UiO : Department of Informatics
University of Oslo

Improving the performance of Web Services in Disconnected, Intermittent and Limited Environments

Joakim Johanson Lindquister

Master's Thesis Spring 2016

Abstract

Lorem ipsum dolor sit amet, cu sed suas apeirian, decore iudicabit at per, pro ne lorem dicit dictas. Cu quo aeque maiorum gubergren, principes complectitur ei ius, numquam veritus minimum mel id. Ea ius vidit soleat. Mel timeam laoreet tractatos no.Pro an sadipscing efficiantur, esse ludus diceret nam in. Vis percipit probatus in. Est noster moderatius dissentiet te. Eirmod latine dissentias in sea, perfecto omittantur at duo, mea vide exerci ut. Nec euismod vocibus consectetuer eu.

Et fierent delectus sapientem eam, id eum dolore nullam. Cu his quod possit utamur, mel offendit copiosae forensibus ut, ius fabulas fierent sapientem an. Sed at vidit mentitum expetendis, utamur insolens ad cum, dicat dicta salutatus ei duo. Est tnce e numquam explicari posidonium. Vim amet nostrud at, ea nam graece mediocritatem, cu fabulas maiorum nostrum vix. Ius id zril nullam aperiam, at sint corpora repudiandae eam.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Military units operate under conditions where the reliability of the network connection may be low. They can operate far from existing communication infrastructure and rely only on wireless communication. Such networks are often characterized by unreliable connections with low date rate and high error rates making data communication difficult. In a military scenario it is necessary for units at all levels to seamlessly exchange information across different types of communication systems. This ranges from remote combat units at tactical level, to commanding officers at operational level in a static headquarters packed with computer support. To the North Atlantic Treaty Organization (NATO), this concept is referred to as Network Enabled Capability (NEC). In a feasibility study, NATO identified the Service Oriented Architecture (SOA) paradigm and the Web Service technology as key enablers for information exchange in NATO[1].

Web service technology is well tested and in widespread use in civil applications where the network is stable and the data rate is abundant. However, certain military networks suffer from high error rates and very low date rate, which can leave Web services built for civilian use unusable. This thesis investigates how these challenges can be overcome by applying different optimization techniques. The main approach looks into how using alternative transport protocols other than HTTP/TCP may increase speed and reliability.

1.1 Background and Motivation

NATO is a military alliance consisting of 28 member countries [2] and which primary goal is to protect the freedom and security of its members through political and military means. In joint military operations the relatively large number of member countries can be a challenge when setting up machine-to-machine information exchange. Differences in communication systems and equipment attribute to making the integration of such systems more difficult. In order to address this issue, NATO has chosen the SOA concept, which when built using open standards facilitates interoperability[1].

1.1.1 Service Oriented Architecture

SOA is an architectural pattern where application components provide services to other components over a network. SOA is built on concepts such as object-orientation and distributed computing and aims to get a loose coupling between clients and services. In their reference model for SOA, the Organization for the Advancement of Structured Information Standards (OASIS) define SOA as [3]:

Service Oriented Architecture is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.

In SOA, business processes are divided into smaller chunks of business logic, referred to as *services*. A service can be business related, e.g a patient register service, or a infrastructure service used by other services and not by a user application. OASIS define a service as [3]:

A service is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description

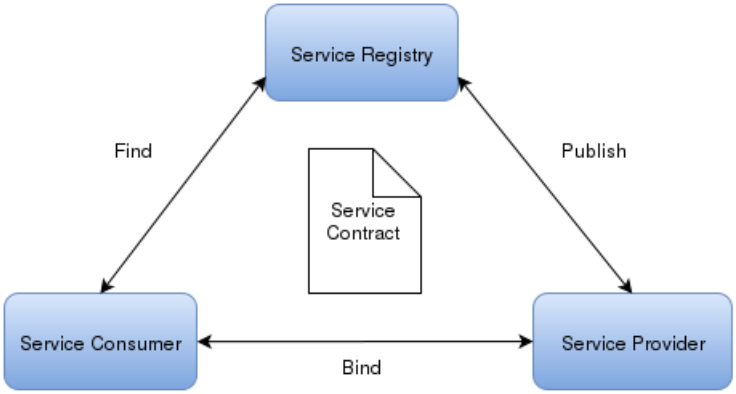

Figure 1.1: The three roles in SOA(from [4])

Services are provided by *service providers* and are consumed by *service consumers*. The service provider is responsible for creating a service description, making the service available to others and implementing the service according to the service description. Services are made available to service consumers through a form of *service discovery*. This can be a static configuration, or more dynamic with a central *service registry*, where service providers publish service descriptions. Service consumers find the services they need by contacting the service registry. The communication between services occur through the exchange of standardized messages.

Following the SOA principles dictates a very loose coupling between services and the consumers of those. This allows software systems to be more flexible, as new components can be integrated with minimal impact on the existing system. Another aspect of loose coupling is with regard to time, which enable services and its consumers to not be available at the same instance of time. This enables asynchronous communication. Loose coupling with regards to location allows the location of a service to be changed without needing to reprogram, reconfigure, or restart the service consumers. This is possible through the usage of runtime service discovery, which is dynamic retrieval of the new location of the service.

Furthermore SOA enables service implementation neutrality. The implementation of service is completely separated from the service description. This allows re-implementation and alteration of a service without affecting the service consumers. Thus this can attribute to keep development costs low and avoiding proprietary solutions and vendor lock-in. Another benefit with SOA is re-usability by dividing common business processes into services, which may help cost reduction and avoids duplication. SOA is only a pattern and the concepts can be realized by a range of technologies. The most common used approach is the Web service family of standards, using the SOAP messaging protocol.

To achieve interoperability between systems from different nations and vendors, NATO has chosen the Web service technology in order to realize the SOA principles[5]. This allows member nations to implement their own technology as long as they adhere to the standards. The Web service technology is discussed in detail in section 2.2. Another approach to realize SOA is Representational State Transfer (REST), an architecture style which uses HTTP over TCP. REST has gained a lot of traction in the civil industry and is discussed in section section 2.2.2.

However, employing Web service solutions directly into military use may not be straight forward. These technologies were not specifically designed to handle conditions found in certain military networks. In the following sections we discuss characteristics of such networks and the possible challenges of using Web services in them.

1.1.2 Military Networks

Military networks are complex and consist of many different heterogeneous network technologies. We can group them into layers, which have different characteristics as can been seen in fig. 1.2. At the highest level, there is fixed infrastructure and relatively static users, meaning that they seldom move around or disconnect. At the lower levels, there are fewer units, but they are much more dynamic. The lower level is called tactical networks, which is discussed in the next paragraph.

Tactical Networks

Tactical networks are characterized by that the units are deployed to operate in a battlefield, which means there is no existing communication in-

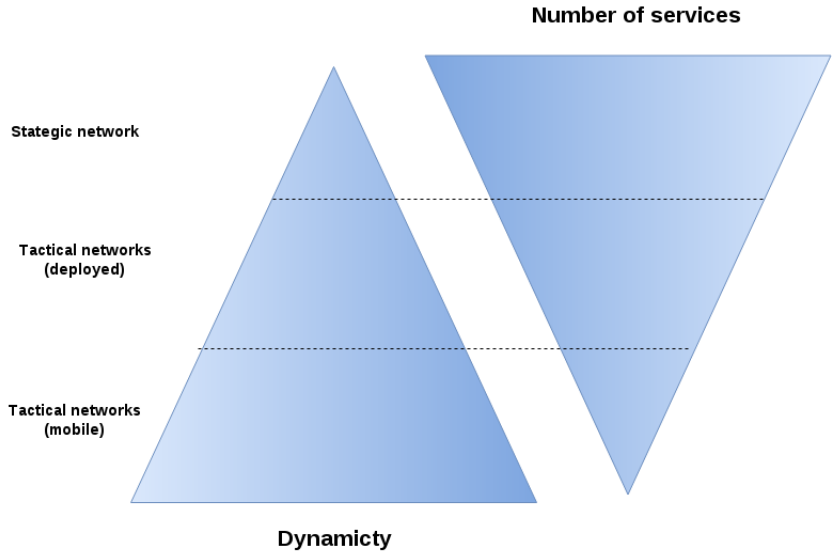

Figure 1.2: Complexity of military networks(from [6])

frastructure. They use tactical communication equipment, which includes technologies like VHF, UHF, HF, tactical broadband and satellites[4].

Examples of such units are mobile units like vehicles, foot soldiers and field headquarters. The tactical network connects deployed headquarters with mobile units. These types of networks are unpredictable and may have very low date rate, possibly high delay, high error rates and frequent disconnections. They are often called disadvantaged grids or Disconnected, Intermittent and Limited (DIL) environments, which is the term used in this thesis. DIL is discussed in section 1.1.3.

NATO studies[7] have identified such networks to have the following characteristics:

Disadvantaged grids are characterized by low bandwidth, variable throughput, unreliable connectivity, and energy constraints imposed by the wireless communications grid that link the nodes .

The characteristics of these networks and what challenges they impose are discussed in further detail in section 1.1.3.

1.1.3 Disconnected, Intermittent and Limited Networks

To improve the performance of Web services in limited military networks, we must understand what limitations we're dealing with. The DIL concept refers to three characteristics of a limited network. As we discussed in the introduction, military tactical networks may suffer from these constraints.

Disconnected Military units that participate in a tactical network are highly mobile and may disconnect from a network either voluntarily

or not. This causes topology changes. Unplanned loss of connectivity can be due to various reasons, such as loss of signal or equipment malfunction. The disconnected term refers to that nodes in the network may be disconnected for a long time, possibly for multiple days.

Intermittent Nodes in a DIL environment may lose connection temporarily before reconnecting. The duration range from seconds to minutes.

Limited The Data rate, how many bits that are sent per second, is limited in DIL networks. Various aspects that affects the date rate are discussed in the next section.

Other constraints

As well as being restricted by the communication link itself, military units may have other limitations as well. Consider that military foot patrols have limited battery capacity as they have to carry it with them in their backpacks. The transmission range of the communication equipment for mobile units may also be limited. Another factor that comes into play for military units is that in some cases they are required to enter radio silence in order to avoid being detected by the enemy. During such circumstances the soldiers may only receive data, but not send any.

1.2 Example scenario

Jeg tenkte her å introdusere et scenario som illustererer problemer og utfordringer med DIL nettverk.

1.3 A suggested solution

The Web service technology enables interoperability between systems, but also increase the information overhead, requiring higher data rate demands. Employing Web services developed for use in civilian networks directly into a DIL environment may not perform satisfactorily. To increase the performance we can apply different optimization techniques. The task-group IST-090[4] investigated which improvements that could be made in order to get SOA applicable at the tactical level. They did not find a magic bullet that would solve all problems, but identified factors that would offer measurable improvements. The most important findings were:

- Foundation on open-standards.
- Ease of management and configuration.
- Transparency to the user.

- The Web services should optimized without the need to incorporate proprietary, ad hoc solutions that ensure tighter coupling between providers and consumers of services.

The last bullet point refers to the issue of that when we have identified optimization techniques, where do we place them? One approach is to modify the Web service application itself. However, this would mean that every application that is used in a tactical network would require modification. This would require a lot of resources and severely limit the flexibility of using Web services. Another solution is, by using proxies, we can apply the optimization there without altering the Web services themselves. The only thing required to do is to configure the application to send and receive data through the proxy. The proxy will take handle of the optimization for tactical networks. This approach is identified in IST-090 and is explored in this thesis.

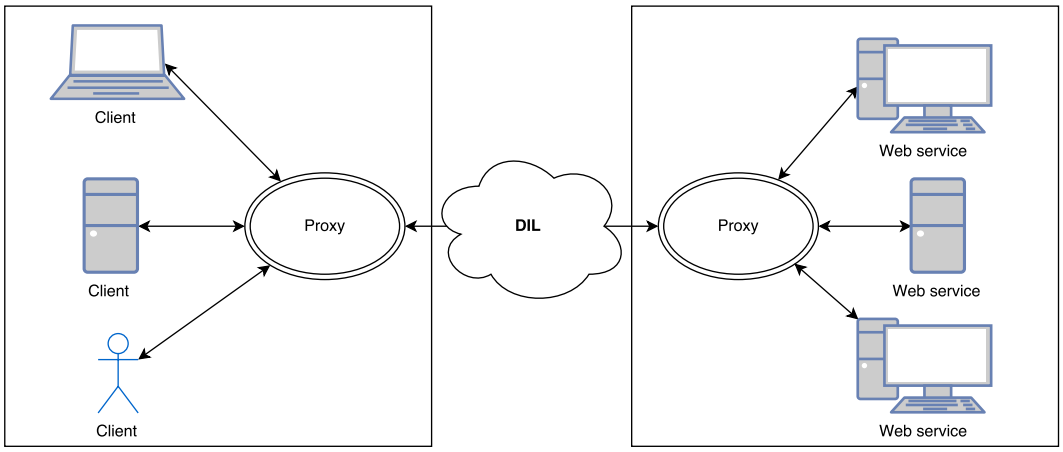

Figure 1.3: Proposed proxy solution

1.3.1 Proxies

A proxy is an application which acts as an intermediary between an client and a server. Proxies are widely in use and their usage and type varies. Example of proxy usage is for load balancing, caching and security. Web proxies are proxies that forward HTTP requests, which is what we are investigating in this thesis. This proxy will support features for compression, delay tolerance and overcome network disruptions.

1.4 Problem Statement

Most of the Web service solutions used today are aimed for civilian use and do not necessarily perform well in military environments. In contrast to civilian networks where the date rate is abundant, mobile tactical networks may suffer from high error rates and low date rate. Adapting Web service solutions meant for civil networks directly for military purposes may not be

possible. Therefore, Web services needs to be adapted in order to handle network challenges. However, it can be very expensive to alter existing Web service technology and incorporate proprietary solutions. A NATO research task group has previously identified the foundation on open standards to avoid tighter coupling between service providers and consumers[4]. It is much better to use Commercial off-the-shelf (COTS) software. By placing the optimization in proxies, the Web services can remain unchanged.

The goal of this thesis is to investigate different optimization techniques that can be applied in order to improve Web service performance in DIL networks. In order for the clients and services to remain interoperable the optimization techniques will be placed in proxies. The Web services will communicate as normal, while all network traffic is tunneled through a proxy. The Web service itself does not need to pay attention to the bad connectivity, the proxy will choose the appropriate protocol and configuration.

1.5 Premises of thesis

The proxy developed as a part of this thesis shall support both REST and Web service communication between machines connected in a DIL network. In order to optimize Web services in DIL environments, the applications themselves should not be required to be customized, all optimization should be placed in proxies. This retains the interoperability with standardized solutions(COTS).

1.6 Scope and Limitations

The goal of this thesis is to investigate optimization techniques for Web services in DIL environments. Security is therefore not addressed in this thesis. However, applications that are to be used in military networks need to be approved by security authorities. If the application is too complex, e.g. it has a very large code base or use a lot of external frameworks, the approvement process will be very lengthy. It is therefore a important that the proxy is relatively simple. Furthermore, some security features such as IPSec will be enabled or disabled as part of the evaluation of the proxy.

When investigating optimization techniques, we limit it to techniques that can be applied at the application or the transport layer of the Internet protocol suite(see table 2.1). The reason for this is that NATO has previously decided "everything over IP", a statement describing that all data communication in NATO should occur with IP packets. We therefore limit our optimization possibilities to the mentioned layers.

Finally, the proxy implemented as a part of this thesis, only accepts Hypertext Transfer Protocol (HTTP) as input from the Web services. As we discuss later in section 2.2, most Web services use HTTP to communicate.

1.7 Research Methodology

Denning.

1.8 Contribution

The outcome of this thesis is a recommendation regarding which optimizations techniques can be used in DIL to enhance the performance of Web services. As well as a prototype implementation of a DIL proxy.

1.9 Outline

Hvordan er resten av oppgaven strukturert.

Chapter 2

Technical Background

Before diving into the design and implementation of the proxy developed in this thesis, we're in this chapter we presenting the technical background of the central concepts and protocols central. We first give a introduction to computer networks in general and how they are organized. Next, we discuss common Web services used for exchanging data in military systems. Then we look into a number of protocols that we can replace HTTP/TCP with in order to increase the performance of Web services. Finally, we introduce the concept of performance testing and network metrics.

2.1 Network layers

To reduce design complexity, networks are organized into layers, each one built upon the one below it. In the Internet Protocol Suite[8], networks is divided into 4 layers. As stated in the scope of this thesis, we only look into optimization techniques for the application and transport layer.

Application Layer
Transport Layer
Internet Layer
Link layer

Table 2.1: The layers of the Internet Protocol Suite

Link layer The lowest layer is the link layer, where link refers to the physical network component used to interconnect nodes in a network. Link layer protocols operate between adjacent network nodes. An example of a link layer protocol is Ethernet.

Internet Layer Where the link layer is only concerned of moving data over a wire to an adjacent node, the Internet layer is concerned of how to deliver data all the way from a source to a destination, possible passing through multiple nodes on its way. It does not guarantee delivery of data,

since data can be lost on the way to the destination. Guaranteed delivery is usually handled on the higher levels of the Internet Protocol Suite.

The core protocol of the Internet layer is Internet Protocol (IP) and its routing function enables sending data over interconnected networks.

Transport layer In the Internet protocol suite model, the transport layer provides end-to-end communication services for applications. It builds on top on the network layer, and takes responsibility of sending data all the way from a process on a source machine to a process on the destination machine. The far most used transport protocol is the Transmission Control Protocol (TCP), which provides reliable transport of data to applications. With reliable transport we mean that if data in transmission is lost or received in the wrong order, this is all handled by the transport protocol. This provides an important abstraction for applications so that they don't need to deal with the characteristics of the physical network itself.

Application layer The top layer is the application layer and is where applications real user use reside. The other layers provide transport services to applications found in this layer. When we talk about application layer protocols, we talk about protocols that applications use to communicate with other applications. Application layer protocols use the communication services the transport layer provides. Examples of application layer protocols is HTTP and File Transfer Protocol (FTP).

2.2 Web services

Web services are client and server applications that communicate over a network and can be used to implement a service oriented architecture. Web services are critical in any data systems and are in widespread use in both civilian and military systems. It is a broad term and can be used to describe different types of services that are available over a network. The most common usage of the term refers to the World Wide Web Consortium (W3C) definition of SOAP-based Web services, but could also refer to more simple HTTP-based REST services.

In this thesis we investigate optimization techniques that should support both W3C Web services and RESTful web services.

2.2.1 W3C Web services

W3C has defined Web services as [9]:

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its

description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

This definition points out a set of standards that enables machine-to-machine interactions. All communication is based on sending XML-based SOAP messages. There exists many definitions of Web services where the core principles are the same, but the finer details may vary. The Web service technology is realization of the SOA principles, which provides loose coupling and ease integration between systems.

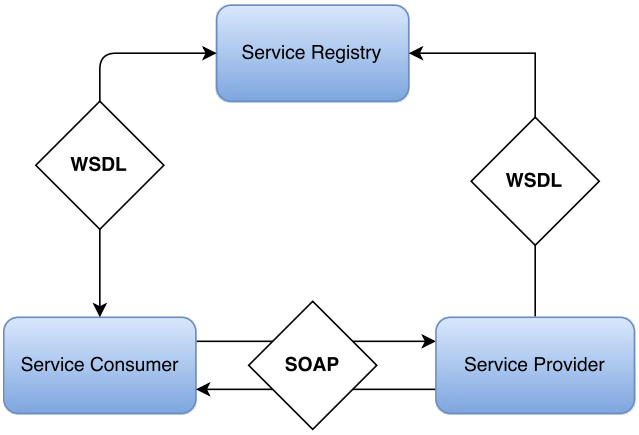

Figure 2.1: W3C Web services

These standards that together makes W3C Web services are presented in the following sections.

XML

The Extensible Markup Language (XML)[10] is considered as the base standard for Web services. An XML document consist of data surrounded by tags and is designed to be both machine and user readable. Tags describe the data they enclose. The tags can be standardized, which allows exchange and understanding of data in a standardized, machine-readable way.

Service descriptions: WSDL

Web Services Description Language (WSDL) is an XML-based interface definition language that describes functionality offered by a Web service[11]. The interface describes available functions, data types for message requests and responses, binding information about the transport protocol, as well as address information for locating the service. This enables a formal, machine-readable description of Web service which clients can invoke.

SOAP

SOAP is an application level, XML-based protocol specification for information exchange[12] in the implementation of Web services. Data commu-

nication in SOAP is done by nodes sending each other whats called SOAP messages. A SOAP message can be considered as an "envelope" consisting of an optional message header and a required message body. The header can contain information not directly related to the message such as routing information for the message and security information. The body contains the data being sent, known as the payload.

It is transport protocol agnostic, which means it can be carried over various underlying protocols. The far most used transport protocol is HTTP over TCP, but other protocols such as UDP and SMTP can be used as well.

2.2.2 Representational State Transfer

In the previous sections we looked into the standards and specifications that compose W3C Web services. However, there also exist other types of Web services which does not follow the these standards. In year 2000, the computer scientist Roy Fielding introduced REST where he presented a model of how the Web *should* work. This idealized model of interactions within a Web application[13] is what we refer to as the REST architectural style. REST attempts to minimize latency and network communication while maximizing the independence and scalability of component implementations. This is done by placing constraints on connector semantics rather than on component semantics like W3C Web services. REST is based on a client-server model where a client requests data from a server when needed. While W3C Web services is service oriented, we can look at REST as being more data oriented.

Web services that adhere to the REST style is called RESTful Web services. They are closely associated with HTTP and use HTTP verbs(e.g GET, POST, DELETE) to operate on information located on a server. RESTful Web services typically expose some sort of information, called resources in REST. Resources are identified by a resource identifier. Table 2.2 illustrates how a component exposes a set of operations of the car resource.

Resource identifier	**HTTP Method**	**Meaning**
/vehicles/cars/1234	GET	Return a car with ID 1234 from the system.
/vehicles/cars/	POST	Create a new car which will be added to the list of cars.
/vehicles/cars/1234	DELETE	Delete a car with ID 1234 from the system.

Table 2.2: Example of REST operations

REST is easy to understand and has gained a lot of traction in the civil industry in the latest years. Although NATO has chosen W3C Web services as the technology to do information-exchange, REST is identified

as a technology of interest to certain groups in NATO[14]. One downside to NATO with REST is that it lack standardization, which may cause interoperability issues.

In the next section we will look into HTTP, which is closely associated with REST and is the most used transport protocol for W3C Web services.

2.3 Hypertext Transfer Protocol

As we have seen in the previous sections, both RESTful and W3C Web services utilizes the HTTP as their way to communicate with other services. The usage of HTTP is very widespread and it is the foundation of data communication for the World Wide Web since the early 90's. It's protocol specification is coordinated by Internet Engineering Task Force (IETF) and the W3C, and is defined as[15]:

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers

HTTP started out as a simple protocol for raw data transfer across the Internet and has since been updated in HTTP/1.0, HTTP/1.1 and most recently a major update with HTTP/2.0. It is a request-reply protocol which means that all data exchanges is initiated with a client invoking a HTTP-request and then waits until a server responds with a HTTP response. A HTTP-request consist of the request method, URI, protocol version, client information, and a optional body. The server responds with a message containing a status line, protocol version, a code indicating the success or error of the request, and a optional body. Both HTTP requests and responses use a generic message format and can contain zero or more HTTP headers. Headers are used to provide information about the request/reply or about the message body, e.g information about the encoding and caching information.

HTTP, being an application level protocol, relies on a transport protocol to actually transfer data to an another machine. HTTP communication most often, but not necessarily, occurs over TCP/IP connections. The only requirement in the HTTP specification is that a reliable transport protocol is used.

HTTP methods

Associated with all HTTP requests are a request method, which indicates the desired action to be performed on a resource located on a Web server. The set of HTTP methods defined in HTTP/1.1 is listed in table 2.3.

HTTP Method	Purpose
OPTIONS	Asks the server which HTTP methods and header field it supports.
GET	Retrieve information identified by the resource indentifier(Request-URI).
HEAD	Identical to GET, except that the HTTP-body is not returned from the server.
POST	Asks the server to accept the message payload from the client as a new resource.
PUT	Similar to POST but allows the client to ask the server to update a resource identified by the request-uri
DELETE	Requests that the resource identified by the request-uri is deleted
TRACE	Echoes the HTTP request. Used for debugging
CONNECT	For use with a proxy that can dynamically switch to being a tunnel

Table 2.3: HTTP methods

2.4 Transmission Control Protocol

TCP is called the workhorse of the Internet because it is so critical for how the Internet works. It is the primary transport protocol of the Internet Protocol Suite[8] and provides reliable, in-sequence delivery of two-way traffic(full-duplex) data. HTTP most often uses TCP as its transport protocol. In this subsection we present the characteristics of TCP and some of the issues we may encounter working with it.

TCP was defined in RFC 793[16] back in September 1981 and has since been improved in various RFC's. The main motivation behind TCP was to provide reliable end-to-end byte streams over unreliable networks.

Reliablility When transferring data over the Internet, the data may pass through various networks, routers and physical networks. Some of the routers may be not working correctly, a bit may be flipped when transferring data wirelessly, or some other factor may come in to play. For those reasons, we have to accept that some of the data will be damaged, lost, duplicated or delivered out of order.

TCP recovers from such faults by assigning sequence number to each packet being sent. It then requires an positive acknowledgement from the receiver that the data was actually received. If the acknowledgement is not received within a timeout interval, the data is transmitted again. For the

receiver the sequence numbers are used to ensure that data is received in the correct order, as well as eliminating duplicates. Furthermore, to detect damaged data, TCP applies checksums to each segment transmitted. At the receiver the checksum is then checked and damaged segments are discarded.

Flow Control If a fast receiver sends data faster than a slow receiver is able to process, the receiver will be swamped with data and may experience serious performance reduction. Flow control is a mechanism to manage the rate of the data transmission to avoid overflowing a receiver. TCP provides this by using a window of acceptable sequence numbers that the receiver is willing to accept. With every acknowledgement sent back to the sender, the window is specified. This allows the receiver to control which segments, and how fast, the sender can send.

Connection TCP is connection-oriented, which means that a connection between a sender and the receiver must be established before any data can be transfered. A connection is specified by a pair of sockets identifying its two sides. Associated with each connection TCP initialize and maintains some status information for each connection. This includes window size, socket information and sequence numbers.

Protocol Computers supporting TCP has a piece of software, which manages TCP streams and interfaces to the IP layer. Most often this software is a part of the kernel[17]. It accepts data streams from local processes, and breaks them up into pieces, before sending them to the IP layer. The pieces are called TCP segments, which consist of a fixed 20 byte header, followed by zero or more data byte. The TCP software decide how big the segment should be, but for performance reasons they should not exceed the Maximum Transfer Unit (MTU) of the link(the physical network). Each segment should be so small that they can be sent in a single, unfragmented package over the entire network. This usually limits the size of each segment to the MTU of the Ethernet, which is 1500 bytes.

When the TCP software receives data from applications, it is not necessarily sent immediately as it may be buffered before its sent. At the receiver, data is delivered to the TCP software, which reconstructs the original byte streams and deliver them to the target application.

2.5 Protocols of interest

Since previous experiments have shown that employing Web services built on HTTP/TCP breaks down in networks with the DIL characteristics, we're in this thesis looking into alternative protocols. One important limitation however, is that NATO has chosen the "everything over IP", which means that all optimization must ocour on the top of the network layer. Of this

follows that we will evaluate protocols in the transport and application layer of the Internet Protocol Suite.

In the following sections we will give a short introduction to the protocols we're investigating in this thesis. We'll get started by discussing User Datagram Protocol (UDP), which alongside TCP is one of the core protocols of the Internet protocol suite. The remaining protocols discussed in the next section all use either TCP or UDP as their transport protocol.

Skrive noe om hvorfor vi har valgt de protokollene som vi velger å presentere i denne seksjonen.

2.5.1 User Datagram Protocol

The Internet has two main protocols in transport layer, UDP and TCP. They have fundamentally different characteristics and use cases, which we go through in this section. UDP was formally defined in 1980 in RFC 768[18] and is a more simpler protocol than TCP. It sends messages, called datagrams, to nodes over the IP network. While TCP provides reliable transmission along with flow control and congestion control, does UDP only support the sending of IP datagrams. Furthermore it is a connectionless protocol, which means that the protocol can send messages *without* first establishing a connection. Since UDP does not provide guaranteed delivery or in-order delivery of messages, it should only be used by applications that does not require this.

To summarize, UDP is a more lightweight protocol than TCP. It has smaller headers and less overhead, which makes it a faster protocol. The downside is that is does not provide any mechanisms for reliability. However, it is worth to note that applications can build their own reliability on top of UDP. This is done by the next protocol we're looking into.

2.5.2 The Constrained Application Protocol

The Constrained Application Protocol (CoAP) is a specialized Web transfer protocol standardized in June 2014 designed for use with constrained nodes and networks[19]. It is designed for machine-to-machine applications, typically in the Internet of Things. Furthermore it is designed with an similar interface as HTTP, in order to easily integrate with Web services. CoAP is based on the REST model, where the server makes resources available under a resource identifier(URI). Clients access these resources using the HTTP-verbs GET, PUT, POST and DELETE. CoAP main features includes:

- UDP transport with optional reliability supporting unicast and multicast requests.
- Asynchronous message exchanges.
- A stateless HTTP mapping, allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way or for HTTP simple interfaces to be realized alternatively over CoAP.

- Low header overhead and parsing complexity.

CoAP works similar to HTTP in the way that they use a client-server interaction model. CoAP requests is sent from a client to request an action on a resource located on a server. The server then responds with a response code and a possible response body. Unlike HTTP which uses TCP as its transport protocol, CoAP uses UDP. Since UDP does not guarantee delivery, CoAP provide mechanisms for optional reliability.

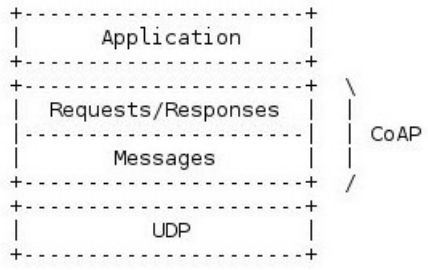

Figure 2.2: Overview of CoAP

2.5.3 Advanced Message Queuing Protocol

Advanced Message Queuing Protocol (AMQP) is an application layer protocol for sending messages. It support both request/response and the publish/subscribe communication paradigms. The protocol is based on the concept of messaging queues, where information producers sends messages into message queues, while information consumers fetch information from queues. A overview of the architecture can be seen in fig. 2.3. Exchanges are the entities that accepts messages from a producer and inserts it into a suitable queue. The implementation of exchanges are known as *brokers*, which runs as a separate decoupled application. AMQP uses TCP for reliable delivery.

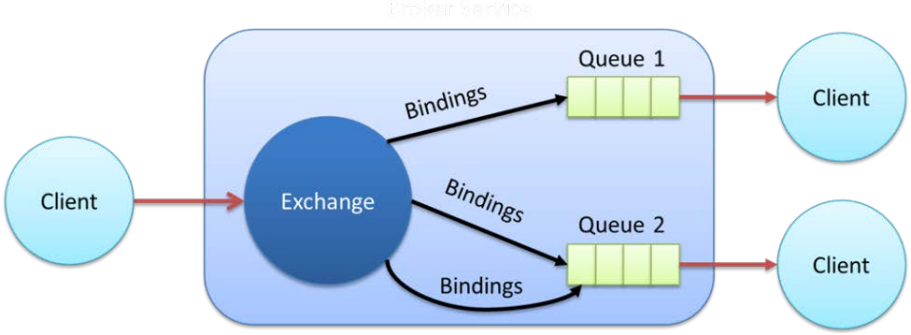

Figure 2.3: Overview of AMQP

An important observation about AMQP is that it has two major versions which are fundamentally different, version 0.9.1 and 1.0. The latter has

been standardized by OASIS[20], and is a more narrow protocol as it only defines the network wire-level protocol for the exchange of messages between two endpoints. Wire-level protocols refers to the means for a application running at one machine to communicate with an application running at another machine using the Internet. Another difference between the versions is that version 1.0 does not specify the details of broker implementation.

Next, we will look into another, more lightweight, publish/subscribe protocol.

2.5.4 MQTT

Like AMQP is MQTT a publish/subscribe messaging transport protocol [21]. However, it is considered to be much more lightweight and is designed for use in networks where the bandwidth is limited. It is also broker-based, where the broker is responsible for delivering messages to clients based on the topic of a message. MQTT run over the TCP/IP protocols.

2.5.5 Stream Control Transmission Protocol

Stream Control Transmission Protocol (SCTP) is transport-layer protocol, which offers functionality from both UDP and TCP[22]. The motivation behind the protocol was that many developers found TCP too limiting, but still required more reliability that UDP could provide. SCTP tries to solve these issues. It is message-oriented like UDP, but ensure reliable, in sequence transport of messages with congestion control like TCP. SCTP is a connection-oriented protocol and provide features like multi-homing and multi-streaming. Multi-homing is the possibility to use more than one network path between two nodes. This increases reliability since if one path fails, messages can still be sent over the other link(s). Multi-streaming refers to SCTP ability to transmit several independent streams of data at the same time, for example sending an image at the same time as a HTML Web page.

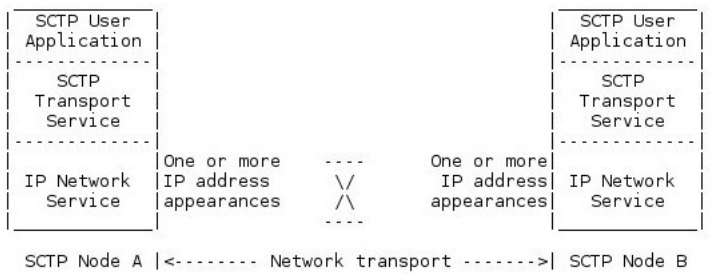

Figure 2.4: Overview of SCTP

2.6 Performance testing

To determine which optimization techniques that have a positive effect on the performance of Web services in DIL environments, we can do performance testing.

2.6.1 Network metrics

Network metrics are used to describe various aspects of data transfer from a point to another.

Link throughput The link throughput is influenced by how large distance there is between the units communicating.

Link reliability How much of the arriving data that is correct. This is called *bit error rate* or *packet error rate*. With high error rates, more data to be transmitted again due to the data arriving being incorrect. This contributes to longer transmission time. In a military setting, an enemy may deliberate sabotage the network with jamming, causing higher error rates.

Link latency The communication technology in use influences how fast data transmission can be done. Long delay may cause that the application sending data timing out.

2.7 Summary

In this chapter we have presented computer networks in general, before we discussed the two most common type of Web services. Moreover, we have discussed the protocols that these Web services use in order to transmit messages over the Internet. We also introduced some new protocols designed to work in "Internet of things" networks, which have many of the same characteristics as DIL networks. Finally, we introduced the concept of performance testing and important network metrics when doing such testing.

Many of the mentioned protocols has been previously researched for use in DIL networks. In the next chapter we will present relevant work in this area.

Protocol	Network layer	Summary
HTTP	Application. Uses TCP.	Widely used. Breaks down in DIL environments.
TCP	Transport	The well-known transport protocol.
UDP	Transport	Lacks reliability, but frameworks exist that provides it.
CoAP	Application. Uses UDP.	Designed for use in the Internet of Things.
AMQP	Application. Uses TCP.	Messaging middleware with store-and-forward capabilities.
MQTT	Application. Uses TCP	Pub/sub designed for use in the Internet of Things
SCTP	Transport	Similar to UDP but also provide reliable, in sequence transport of messages like TCP.

Table 2.4: Summary of protocols

Chapter 3

Related Work

In this chapter we will discuss earlier relevant work in the area of improving the performance of Web services in DIL environments. Improving Web services is critical for both civil and military users as increasing the performance means that applications become faster, less money needs to be put into network bandwidth. From this comes that quite amount of research has been done in the area of optimizing network applications. Improving the performance of Web services in DIL environments has been explored, but mostly for SOAP-based Web services and not REST.

In the following sections we identify results and recommendations that are applicable to this thesis. We get started by presenting compression, the obvious technique of reducing the size of data being sent over the network.

3.1 Compression

Quite an amount of research has been done in the area of compression. Data compression is the technique of encoding information using fewer bits than the original representation. The goal is to reduce data transmission time or the storage requirements. We divide compression lossy and lossless compression. Lossy compression is used to compress data such as images and movies where the consequence of loosing some of the data is not critical. Lossless compression utilize repeating patterns in the data in order to represents the same data in a more efficient way.

XML is the data format used by Web services and has a significant overhead. Previous studies have evaluated different compressions algorithms for Web services. Previous experiments shows EFX has the best compression results with GZIP as the second best alternative[23].

3.2 Making SOA applicable at the tactical level

In the report IST-090, a task group investigated solutions for making SOA applicable at the tactical level. As a follow-on to IST-090, the research group IST-118 was created with the goal to create a recommendation for using SOA in DIL networks. In the paper IST-118 they summarized the

findings of IST-090. Although the papers only looked into W3C Web services, many of their recommendations are also applicable to RESTful Web services. They identified three key issues that needs to be addressed in order to apply Web services in tactical networks[4, 24]:

End-to-end connections

Web services depend on a direct, end-to-end connection between the client and the service. Attempting to establish and maintaining connections in DIL environments can lead to increased communication overhead and possible complete breakdown of communication. Most Web services use TCP as the transport protocol, which is a connection-oriented protocol designed for wired networks. In DIL environments with high error rates and high latencies, the congestion control of TCP will cause sub-optimal utilization of the network due to frequent connection timeouts. Similar, HTTP, which is the application layer protocol most often used together with TCP, struggles in such environments. HTTP is a synchronous protocol, which means that the HTTP connection is kept open until a response is received. Long response times cause timeouts. IST-090 points out the obvious solution to replace HTTP and TCP with other, more suitable protocols.

The report[4] mentions two approaches to replace HTTP/TCP. The clients and services themselves can be modified to support other protocols, or proxies which support alternative protocols can be used. With employing a proxy solution, standards compliance can be retained.

Network heterogeneity

Another issue is when heterogeneous networks are interconnected. Different performance in networks may lead to buildup of data in buffers, risking loss of information. A proposed solution to this is to have store-and-forward support, which can support that messages are not dropped, but stored and forwarded when possible.

Web service overhead

W3C Web services are associated with a considerable amount of overhead. Web Service technology is based on SOAP, which use XML-based messages. It is a textual data format and produce much larger messages than binary formats. Optimization approaches should seek to reduce the network traffic generated by Web services by using techniques as compression to reduce the size of messages. Another approach is to reduce the number of messages being sent, which was looked into in IST-090[4]. In their work they investigated three different ways to do this:

1. Employing caching near the client in order to reuse older messages.

2. Using publish/subscribe paradigm, which allows clients to subscribe to information instead of requesting it. This allows the same message to be sent to multiple clients.

3. Employing content filtering, which filters out unnecessary data.

3.3 Previous evaluations of alternative protocols

Previous studies have investigated potential gains from replacing HTTP/TCP with alternative protocols [25]. They looked into how TCP, UDP, SCTP and AMQP for conveying Web services traffic under typical military networking conditions. The researchers found that SCTP had the highest success rate in military tactical communication. However, on the lower bandwidth links the protocols tends to generate more overhead than TCP. They pointed out that this was due to SCTP having a more complex connection handshake procedure and in addition use heartbeat packets.

3.4 Tuning application server parameters

When setting up an application server, several parameters which can affect the performance of Web services running on the application server can be configured. Wrong or bad configuration may cause inaccurate timeouts and congestion in the network. In a paper written by researchers at Norwegian University of Science and Technology (NTNU) and FFI, investigated how such tuning affected the performance of Web services in different emulated tactical networks[14]. In their study they investigated how tuning server parameters affect the performs of both REST and SOAP Web services. They identified a number of key HTTP and TCP tuning parameters:

HTTP Timeout Controls how long a HTTP connection can be deemed as idle and kept in the "keep-alive" state. Having a to low timeout on networks with low bandwidth, can potentially flood the network with packets that have timed out. Consideration should therefor be taken when setting setting this parameter for mobile tactical networks.

HTTP Compression Enables HTTP/1.1 GZIP compression.

HTTP Chunking Allows the server to send data in dynamic chunks.

HTTP Header and Send Buffer Sizes Can vary the size of the buffers that hold the request and send data, respectively.

TCP Idle Key Timeout Sets the time before an idle TCP channel closes.

TCP Read and Write Timeouts Set the timeout for TCP read and write operations, respectively.

TCP Selector Poll Timeout Sets the time a Java new/non-blocking I/O (NIO) selector will block waiting for user requests.

TCP Buffer Size Sets the size of the buffer that holds input streams created by the network listener.

TCP Batching/TCP NO_DELAY Batches together small TCP packets into larger packets.

MTU Size The maximum transmission unit size regulates the largest data unit that can be passed onwards. In tactical military communication the MTU size can be very low(down to 128 bytes).

After running their experiments they concluded that few of the parameters actually had any significant impact on the performance of the Web Service. However, they identified HTTP Chunking configuration as having the most impact on the performance. It significantly improved the performance in different types of networks and for both SOAP and RESTful Web services.

3.5 Proxy optimization

One of the recommendations of IST-090 was the usage of proxies. This recommendation has been picked-up by other research group and a set of proxies for optimizing Web services in DIL networks already exist. However, many of them does not fulfill all the requirements we have for our proxy. Some of them does only support SOAP Web services and others are unusable due to security reasons. This section lists and discuss previous implementations of such proxies.

3.5.1 DSProxy

DSProxy is a proxy solution developed by Norwegian Defence Research Establishment (FFI), which transports SOAP messages over DIL networks[26]. It reduces bandwidth needs by employing different optimization techniques such as compression. DSProxy also provides delay tolerance, which allows COTS clients to function in DIL networks.

The downside with DSProxy is that it only support SOAP, which leaves RESTful Web services out of the picture.

3.5.2 AFRO

Adaption Framework foR Web Services prOvision (AFRO) is an edge proxy which offers different levels of Quality of Service (QoS) to Web services through performance monitoring and application of the context-aware service provision paradigm[4]. It perform so called adaption actions, which modifies the SOAP XML messages by changing their encoding to more efficient data representation. It also cuts out information that is accepted to be removed by the service requester.

However, since the proxy modifies the data being sent, the checksum of the data is also changed. In applications where we want to be sure that no one has tampered with the data before arriving, checksums are often used. Therefore this solution would not work for such applications.

3.5.3 Suri

To be done.

3.6 Summary

In this chapter we looked into efforts previously undergone in order to improve the performance of Web services in networks with the DIL characteristics. We identified compression with EFX as proven technique to reduce the size of messages sent over the network. Next, we looked into the paper IST-090 and the challenges that comes with DIL networks. We saw how IST-090 pointed to the usage of proxies. Furthermore, we investigated previous attempts with the usage of alternative transport protocols, before we looked into previous efforts in the area of tuning application server parameters. Finally, we discussed previously developed proxies for DIL networks and discovered that they do not fulfill all the requirements that we have. Proxies previously created are either limited to SOAP-based Web services or are inadequate to be used due to security reasons.

Protocol Stack	**optimization possibilities**
The application	Optimize the application
Web service messaging: SOAP	Optimize SOAP, e.g XML compression
HTTP/TCP, UDP or other transport protocols	SOAP is transport agnostic. Other protocols can be used.
IP	NATO NEC feasibility study states that all protocols should be over IP.
Lower layers	Not in the scope of this thesis.

Table 3.1: Optimization possibilities.

Chapter 4

Requirement Analysis

In this chapter we discuss the requirements for the proxy being developed as a part of this thesis. Those requirements builds on the scope and premises discussed in the introduction. To recap, these defining requirements were:

1. Support both REST and Web service communication between machines connected in a DIL network.
2. Work on top of the IP-layer.
3. All optimization techniques must be placed in a proxy, and not in the Web service applications themselves.

4.1 A DIL HTTP Proxy

The first requirement implies that our proxy must accept HTTP, as this is the far most used Web service protocol. Our proxy must be able to accept a HTTP requests from a Web service, forward it to the other proxy, which in turn delivers it to the intended receiver. The communication between the proxies are not required to be HTTP, but rather a protocol than deals with DIL networks in a better way. However, since ultimately a HTTP request should be delivered to the intended receiver, the HTTP properties must be retained. This means that the proxy must preserve the HTTP Method and HTTP headers. Also, since REST is payload agnostic, the proxy must be able to support different types of data being sent through it(XML, JSON etc.).

Furthermore, the proxy must be able to handle the difficult network conditions of DIL. The specific requirements are outlined in the following sections.

4.1.1 Disconnected

Support disconnects over a longer period of time. Previous work identified the removal of end-to-end dependencies as important. By employing proxies, the end-to-end dependency is instead between the client and the

proxy locally. However, the connection between the proxies over a DIL network can still be lost. This means that the proxy must be able to maintain the connection with the local application, while managing loss of connection with the other proxy. When the connection are reestablished, the proxy should continue sending the data and finally delivering a response back to the client.

4.1.2 Intermittent

Handle brief disconnects. Same requirements as for disconnected. The proxy should "hide" disconnects from the client.

4.1.3 Limited

It must handle very low data rates.

4.2 Support optimization techniques

4.2.1 Compression

In order to perform compression the proxy must be able to modify the payload of the message. Due to security mechanisms that detect changes to the payload(checksums), the payload must be restored back to its original form before being forwarded to the final receiver.

4.2.2 Proxy protocol communication

One of the optimization techniques identified was the usage of alternative protocols. The proxy should therefor support a sub set of them, and be easily configured to use other protocols for testing purposes.

4.3 Other considerations

Since we're creating a proxy aimed for use in a military context, military operational requirements are part of the requirements for this proxy. Mobile units have to carry batteries with them and the capacity is therefore limited. Advanced compression techniques may reduce the overhead, but also requires more battery. This trade-off needs to be considered.

4.4 Summary

In this chapter we have discussed the requirements for our proxy, which are summarized in table 4.1. Next we discuss the design and implementation of our proxy.

Requirement	Priority
Receive and forward HTTP requests	1
Allow modifications on the payload	1
Retain the checksum of the payload	1
Allow configuration of HTTP timeouts	1
Keep HTTP-connection alive	1
Support protocol X and y	2
Handle very low data rate	1
Have store-and-forward capabilities	1
Handle frequent disconnects	1
Handle disconnects over longer periodes of time	1

Table 4.1: Summary of proxy requirements

Chapter 5

Design and Implementation

In this chapter we will introduce the design and implementation details of our proxy solution for improving the performance of Web services in DIL environments. We will first look into the overall design before we dive into the details.

5.1 Overall Design

The proposed design in this thesis includes a proxy pair.

5.2 Proxy

5.2.1 Apache Camel

This does the work of converting the specific input/request (like an HTTP request) into something generic - a Camel Exchange - that can travel down a Route. - quoute. må skrives om.

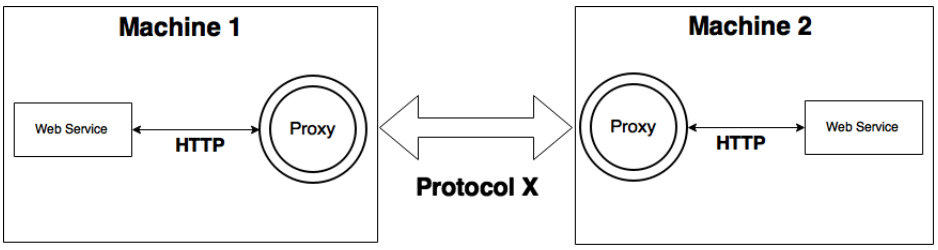

Figure 5.1: Architectural overview of proposed design

5.3 Summary

Chapter 6

Testing and Evaluation

6.1 Introduction

In order to answer our research questions of how to improve the performance of Web services in DIL environments, we developed test setups and environments to evaluate the proxy. The goal is to measure any possible improvements(or deterioration) of performance when we use the proxy developed as a part of this thesis. In this chapter we present how the testing was performed and finally present the results we obtained. Since the proxy is being developed as a prototype for military usage, we wanted to use test scenarios that resembles actual military and civilian usage. While W3C services only uses HTTP as a transport mechanism, REST utilizes the different HTTP methods to indicate which operation to perform on a resource. Each test scenario is therefor performed with both a W3C Web service applications and RESTful Web services.

To evaluate how different network properties affects performance, the tests was performed on networks with different characteristics. The base case was to test without any intentional limitations to the network and without the actual usage of the proxy. Then we introduced usage of the proxy and evaluated it in different types of networks. An infinite number of possible network combinations exists, so we have in this thesis chosen to focus on five different network types identified by the task group IST-118 for DIL-testing. The different networks are summarized in table 6.8.

Furthermore we performed tests with two setups, first with machine-to-machine over an Ethernet cable, then over actual military communication equipment. The usage of actual military equipment allowed us to get as realistic results as possible.

6.2 Evaluation Tools

In order to simulate DIL environments we need some way to control the properties of the network traffic. In this thesis we use two approaches, the first one connecting two machines through a third machine. The third machine will use a component of the linux kernel to control the flow of the network traffic flowing through it, allowing us to simulate a DIL

network. The second approach involves using actual military equipment in laboratory at FFI. The benefit of using actual equipment, is that we get as realistic tests as possible.

6.2.1 Network Emulator

Fortunately, the Linux kernel offers a rich set of tools for managing and manipulating the transmission of packets. Network Emulator (NetEm) is an enchancement of the traffic control facilities that allows us to control delay, packet loss and other characteristics to packets outgoing from a selected network interface. Figure 6.1 illustrates the setup of this approach.

tc(traffic control) is a linux program to configure and control the linux kernels Network scheduler. This program allows us to emulate many of the network characteristics that DIL has and the different setups are explained in the coming sections.

Delays

NetEm can emulate delays on packets on a specific link.

Listing 6.1: "Emulating delay"

```
tc qdisc add dev etho root netem delay 100ms
```

In this example we add a fixed delay on 100 ms to all packets going out of local Ethernet.

6.3 Test Setup

The majority of testing was performed on machines located at the FFI-lab at Kjeller. Although testing on regular machines gives us a indication, to get as realistic results as possible we also performed tests on military communication equipment.

The Web service and client are connected to each other through a third computer, acting as a router. This router machine had two network cards and networked together the other machines by Ethernet cables. In order for the router machine to forward IP packets back and forth between the client and server, IP forwarding was enabled on the kernel. The specifications of the machines involded in the testing are listed in table 6.1.

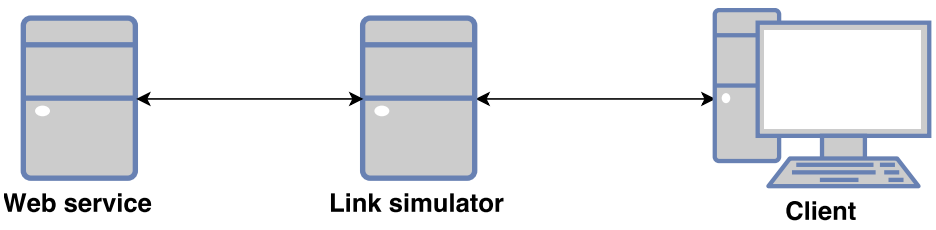

Figure 6.1: Testing environment

Role	OS	Kernel version
Client	Debian	2.16
Web service	Ubuntu	2.15
Router	Ubuntu	2.15

Table 6.1: Machines involved in the testing

The server and client are assigned an IP address in two different subnets. This is done by the Linux network interface administration program *ifconfig*. In listing 6.2 the client machine is is assigned the IP address 192.168.2.44.

Listing 6.2: "Configuring a network interface of the router"

```
ifconfig eth0 192.168.2.1 up
```

After setting up the IP addresses we need to configure the routing so that the kernel know where to route the network traffic. In this case we want all traffic to go through the routing machine. In listing 6.3 we configure all IP traffic boud for the subnet 192.168.1.X to be routed through the router machine with IP 192.168.2.1.

Listing 6.3: "Configuring routing rules for the client"

```
ip route add unicast 192.168.1.0/24 via 192.168.2.1
```

6.3.1 Enabling proxies

In order to enable the applications to tunnel all their HTTP traffic through our proxy, we needed a way to setup a proxy without altering the applications themselves. Fortunately, Java provide mechanisms to deal with proxies[27]. We configured the Java Virtual Machine (JVM) to get the applications to tunnel all HTTP traffic through our proxy. This is done by setting properties to the JVM:

Listing 6.4: "Setting a proxy on the JVM"

```
java -Dhttp.proxyHost=localhost \
-Dhttp.proxyPort=3001 \
-Dhttp.nonProxyHosts= \
-jar target/client.jar
```

In listing 6.4 the application **client.jar** is started and all HTTP traffic will go through the proxy server at localhost on port 3001.

6.3.2 Emulating networks

Since all network traffic passes through the routing machine, we can control the flow of IP packets here. As previously discussed, we use Netem. For each network configuration, a bash script is run. This script configures the network interfaces in order to get the correct network behaviour. Both interfaces are configured so the network is symmetrical in both directions.

6.4 Test Execution

In our tests we use two different sets of applications. One for W3C Web services and one for RESTful Web services. Data being sent between the client and server is sent uncompressed.

6.4.1 W3C Web service test applications

For the purpose of testing W3C Web service applications we simulated a system which allows units in the field to report positions of friendly forces back to a headquarter. The position reports use the NATO Friendly Force Information (NFFI) format, which has an associated XML schema with it.

The client requests these data over the network. This is repeated x times and the the average round-trip time is calculated. The test service was deployed with Glassfish 4, an application server. The client was ran through the IDEA Netbeans.

6.4.2 RESTful Web service test applicatons

For testing we developed a small RESTful Web service, an example service keeping order of cars in a "car system". The service exposes an Application Program Interface (API) which offers different operations to manage the car system. Clients can invoke these operations by using HTTP requests and utilizing the associated HTTP method to indicate what to do with an resource. Since RESTful services are payload agnostic, we choose JSON to represent the data being sent between the server and the client. JSON is a lightweight data-format.

Each test case consist of a client sequentially invoking the server with different API requests. The most common HTTP-methods GET, PUT, POST, and DELETE are all part of the testing.

6.4.3 Test parameters

- Compression on/off.
- Proxy on/off.

6.4.4 Testing on military communication equipment

Kongsberg radio.

6.5 Function tests

The first phase of the testing was performed without any actual intended limitations to the network. The objective of this testing is to validate that the proxy is working correctly and have a benchmark to compare other results with. This phase was again divided into two phases, one without

the usage of proxy and one with the usage of it. This allows us to investigate any potential overhead associated with the usage of the proxy.

6.5.1 Execution

The Web service client and the service itself was started on separately machines interconnected through a third machines acting as a router as discussed in section 6.3. However for the first phase, the client and server did not use any proxy.

Warm-up.

Number of tests.

6.5.2 Results and Analysis

Enabling compression yields an improvement in the performance. We also notice that HTTP and CoAP has a almost identical performance, while AMQP has significant longer RTT.

Test	Avg. RTT	With compression
Without proxy	117 ms	N/A
Proxy with HTTP	148 ms	93 ms
Proxy with AMQP	631 ms	509 ms
Proxy with CoAP	255 ms	102 ms

Table 6.2: W3C Web service results

Test	Transfer time	With compression
Without proxy	1000 ms	N/A
Proxy with HTTP	1200 ms	1000
Proxy with AMQP	1200 ms	1000
Proxy with CoAP	1200 ms	1000

Table 6.3: RESTful Web service results

6.6 DIL Tests - Disconnected

In this scenario we evaluate the performance with the DIL characteristic *disconnected*, which refers to the network suddenly going down when the application is sending data. The objective of this testing is to evaluate how the proxy manages disconnects over longer periods of time. We define the success criteria for this test to be that the client is able to eventually process his request after the connection is reestablished. The client HTTP request should not be interrupted in any way, other than it taking longer time to process the request.

6.6.1 Execution

The tests are performed on a unlimited network. During testing we physically remove the Ethernet cable during transmission. 60 seconds?

6.6.2 Results

Test	Result
Without proxy	Timeout
Proxy with HTTP	Success
Proxy with AMQP	Success
Proxy with CoAP	Success

Table 6.4: W3C Web service results

Test	Result
Without proxy	Timeout
Proxy with HTTP	Success
Proxy with AMQP	Success
Proxy with CoAP	Success

Table 6.5: RESTful Web service results

6.7 DIL Tests - Intermittent

Intermittent refers to the network connection being lost, but then regained again. The objective of this testing is to evaluate how the proxy manages frequent temporary loss of connections. The success criteria is the same as for disconnected, the client should not notice any disruption of service.

6.7.1 Execution

Make script for handling this? Disconnects which lasts ms and seconds.

6.7.2 Results

Test	10 sec	30 sec	60 sec
Without proxy	Timeout	Timeout	Timeout
Proxy with HTTP	1400 ms	40000	Timeout
Proxy with AMQP	1400 ms	40000	Timeout
Proxy with CoAP	1400 ms	40000	Timeout

Table 6.6: W3C Web service results

Test	10 sec	30 sec	60 sec
Without proxy	10000 ms	Timeout	Timeout
Proxy with HTTP	1200 ms	12000	60000
Proxy with AMQP	1200 ms	12000	60000
Proxy with CoAP	1200 ms	12000	60000

Table 6.7: RESTful Web service results

6.8 DIL Tests - Limited

The third DIL characteristic, *limited*, refers to different ways a network can be limited. This includes high delays, packet loss and low bandwidth. The different types of networks seeks to emulate properties of actual communication devices used by the military. These include satellite link networks(SATCOM), line-of-sight(LOS), combat network radio(CNR) and WIFI. Wifi is divided into two types to illustrate both with good connection and one with less. We also investigate Long-Term Evolution (LTE), commonly known as 4G, a network technology which has become in widespread use in the latest years. The reason for including LTE in addition to the ones from IST-118, is that the Norwegian Defense is looking into the possibility of using LTE. Thus making it interesting for us to investigate the performance under this type of network as well.

The metrics we use to characterize a network is data rate, delay and Packet Error Rate (PER). The term data rate refers to the amount of data that can be transmitted over a network per unit of time. Delay refers to the time it takes for a bit of data to travel across the network from machine to machine. PER means the number of incorrectly received packets divided by the total number of received packets. A packet is incorrect if at least one bit is transmitted erroneous. This is used to simulate a network which transmits corrupt, lost or duplicate packets.

Network	Data Rate	Delay	PER
Satellite Communication	250 kbps	550 ms	0 %
Line of Sight	2 mbps	5 ms	0 %
Wireless Fidelity (WiFi) 1	2 mbps	100 ms	1 %
WiFi 2	2 mbps	100 ms	20 %
Combat Net Radio with Forward Error Correction	9.6 kbps	100 ms	1 %
LTE (4G)	20 mbps	100 ms	1 %

Table 6.8: Different network types

In the following sections the test cases is ran for each network configuration.

6.8.1 Satellite communication

In this test scenario we emulate Satellite Communcation (SATCOM). Low data rate, high delay.

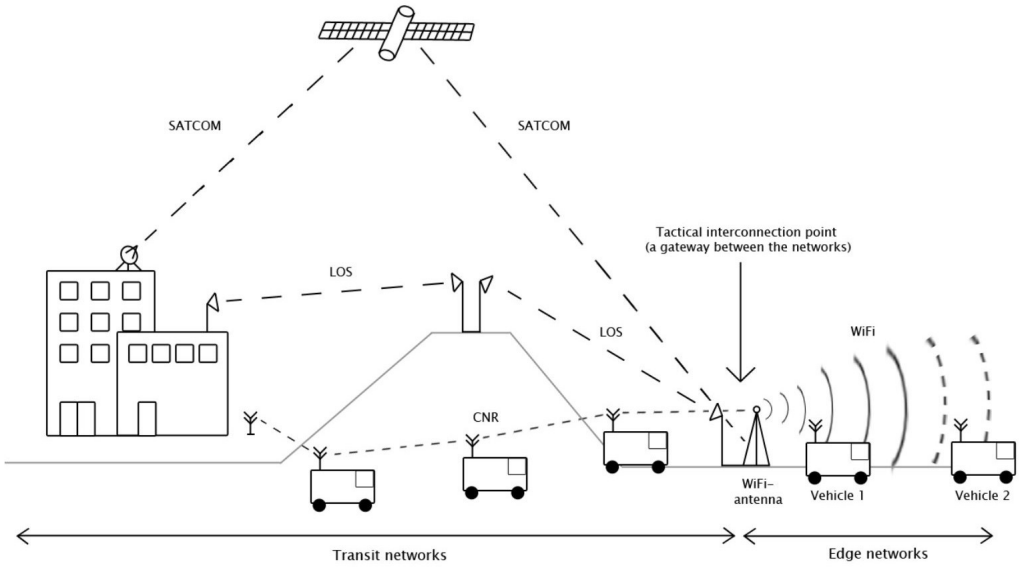

Figure 6.2: Overview of tested networks

Test	Transfer time	With compression
Without proxy	6235 ms	N/A ms
Proxy with HTTP	X ms	3504 ms
Proxy with AMQP	X ms	24414 ms
Proxy with CoAP	X ms	11210 ms

Table 6.9: RESTful Web service results

Results and analysis

6.8.2 Line-of-Sight

In this test scenario we emulate so-called Line of Sight (LOS) networks, which are characterized by being a radio-based type of network with no physical obstacles between the nodes in the network. High data rate, low delay and zero error.

6.8.3 WiFi 1

Placeholder.

6.8.4 WiFi 2

Placeholder

Test	Transfer time	With compression
Without proxy	6235 ms	N/A ms
Proxy with HTTP	X ms	3504 ms
Proxy with AMQP	X ms	24414 ms
Proxy with CoAP	X ms	11210 ms

Table 6.10: RESTful Web service results

6.8.5 Combat Net Radio with Forward Error Correction

6.9 Summary

In this section the results from the tests are presented. These results lead up to the discussion and conclusion in the next chapter.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

7.2 Future Work

Bibliography

[1] P. Bartolomasi et al. *NATO network enabled capability feasibility study*. 2005.

[2] NATO. *NATO - Member Countries*. http://www.nato.int/cps/en/natohq/nato_countries.htm. Accessed: 2015-05-04.

[3] OASIS et al. *Reference Model for Service Oriented Architecture 1.0 OASIS standard*. http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf. Accessed: 06. 10. 2015. Oct. 2006.

[4] F. Annunziata et al. *IST-090 SOA challenges for disadvantaged grids*. https://www.cso.nato.int/pubs/rdp.asp?RDP=STO-TR-IST-090. Apr. 2014.

[5] NATO C3 Board. *Core Enterprise Services Standards Recommendations - The SOA Baseline Profile*. 1.7. 2011.

[6] Frank T. Johnsen. "Pervasive Web Services Discovery and Invocation in Military Networks". In: *FFI-rapport 2011/00257* (2011).

[7] A. Gibb et al. "Information Management over Disadvantaged Grids". In: *Task Group IST-030/ RTG-012, RTO-TR-IST-030* (2007). Final report of the RTO Information Systems Technology Panel.

[8] R. Braden. *RFC 1122 – Requirements for Internet Hosts – Communication Layers*. https://tools.ietf.org/html/rfc1122. Accessed: 06. 01. 2016. Oct. 1989.

[9] Hugo Haas and Allen Brown. *Web Services Glossary*. http://www.w3.org/TR/ws-gloss/\#webservice. Accessed: 2015-05-06.

[10] W3C. *Extensible markup language (XML) 1.0*. Nov. 2008. URL: https://www.w3.org/TR/REC-xml/ (visited on 02/25/2016).

[11] Erik Christensen et al. *W3C - Web service definition language (WSDL)*. Mar. 2001. URL: https://www.w3.org/TR/wsdl (visited on 02/27/2016).

[12] Martin Gudgin et al. *W3C - SOAP version 1.2 part 1: Messaging framework (Second edition)*. Apr. 2007. URL: https://www.w3.org/TR/soap12-part1/ (visited on 02/27/2016).

[13] Roy T. Fielding and Richard N. Taylor. “Principled Design of the Modern Web Architecture”. In: *Proceedings of the 22Nd International Conference on Software Engineering*. ICSE ’00. Limerick, Ireland: ACM, 2000, pp. 407–416. ISBN: 1-58113-206-9. DOI: 10.1145/337180.337228. URL: http://doi.acm.org/10.1145/337180.337228.

[14] Frank. T Johnsen and Trude Bloebaum. “Recommendations for increased efficiency of Web services in the tactical domain”. In: ().

[15] R. Fielding et al. *RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1*. https://tools.ietf.org/html/rfc2616. Accessed: 10. 02. 2016. June 1999.

[16] Information Sciences Institute - University of Southern California. *RFC 793 – Transmission Control Protocol*. https://tools.ietf.org/html/rfc793. Accessed: 10. 02. 2016. Sept. 1981.

[17] David J. Wetherall Andrew S. Tanenbaum. *Computer Networks*. Fifth Edition. Pearson New International Edition.

[18] J Postel. *RFC 768 - User Datagram protocol*. Aug. 1980. URL: https://tools.ietf.org/html/rfc768 (visited on 02/28/2016).

[19] Z. Shelby et al. *RFC 7252 – The Constrained Application Protocol (CoAP)*. https://tools.ietf.org/html/rfc7252. Accessed: 10. 02. 2016. June 2014.

[20] OASIS. *Advanced message queuing protocol (AMQP) version 1.0*. Oct. 2012. URL: http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html%5C#toc (visited on 02/28/2016).

[21] OASIS, Andrew Banks, and Rahul Gupta. *MQTT Version 3.1.1 Specification*. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html. Accessed: 06. 01. 2016. Oct. 2014.

[22] R Stewart. *RFC 4960 - Stream control transmission protocol*. Sept. 2007. URL: https://tools.ietf.org/html/rfc4960 (visited on 02/29/2016).

[23] Frank T. Johnsen and Trude Bloebaum. “Using NFFI Web Services on the tactical level: An evaluation of compression techniques”. In: 13th International Command and Control Research and Technology Symposium (ICCRTS). Seatle, WA, USA, 2008.

[24] Frank T. Johnsen et al. “IST-118 - SOA recommendations for Disadvantaged Grids in the Tactical Domain”. In: *18th ICCRTS* (2013).

[25] Frank T. Johnsen et al. “Evaluation of Transport Protocols for Web Services”. In: *MCC 2013* (2013).

[26] Ketil Lund et al. “Robust Web Services in Heterogeneous Military Networks”. In: *IEEE Communications Magazine, Special Issue on Military Communications* (Oct. 2010).

[27] Oracle. *Java Networking and Proxies*. https://docs.oracle.com/javase/8/docs/technotes/guides/net/proxies.html.

Acronyms

AFRO Adaption Framework foR Web Services prOvision. 35

AMQP Advanced Message Queuing Protocol. 27

API Application Program Interface. 46

CoAP The Constrained Application Protocol. 26, 27

COTS Commercial off-the-shelf. 17, 34

DIL Disconnected, Intermittent and Limited. 14, 15, 17, 31, 43

FFI Norwegian Defence Research Establishment. 34

FTP File Transfer Protocol. 20

HTTP Hypertext Transfer Protocol. 17, 20, 23

IETF Internet Engineering Task Force. 23

IP Internet Protocol. 20, 26

JVM Java Virtual Machine. 45

LOS Line of Sight. 50

LTE Long-Term Evolution. 49

MTU Maximum Transfer Unit. 25

NATO North Atlantic Treaty Organization. 11

NEC Network Enabled Capability. 11

NetEm Network Emulator. 44

NFFI NATO Friendly Force Information. 46

NIO Java new/non-blocking I/O. 34

NTNU Norwegian University of Science and Technology. 33

OASIS Organization for the Advancement of Structured Information Standards. 12, 27

PER Packet Error Rate. 49

QoS Quality of Service. 35

REST Representational State Transfer. 13, 20, 22, 33

SATCOM Satellite Communcation. 49

SCTP Stream Control Transmission Protocol. 28, 33

SOA Service Oriented Architecture. 11–13, 21, 31

TCP Transmission Control Protocol. 20, 24, 26, 28

UDP User Datagram Protocol. 26–28

W3C World Wide Web Consortium. 20, 23

WSDL Web Services Description Language. 21

XML Extensible Markup Language. 21, 31