. So it may happen that packages get lost or arrive in a wrong order. The incoming SSH stream is received at one server port, processed and sends to websites over another port with a Transmission Control Protocol (TCP) connection, like illustrated in **Figure 6**. The TCP connection between server and website ensures that data packages will arrive and are also in the right order. The transmission is slower caused by the validation process.

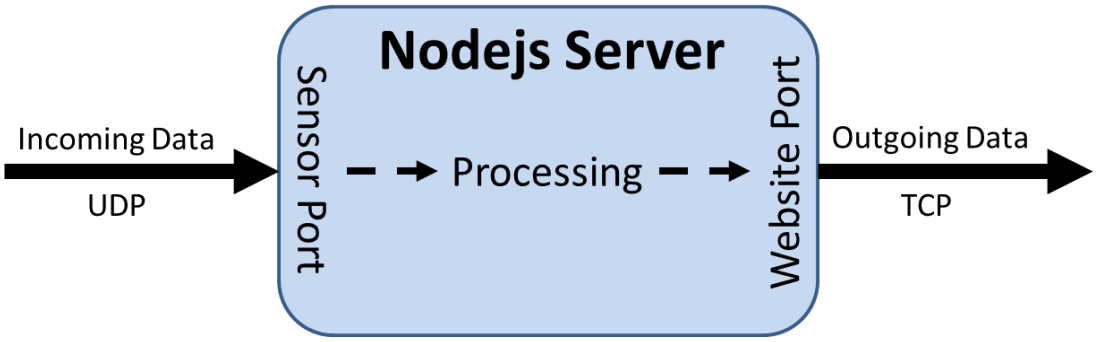

Figure 6: Node.js Server Instance

Due to the UDP connection which does not synchronize the incoming data, it may also happen that packages are received from the server multiple times. A processing step avoid similar packages be forwarded to websites. The incoming data packages are strings in Java Script Object Notation (Json) format. Each package has a timestamp which distinguish it from others. Reading the timestamp would require de-serializing, parsing and serializing, which makes the processing slow. To avoid this, only the hash value of each object will be compared with the hash value of the last object. A new timestamp inside the object generates each time a unique hash value. Similar packages have a similar hash value.

The server allows transmitting data with a variable update rate and the transmission of any kind of text based data. These properties meet the requirements of ***i. Scalability*** and ***ii. Visualisation update rate***.

5.2. Registration Service

A registration service was implemented to test the registration process and analyse the attributes for the registration. The service is an ASP.NET WCF service and consists of a database and a communication interface.

```
1.  [ServiceContract]
2.  public interface ISensorRegistrationService
3.  {
4.      [OperationContract]
5.      Registration RegisterSensor(string username, int sensorid);
6.
7.      [OperationContract]
8.      Response AddSensor(string username, int amountOfValues, int updateRate,
9.                         string code);
10.
11.     [OperationContract]
12.     Response DeleteSensor(string username, int sensorid, string code);
13.
14.     [OperationContract]
15.     Response GetSensor(string username, string code);
16.
17.     [OperationContract]
18.     Response GetAllConnections(string code);
19. }
```

Source Code 1: Registration Interface

The interface allows the SSH to register a sensor which exists in the database by sending the username and sensor id. All other methods are secured by a code and can be only accessed by the system itself for communication purposes. New sensors can be added with a configuration and existing sensors can be deleted. The interface allows retrieving all sensors of one user and also all sensors at once from the database.

The database is illustrated in **Table 2**. It has a unique id for each record and stores username, sensor id, Node.js server address and port, client website port, access token, security certificate, visualisation update rate and amount of values to stream.

	Name	Data Type	Allow Nulls
🔑	Id	int	☐
	Username	nvarchar(MAX)	☐
	SensorID	nvarchar(MAX)	☐
	NodeURL	nvarchar(MAX)	☐
	ClientPort	int	☐
	SensorPort	int	☐
	AccessToken	nvarchar(MAX)	☐
	SecurityCertificate	nvarchar(MAX)	☐
	UpdateRate	int	☐
	AmountOfValues	int	☐

Table 2: Database Schema

Two applications were implemented for the communication with the registration service. The first is a management application with GUI to test the WCF registration service interface. The second application is a console application to simulate the registration process accomplished by the SSH.

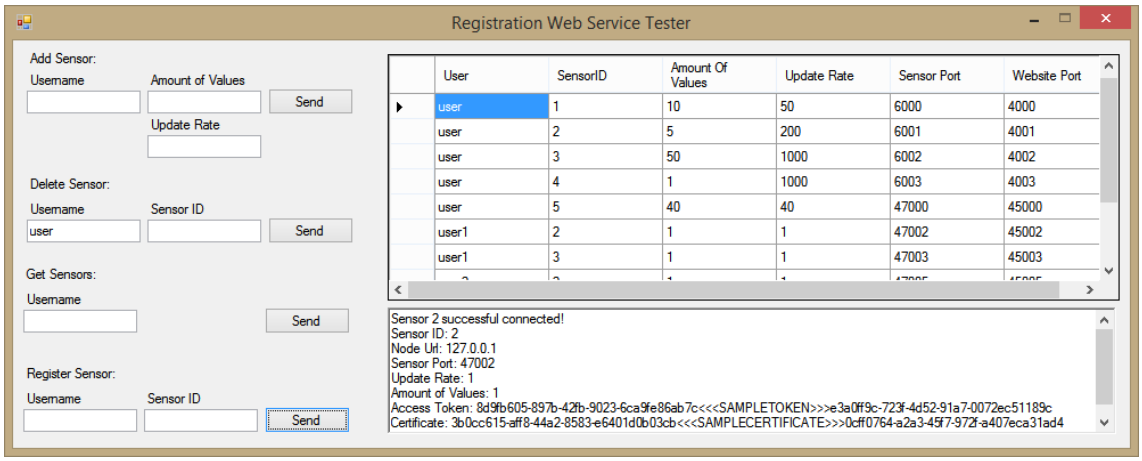

Figure 7: Registration Management Application

The management application, illustrated in **Figure 7**, allows testing the registration interface. A sensor can be added by entering the username, amount of values and update rate. If the user does not exist the registration service will create a new one. The registration service creates all other attributes automatically, like the unique sensor id, Node.js server address with unique ports, access token and security

certificate. The combination of username and sensor id needs to be unique, that allows the user to combine multiple sensor streams. The ports need to be also unique, because the server instance is running on a physical machine and the ports are limited to 65.535. Each port can be used for one incoming and one outgoing connection on server side. One port is used for incoming data from the SSH and the other is used for the outgoing communication, a broadcast to other websites.

The registration application is a console application, which simulates the registration process from SSH point of view. The application takes a username and sensor id. In production state this would be configured by the user on the SSH. The registration application knows the static registration service address and connects to it automatically. The incoming data are checked on the registration server and if they are valid, the registration service response with all needed information for connection to the Node.js instance and also the update rate, amount of values to stream and the security certificate for a save TLS connection. A Node.js server instance was created before in the background which would be created by another server in production state. The registration application starts to stream random data in Json-format to the Node.js instance. The data is visible in the Node.js console and can be also viewed by a local website.

In a production state scenario the SSH configuration would be done by connecting the SSH to a local network and visiting the menu website by a local IP address owned by the SSH like in a modern Wi-Fi router. Username and id can be entered and also additional security token could be added to make the service saver.

This setup fits to requirements ***iii. Configurability*** and ***iv. Sensor Data Sharing***. It is possible to add sensors through a GUI, the configuration of the SSH is reduced to a minimum effort and can be easily extended with a GUI interface. It is easily possible to extend the service by a web portal to allow sensor data sharing.

An acceptance tests for the registration service were defined to test the web service interface and investigate the operational capability of the registration service. The acceptance test was conducted with the registration management application.

5.2.1. Acceptance Test: Unique username and sensor ID combination

Each user may have several SSH which are connected to the service. For identification purpose, the combination of username and sensor ID needs to be unique. Similar sensor IDs may exist but need to be owned by different users.

Examples

Adding new sensors creates a unique username and sensor ID combination. The following is expected:

User	Sensor ID	Unique
User1	1	yes
User1	2	yes
User1	3	yes
User2	1	yes
User2	2	yes
User2	3	yes
User3	1	yes
User3	2	yes
User3	3	yes
...		

Table 3: Acceptance Test Unique ID

The test was successfully passed. Each username and sensor ID combination is unique.

5.2.2. Acceptance Test: Unique ports

After adding a new sensor, the service needs to find free ports for the Node.js server instance to avoid port conflicts.

Examples

Adding new sensors create each time new port numbers. The following is expected:

User	Sensor Port	Website Port	Unique
User1	45000	50000	yes
User1	45001	50001	yes
User1	45002	50002	yes
User2	45003	50003	yes
User2	45004	50004	yes
User2	45005	50005	yes
User3	45006	50006	yes
User3	45007	50007	yes
User3	45008	50008	yes
...			

Table 4: Acceptance Test Unique Port

The test was successfully passed. The service chooses each time new ports.

5.2.3. Acceptance Test: Set sensor ID free after deleting

After deleting a sensor of a user, the sensor ID should be free to use for the next sensor.

Examples

Adding some sensors and deleting the first will set the sensor ID free for the next added sensor. The following is expected:

User	Sensor ID	Action
User1	1	Add
User1	2	Add
User1	3	Add
User1	4	Add
User1	5	Add
User1	2	Delete
User1	4	Delete
User1	2	Add
User1	4	Add
...		

Table 5: Acceptance Test Delete ID

The test was successfully passed. The service set the ID from deleted sensors free. The ID is used for the next added sensor.

5.2.4. Acceptance Test: Set ports free after deleting

After deleting a sensor from a user the ports should be free to use for the next sensor added.

Examples

Adding some sensors and deleting the first will set the ports free for the next added sensor. The following is expected:

User	Sensor ID	Sensor Port	Website Port	Action
User1	1	45000	50000	Add
User1	2	45001	50001	Add
User1	3	45002	50002	Add
User2	1	45003	50003	Add
User2	2	45004	50004	Add
User1	1	45000	50000	Delete
User2	1	45003	50003	Delete
User3	1	45000	50000	Add
User3	2	45003	50003	Add
User3	3	45005	50005	Add
…				

Table 6: Acceptance Test Delete Port

The test was successfully passed. The service set not used ports free and use them for the next added sensor.

5.2.5. Acceptance Test: Get sensors of one user

The service need to provide an overview of created sensors of a specific user.

Examples

Adding some sensors for different users and retrieving the sensors by username will return a list of sensors owned by the user. The following is expected:

User	Sensor ID	Action
User1	1	Add
User1	2	Add
User1	3	Add
User2	1	Add
User2	2	Add
User3	1	Add
User1	1, 2, 3	Get
User2	1, 2	Get
User3	1	Get
…		

Table 7: Acceptance Test User Sensors

The test was successfully passed. The service provides a list of sensors owned by a specific user.

5.2.6. Acceptance Test: Get all sensors

The service need to provide an overview of all available sensors stored in a database.

Examples

Adding some sensors for different users and retrieving them by a command will return a list of all sensors stored in the database. The following is expected:

User	Sensor ID	Action
User1	1	Add
User1	2	Add
User1	3	Add
User2	1	Add
User2	2	Add
User3	1	Add
User1	1, 2, 3	Get
User2	1, 2	
User3	1	
…		

Table 8: Acceptance Test All Sensors

The test was successfully passed. The service provides an overview of all stored sensors.

5.2.7. Acceptance Test: Register sensor

The SSH can register itself by sending a username and a sensor ID to the service. The service needs to validate the data and give a feedback to the SSH.

Examples

Adding some sensors and register the SSH to the service with a valid username and sensor ID results in a positive feedback. Attempts to register with a not existing username and sensor ID combination should result in a negative feedback. The following is expected:

User	Sensor ID	Message	Action
User1	1		Add
User1	2		Add
User2	1		Add
User2	2		Add
User1	1	Successful	Register
User1	2	Successful	Register
User1	3	Not Available	Register
User2	1	Successful	Register
User2	2	Successful	Register
User2	3	Not Available	Register
…			Register

Table 9: Acceptance Test Register Sensors

The test was successfully passed. The service allows to register the SSH with valid username and sensor ID and responses with a positive feedback. Non valid username and sensor ID combinations result in a negative feedback.

5.3. Data Visualisation

Different websites for visualisation of sensor data were implemented. The websites are totally independent from the Node.js server and can be deployed on other servers. To evaluate the independence, a Raspberry Pi was configured as a server and several websites were hosted on it. **Figure 8** shows the setup.

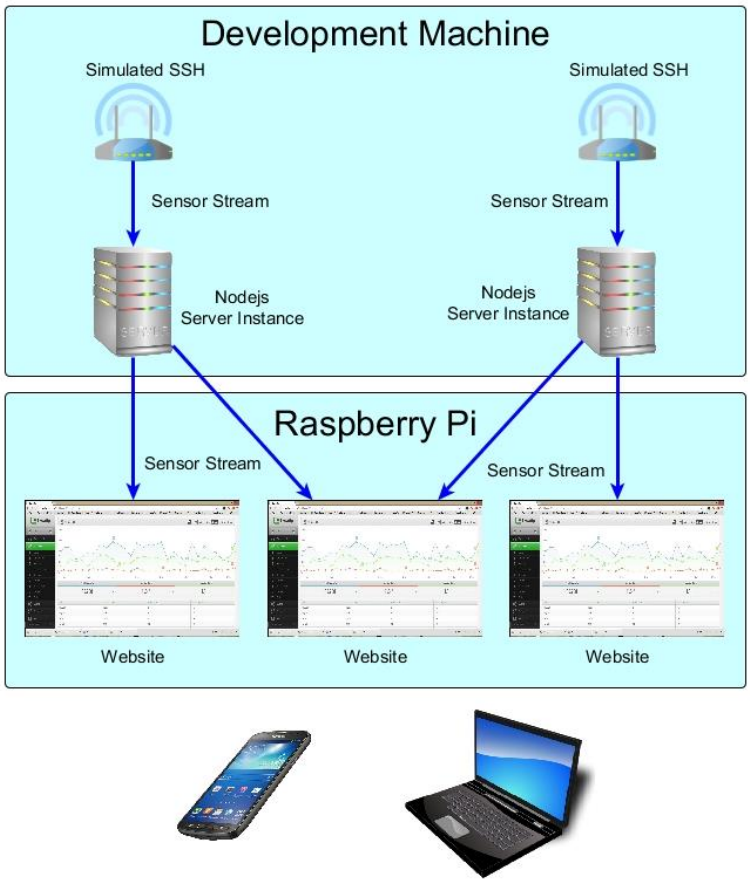

Figure 8: Setup

The websites can be visited from the local network by a smartphone or laptop. Simultaneous streaming from multiple SSH was simulated with the help of the console applications, which was introduced in the previous chapter. The sensor simulation application sends a Json string which appears on the website as a text. It is possible to receive multiple streams on one website and also different streams on different websites.

The Java Script code for receiving the sensor data string on the website is very simple, like shown below.

```
1.  var socket = io.connect('http://NodejsAdress:port/', {secure: true});
2.  socket.on('notification', function (data) {
3.      var object = jQuery.parseJSON( data.message );
4.      $("#ObjectId").html(object.attribute);
5.  });
```

Source Code 2: Receiving Data on Website

A WebSocket connection is opened, which receives the data string and de-serialize it to an object. Afterwards all object attributes can be used inside the HTML website. If more than one different SSH sources need to be combined on one website, the code needs to be copied for each source and the address and port needs to be adjusted.

The sensor data was injected into a graph to prove that the data can be used. The open source library Smoothie Charts (Walnes & Noakes, 2014) was used for visualization. Random single values where streamed over the Node.js server to the website and visualized in real time. An example is shown in **Figure 9**. Incoming data is appended at the right side.

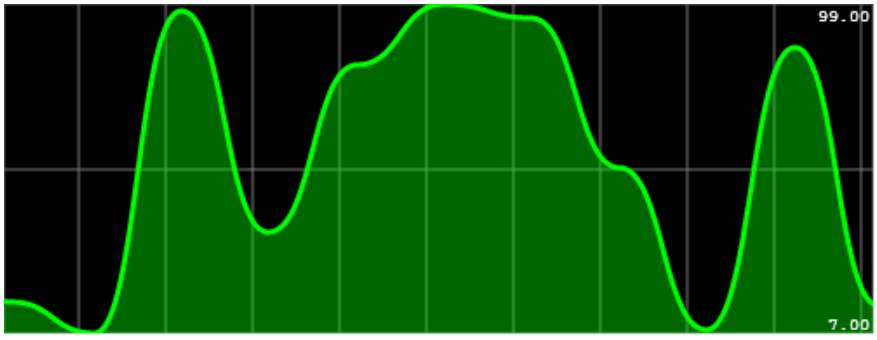

Figure 9: Smoothie Chart

In a second step a 720 Hz ECG signal was read by the application out of a file and send in sets of 720 values each second to the website. The visualisation of the ECG is not equivalent with the real ECG graph because it was not possible to serve the chart library with the right timestamp format. The ECG signal should look like illustrated in **Figure 10**.

Figure 10: ECG Signal (Association for the Advancement of Medical Instrumentation, 2002)

The streamed ECG signal is visualized like shown in **Figure 11**.

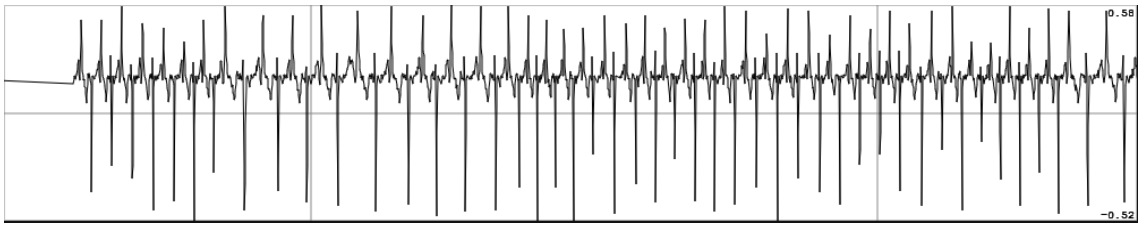

Figure 11: ECG Visualisation

It is noticeable that the pattern of the y-values seems right. The x-axis is compressed. It could be surly possible to visualize the ECG properly with more programming effort and another visualisation library. In this case the data consists of a very large array containing 720 objects filled with key value pairs. The visualisation of such an amount of values costs a lot of Central Processing Unit (CPU) load on client side due to the parsing of a long array. This may cause heating up the CPU and turning on the CPU van, which can be annoying for the user. Adding the data directly to the object, thus not in an array, may lower the CPU load.

These tests prove that sensing data can be used in real time on a website. This meets the requirements ***v. Frontend Development***. The user needs only frontend development skills to create real time applications. The tests show also that the data can be injected and used in Java Script applications which meet the requirement ***vi. Visualising Data***.

5.4. Database adaptation

The adaptation of a database was not tested because the possibility is proved by the previous chapter. It works on the same way than the visualisation process. To adapt a database, the website needs a server backend like ASP.NET Web API or another programming language. The data will be received on the website and injected in a Java Script code. The script sends the data to the server backend which stores the data into a database. Afterwards it can be visualized in the browser. For this reason, it fulfils the requirement of ***vii. Database Adaptation***.

5.5. Security Setup and Issues

Three security setup possibilities were planned, Transmission of raw data, Transport Layer Security and a complete encryption. Two versions, one transmitting raw data and another with TLS were implemented and tested. A total encryption would influence mostly the websites by more processing effort for decryption and was not considered at this stage. A big problem when using TLS with untrusted self-signed certificates is that a confirmation is needed each time a connection is created, which slowed down the experiment process. It is also a problem for mobile devices which cannot confirm an untrusted certificate. The solution would be to buy trusted certificates by a certificate authority.

A security issue was identified. The Node.js instance allows to transmit strings to a website not only from a SSH, a computer can be connected to the Node.js instance as well. This allows sending sensor values, messages, but also scripts to client websites. An offender could create a harmless looking website and prepare his computer sending messages to the website. There would be no suspect code on the page and the antivirus scanner would not find anything suspicious. The offender could then send Java Script to the website. The script would be executed on each connected client.

Web browsers keep Java Script code in a sandbox for security reason, which means the code has no access to the file system. But it can be used to gather information

about the client or to pop up messages which are mostly annoying and hard to close. The worst thing could be to run a script which steals session cookies from other browsers. The offender can load the stolen cookies in a browser. If a cookie is from an active session, the website identifies the offender as another user. The offender has now the possibility to access and hijack accounts, like Facebook, email account or also banking accounts. A deeper analysis of this issue is beyond the scope of this work. It requires a larger focus on hacking techniques.

The tests meet the requirements of ***viii. Security*** and ***ix. Privacy***. Different security configurations were implemented and tested. Potential threats were identified and analysed. The sensor publisher cannot be traced back because only the Node.js instance address is visible.

5.6. Server Instance and Limits

This chapter investigates the performance of the system. Three different applications were implemented to test the system by monitoring the CPU load and the available Random-Access Memory (RAM) in different operational capabilities. The first experiment will analyse the instantiation process. The second experiment will concentrate on data streaming speed and the third will focus on data irregularity during the streaming process.

The CPU load and RAM usage was recorded directly in the application by using the Windows performance counter build in function. It uses the same functions like the Windows Task Manager Performance Counter. After sending a data package, the interval stopwatch will start and before the next package will be send the performance values will be recorded. This gives the performance counter some time to actualize the performance values.

All tests were carried out with an Intel(R) Core(TM) i5-3317U CPU @ 1.70GHz 4 GB RAM with Windows 8.1. These experiments serve the requirement ***xi. System Limits*** and will identify the system limits and operational capability of the system.

5.6.1. Instantiation Limits

The following experiments will identify the maximum amount of instances which can be created on one machine, how fast instances can be created and the influence on CPU load and RAM. The aim is to define specifications for a stable running system.

A console application was developed to test the instantiation process. The application runs an automated test. The user need to enter two values, the amount of instances to create and the initiation interval in which each instance will be created. The application creates instances in the given interval and logs the machine CPU load and the available machine RAM after an instance is created. If the application created all instance, it closes them and saves the logged values in a spread sheet.

Various series were carried out with different number of instances and in different intervals. The results are listed in **Table 10**.

		RAM usage [MB]				Total Average RAM usage [MB]				CPU Load [%]			
Instances	Interval [ms]	Average	St.Deviation σ	σ max	σ min	Total Average	St.Deviatio	σ max	σ min	Average	St.Deviation σ	σ max	σ min
100	50	8.09	1.15	9.24	6.94					99.14	6.01	105.2	93.13
100	200	16.47	1.4	17.87	15.07					90.01	10.56	100.6	79.45
100	500	16.8	1.15	17.95	15.65					27.87	7.87	35.74	20
100	1000	16.09	1.34	17.43	14.75	16.45	1.30	17.75	15.16	12.14	6.21	18.35	5.93
200	50	7.05	0.45	7.50	6.60					99.58	4.37	104	95.21
200	200	15.37	2.05	17.42	13.32					91.2	8.91	100.1	82.29
200	500	15.2	2.55	17.75	12.65					31.56	9.9	41.46	21.66
200	1000	14.82	2.22	17.04	12.60	15.13	2.27	17.40	12.86	16.58	9.71	26.29	6.87
300	50	11.11	2.56	13.67	8.55					99.75	3.29	103	96.46
300	200	9.22	4.09	13.31	5.13					99.97	0.37	100.3	99.6
300	500	12.51	4.07	16.58	8.44					30.16	9.25	39.41	20.91
300	1000	11.6	3.8	15.40	7.80	11.11	3.99	15.10	7.12	18.84	10.4	29.24	8.44
350	50	4.22	0.59	4.81	3.63					99.96	0.7	100.7	99.26
350	200	11.46	3.7	15.16	7.76					99.89	1.51	101.4	98.38
350	500	11.99	4.17	16.16	7.82					29.69	8.37	38.06	21.32
350	1000	11.72	3.98	15.70	7.74	11.72	3.95	15.67	7.77	25.78	11.22	37	14.56
400	50	5.65	0.69	6.34	4.96					99.78	3.51	103.3	96.27
400	200	9.11	4.33	13.44	4.78					96.11	6.75	102.9	89.36
400	500	10.29	4.52	14.81	5.77					31.38	9.65	41.03	21.73
400	1000	10.88	4.34	15.22	6.54	10.09	4.40	14.49	5.70	22.5	11	33.5	11.5

Table 10: Instances Experiment

The experiment was carried out in series of 100, 200, 300, 350 and 400 maximal numbers of instances. For each step the test was run in initiation intervals of 50, 200, 500 and 1000 milliseconds. Each test run produced a table with CPU load and RAM. The raw data can be found in the appendix (CD). **Table 10** list the average CPU load

in percent and used RAM in Mega Byte with the standard deviation during each test run. This allows a statement about the machine state and stability. As lower the CPU load and deviation is, as more stable is the system. The average used RAM for an instance was calculated in each series.

The series with an interval of 50 milliseconds was not considered in the results since it was found that there is a measurement inaccuracy. The RAM value is actualising to slow, so that wrong values were logged. This was monitored by observing the Windows Task Manager Performance tool.

Figure 12 shows the results for the average RAM usage of each instance. The green and red lines are the maximum and minimum standard deviation.

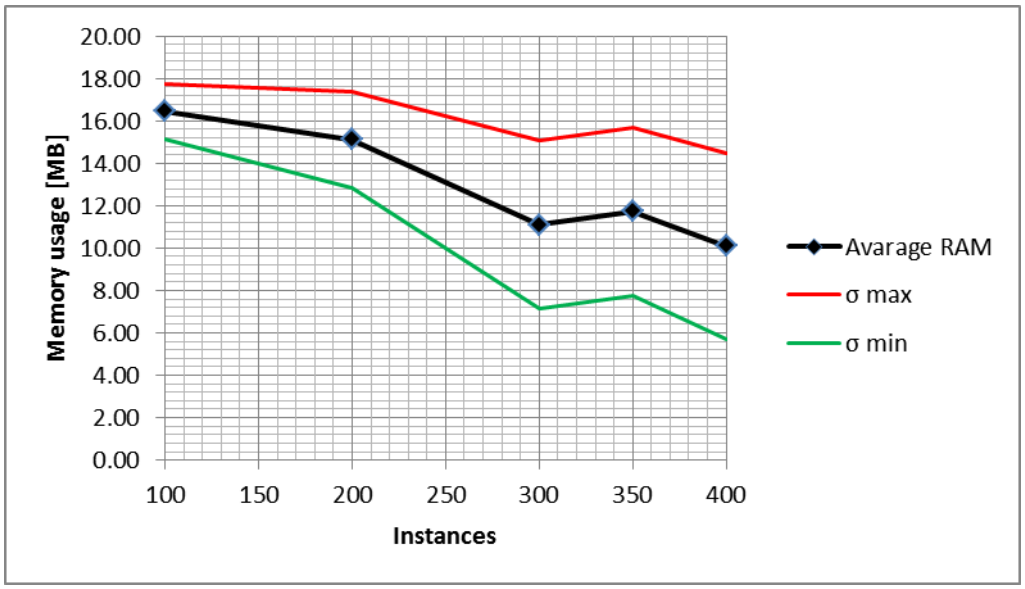

Figure 12: Instances Experiment - Results Memory

The experiment showed that one instance uses approximately 16 MB of RAM at the beginning and that the instances getting smaller as closer the system memory gets to a usage of 100% which was reached on the tested machine after approximately 350 instances. For verification the Windows Task Manager was monitored and it showed that one part of the instances is still using 16 MB but others are using less than 2 MB and some only 0.3 MB, which explains the rising spread of the standard deviation. One of the small instances was tested for function and it worked but expands to a

RAM usage of 15 MB. It suggests that a working instance will need a minimum of 15 MB to work probably. The maximum amount of instances is limited by the available RAM. Not used instances can get smaller to make space for instances in use. The CPU load is only affected during the initiation process but not when the instances are running.

Figure 13 to **Figure 17** show the average CPU load for different initiation intervals for each test series. The green and red lines are the standard deviation. A high deviation is an indicator for a high fluctuation of the CPU load. The aim is to keep the CPU load and the deviation as small as possible.

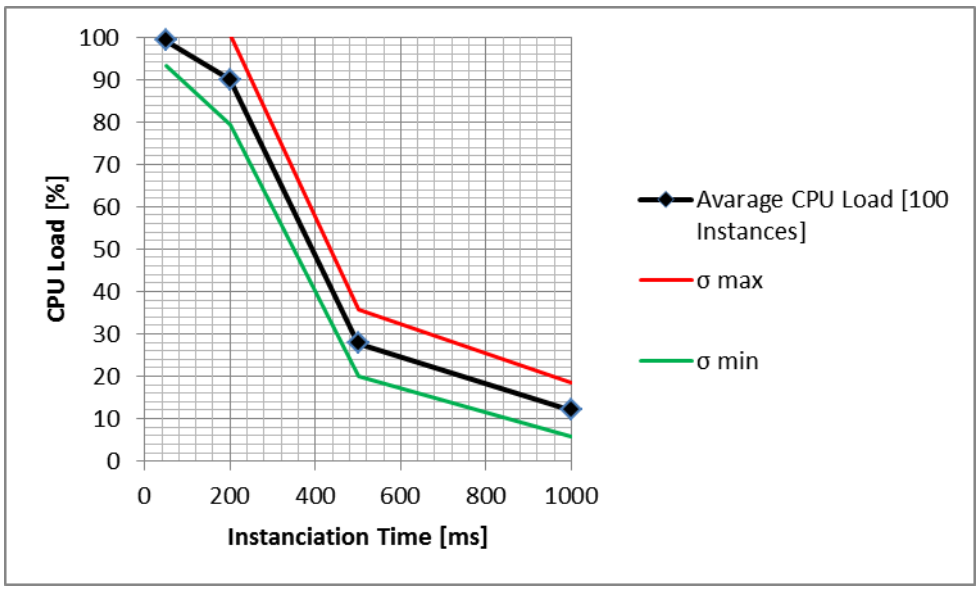

Figure 13: Instances Experiment - Results CPU 1

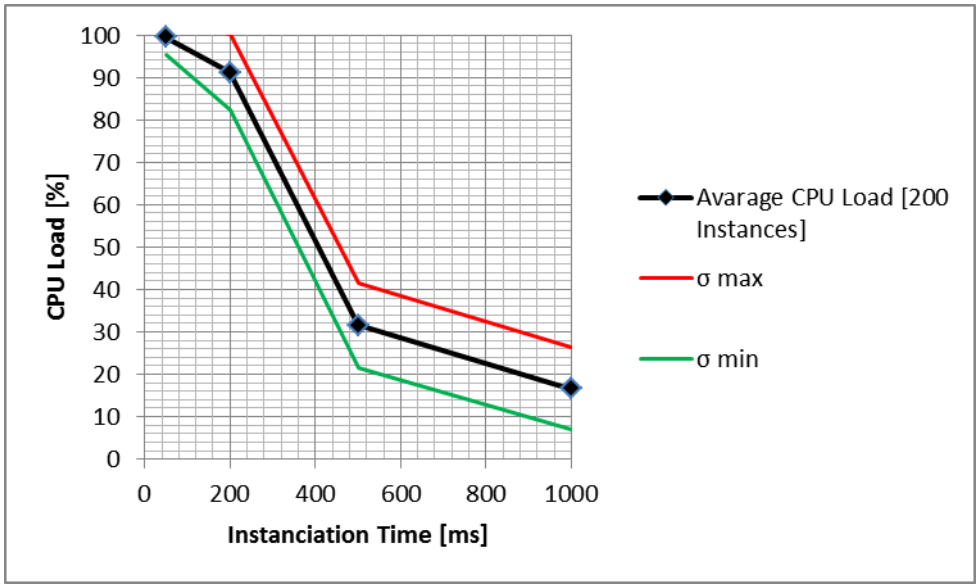

Figure 14: Instances Experiment - Results CPU 2

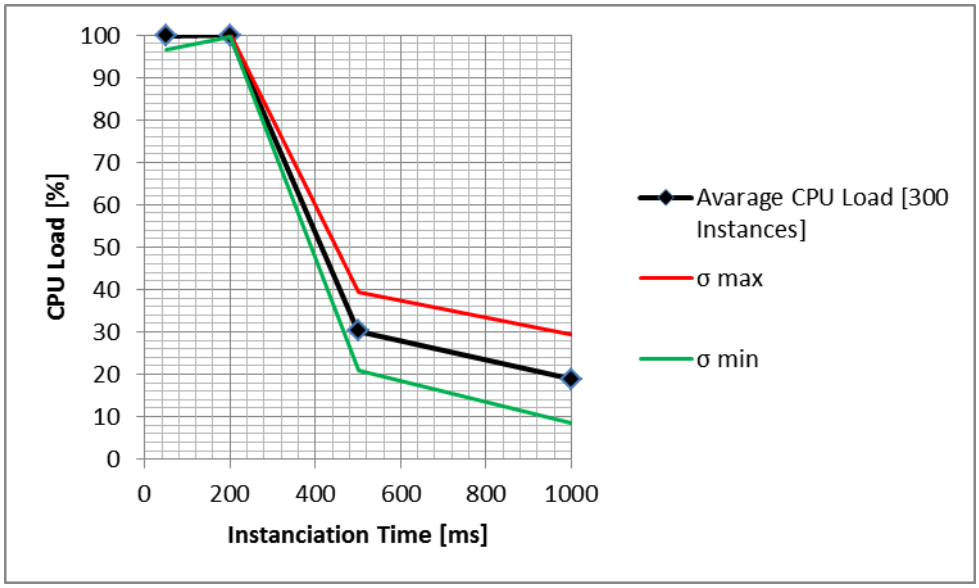

Figure 15: Instances Experiment - Results CPU 3

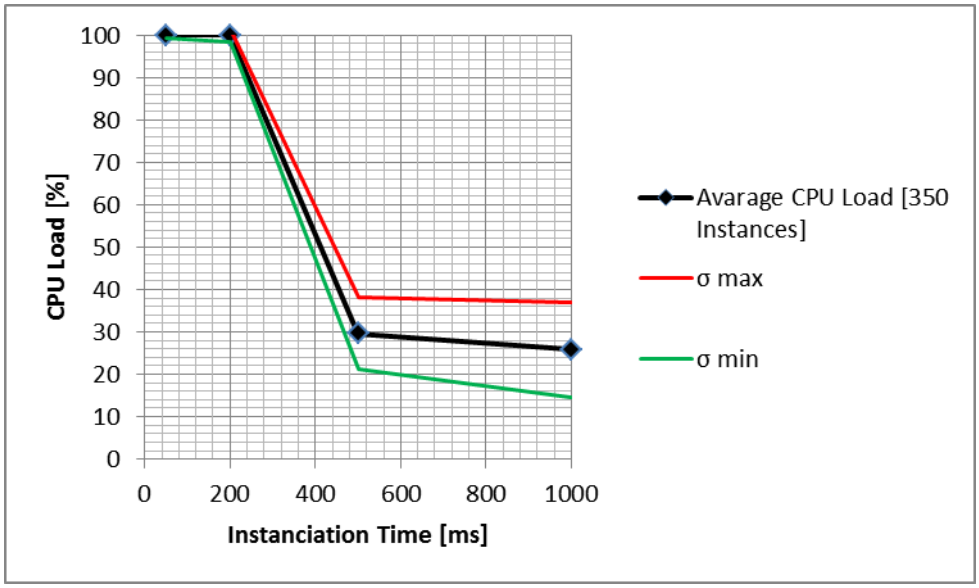

Figure 16: Instances Experiment - Results CPU 4

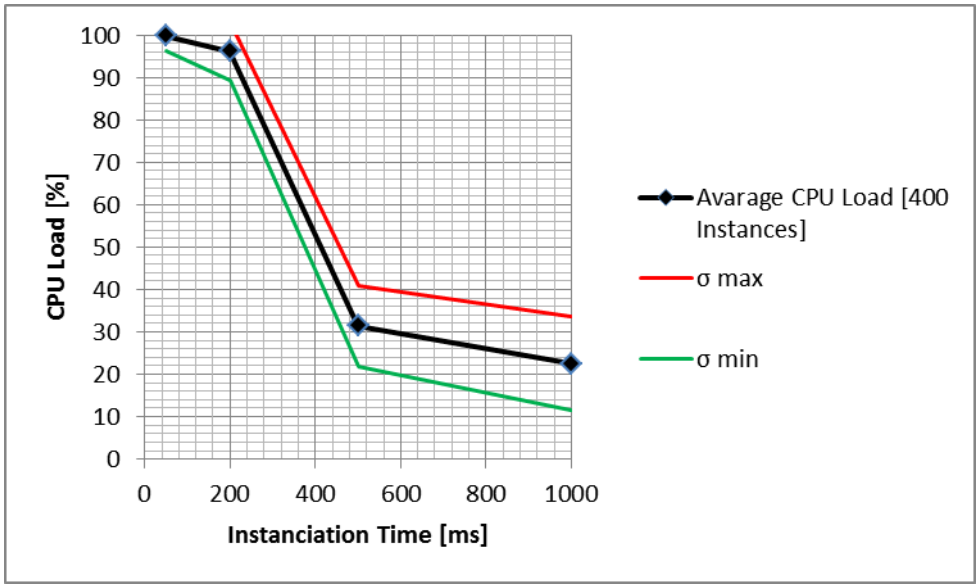

Figure 17: Instances Experiment - Results CPU 5

The results show that an initiation interval faster than 500 milliseconds increase the CPU load drastically and an interval faster than 200 milliseconds is not practicable. The similar average CPU load between the series in the range from 500 to 1000 milliseconds proves that the amount of running instances does not affect the CPU load.

To avoid high CPU load the system needs an instantiation load balancer which creates new instances in an interval of more than 500 milliseconds. The amount of instances needs to be limited and this is also the bottleneck of the system. The maximal amount depends on the RAM of the server. Each instance needs approximately 16 MB of RAM.

5.6.2. Streaming Limits

The following experiments will identify the impact of different streaming speed and data size on the CPU load and RAM usage. The target is to find specify limits and test the operational capability of the system.

As in 5.6.1., a console application was developed logging CPU load and RAM usage. The console application starts a server instance and start to stream data to the server. The transmission interval, amount of values to stream and the duration of the stream can be configured at the beginning. After the application finishes it saves the results in a spread sheet.

The experiment was accomplished in various series, started with 10.000, 1.000, 100 and 10 values streamed in one interval. Also the interval speed varied from 500, 200, 50, down to 10 milliseconds. The duration of the stream was set to 10 seconds. **Table 11** shows the results of the experiment.

Package Size	Choosen Rate [ms]	Streamed Time [s]	CPU Load [%]			
			Average	St.Deviation σ	σ max	σ min
10000	500	10	35	17.3	52.3	17.7
10000	200	10	34.46	10.4	44.86	24.06
10000	50	10	44.13	13.41	57.54	30.72
10000	10	10	50.06	15.06	65.12	35
1000	500	10	15.11	11.94	27.05	3.17
1000	200	10	20.48	13.74	34.22	6.74
1000	50	10	22.6	8.92	31.52	13.68
1000	10	10	44.31	13.81	58.12	30.5
100	500	10	2.55	1.76	4.31	0.79
100	200	10	5.17	4.43	9.6	0.74
100	50	10	24.45	20.06	44.51	4.39
100	10	10	42.98	26.24	69.22	16.74
10	500	10	1.8	1.67	3.47	0.13
10	200	10	2.29	2.81	5.1	-0.52
10	50	10	6.88	7.22	14.1	-0.34
10	10	10	23.69	35.03	58.72	-11.34

Table 11: Transmission Speed Experiment

The average CPU load was calculated and also the standard deviation. It has been found that streaming data has no effect on RAM usage. The RAM usage was not considered caused by the constant level. The raw data can be found on the CD.

Figure 18 to **Figure 21** shows the average CPU load in relation to the update rate.

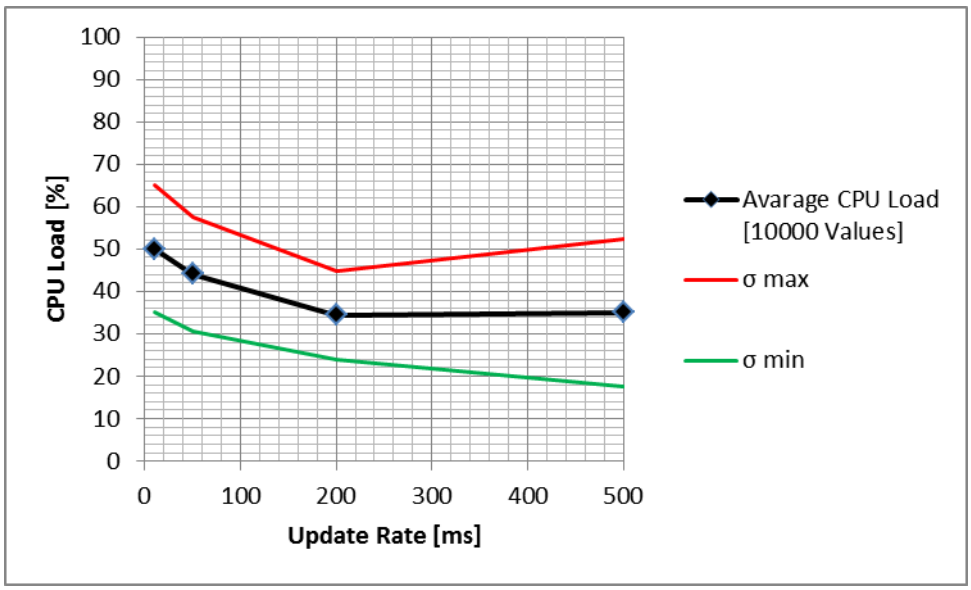

Figure 18: Transmission Speed Experiment - Results CPU 1

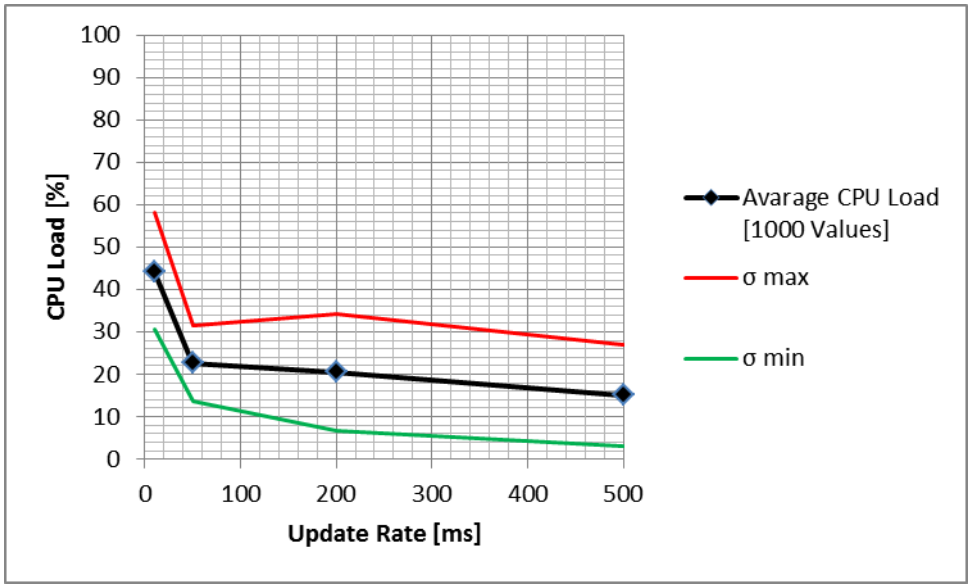

Figure 19: Transmission Speed Experiment - Results CPU 2

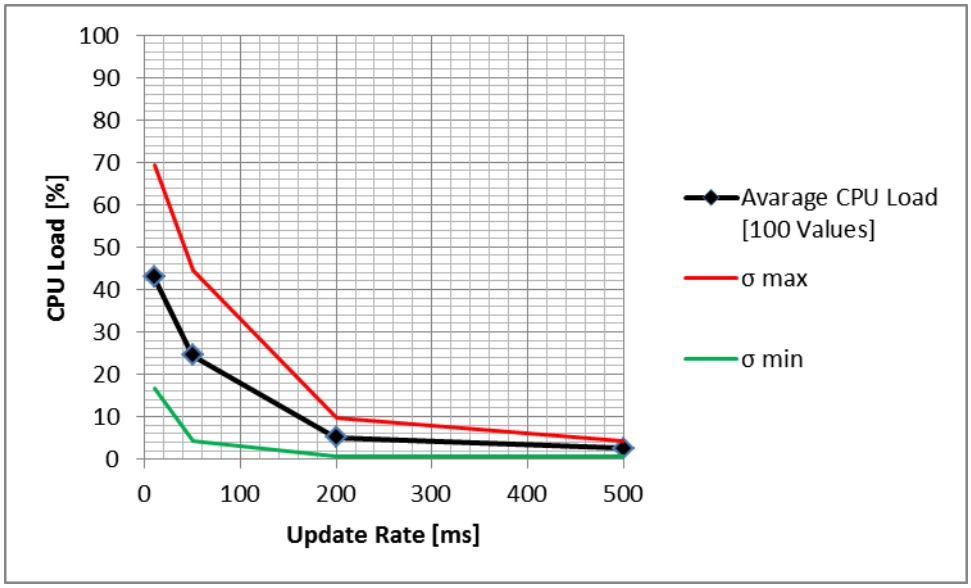

Figure 20: Transmission Speed Experiment - Results CPU 3

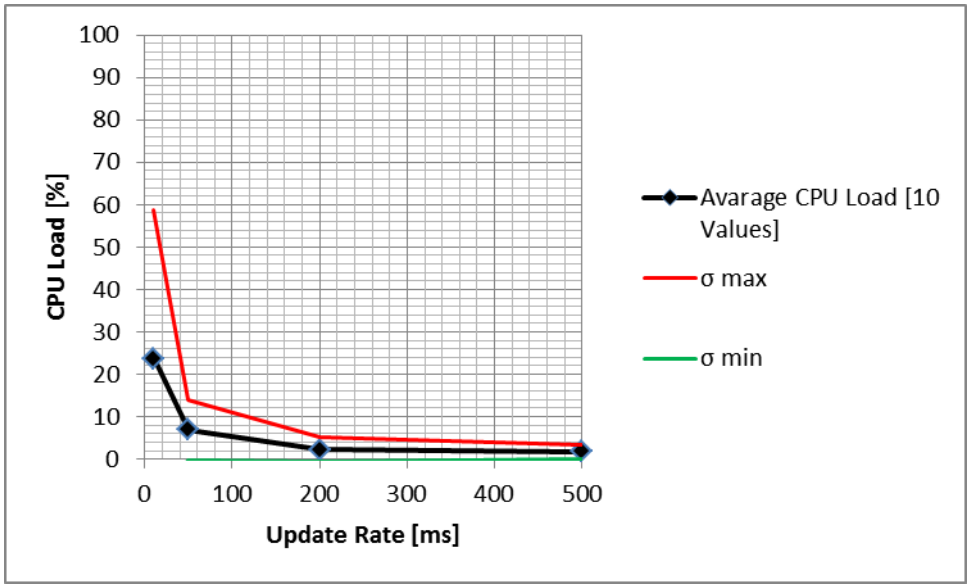

Figure 21: Transmission Speed Experiment - Results CPU 4

It should be noted that the results concern only one instance on the server. For that reason it is very important to keep the CPU load as low as possible. **Figure 18** shows the CPU load vacillates between 20 and 50 percent. It suggests that streaming a set of 10.000 values at once is not useful. Streaming a set of fewer than 100 values with a maximum update rate of 200 milliseconds is feasible. As well as streaming a set fewer than 10 values with a maximum update rate of 50 milliseconds. The maximum amount of values which can be streamed with the used setup is approximately 22.000. This is due to the limited UDP package size of 65 kb.

5.6.3. Packet Loss

The experiment focuses on the accuracy and lost packages by streaming data over a UDP connection.

A console application was developed with the ability to log CPU load and RAM usage. The application creates a server instance and stream data to the server over UDP. The update rate and amount of values to stream can be configured. It also logs the sent messages and stores each message in a file. The Node.js server instance is modified and stores each received message in another file as well. Both files are compared to each other at the end of the experiment to analyse data irregularities

caused by the UDP. KDiff3 is a tool based on the LINUX command “diff” and was used to calculate differences between the two files and visualise it like illustrated in **Figure 22**.

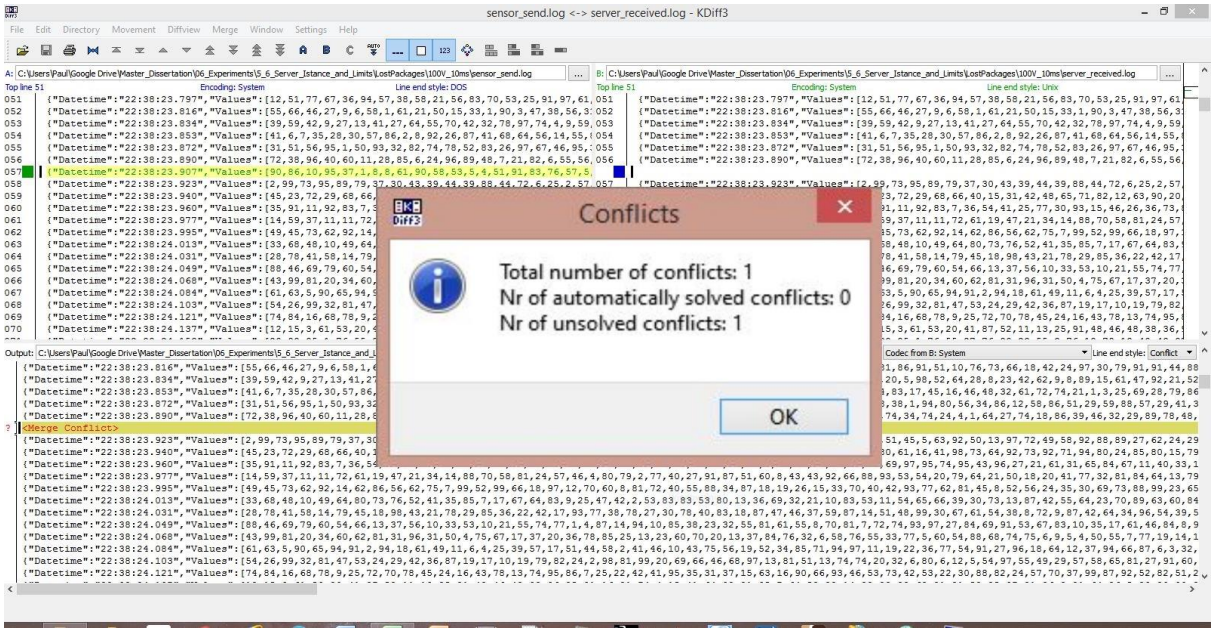

Figure 22: KDiff3 - Count Irregularities

The experiment was conducted in series, like in 5.6.1 and 5.6.2 but in shorter intervals of 50, 25, 5 and 1 milliseconds. The short interval was chosen to raise the risk to produce data irregularities. The mount of values streamed was set to 100, 10 and 1 value. The duration of the stream was set to 10 seconds. The set update rate varies from the real update rate caused by the overhead of the calculation and creation of mock data, the latency and the process of writing in files. The record of the send and received packages make it possible to calculate the true update rate and the overhead by comparing the amount of transmitted packages with the expected amount of received packages. **Table 12** shows the results.

Package Size [Values]	Choosen Rate [ms]	Streamed Time [s]	Effective Transmitt	True Rate	Overhead [ms]	Irregularities	Accuracy [%]	CPU Load [%]			
								Averag	St.Deviatio	σ max	σ min
100	50	10	171	58.48	8.48	0	100.00	34.99	16.14	51.13	18.85
100	25	10	306	32.68	7.68	0	100.00	40.41	19.33	59.74	21.08
100	10	10	560	17.86	7.86	1	99.82	70.63	20.2	90.83	50.43
100	5	10	777	12.87	7.87	1	99.87	80.2	23.48	103.68	56.72
100	1	10	1091	9.17	8.17	1	99.91	90.17	21.2	111.37	68.97
10	50	10	184	54.35	4.35	0	100.00	15.61	10.24	25.85	5.37
10	25	10	345	28.99	3.99	0	100.00	28.96	20.45	49.41	8.51
10	10	10	720	13.89	3.89	0	100.00	54	34.03	88.03	19.97
10	5	10	1099	9.10	4.10	0	100.00	72.86	40.06	112.92	32.8
10	1	10	1992	5.02	4.02	3	99.85	88.47	29.92	118.39	58.55
1	50	10	188	53.19	3.19	0	100.00	13.51	9.85	23.36	3.66
1	25	10	355	28.17	3.17	0	100.00	24.58	20.44	45.02	4.14
1	10	10	755	13.25	3.25	0	100.00	47.57	34.95	82.52	12.62
1	5	10	1225	8.16	3.16	0	100.00	65.27	44.58	109.85	20.69
1	1	10	2431	4.11	3.11	6	99.75	88.14	32.05	120.19	56.09

Table 12: Lost Packages Experiment

The calculation overhead takes longer for larger packages like expected, for 100 values approximately 8 milliseconds and for a single value approximately 3.2 milliseconds. Unexpectedly, irregularities in the data were very low, also with very fast update rates of 8 milliseconds. There are a larger amount of irregularities with an update rate of 4.11 milliseconds but in total the accuracy is still higher than 99.75%. The irregularities occur at high CPU load from nearly 100% which is caused by the stream itself but also by calculation of the mock data and writing the data transfer into files.

Figure 23 to **Figure 26** illustrates the average CPU load for different update rates and amount of streamed values. The green and red lines are the standard deviation.

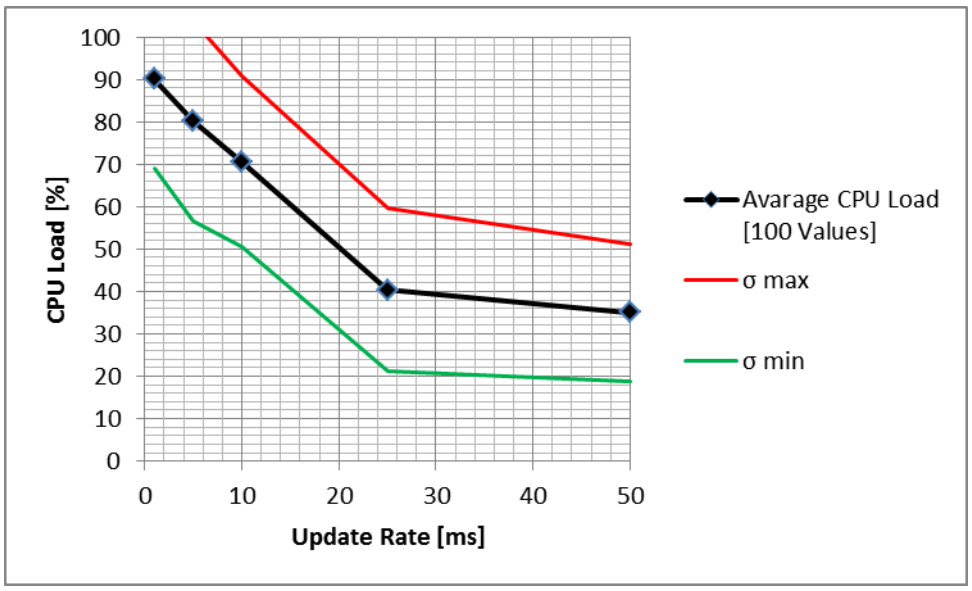

Figure 23: Lost Packages Experiment - Results CPU 1

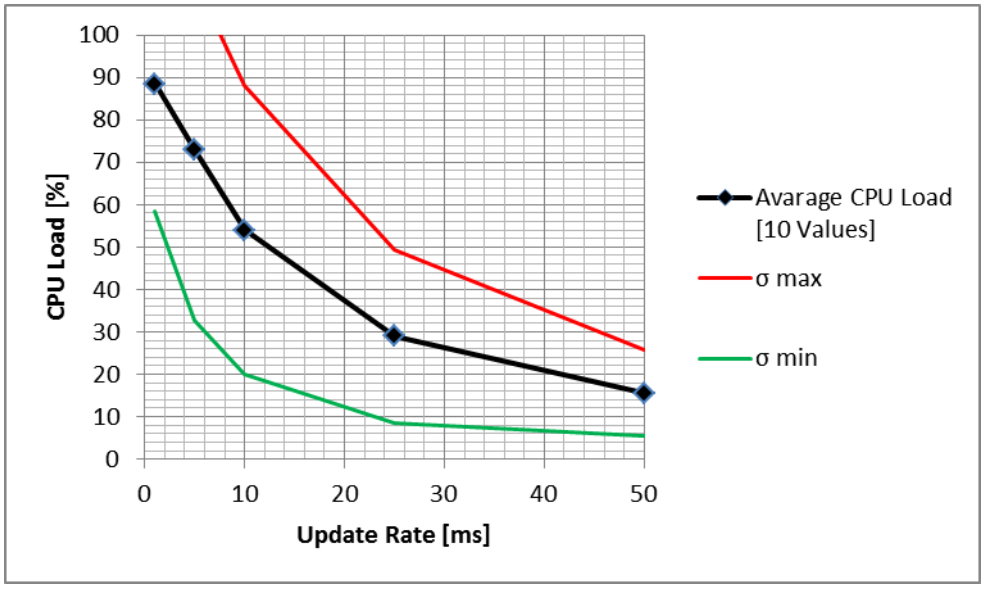

Figure 24: Lost Packages Experiment - Results CPU 2

Comparing **Figure 24** with **Figure 21** of previous experiment (5.6.2.) shows the high influence of logging the data transfer on the CPU load. 15.61% CPU load by an update rate of 50 milliseconds with data logging against 6.88% without data logging.

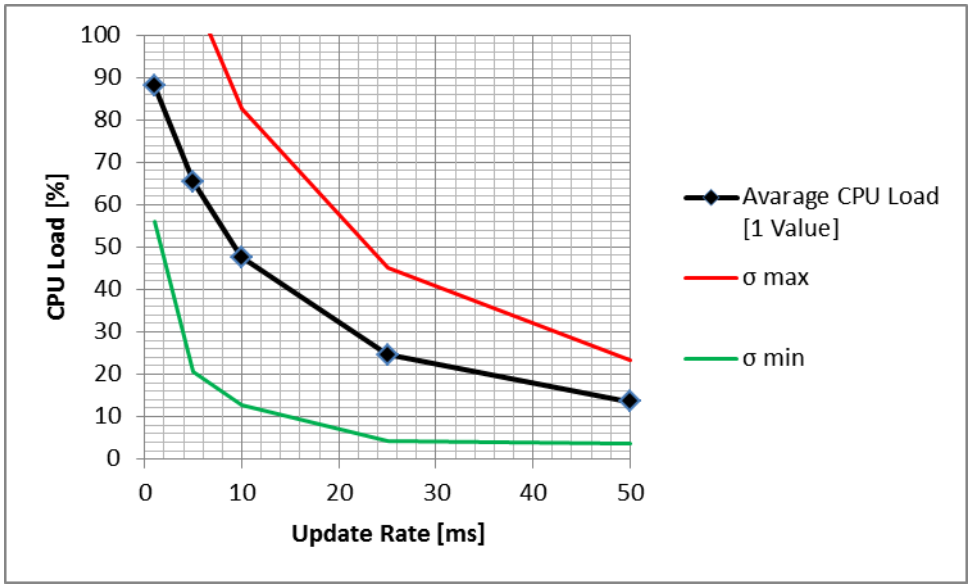

Figure 25: Lost Packages Experiment - Results CPU 3

The experiment shows that it is possible to send data in a high update rate of 25 to 50 milliseconds and with high background processing with an accuracy of 100% with an UDP connection between SSH and server. A visual update rates in the area of a smooth visualisation of 62.5 milliseconds, like mentioned in section 3.1., and reasonable CPU load, can be reached by streaming less than 10 sensor values at once.

6. Evaluation

This chapter is an evaluation of the project. The project management, scope, experiments and the results will be summarised and evaluated. The project will be analysed and compared to the requirements and the work of others.

The project plan has been followed and all five milestones are reached in time. However, some parts of the work could not be carried out in as much detail as wished at the beginning of the project. On the one hand the literature review took more time than planned due to the interdisciplinary subject with a large amount of papers to review. On the other hand there was a shift of the focus from frontend technologies to server technology due to the finding that server technology is of more interest for the feasibility of the project.

The registration service as an interface between SSH and user management system was implemented and tested. Acceptance tests showed that the manageability of the system is not an issue. GUIs and management web service could not be implemented due to the short time of the project. However, it was not necessary for proving the operational capability and it can be easily extended with standard web development technologies like ASP.NET.

An investigation of various frontend technologies could not be carried out in the planned scope, like mentioned before, caused by questions on the performance of the broadcasting server which arose during the development. It seemed more important to analyse the server first and investigate frontend technologies after setting up an evaluated infrastructure. Nevertheless, the functional capabilities to visualise streamed data in real time in a browser, combining different streams and injecting the streamed data in java script applications was proved.

The security of the system was analysed in two ways. The first was the general security of the system during data transfer. To do this, three versions of the server were designed. One transmitting raw data, a server with TLS secured connection and a complete encrypted data transfer, however, the first two versions could be

implemented. Full encryption was not useful at this stage of development and would require some extra time which was beyond the scope of this project. The second direction was an investigation of the abuse of the system for malicious purposes. It was identified that the system could be abused by offenders to attack online users by creating fake websites and get access to user information and possibilities to hijack user accounts. It is worth investigating this security issue in more detail in future.

Different experiments were conducted to determine the system limits. Effects on CPU load and RAM usage by the system were identified. First, the amount of instances running on one physical server is limited by the available RAM. Over 300 server instances can be started on a machine with only 4 GB of RAM. Second, the CPU load is affected by the instantiation speed. This needs to be controlled by a load balance system. Third, streaming data has no effect on the RAM usage of the physical server and also high update rates of 25 to 50 milliseconds create a pleasant CPU load and would allow smooth visualisation. And fourth, the high data accuracy of 99.75% with an interval faster than 5 milliseconds and 100% with 25 milliseconds in spite of using UDP connection is a good result as well. The definition of these limits gives a feeling for the system and allows scaling the system up in future. Due to the short time and resources, it was not possible to run the test in a reality close environment by separation of SSH simulation and server on different physical machines. This would make detailed tests of network latency possible. It would be of interest as well to compare the results by running the experiments on machines with different performance and compare the results to make statements about benefits of special processors or other hardware.

In total the main objectives were reached. One objective was to keep the system as universal as possible. For that reason different use cases and application areas were described and analysed by focusing on their priorities and requirements. The requirements of a similar system were analysed (J. Zhang et al., 2013). Based on this analysis, requirements for the system were defined and listed which serve the different application areas and priorities. Parts of the system were implemented and tested to prove the feasibility of the requirements.

All implementations and experiments address and confirm the requirements like mentioned in chapter 5, such as ***i. Scalability*** and ***ii. Visualisation update rate*** are addressed by the possibility to vary the visualisation update rate for different application areas and the possibility to transmit any kind of text based data. The implemented registration service addresses the requirements ***iii. Configurability*** and ***iv. Sensor Data Sharing***. It is possible to add sensors through a GUI, the configuration of the SSH is reduced to a minimum effort and can be easily extended with a GUI interface. It is easily possible to extend the service by a web portal to allow sensor data sharing. Tests proved that sensing data can be used in real time on a website by little programming effort. This meets the requirements ***v. Frontend Development***. The user needs only frontend development skills to create real time applications. The tests in section 5.3 show that the data can be injected and used in Java Script applications which meet the requirement ***vi. Visualising Data*** as well as the requirement ***vii. Database Adaptation*** by redirecting the data over a Java Script code to the server backend. Different security configurations were implemented and tested and potential threats were identified and analysed. The tests met the requirements of ***viii. Security*** and ***ix. Privacy***. The investigation of the system limits and operational capability of the system in section 5.6 serve the requirement ***xi. System Limits***. Only the ***x. Reliability*** could not be analysed. It would require a working prototype and a mobile application which could not be provided in the scope of the project.

The system shares the idea of streaming sensing data with other platforms. The SensibleThings platform from Forsström has a P2P architecture compared to the data centric architecture of proposed system. Forsströms system appeals to more coding experienced users and mobile application developers (Forsström et al., 2014). The proposed system can be used by less coding skilled users and it is focused on high configurability over simple GUI.

Zhang presented a service oriented platform with the aim to maximize the sharing and utility of available sensor data sources (J. Zhang et al., 2013). One difference is that the proposed system does not use databases for storing sensing data in the

architecture. The requirements of both systems are mostly the same with the difference that proposed system has a larger focus on configurability and GUIs for manageability by the user. Zhangs system transfers as well as the proposed system a Json string with multiple sensor data. However Zhangs system concentrates in big data analytics and has no focus on high update rates. It actualizes the data in intervals of a few seconds.

Many systems concentrate mostly on one application area. E.g. monitoring drought (Deng et al., 2013), agriculture (Cai et al., 2011), landslides (Teja et al., 2014) or floods (Y. Zhang, Li, Li, & Guo, 2009). These systems send their data at intervals to their base stations or to a centralized computer where the data will be stored for long term research. A GPS tracking system was introduced by El-Medany using ASP.NET and TCP/IP sockets. The system sends data in intervals of one minute. The proposed system focuses on universal usability in many application areas. It is not designed for one specific area and can be used or consumed by users.

In total the project is a success. It proved the feasibility of the architecture and forms a basis which offers some points for further research in the area of detailing the experiments, evaluation of visualisation technologies and analysis of security issues. The experiments can be repeated on other machines to get feedback about the influence of different hardware, e.g. different amount of CPUs. In any case the experiments should be conducted in a more production-like environment. The streaming server, website and SSH need to be separated on different physical machines to investigate the latency of the system. The visualisation part can be done in more detail. The amount of different visualisation libraries is enormous. An evaluation of different libraries with focus on low processing on client side would be of interest. The identified security issue, hijacking of accounts by steeling session cookies, should be investigated by detailed experiments because it is of general interest if Node.js can be easily abused for cybercrime.

The present work fits into present research of the WoT and delivers a system concept which could rival with commercial systems like TempoDB, Xively and Sensor Cloud in future.

7. Conclusion and Future Work

The present work contributes to the infrastructure of the internet by providing an architecture concept for sensor data streaming, processing and live visualisation of streamed data over the web.

The idea of a sensor streaming system belongs to the area of the Web of Things. Similar systems exist in the commercial sector nevertheless they all have different strengths and targeting different application areas. As well as open source systems in the area of research with different architectures and different degree of difficulty to deploy own sensors. The present work delivers a universal system trying to combine different application areas and streaming tasks in one easy manageable system.

The system requirements were defined by creating use cases in different application areas. An identification of their priorities and an investigation of requirements of similar systems results in a list of system requirements with a large focus on universality and manageability. The system requirements were used to prove the architecture concept by implementing critical parts of the system and carrying out experiments.

The architecture was designed and discussed. The starting point is a Sensor Streaming Hardware (SSH) and the endpoint a website. The architecture defines not only the infrastructure to transfer the data between SSH and website but as well ensures an easy sensor deployment, manageability of the sensors and security. The architecture consists of various elements and servers. A management service allows the user to create an account and to configure the SSH online. The data is transferred to a second registration service which stores the data into a database. In the meanwhile the management server creates a virtual streaming server with the configured setup. The user needs only to access the SSH over the local network by an IP address and configure the username and sensor ID. After starting the SSH, the SSH will automatically connect to the registration service where the username and sensor ID will be validated. The SSH will get all configured data and the streaming

server address and connect to the streaming server instance. The data stream will start immediately with the configured setup. A monitoring service running on a separate physical machine is used to monitor the status of the virtual server instances and will handle exceptions.

As mentioned before, important parts of the system were implemented to evaluate the feasibility of the system. The registration service is an interface between the management system, the SSH and the database and was implemented and tested by acceptance tests. The streaming server was implemented with Node.js and Java Script in two versions, one with TLS and the other transferring raw data. The server was modified and instantiated by a C# application. This was the basis for the different experiments to define the system limits. Three different experiments were conducted focusing on effects of creating large amounts of virtual streaming server instances, data streaming of large packages and accuracy of getting the right data on the server through the UDP connection. Security issues were analysed and discussed. Especially sending and execution of malicious java script over the connection was identified as a possible abuse. Section 5.3 showed that sensing data can be easily injected and used in browser applications. The sensing data can be also processed by Java Script code on client side.

Like mentioned in chapter 6, further research can be done in the area of detailing the experiments by repeating them on other machines to get a feedback about the influence of different hardware and the behave in a more production-like environment. A more detailed evaluation of visualisation technologies can be done and the analysis of the identified security issues in section 5.5.

vi. References

American Heart Association. (2014). Target Heart Rates. Retrieved from http://www.heart.org/HEARTORG/GettingHealthy/PhysicalActivity/Target-Heart-Rates_UCM_434341_Article.jsp

Andersson, K., & Johansson, D. (2012). Mobile e-services using HTML5. *37th Annual IEEE Conference on Local Computer Networks -- Workshops*, 814–819. doi:10.1109/LCNW.2012.6424068

Association for the Advancement of Medical Instrumentation. (2002). Cardiac monitors, heart rate meters, and alarms [American National Standard (ANSI/AAMI EC13:2002)]. Arlington. Retrieved from http://www.physionet.org/physiobank/database/aami-ec13/

Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, *54*(15), 2787–2805. doi:10.1016/j.comnet.2010.05.010

Australia, C. of. (2014). Deep Ocean Tsunami Detection Buoys. Retrieved July 06, 2014, from http://www.bom.gov.au/tsunami/about/detection_buoys.shtml

Bendel, S., Springer, T., Schuster, D., Schill, A., Ackermann, R., & Ameling, M. (2013). A service infrastructure for the Internet of Things based on XMPP. *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, 385–388. doi:10.1109/PerComW.2013.6529522

Cai, K., Liang, X., & Wang, K. (2011). Development of Field Information Monitoring System Based on the Internet of Things *, 675–680.

Chadil, N., Russameesawang, A., & Keeratiwintakorn, P. (2008). Real-time tracking management system using GPS, GPRS and Google earth. *2008 5th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, 393–396. doi:10.1109/ECTICON.2008.4600454

Chen, B. (2011). A framework for browser-based Multiplayer Online Games using WebGL and WebSocket. *2011 International Conference on Multimedia Technology*, 471–474. doi:10.1109/ICMT.2011.6001673

Dahl, T., Koskela, T., Hickey, S., & Vatjus-Anttila, J. (2013). A Virtual World Web Client Utilizing an Entity-Component Model. *2013 Seventh International Conference on Next Generation Mobile Apps, Services and Technologies*, 7–12. doi:10.1109/NGMAST.2013.11

Deng, M., Di, L., Han, W., Yagci, A. L., Peng, C., & Heo, G. (2013). Web-service-based Monitoring and Analysis of Global Agricultural Drought, *22030*.

Duquennoy, S., Lifl, I., & Grimaud, G. (2009). The Web of Things : interconnecting devices with high usability and performance, *2009*.

El-Medany, W., Al-Omary, A., Al-Hakim, R., Al-Irhayim, S., & Nusaif, M. (2010). A Cost Effective Real-Time Tracking System Prototype Using Integrated GPS/GPRS Module. *2010 6th International Conference on Wireless and Mobile Communications*, 521–525. doi:10.1109/ICWMC.2010.104

Forsström, S., & Kanter, T. ubiquitous sensor-assisted applications on the internet-of-things. (2013). Enabling ubiquitous sensor-assisted applications on the internet-of-things. *Personal and Ubiquitous Computing*, *18*(4), 977–986. doi:10.1007/s00779-013-0712-9

Forsström, S., Kardeby, V., Österberg, P., & Jennehag, U. (2014). Challenges when Realizing a Fully Distributed Internet-of-Things – How we Created the SensibleThings Platform, (c), 13–18.

Goldberger, A. L., Amaral, L. A. N., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., … Stanley, H. E. (2000). {PhysioBank, PhysioToolkit, and PhysioNet}: Components of a New Research Resource for Complex Physiologic Signals. *Circulation*, *101*(23), e215–e220.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, *29*(7), 1645–1660. doi:10.1016/j.future.2013.01.010

Ha, S. W., Lee, Y. K., Vu, T. H. N., Jung, Y. J., & Ryu, K. H. (2012). An environmental monitoring system for managing spatiotemporal sensor data over sensor networks. *Sensors (Basel, Switzerland)*, *12*(4), 3997–4015. doi:10.3390/s120403997

IRIS. (2014). Incorporated Research Institutions For Seismology. Retrieved July 06, 2014, from http://www.iris.edu/

Jones, V., Gay, V., & Leijdekkers, P. (2010). Body sensor networks for mobile health monitoring: Experience in europe and australia. *Digital Society, 2010. ICDS'10. …*.

Joyent Inc. (2014). Nodejs. Retrieved August 01, 2014, from http://nodejs.org

LogMeIn Inc. (2014). Xively. Retrieved August 01, 2014, from https://xively.com/

Microsoft. (2014). ASP.NET. Retrieved July 06, 2014, from http://www.asp.net/

MicroStrain. (2014). SensorCloud. Retrieved August 01, 2014, from http://www.sensorcloud.com/

Neumeyer, D., & Brown, J. (2014). Audio-Visual Palimpsests: Resynchronizing Silent Films with "Special" Music. In *The Oxford Handbook of Film Music Studies* (p. 588). Oxford University.

Orduña, P., & Angulo, I. (2014). Graphic Technologies for Virtual , Remote and Hybrid laboratories : WebLab-FPGA hybrid lab, (February), 163–166.

Rao, A. S., Izadi, D., Tellis, R. F., Ekanayake, S. W., & Pathirana, P. N. (2009). Data monitoring sensor network for BigNet research Testbed. *2009 International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, 169–173. doi:10.1109/ISSNIP.2009.5416816

Sample, A. P., Braun, J., Parks, A., & Smith, J. R. (2011). Photovoltaic enhanced UHF RFID tag antennas for dual purpose energy harvesting. *2011 IEEE International Conference on RFID*, 146–153. doi:10.1109/RFID.2011.5764615

Shamszaman, Z. U., Ara, S. S., Chong, I., & Jeong, Y. K. (2014). Web-of-Objects (WoO)-based context aware emergency fire management systems for the Internet of Things. *Sensors (Basel, Switzerland)*, *14*(2), 2944–66. doi:10.3390/s140202944

Singh, D., Tripathi, G., & Jara, A. J. (2014). A survey of Internet-of-Things: Future vision, architecture, challenges and services. *2014 IEEE World Forum on Internet of Things (WF-IoT)*, 287–292. doi:10.1109/WF-IoT.2014.6803174

Teja, G. N. L. R., Harish, V. K. R., Nayeem Muddin Khan, D., Krishna, R. B., Singh, R., & Chaudhary, S. (2014). Land Slide detection and monitoring system using wireless sensor networks (WSN). *2014 IEEE International Advance Computing Conference (IACC)*, 149–154. doi:10.1109/IAdCC.2014.6779310

TempoDB. (2014). TempoDB. Retrieved August 01, 2014, from https://tempo-db.com/

Walnes, J., & Noakes, D. (2014). Smoothie Chart. Retrieved July 26, 2014, from http://smoothiecharts.org/

Wessels, A., Purvis, M., Jackson, J., & Rahman, S. (Shawon). (2011). Remote Data Visualization through WebSockets. *2011 Eighth International Conference on Information Technology: New Generations*, 1050–1051. doi:10.1109/ITNG.2011.182

YEI Corporation. (2014). YEI Technology. Retrieved July 16, 2014, from https://www.yeitechnology.com/

Zhang, J., Iannucci, B., Hennessy, M., Gopal, K., Xiao, S., Kumar, S., … Rowe, A. (2013). Sensor Data as a Service -- A Federated Platform for Mobile Data-centric Service Development and Sharing. *2013 IEEE International Conference on Services Computing*, 446–453. doi:10.1109/SCC.2013.34

Zhang, Y., Li, J., Li, Z., & Guo, L. (2009). Real-Time Flood Forecasting System Based on B/S Mode. *2009 International Conference on Management and Service Science*, 1–4. doi:10.1109/ICMSS.2009.5303965

A. Appendices

A.1. Evaluation of Cases

	Evaluation of Cases				
Area	Healtcare	Engineering	Transport	Private Sector	HCI
Measuring	ECG	Manufacturing Tolerances	Vehicle Telemetry	Smartphone Sensors	Position/ Acceleration sensors
Visualisation Technology	2D-Graph	Values, 2D-Model	Values, 2D-Graph/Map	2D-Interaction	3D-Interaction
Scalability	High	Low	Low	Low	Medium
Visualisation Rate	Low	Low	Medium	High	High
Mobility	High	Low	High	Medium	Medium
Reliability	Medium	High	Medium	Medium	Medium
Security	High	High	Medium	Low	Low
Privacy	High	Medium	Medium	Low	Low

	Legend	High	Medium	Low
Area	Application areas			
Measuring	Describes the data to transmit.			
Visualisation Technology	Describes efford how measurment could be logicly visualised.			
Scalability	Describes how many values will be transferred at once.	>100	<100	<10
Visualisation Rate	Describes the needed visualisation update rate for the sensor.	<0.1s (10fps)	<=1s	>1s
Mobility	Describs the acting radius of the sensor.	Everywhere (WiFi/3G)	In specific area (WiFi)	Stationary (LAN)
Reliability	Describes the reliability of the system focused on the connection.	No failure tolerated	May lose connection	Reliability not important
Security	Describes the security of the system based on the connection.	Encryption needed	Only Transport Security	Raw Readable Data
Privacy	Describes the importance of the data regarding on privacy.	Can harm privacy extremly	Privacy relevant	Privacy is not important

Table 13: Rating of Cases

A.2. Node.js code

A.2.1. Raw Data transmission

```
1.  var http = require('http');
2.  var url = require('url');
3.  var socket = require('socket.io');
4.  var md5 = require('MD5');
5.
6.  // Ports
7.  var portClient = !isNaN(process.argv[2]) ? process.argv[2] : 8443;
8.  var portSensor = !isNaN(process.argv[3]) ? process.argv[3] : 41181;
9.
10. //http listen
11. var httpServer = http.createServer().listen(portClient);
12. var httpSocketIo = socket.listen(httpServer);
13. var oldData = null;
14.
15. //udp server on portSensor
16. var server = require("dgram").createSocket("udp4");
17.
18. // triggered on incomming message throught the udp server launched after
19. server.on("message", function (msg, rinfo) {
20.   // Store Hash from object
21.   var temp = md5(msg);
22.
23.   // Avoid similar objects to update the client website
24.   // Compare with Object hash with last object hash
25.   if(oldData != temp){
26.     // Send message to client website
27.     httpSocketIo.sockets.emit('notification', {'message': String.fromCharCode.apply(
    null, new Uint16Array(msg))});
28.     // Log in console
29.     console.log("msg: " + msg);
30.   }
31.   // Store actual object hash
32.   oldData = temp;
33. });
34.
35. // Start listening on udp server port portSensor
36. server.on("listening", function () {
37.   var address = server.address();
38.   console.log("udp server listening " + address.address + ":" + address.port);
39.   console.log("and broadcasting to " + address.address + ":" + portClient);
40. });
41. server.bind(portSensor);
```

Source Code 3: Node.js Script

A.2.2. With TLS

```
1.  var https = require('https');
2.  var url = require('url');
3.  var fs = require('fs');
4.  var socket = require('socket.io');
5.  var portClient = !isNaN(process.argv[2]) ? process.argv[2] : 8443;
6.  var portSensor = !isNaN(process.argv[3]) ? process.argv[3] : 41181;
7.
8.  var options = {
9.    key: fs.readFileSync('privatekey.pem'),
10.   cert: fs.readFileSync('certificate.pem')
11. };
12.
13. //https listen
14. var httpsServer = https.createServer(options).listen(portClient);
15. var httpsSocketIo = socket.listen(httpsServer);
16.
17. //udp server on portSensor
18. var server = require("dgram").createSocket("udp4");
19.
20. // triggered on incomming message throught the udp server launched after
21. server.on("message", function (msg, rinfo) {
22.   console.log("msg: " + msg);
23.   httpsSocketIo.sockets.emit('notification', {'message': String.fromCharCode.apply(n
    ull, new Uint16Array(msg))});
24. });
25.
26. // Start listening on udp server port portSensor
27. server.on("listening", function () {
28.   var address = server.address();
29.   console.log("udp server listening " + address.address + ":" + address.port);
30. });
31. server.bind(portSensor);
```

Source Code 4: Node.js Script with TLS

A.3. MSc Research Proposal

1. Student details

Last (family) name	Lapok
First name	Paul
Napier matriculation number	40132336

2. Details of your programme of study

MSc Programme title	MSc Computing
Year that you started your diploma modules	1 Year
Month that you started your diploma modules	September
Mode of study of diploma modules	Full-time
Date that you completed/will complete your diploma modules at Napier	31th August 2014

3. Project outline details

Please suggest a title for your proposed project. If you have worked with a supervisor on this proposal, please provide the name. NB you are strongly advised to work with a member of staff when putting your proposal together.

Title of the proposed project	Cloud Based Visualisation of Real Time Sensor-Data Streams
Name of supervisor	Alistair Lawson
I do not have a member of staff lined up to supervise my work	

4. Brief description of the research area - background

Please provide background information on the *broad research area* of your project in the box below. You should write in narrative (not bullet points). The academic/theoretical basis of your description of the research area should be evident through the use of references. Your description should be between half and one page in length.

Sensor data was streamed mostly concentrated on environment monitoring. This has included systems monitoring drought (Deng et al., 2013), seismography, tsunamis, landslides (Teja et al., 2014) or flood (Y. Zhang et al., 2009) for the purpose of alerting to potentially dangerous situations. Systems like these send their data in intervals to their base stations or a centralized computer (Ha et al., 2012), sometimes directly over the web where it will be stored in databases used as historic data for longer term research. The sampling rate may vary between a few seconds up to hours which meet the requirements of these systems. A high sampling rate compared to real-time computing is not needed, like in industrial applications, e.g. machine control.

There are also existing systems in the area of object tracking over the web. Tracking systems consisting of hardware and server using the Global System for Mobile Communications (GSM), General Packet Radio Service (GPRS) and

often the Google API to visualize the tracked objects as discussed in (Chadil et al., 2008), an open source tracking system using commodity hardware as described in (El-Medany et al., 2010), a system using ASP.NET and TCP/IP sockets sending IP data packets over the connection with an time interval of 1 minute. Other systems using the Short Message Service (SMS) to transfer the data with an interval of 12 seconds (Rao et al., 2009). Most of these systems store the data in databases which can be accessed over the web.

A big area is the Internet of Things (IoT). The vision is the unique identification of real world objects and interaction of them with other objects or services. Depending on technologies or application areas the Internet of Things can be defined in a number of ways. Some of these definitions focus on uniquely addressable interconnected objects by using Radio-Frequency Identification (RFID) technologies, others focus on communication and interaction between objects or investigation in infrastructures and others concentrate on user centric approaches (Gubbi et al., 2013). The data transmission between objects is the core statement of these definitions.

The aim of the project is to design an architecture for real time sensor data streaming, management and live visualisation over the web. State of the art technologies will be reviewed and potential application areas for sensor data streaming will be identified.

The focus will be on identification and testing of front end technologies to allow an evaluation of operational capability in different application areas. One big difference will be that the system does not store sensor data. It will be redirected to client applications, where data will be consumed directly, e.g. visualisation of data. The decision to store data is on client side.

5. Project outline for the work that you propose to complete

Please complete the project outline in the box below. You should use the emboldened text as a framework. Your project outline should be between half and one page in length.

The idea for this research arose from:

The idea arose from my work on web services for the Pro-talk Project lead by Alistair Lawson and from the courses I had chosen during the master program in software engineering and web development. I am also interested in measuring and like the idea of “the internet of things”.

The aims of the project are as follows:

The aim of the project is to design an architecture for real time sensor data streaming, management and live visualisation over the web. State of the art technologies will be studied and potential application areas for sensor data streaming will be identified.

The focus will be on identification and testing of front end technologies to allow an evaluation of operational capability in different application areas.

The main research questions that this work will address include:

Is it possible to build a manageable real time sensor streaming system?

Which areas of application related to sensor data streaming can be covered with the identified technologies?

Where are the limits of these identified technologies?

The software development/design work/other deliverable of the project will be:

The Project will deliver some software for testing and evaluation of some technologies.

The project will involve the following research/field work/experimentation/evaluation:

The first part will be an evaluation of state of the art technologies. A requirement analysis and survey over areas of application will be done for definition of the system requirements. An experimental setup will allow an investigation of client side web technologies and measurement of performance which will allow an evaluation of operational capabilities.

This work will require the use of specialist software:

The work will require open source tools for measurement of network traffic to evaluate operational capabilities.

This work will require the use of specialist hardware:

It does not require specialist hardware, because sensor streaming hardware will be simulated.

6. References

Please supply details of all the material that you have referenced in sections 6 and 7 above. You should include at least three references, and these should be to high quality sources such as refereed journal and conference papers, standards or white papers. Please ensure that you use a standardised referencing style for the presentation of your references, e.g. APA, as outlined in the yellow booklet available from the School of Computing office and http://www.soc.napier.ac.uk/~cs104/mscdiss/moodlemirror/d2/2005_hall_referencing.pdf

American Heart Association. (2014). Target Heart Rates. Retrieved from http://www.heart.org/HEARTORG/GettingHealthy/PhysicalActivity/Target-Heart-Rates_UCM_434341_Article.jsp

Andersson, K., & Johansson, D. (2012). Mobile e-services using HTML5. *37th Annual IEEE Conference on Local Computer Networks -- Workshops*, 814–819. doi:10.1109/LCNW.2012.6424068

Association for the Advancement of Medical Instrumentation. (2002). Cardiac monitors, heart rate meters, and alarms [American National Standard (ANSI/AAMI EC13:2002)]. Arlington. Retrieved from http://www.physionet.org/physiobank/database/aami-ec13/

Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, *54*(15), 2787–2805. doi:10.1016/j.comnet.2010.05.010

Australia, C. of. (2014). Deep Ocean Tsunami Detection Buoys. Retrieved July 06, 2014, from http://www.bom.gov.au/tsunami/about/detection_buoys.shtml

Bendel, S., Springer, T., Schuster, D., Schill, A., Ackermann, R., & Ameling, M. (2013). A service infrastructure for the Internet of Things based on XMPP. *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, 385–388. doi:10.1109/PerComW.2013.6529522

Cai, K., Liang, X., & Wang, K. (2011). Development of Field Information Monitoring System Based on the Internet of Things *, 675–680.

Chadil, N., Russameesawang, A., & Keeratiwintakorn, P. (2008). Real-time tracking management system using GPS, GPRS and Google earth. *2008 5th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, 393–396. doi:10.1109/ECTICON.2008.4600454

Chen, B. (2011). A framework for browser-based Multiplayer Online Games using WebGL and WebSocket. *2011 International Conference on Multimedia Technology*, 471–474. doi:10.1109/ICMT.2011.6001673

Dahl, T., Koskela, T., Hickey, S., & Vatjus-Anttila, J. (2013). A Virtual World Web Client Utilizing an Entity-Component Model. *2013 Seventh International Conference on Next Generation*

Mobile Apps, Services and Technologies, 7–12. doi:10.1109/NGMAST.2013.11

Deng, M., Di, L., Han, W., Yagci, A. L., Peng, C., & Heo, G. (2013). Web-service-based Monitoring and Analysis of Global Agricultural Drought, *22030*.

Duquennoy, S., Lifl, I., & Grimaud, G. (2009). The Web of Things : interconnecting devices with high usability and performance, *2009*.

El-Medany, W., Al-Omary, A., Al-Hakim, R., Al-Irhayim, S., & Nusaif, M. (2010). A Cost Effective Real-Time Tracking System Prototype Using Integrated GPS/GPRS Module. *2010 6th International Conference on Wireless and Mobile Communications*, 521–525. doi:10.1109/ICWMC.2010.104

Forsström, S., & Kanter, T. ubiquitous sensor-assisted applications on the internet-of-things. (2013). Enabling ubiquitous sensor-assisted applications on the internet-of-things. *Personal and Ubiquitous Computing*, *18*(4), 977–986. doi:10.1007/s00779-013-0712-9

Forsström, S., Kardeby, V., Österberg, P., & Jennehag, U. (2014). Challenges when Realizing a Fully Distributed Internet-of-Things – How we Created the SensibleThings Platform, (c), 13–18.

Goldberger, A. L., Amaral, L. A. N., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., … Stanley, H. E. (2000). {PhysioBank, PhysioToolkit, and PhysioNet}: Components of a New Research Resource for Complex Physiologic Signals. *Circulation*, *101*(23), e215–e220.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, *29*(7), 1645–1660. doi:10.1016/j.future.2013.01.010

Ha, S. W., Lee, Y. K., Vu, T. H. N., Jung, Y. J., & Ryu, K. H. (2012). An environmental monitoring system for managing spatiotemporal sensor data over sensor networks. *Sensors (Basel, Switzerland)*, *12*(4), 3997–4015. doi:10.3390/s120403997

IRIS. (2014). Incorporated Research Institutions For Seismology. Retrieved July 06, 2014, from http://www.iris.edu/

Jones, V., Gay, V., & Leijdekkers, P. (2010). Body sensor networks for mobile health monitoring: Experience in europe and australia. *Digital Society, 2010. ICDS'10. …*.

Joyent Inc. (2014). Nodejs. Retrieved August 01, 2014, from http://nodejs.org

LogMeIn Inc. (2014). Xively. Retrieved August 01, 2014, from https://xively.com/

Microsoft. (2014). ASP.NET. Retrieved July 06, 2014, from http://www.asp.net/

MicroStrain. (2014). SensorCloud. Retrieved August 01, 2014, from http://www.sensorcloud.com/

Neumeyer, D., & Brown, J. (2014). Audio-Visual Palimpsests: Resynchronizing Silent Films with "Special" Music. In *The Oxford Handbook of Film Music Studies* (p. 588). Oxford University.

Orduña, P., & Angulo, I. (2014). Graphic Technologies for Virtual , Remote and Hybrid

laboratories : WebLab-FPGA hybrid lab, (February), 163–166.

Rao, A. S., Izadi, D., Tellis, R. F., Ekanayake, S. W., & Pathirana, P. N. (2009). Data monitoring sensor network for BigNet research Testbed. *2009 International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, 169–173. doi:10.1109/ISSNIP.2009.5416816

Sample, A. P., Braun, J., Parks, A., & Smith, J. R. (2011). Photovoltaic enhanced UHF RFID tag antennas for dual purpose energy harvesting. *2011 IEEE International Conference on RFID*, 146–153. doi:10.1109/RFID.2011.5764615

Shamszaman, Z. U., Ara, S. S., Chong, I., & Jeong, Y. K. (2014). Web-of-Objects (WoO)-based context aware emergency fire management systems for the Internet of Things. *Sensors (Basel, Switzerland)*, *14*(2), 2944–66. doi:10.3390/s140202944

Singh, D., Tripathi, G., & Jara, A. J. (2014). A survey of Internet-of-Things: Future vision, architecture, challenges and services. *2014 IEEE World Forum on Internet of Things (WF-IoT)*, 287–292. doi:10.1109/WF-IoT.2014.6803174

Teja, G. N. L. R., Harish, V. K. R., Nayeem Muddin Khan, D., Krishna, R. B., Singh, R., & Chaudhary, S. (2014). Land Slide detection and monitoring system using wireless sensor networks (WSN). *2014 IEEE International Advance Computing Conference (IACC)*, 149–154. doi:10.1109/IAdCC.2014.6779310

TempoDB. (2014). TempoDB. Retrieved August 01, 2014, from https://tempo-db.com/

Walnes, J., & Noakes, D. (2014). Smoothie Chart. Retrieved July 26, 2014, from http://smoothiecharts.org/

Wessels, A., Purvis, M., Jackson, J., & Rahman, S. (Shawon). (2011). Remote Data Visualization through WebSockets. *2011 Eighth International Conference on Information Technology: New Generations*, 1050–1051. doi:10.1109/ITNG.2011.182

YEI Corporation. (2014). YEI Technology. Retrieved July 16, 2014, from https://www.yeitechnology.com/

Zhang, J., Iannucci, B., Hennessy, M., Gopal, K., Xiao, S., Kumar, S., … Rowe, A. (2013). Sensor Data as a Service -- A Federated Platform for Mobile Data-centric Service Development and Sharing. *2013 IEEE International Conference on Services Computing*, 446–453. doi:10.1109/SCC.2013.34

Zhang, Y., Li, J., Li, Z., & Guo, L. (2009). Real-Time Flood Forecasting System Based on B/S Mode. *2009 International Conference on Management and Service Science*, 1–4. doi:10.1109/ICMSS.2009.5303965

7. Ethics

If your research involves other people, privacy or controversial research there may be ethical issues to consider (please see the information on the module website). If the answer below is YES then you need to complete a research Ethics and Governance Approval form (available on the website: http://www.ethics.napier.ac.uk).

Does this project have any ethical or governance issues related to working with, studying or observing other people? (YES/NO)	**No**

8. Supervision timescale

Please indicate the mode of supervision that you are anticipating. If you expect to be away from the university during the supervision period and may need remote supervision please indicate.

Weekly meetings over 1 trimester	X
Meetings every other week over 2 trimesters	
Other	

9. Submitting your proposal

Please save this file using your surname, e.g. macdonald_proposal.doc, and e-mail it to the module leader in time for the next proposal deadline.

A.4. Project Plan

Tasks	CW21	CW22	CW23	CW24	CW25	CW26	CW27	CW28	CW29	CW30	CW31	CW32	CW33
Literature Review				M1									
Search for Papers													
Read Papers													
Write Review													
Project Proposal													
Write Initial Report													
Methodology						M2							
Requirement Analysis													
Sensor Classification													
Visualisation Technologies													
Planing Implementation													
Planing Experiment details													
Writing Methodology													
Implementation										M3			
Requirements													
Sensor Simulator													
(Mobile Application)													
Web Service													
Tesing / Experiments													
Results and Evaluation											M4		
Correcting Dissertation													M5

	Milestone 1:	**15th June 2014**	Finish literature review
			Finish project proposal
			Finish initial report
	Milestone 2:	**29th June 2014**	Finish Analysis
	Milestone 3:	**27th July 2014**	Finish implemetation
			Finish testing
			Finish experiments
	Milestone 4:	**3th August 2014**	Summarize Results and finish evaluation
	Milestone 5:	**17th August 2014**	Finish, print Report, put together documents

Figure 26: Project Plan

A.5. Digital Appendix

The following additional content can be found on the attached CD.

- Dissertation as PDF
- Research Proposal
- Initial Report
- Figures of Architecture (chapter 4)
- Measurement data of Experiments (chapter 5)
- Acceptance Test (Results of chapter 5.2)
- Applications and Source Code
    - Frontend Data Visualisation (Websites chapter 5.3)
    - Applications (Virtual Studio 2013 Project)
        - ClientWebsite (example of client website)
        - ECGSignal (Streaming ECG Signal data to Server)
        - InstanceTest (Experiment chapter 5.6.1)
        - LoadTest (Experiment chapter 5.6.2)
        - LostPackages (Experiment chapter 5.6.3)
        - ManageSensors (Console application for communication with Registration Service chapter 5.2)
        - NodeIterator (Example application – Sending data to Server)
        - RegistrationServiceTester (GUI for Registration Service chapter 5.2)
        - SendMessage (Investigation of Security chapter 5.5)
        - SensorRegistrationService (Registration Service and database)
    - Node.js Server Scripts
        - With TLS
        - Without TLS
        - Prepared for data logging