Reducing Software Complexity in a Distributed Publish-Subscribe system using Multicast communication

ARAXI MEKHITARIAN
Abstract

Systems of distributed character are increasing in size and becoming more complex. Managing and adapting to constant changes of requirements is a challenge during the entire system development life cycle. While new functionalities are implemented, the software may change in design and may lead to poor software quality and increased system complexity.

This thesis focuses on tackling the complexity issue in a distributed Electronic Warfare system used in military aircraft. The system consists of a server and several clients which acts as publishers or subscribers for different events sent in the system. The communication is based on unicast and uses a publish/subscribe pattern for the client nodes to register as publishers or subscribers to the server. The system is created to handle message passing in high rate and is sensitive for message delays. Due to this, the system is dependent on a reliable network structure with a continuous necessity for development.

An implementation of a multicast prototype will be replacing the topic-specific unicast communication and the publish/subscribe registration process to the server. The system will be evaluated by a comparison of the old communication version with the new multicast implementation using software metrics. The result is to evaluate if the behavior and functionality of the distributed Electronic Warfare system change.

Keywords

Distributed systems, Publish/Subscribe, Electronic Warfare, Multicast, Software complexity
Sammanfattning

System av distribuerad karaktär ökar i storlek och blir alltmer komplex. Att hantera och anpassa sig till ständiga kravändringar är emellertid en utmaning under hela systemets utvecklingsprocess. Medan nya funktioner implementeras kan mjukvaran ändras i design vilket kan leda till dålig programkvalitet och ökad systemkomplexitet.

Denna rapport fokuserar på att hantera komplexiteten i ett distribuerat telekrigföringssystem som används i militära flygplan. Systemet består av en server och flera klienter som publicerar och prenumererar för olika typer av meddelanden som skickas i systemet. Kommunikationen baseras på enkelsändning och använder ett publish/subscribe meddelandemönster där klienterna registrerar sig som publicerare eller prenumeranter till servern. Systemet är skapat för att kunna hantera höga meddelandehastigheter och har låg tolerans för meddelandeförseningar. På grund av detta är tillförlitlighet i nätverksstrukturen ett essentiellt kvalitetsattribut då nätverket är i behov av en ständig utveckling.


Nyckelord

Distribuerade system, Publish/Subscribe, Telekrigföringssystem, Multisändning, Kodkomplexitet
# Table of Contents

1 Introduction ............................................................................................................. 1
   1.1 Background ........................................................................................................ 1
   1.2 Problem ............................................................................................................. 2
   1.3 Purpose ............................................................................................................. 3
   1.4 Goals ................................................................................................................ 3
      1.4.1 Benefits, Ethics and Sustainability .......................................................... 4
   1.5 Research Methodology ..................................................................................... 4
   1.6 Stakeholder .................................................................................................... 6
   1.7 Delimitations ................................................................................................. 7
   1.8 Outline .......................................................................................................... 7

2 Group communication methods in distributed systems ........................................ 8
   2.1 Unicast, Broadcast, Multicast .......................................................................... 8
   2.2 Multiple unicast versus Multicast .................................................................. 10
   2.3 Broadcast versus Multicast ........................................................................... 10
   2.4 IP Multicast .................................................................................................. 11
      2.4.1 Addressing ............................................................................................... 11
      2.4.2 Group Management ................................................................................. 12
      2.4.3 Routing ................................................................................................... 12

3 Software quality ................................................................................................. 13
   3.1 Achieving good software quality .................................................................... 13
   3.2 Software metrics ............................................................................................ 14
      3.2.1 Lines of code ............................................................................................ 14
      3.2.2 Number of members per type .................................................................... 15
      3.2.3 Number of parameters per function ......................................................... 15
      3.2.4 Efferent coupling ....................................................................................... 15
      3.2.5 Cyclomatic complexity ........................................................................... 15

4 The Electronic Warfare system architecture ..................................................... 17
   4.1 Overview ......................................................................................................... 17
   4.2 Events, domains and switches ....................................................................... 18
   4.3 The server ....................................................................................................... 18
   4.4 The client hosts ............................................................................................. 19
   4.5 The publish-subscribe registration phase ..................................................... 19

5 Method .................................................................................................................. 21
   5.1 Data collection techniques ............................................................................. 21
   5.2 Data analysis techniques ................................................................................. 21
   5.3 Quality assurance ........................................................................................... 22
   5.4 Software development process ....................................................................... 23
      5.4.1 Requirements specification ...................................................................... 23
      5.4.2 Design, Implementation and Test ............................................................ 24
      5.4.3 Evaluation ................................................................................................. 24

6 Software complexity reduction process in the publish-subscribe distributed system ................................................................................................. 25
   6.1 Requirements collection .................................................................................. 25
   6.2 Refactoring process ........................................................................................ 26
1 Introduction

Software has the ability to evolve and transform over time. During the entire system life cycle, software is improved, modified and adapted to new requirements [1]. However, managing the constant change of conditions is challenging. As new functionalities are implemented, the software complexity may increase and the system can change from its original design, potentially resulting in poor software quality. While systems are growing in size, the difficulty level for maintaining code structure and quality rises [2]. The problem occurs when the system reaches a point where there is an overload of functionality in the system, which can lead to duplicity, degradation of code quality and maintainability issues. This makes the balance of development, improvement and complexity of a system a very difficult task.

One approach to manage software complexity and preserve good system quality is by software maintenance. Software maintenance is according to ISO/IEC-24765 defined as; “The process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment” [3]. Maintaining software is a continuous process that begins after the first initial release until software retirement with the aim to achieve clean and easily understood software design. There are several techniques used for software maintenance and one of the most common is called refactoring [4], which is a systematic approach to clean up code and reduce chances of implementation errors. Refactoring incrementally improves software quality by replacing, extracting and moving internal functionality of the existing code without changing the external behavior of the system.

1.1 Background

Identifying and managing the complexity of distributed systems are important aspects of the entire software development process. As with any other system, distributed software continuously need to adjust to new conditions and requirements in order for it to be reliable and useful [2].

A distributed system is a set of independent computers, each containing several processors, that collaborates together to achieve a common goal [5] [6]. The computers have the ability to share resources between the autonomous components in form of message passing through a communication network, which is one of the main reasons for its usage [7]. There are mainly three different ways resources can be transferred from a host over a communication network in a distributed system; one-to-one communication (unicast), one-to-all communication (broadcast) and one-to-many communication (multicast) [8]. Due to the independence of the nodes, in order to obtain a well-functioning distributed system, the reliance of the underlying communication network is crucial. Since the information is distributed, many difficulties can arise regarding the network design of the system.
Because of the diversity of communication network designs in distributed systems, it is important that the system can handle delays and a range of different types of failures, regardless of its network structure [7]. This makes design goal decisions a vital part of the functionality and intricacy of the system.

It is critical that the requirements and design issues are recognized and maintained in an early stage of the software development process and that the engineers have a thorough understanding of the theoretical and practical features of the system implementation [6]. The software engineers have a major part in the success of the software development process. Making the wrong design and implementation decisions could increase unnecessary complexity and lead to higher chances of unexpected bugs and errors. In order to achieve good quality of code in distributed systems the design goals that developers should strive towards are functionality, reliability, usability, efficiency, maintainability and portability [5].

1.2 Problem
Designing software systems is a difficult software engineering activity that requires deep knowledge of design features and years of experience and skills. Nowadays, systems are expected to be more reliable, robust and handle more information within a short time frame where requirements are frequently changing [1].

When dealing with systems of a distributed character, the intricacy and functionality are increased radically [2]. Each host in a distributed system can consist of different operating systems, run on a selection of processors and use middleware platforms with different protocols [9]. Handling these requirements and integrating them into the system is a challenging task to accomplish. Because there is a limited timeframe set for development, the outcome may be naive alternative solutions called “workarounds” which can ruin the structure and the quality of the software and make the system more complex [10].

Producing software systems that fail in quality lose value, purpose and resources. As the expectations increase from third parties, the systems must ensure sustainable quality. This is also the case for the system that this research project is based on, which is a distributed system used for self-protection in military aircraft. The system consists of a server and several clients which act as publishers or subscribers for different types of messages sent in the system. Since the system relies on a complex communication design, it is desirable to improve the network design and find a balance of good code quality and low software complexity.
A problem in the current system is that the communication is based on unicast. If a host wants to send the same message to several hosts, it has to copy and send the message once for every host that subscribes for that type of message, leading to higher chances of receiving faults during message transmission.

An additional problem with the current system is that it is based on a publish/subscribe pattern. Before any client communication is possible, the system requires event registrations to a server for all client hosts. Each client either registers as a publisher or a subscriber to the server for specific events. The registration phase in the distributed system is a crucial part for the system to operate, however, many steps are performed during the process which makes it time-consuming. This wants to be avoided with a new implementation which is not dependent on the client publish/subscribe registrations to the server.

From the mentioned problems, the research question is stated as:

How can software complexity be reduced in a distributed publish/subscribe system by replacing the network communication with multicast, without affecting and changing the external behavior of the system?

1.3 Purpose

Since there is a need to upgrade the current network design of the system in focus, it is necessary to discover an alternative solution for the unicast communication and the publish/subscribe registration phase to the server, without changing the external behavior of the system. This would lead to clients communicating with each other more efficiently.

The purpose of this thesis is to replace the publish/subscribe pattern as well as implementing a better communication method than the available unicast. This will be done by implementing a multicast communication prototype in the distributed system. The external behavior is preserved by not disrupting the traffic flow in the system after the changes with the multicast implementation is completed. The prototype is evaluated by a software complexity comparison with regards to the previous unicast communication and the publish/subscribe pattern using software metrics.

1.4 Goals

The goal of the thesis is to improve the communication in the distributed system in order to make it more scalable and handle increasing and decreasing workload with minimal impact on performance. With the replacement of the multicast communication, the functionality of the publish/subscribe pattern is not needed as well as the client registrations to the server, leading to a decreased startup time of the system.
It is also desirable to constantly improve code quality, maintainability and productivity during software development. Continuous software maintenance may avoid early system retirement and can result in a well-functioning system that is scalable, modifiable, reliable and can more efficiently adapt to new requirements.

1.4.1 Benefits, Ethics and Sustainability

An impact the multicast solution will have in the system is not needing the publish/subscribe registration phase to the server. The server has a significant role during the registration process since it keeps track of all clients and their event publications and subscriptions. By not requiring the registration phase, the workload on the server will decrease and may be used for monitoring purposes only. This will in turn benefit the client nodes not being reliant on the server, which will decrease time consumption and reduce chances of failures during system setup.

A benefit is that the workload will be reduced for the server and all client nodes in the system with the implementation of multicast. Since multicast decreases the network load by eliminating redundant traffic, the energy consumption in the overall system will decrease which is beneficial in terms of ecological sustainability.

Another benefit of the multicast solution is the reduction of code dependencies and test cases in the system. The registration phase covers large parts of the system codebase. With the removal of the registration process, a large portion of the code dependencies and test cases will be reduced. This will lead to more understandable code and may increase code quality, maintainability and productivity. In regard to economic sustainability, this would lead to employees saving more time and resources for less troubleshooting and testing.

There are some ethical aspects in regard to the work done in this field. Since the implementation touches the area of military, much of the information is confidential and need to be handled with respect to the rules and restrictions of the stakeholder. All information is therefore controlled by higher authorities before publication in order to avoid leakage of company restricted material.

1.5 Research Methodology

There are a variety of different methods and methodologies, each with its own features and approaches, that have to be considered when conducting an academic research. The selection of these methods and methodologies contributes to valid and suitable research outcomes.
There are mainly two different research approaches. An inductive approach begins with observations, theories and experiences and thereafter generates a conclusion. The inductive approach usually uses qualitative methods to analyze the phenomenon [12]. A deductive approach focuses on verifying or falsifying hypotheses and usually uses quantitative methods to test the theories [13].
This research project is based on an empirical research method using a positivism philosophical assumption with an inductive research approach. The research combines qualitative data collection with a quantitative evaluation of the system. Data will be collected by a literature study of group communication techniques, software quality essentials and software metrics. Interviews as well as analyzing documents and code will be performed, which will be the basis for the multicast implementation. Thereafter, an experiment will be carried out and evaluated using software metrics.

1.6 Stakeholder

Saab is one of the leading companies in military defense and civil security. The company is divided into several business areas that provide advanced air, marine and civil security solutions and products with the aim to have scalable, flexible and upgradable systems [14].

Surveillance is one of Saab's major system development areas which includes collecting, monitoring and managing information to be able to detect and protect against various types of threats [14]. Surveillance covers airborne, ground-based and marine radar systems that are used for situation awareness and self-protection.

One of the major fields in Surveillance is the airborne Electronic Warfare (EW), which is a collection of different technologies e.g., Radar Warning Receiver (RWR) systems and Electronic Countermeasures (ECM) systems. RWR systems detect and warn for any threats and ECM systems deceive and prevent detection or/and attacks [15][16]. This technology is divided into a distributed system of computers, also called hosts, placed in different locations inside the aircraft that communicates together in order to keep it safe and undetected.

The communication network design in the current EW system is based on a decentralized topic-based publish/subscribe communication pattern. Each host in the distributed EW system consists of several running services that internally register as subscribers or publishers to events which are sent to the server through a Data Distributor (DD). Each host contains one DD which is shared by all the host's services. The DD handles the outgoing communication to the server and the other computers. Messages are sent via unicast communication in the system and before any interaction, the computers register as publisher or subscriber to the server. The server has the responsibility to register the hosts and to inform the publishers of available subscribers. This described distributed EW system is the focal point in this report.
When working with military defense and civil security, it is essential to have technologies that constantly are in development, adaption and improvement [17]. Since Saab is constantly looking for flexible, scalable and robust solutions, there is a continuous need for development and change in the network design of the distributed EW system. It is crucial that the communication network in the system is performing as required, can process data fast and handle faults and errors. Since the EW system controls the safety of the aircraft, it is essential to produce good code quality and design principles.

1.7 Delimitations
No security aspects during message passing in the distributed EW system will be covered in this thesis. Also, guaranteeing and confirming the order in which the messages are sent and received is out of scope.

The focus in this degree project is not on performance perspectives. The multicast solution will only cover and evaluate software complexity aspects in form of software metrics and does not focus on performance assessments.

1.8 Outline
The outline of the thesis is structured in the following chapters:

Chapter 2 discusses different group communication methods for distributed systems and how IP multicast can be deployed.

Chapter 3 presents information regarding software quality, its definition and why it is important to achieve good quality of software. The chapter also discusses different software metrics used for understanding software quality in systems.

Chapter 4 provides a general background of the current distributed Electronic Warfare system architecture.

Chapter 5 presents the methods used in order to conduct the thesis. It contains different data collection and analysis techniques as well as the system development process of the multicast implementation.

Chapter 6 demonstrates the refactoring process, which consists of the design and implementation of the multicast prototype and the configuration of the switch.

Chapter 7 contains the evaluation of the results from Chapter 6.

Chapter 8 presents the final conclusions, thoughts and future work containing improvement possibilities.
2 Group communication methods in distributed systems

In distributed systems, hosts communicate with each other by exchanging messages. There exist different methods of communication in the system. The messages can be sent directly from one source to a destination or sent to a group of nodes that share a common interest or task. This chapter introduces different communication techniques used for resource sharing in distributed systems.

2.1 Unicast, Broadcast, Multicast

The simplest form of communication is point-to-point communication, also called unicast [18][19]. Unicast has a one-to-one relationship since messages are sent from one source to one destination. Figure 1 demonstrates unicast communication and gives an example of how a source node (publisher) sends a message to only one destination node (subscriber) in a system divided into three subnetworks. If there are other interested subscribers of the same event in the system the publisher will have to copy and resend the message once for every interested node, which will be redirected by the switches to the correct destinations.

![Figure 1 – Representation of unicast communication](image)

Another method of communication is called broadcasting [18]. In this case, there is a source node sending a message to all other hosts in the network. Since there only is a single node initiating the broadcast to the other hosts, the communication forms a one-to-all relationship. In broadcast communication, there is no need to duplicate and resend the message as in unicast, the switch will receive a message once and forward it to the others. Figure 2 illustrates the same network architecture as in Figure 1, but in this case the source host forwards the message to the switch which broadcast the message to all other subnetworks in the system.
An alternative for unicast and broadcast is multicasting [19]. Multicast communication reminds of broadcast but instead of sending a message to all nodes in the network, there are only some that receive it. Therefore, the multicast communication has a one-to-many relationship. Figure 3 represents multicasting which also has the same network structure as Figure 1 and 2. As in broadcast, there is still a single node sending a message, but for multicasting, there is a group of receivers instead. Seen in Figure 3, the message is sent from the source and is forwarded to those that belong to the group, the other nodes never receive the event. The group in this example includes 3 out of 6 nodes and are placed in two of the subnetworks in the system.

Today, there are many applications that use multicasting. One example is in distributed databases where information can be stored in multiple locations [19]. Since the information is spread across several units, the database does not keep track of where everything is stored. Therefore, whenever a user wants to access specific data, the server can multicast the user’s request to all units and wait for a response from the location that has the stored information.
Multicast can also be used for information dissemination [18]. If a company is about to launch a new product or inform customers about software updates, they can use multicast to distribute the data easier to their users. Multicast is also used for distance learning, where a group of students receives the course content without attending class.

Teleconferencing is another area that uses multicasting [18]. Whenever a temporary or permanent group of people are expecting the same data when attending a conference, they can use multicast for that purpose. For example, a group of work colleagues attending a meeting conference. The group may not be at the same location or even in the same country, but with multicast they can still have their conference meeting.

### 2.2 Multiple Unicast versus Multicast

There are a couple of reasons why multicast requires less bandwidth and has better performance than multiple unicasting. Some differences between unicast and multicast are explained below.

During unicasting, the messages are replicated and sent as a copy to each recipient from the source [18]. This means that if there are 10 nodes waiting for the message, the unicast specifies 10 different destination addresses, copies the message and send it one at a time to the nodes. While for multicasting there is a single message sent from the source which is duplicated in the switch. The duplicated messages, sent from the switch, have all the same destination address for its recipients.

Also, the event creation time in multiple unicasting can have a negative impact of the overall performance in the system since the messages are created one by one [18]. As an example, if there are 1000 locations expecting an event, the time interval between the first and the last event sent could be unacceptable and may even break some system requirements. In multicast, this kind of time delay does not exist since there only is one message created and sent from the source.

### 2.3 Broadcast versus Multicast

One of the main differences between broadcast and multicast is group management [20]. Broadcast does not require group management since all the nodes in the network will receive the messages. For multicast, it is required to have some sort of group management to be able to send the messages to the right group of recipients.

Since messages are sent to all nodes in the system during broadcast, there are nodes receiving messages they are not interested in. This causes unnecessary use of available bandwidth in the network [20]. The message broadcasting does also slow down the system due to the large number of events being sent in the network. In multicast, the bandwidth is effectively used and traffic is manageable and controllable since the messages are sent to intended nodes only.
2.4 IP Multicast

Internet Protocol (IP) multicast is used in many multimedia applications such as video or audio streaming [21]. It is a specific version of multicast that uses the data link layer and Internet layer to send packages to a group of receivers in a single transmission. To manage multicast group transmissions, the network needs to be set up in such a way that the hosts, routers and switches understand whenever multicast traffic is deployed. This section will explain some fundamental parts for configuring and deploying IP multicast in networks.

2.4.1 Addressing

The hosts in the network need to differentiate between IP packets of unicast and multicast character. For multicast traffic, there are specific prefixes that identify multicast IP addresses and MAC addresses which can be seen in Figure 4.

Each IP multicast address is mapped to a multicast MAC address. Addresses that start with 01-00-5e-XX-XX-XX are known as a multicast MAC address by the network. When receiving a MAC address of multicast character, the data link layer strips the MAC address off and send the rest to the above layer (Internet layer) for further processing.

IP addresses are divided into different scopes and are used for different purposes. Addresses of class D are designated for multicast communication and is in the range of 224.0.0.0 to 239.255.255.255 [21]. Some of these addresses are reserved for routers and other protocols, therefore the range 224.0.1.0 to 238.255.255.255 is used globally for multicast data. Whenever a node in the network receives an IP address in this range it will classify and handle it as a multicast package.

![Mapping of multicast IP address to multicast MAC address](image-url)

**Figure 4 – Mapping of multicast IP address to multicast MAC address**
Figure 4 demonstrates the mapping of an IP multicast address to MAC address. The IP multicast prefix represents 224 which differentiate the IP address as multicast. The last 23 bits of the 28-bit long IP multicast address are the ones added to the MAC address. As seen in Figure 4, the MAC address has its multicast prefix corresponding to 01-00-5e. By this address mapping, the switches in the system can use the MAC addresses to recognize multicast packets and make efficient forwarding decisions.

### 2.4.2 Group Management

Some sort of multicast group management is required for the network to distinguish which nodes that are expecting which messages. Therefore, hosts in the network join multicast groups to receive messages of interest. Each multicast group is identified by a single IP multicast address. Any address in the scope 224.0.1.0 to 238.255.255.255 corresponds to a multicast group. All hosts expecting multicast traffic must implement the Internet Group Management Protocol (IGMP) [22] which enables hosts to join or leave multicast groups. Whenever a host wants to send a message to a group of receivers, it specifies the correct multicast address as the destination address of the IP datagrams and sends it out on the network. Just like the network needs to support multicast transmission and allow multicast packages to arrive at its destination, all hosts interested in multicast traffic must support IGMP to join multicast groups [22].

### 2.4.3 Routing

Routing multicast traffic is harder than unicast and broadcast communication since there are only some of the hosts expecting certain messages and not one or all the hosts in the network. The switches and routers in the network have the responsibility of distinguishing multicast traffic from ordinary unicast and broadcast and to replicate and redirect the multicast packets to the correct receivers [21].

There are many types of routing protocols used in networks. A routing protocol defines how routers and switches communicate with each other in order to redirect the traffic by selecting the best routes. Protocol-Independent Multicast (PIM) [21] is a collection of routing protocols that are specifically used for multicast traffic in IP networks in order to redirect the multicast data more efficiently. The different versions of PIM routing protocols are out of scope and are not covered in this report.
3 Software quality

Poor quality in software systems loose value, therefore, software needs to evolve in order to be successful and useful. As new functionality is implemented the design may degrade, be more complex and harder to evolve. This chapter introduces how to achieve good quality in software systems and explains how to cope with complexity using different software metrics.

3.1 Achieving good software quality

Software quality is applied throughout the software development process and can be defined as; a software process to make products useful and provide quantifiable measurements for both the individuals who deliver and those who utilize it [23]. In order to build and assure software of high quality, proper software engineering methods, project management techniques and solid quality controls are essential [23]. Suitable software engineering methods are required to ensure that the design of the software meets the quality attributes. The methods are also used to understand and handle encountered problems during the development process. Good project management procedures in teams include risk and test planning phases and scheduled meetings which benefits by better decision makings throughout the system development process. Solid quality controls incorporate activities such as design, application and interface test cases to ensure and validate the quality of the system.

A range of different attributes has been proposed for defining quality in software. Table 1, adapted from [23] and [24], describes different characteristics as indicators for software quality.

<table>
<thead>
<tr>
<th>Quality attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functionality</strong></td>
<td>How accurate, suitable and secure the software is</td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
<td>How reliable the software is in terms of tolerating faults and recovering from difficult conditions</td>
</tr>
<tr>
<td><strong>Usability</strong></td>
<td>How easy the software is to use in terms of operability, learnability and understandability</td>
</tr>
<tr>
<td><strong>Efficiency</strong></td>
<td>How optimal the usage is of resource and time attributes in the software</td>
</tr>
</tbody>
</table>
By following the quality characteristics illustrated in Table 1, the delivery of software with good quality can be obtained. Each particular project is composed differently and needs to define which quality attributes that are essential for the project’s demands. It is therefore critical that the customer and the project teams have the same understanding of which features that are essential to the project in order to avoid conflicts and additional expenses.

### 3.2 Software metrics

The constant need for system progression implicates that changes and improvements in software design are necessary. The software evolvement also leads to system growing in size and complexity which, if not controlled properly, could become unmanageable.

Software metrics could be used to understand the quality of the current software and to identify potential design flaws [25]. By analyzing the system with different metrics, development teams can detect and prioritize parts of the software that lack in quality and apply it as a starting point for deeper analysis.

Refactoring is a technique used for improving internal code structure without changing the system behavior [26]. It can be applied at different abstraction levels such as source code, design models, documents analysis and software architectures [27]. By using software metrics, it is possible to detect areas in the code that could indicate deeper problems and in the long run, affect design quality negatively which can be addressed with refactoring techniques [25]. Software metrics can also be used to estimate the effects of the system after applied refactoring [27].

This section introduces some software metrics relevant to this thesis that can be used as a tool to identify potential design problems in source code.

#### 3.2.1 Lines of code

One of the most common methods for measuring a program's complexity is Lines of Code (LOC) [28]. It is a simple metric used to measure the size of the program by counting the number of lines in the code. There are different types of LOC; blank lines, commented lines and source code lines. Something that has to be taken into account is that LOC is not always a precise method because the number of lines of code may differ depending on developer experience and could also vary in different languages, even if the implemented functionality is the same.
3.2.2 Number of members per type
Number of members per type is a software metric that counts the members of a type. A type in C++ is either a class, struct or union. There are mainly three different member types; data members, function members and nested types (including class, struct, union, enum) [29]. By counting the number of members in a class, it is easier to define how large the type is and can break out parts as an attempt to improve code quality.

3.2.3 Number of parameters per function
It is important to keep functions small and well-written to perform their specific task. Sometimes, while changing the code, it may be that several different types of functionalities are added to the same function, resulting in a lot of dependencies [4]. This causes the function to get a long parameter list and its internal functionality becomes more difficult to understand and handle.

Using a metric that calculates the parameters of the function in the program could help developers detect and adjust functions with long parameter lists. Reducing the number of parameters in a function requires proper understanding of the design of the function. For example, parameters can be grouped into a new function if they are logically connected or be created to a new object [4]. On some occasions, features may contain redundant parameters that use the same value for each call but are not used in the function and can be deleted.

3.2.4 Efferent coupling
Efferent coupling is a software metric that measures the outgoing dependencies per type, i.e. how many types the type of interest is dependent on [30]. If a type is referenced directly in the type of interest, then a dependency exists. Types that rely on other types can be hard to handle since changing one type requires changes in other types since their dependent of each other [30]. Moreover, if a type has a high dependency to other types it may be considered to lack cohesion. Reducing efferent coupling could be done by splitting a class into smaller pieces.

3.2.5 Cyclomatic complexity
All programs consist of different execution statements (if, else, while, switch-cases) in order to perform a certain task. The more decision-making a program executes, the more complex the program is to understand. McCabe’s Cyclomatic Complexity (CC) [31] is a software metric used to calculate the number of linearly independent paths in a function and to understand the potential flows in a program. Additionally, CC is associated with the number of test cases in the program, i.e. lower CC leads to fewer test cases which result in a lower testing effort [31].
A program can be formulated as a flow graph to calculate the complexity of it. The CC of a program can be calculated using equation [32]:

$$V(G) = e - n + 2$$  \hspace{5cm} \text{Equation 1}$$

In Equation 1, variable $G$ represents the graph, variable $e$ stands for the number of edges that exists in a flow graph and $n$ is for the number of nodes.

Figure 5 – Calculating CC of a program

Figure 5 demonstrates an example of calculating the CC of a small program. The source code is transformed into a flow graph. Seen from Figure 5, each statement and expression is converted to nodes. Links are drawn between the nodes which correspond to edges. The number of nodes from the flow graph can be calculated to 5 and the number of edges also to 5, which results in a CC of 2 in the program.
4 The Electronic Warfare system architecture

The current distributed EW system was analyzed in order to understand the stakeholder’s needs and requirements to properly design and implement the multicast prototype. The gathered information about the system architecture is collected from documents and materials provided by the stakeholder. This chapter will give an overview of the current system architecture, demonstrate the responsibilities and the area of use of the different components in the system.

4.1 Overview

The distributed system is based on a peer-to-peer architecture and consists of 10-20 nodes. Figure 6 gives an overview of the current EW system which consists of a server (also called a master), several publisher and subscriber clients, domains and switches. The nodes represent computational devices located around the aircraft in order to protect and detect danger. The nodes communicate by unicast in a best-effort manner. The system uses UDP, which is a connectionless protocol and compared to TCP does not require acknowledgments when sending messages in the network.

Figure 6 – An overview of the distributed EW system demonstrating the server, clients, domains and switches
The communication architecture in the EW system is currently based on a decentralized topic-based publish/subscribe communication pattern. Each client node consists of internal services that either register as a publisher or/and a subscriber to the server node. A client can be a subscriber, publisher or both depending on which events the services are interested in sending or receiving. The server and each client consists of a Data Distributor (DD) which is an intermediate component between the internal services and the outgoing communication to other hosts in the network. Whenever a service wants to publish an event out on the network, the message passes through the DD to an Event Sender which forwards the message to a socket handler. From there, the message is sent for each subscriber as a unicast message. The server and the clients are connected to switches to be able to communicate with each other. The messages sent passes through the switches which redirect the messages to the correct receiver/s. Each domain contains several different events and the client nodes can belong to several domains depending on which events they publish or subscribe to.

4.2 Events, domains and switches
The distributed EW system is divided into domains. A domain is a subset of nodes that communicate and share information with each other. Each domain contains a set of events. An event is a message sent from a publisher to a subscriber node. To be able to receive events from a publisher node, the subscriber must be registered to the same domain. A domain combined with a specific event represents a topic. The nodes can belong to more than one domain. Some of the domains are intertwined, which means that the nodes in the system can publish data for a topic in Domain A, and subscribe and process data for a specific topic in domain B. The nodes communicate over Ethernet and uses switches to forward events to and from publisher and subscriber nodes. The switches are also used to and from the server node. The events sent does not require acknowledgments back from receiver nodes.

4.3 The server
The server node has a vital role in the system and is responsible for managing many tasks. One of its responsibilities is to receive and handle domain registration messages from the client hosts and to control that they all have the same version of the message to be able to avoid faults during domain registration.

The server is also responsible for publication and subscription of client nodes. The server confirms that the client nodes are registered in the specific domain before accepting registration as a publisher or subscriber for an event in that domain. It sends out heartbeat messages to all client nodes to verify that they are active and running. If a node does not respond to a heartbeat during a time interval it is considered dead and is removed from its registered topics. The node has to redo the registration process against the server for publication or/and subscription.
4.4 The client hosts

There are two types of client nodes, publishers and subscribers. All client nodes are responsible for informing the server which domains they want to register to. After domain registration, they update the server with topic publication and subscription information. The nodes are also responsible for responding to heartbeat messages to the server in order to tell if they are running.

The publisher node is responsible for registering the subscribers that are interested of its content by adding them to its internal subscriber list. Whenever the server receives a new subscriber it will send the subscriber’s information to the correct publisher which will add the new subscriber to its list. Whenever data is updated in a topic, the responsible publisher will notify the interested subscribers by iterating through its subscriber list. The subscribers will receive a message with the updated topic and process the data.

4.5 The publish-subscribe registration phase

In order for the nodes to send messages to each other, they must register as publishers or/and subscribers to the server. Figure 7 illustrates an example of two clients C1 and C2 and a server node S. An example will be demonstrated where C1 will be registered as a subscriber and C2 as a publisher for a particular event “HelloWorld” in a domain called Domain A. The steps 1-9 in Figure 7 will be described based on this example.

![Figure 7 - The registration phase to a server (S) for a subscriber called client 1 (C1) and a publisher called client 2 (C2)](image)

The registration process for the subscriber node C1 and publisher node C2 to the server is shown in Figure 7. It is divided into nine steps which are explained in detail below.

1. The server is being launched and clients C1 and C2 do an internal registration. The internal registration specifies information regarding domains and their corresponding events the client’s internal services will publish and subscribe to. Empty publisher and subscriber lists are generated for every domain that the clients do an internal registration on.
The lists will be filled with interested publishers and subscribers throughout the registration process. As for the example given, after the internal registration, C1 will know internally that it is a subscriber for the event "HelloWorld" in domain A and C2 will know it is a publisher for the same event and domain. The server does not know this information yet but will be provided with it during the coming steps.

2. C1 does a domain registration to the server, i.e. it sends a registration event telling the server it wants to be registered to domain A.

3. The server registers C1 to domain A and sends an acknowledgment message to the client if it was a successful registration or not.

4. After successful domain registration, C1 does a subscriber registration to the server, i.e. it tells the server it wants to be registered as a subscriber to the event “HelloWorld”.

5. The server searches for interested publishers of event “HelloWorld” in domain A. It does this continuously for all client nodes as soon as they passed step 4 in the publish/subscribe registration process. When any interested publisher or subscriber is found, it will send it/them one by one to the client node. In this case, it did not find any interested publishers for C1 and does not send anything.

6. The server sends an acknowledgment that it has registered C1 as a subscriber for event “HelloWorld” in domain A.

The same process from step 2-5 is done for C2 but instead of registering as a subscriber to event “HelloWorld” on step 4, it will do it as a publisher.

7. C2 has now done a registration as a publisher to event “HelloWorld” in domain A. The server searches for interested subscribers of the event “HelloWorld” in domain A and sends it to publisher C2. In this case, the server will find C1 as an interested subscriber and send it to C2 which will add C1 to its subscriber list.

8. Publisher C2 searches through its subscriber list and finds subscriber C1.

9. Publisher C2 sends the “HelloWorld” event to subscriber C1.

The registration process for the subscriber node C1 and the publisher node C2 is now completed and they can communicate freely without client interactions with the server.
5 Method

This chapter introduces different data collection and analysis techniques and demonstrates the software development process for the multicast implementation.

5.1 Data collection techniques

Data collection is a fundamental part when conducting a research. There are several qualitative and quantitative methods, all of them with different purposes and approaches, that can be used for data collection. Some examples of data collection methods are; Observation, Questionnaire, Interview, Case study [33], Document analysis [11], Experiments and Language and Text [12]. A brief explanation of the data collection methods is described below.

During Observational data collection, the researcher observes the behavior of objects and phenomena. The observation method is categorized as; Participant observation, Non-participant observation, Structured observation and Unstructured observation.

The Questionnaire data collection technique formulates statements and open or/and closed questions which covers research questions, objectives and variables. Some questions or statements may require some quality assurance with validity and reliability tests.

An Interview consists of an interviewer and an interviewee, which directly or indirectly interact with each other. The interviewee answers the questions given by the interviewer. The interview questions can be formulated as structured, unstructured or semi-structured depending on how flexible or in-order the interview must be handled.

Case study consists of a study of a single case which can include a research design and different data collecting methods such as interviews, questionnaires and observation in order to analyze the study.

Documents analysis includes data collections from different types of sources and materials that are used and analyzed by the researcher in order to define what information that is available in the interested research area.

An Experiment is another way for collecting data. This method uses a collection of large data sets for variables used in the experiment.

The Language and Text data collection approach is used for interpretation of conversations, documents and texts.

5.2 Data analysis techniques

Once the data collection is completed, the research moves to quantitative or/and qualitative data analysis. An analysis of the collected data is made to get a better understanding of the current situation and to decide whether further data collection is necessary.
A quantitative data analysis is done to examine, measure, test and construct concepts and theories of the collected data [13]. A qualitative data analysis is performed to scan, organize and sort the collected data [11]. Some examples of data analysis techniques are Statistics [13], Computational Mathematics, Coding, Analytic and Narrative Analysis [12].

A Statistical test is categorized into descriptive and inferential statistical testing. Descriptive testing analyzes how variable values are distributed and formulated and discover relationships of the data. Inferential testing analyses samples in relation to populations.

Computational Mathematics is a data analysis method that uses algorithms and numerical methods to produce models and simulations.

Coding is the process of turning qualitative data into quantitative by organizing information gathered during data collection. The process includes labeling and categorizing concepts and strategies.

Analytic Induction and Grounded Theory is an iterative analyzation technique which iteratively analyses collected data to verify or falsify a stated hypothesis or theory until a solution is found or/and validated.

Narrative Analysis is a data analysis technique that uses different procedures to test and analyze documents.

5.3 Quality assurance

All studies follow some quality assessment criteria for quantitative and qualitative verification and validation of the collected and analyzed data. The focus on quality assurance in quantitative and qualitative research is validity, reliability and dependability, replicability, confirmability, transferability and ethical aspects [12].

Validity ensures that the instruments used during testing are measuring expected data. Reliability corresponds to dependability and refers to how trustworthy, correct and consistent the research outcomes are. The replicability aspect is used to making sure others can replicate the measurements and reach to the same result and conclusions. Confirmability confirms the correctness of the research results and makes sure it is performed without being affected by personal assessments. Transferability refers to generalizing and transferring the research so it can be used as a database for other researches. Ethics assures that the research is following moral guidelines by preserving confidentiality and handling personal and sensitive information securely.
5.4 Software development process

The prototype of the multicast implementation is conducted of an agile software development process [34] which is presented in Figure 8. The development process is divided into several iterative steps. After each phase, the stakeholder is notified to be updated with the status of the ongoing prototype. Discussions regarding requirements are also made and feedback is given to steer the development of the prototype to the wanted outcome. This is done until final prototype deployment.

![Diagram of software development process]

Figure 8 – The software development model for the multicast implementation

This section presents the steps of the development process for the multicast implementation described in Figure 8 as well as the data collection and analysis techniques used.

5.4.1 Requirements specification

The technique used for gathering the stakeholder’s requirements regarding design and implementation details were from non-structured informal interviews. The interviews were held during group discussions and meetings with supervisor, system architecture, scrum master, and software engineers of the EW system for gathering information and discussing the current progress of the implementation.

Another technique used to gather knowledge about the current EW system was from document analysis. Documents were analyzed to prepare questions for the interviews. The source code of the EW system was analyzed to find refactoring areas to be able to replace the unicast and the publish/subscribe registration process with the multicast communication.

The gathered information from the interviews together with document analysis were the basis for structuring the requirements of the development of the multicast communication in the EW system. These requirements could change during each iteration if new discoveries were found or goals were set up by the stakeholder.
5.4.2 Design, Implementation and Test
A prototype of multicast communication will be designed and implemented with the aim to replace the current unicast communication and the publish/subscribe registration process in the distributed EW system. Each iterative step of the design and implementation phase involves feedback from the stakeholder, problem reports and addition of new requirements that can affect the final design and implementation.

The replacement is conducted in a refactoring process that iteratively restructures the system’s unicast communication and the publish/subscribe registration process with multicast. During the refactoring process, the source code is tested and the program is continuously checked to avoid changing the external behavior. After implementing the multicast prototype, a verification test scenario is performed to ensure that the communication works between the clients on the network.

5.4.3 Evaluation
Experiment is chosen as the data collection method for testing and evaluating the multicast implementation. The experiment will be evaluated with a before-and-after comparison of the previous unicast communication architecture with the new multicast solution using software complexity metrics. The data analysis method for the experiment is statistics, which is used to discover relationships and draw conclusions from the current communication architecture with the new multicast implementation.

A tool called Metriculator [35] was used which is a plugin for Eclipse used for statically analyzing different software complexity metrics for C++ source code. Since the software for the distributed EW system is written in C++ made Metriculator a suitable tool.
6 Software complexity reduction process in the publish-subscribe distributed system

This chapter presents the refactoring process of the communication architecture in the distributed system with a multicast implementation. More specifically, the refactoring process for the multicast solution was divided into two parts. The first part involved removing the publish/subscribe registration phase, the second part was to replace the unicast communication. Section 6.1 presents the collecting of the requirements for the implementation. The steps taken of the refactoring process are explained in 6.2. Section 6.3 to 6.6 explains each step thoroughly in the refactoring process.

6.1 Requirements collection

The requirements of the system were collected in order to understand the stakeholder's expectations and requests of the multicast prototype. The requirements were divided into system and implementation requirements.

The system requirements describe the functionality of the distributed EW system which replaced the publish/subscribe pattern and the unicast communication with multicast. These requirements are listed below:

- The system should work without server functionality, i.e., no client registrations to the server should be made and the system communication should work with multicast
- The external behavior of the system should stay the same even after replacement of unicast and removal of the publish/subscribe pattern
- The system should follow the quality features; functionality, reliability, usability, maintainability and portability described in section 3.1
- The refactoring process should not introduce new bugs and errors in the system

The implementation requirements were designed for program-specific purposes and were used as a basis for the refactoring process. These requirements consisted of:

- The receiver should still able to receive desired events as before refactoring process was carried out
- The sender should still be able to send desired events as before refactoring process was carried out
- Source code in regard to client interaction with the server is unused, i.e. event publications and subscriptions to the server are not needed
- The internal communication among the services in a host should still function properly after the refactoring process
- Internal services should still be able to send and receive events from external hosts after the refactoring process

To be able to provide a solution, the above-mentioned requirements were followed throughout the implementation process.
6.2 Refactoring process

The refactoring process was based on the requirements formulated in 6.1. The process focused on restructuring parts of the communication architecture of the distributed EW system. This was written as a systematic approach for rearranging and changing source code connected to the client publish/subscribe registrations to the server and the unicast communication. The refactoring process consisted of:

1. Identifying refactoring areas in source code
2. Apply refactoring
3. Guarantee that the refactoring areas preserve the same external behavior
4. Evaluate the effects of the refactoring areas by code complexity analysis using software metrics

The remaining sections of this chapter will cover the above-mentioned steps of the refactoring process.

6.3 Identify refactoring areas in source code

The most difficult and demanding part of the refactoring process involved identifying areas that would be replaced by multicast logic. In this step, the refactoring was applied to the program, i.e. the source code of the distributed EW system and not on abstract levels such as design models, analysis documents or architectures. The focus was on the packages, classes and class functions containing source code of the communication for the EW system.

The source code is divided into server-side logic and a client-side logic. Each side has their own packages and classes that are a part of the communication in the system and were therefore considered important for the refactoring, these are described in Table 2.

<table>
<thead>
<tr>
<th>Class/Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Client/Server Data Distributor class</strong></td>
<td>The server and the clients have each a DD class used for sending messages to external hosts</td>
</tr>
<tr>
<td><strong>Client/Server Event Sender class</strong></td>
<td>Each DD communicates with an Event Sender that is responsible for sending events with the underlying communication layers via a socket.</td>
</tr>
<tr>
<td><strong>Client/Server Data Distribution package</strong></td>
<td>This package contains a variety of subclasses containing functionality regarding the DD both for the server and client.</td>
</tr>
</tbody>
</table>
Because the server logic was not needed after the removal of the publish/subscribe registration phase, the focus relied solemnly on classes and packages connected to the client functionality. The clients should still be able to send and receive messages without the server side logic.

The DD and Event Sender on the client side contains registration logic described in the registration phase in section 4.5. The logic within the source code of the DD and the Event Sender handles and process internal and external events such as registration of different domains, publications and subscriptions.

During the registration phase, the DD communicates with the Event Sender and forwards information of registration events which the Event Sender acts upon and thereafter send the appropriate event through the socket to the server. The server replies with an acknowledgment to the client. Furthermore, the DD receives heartbeat requests from the server and thereafter sends a reply via the Event Sender back to the server to inform that it is still alive.

Initially, the internal domain registration is performed and the client host reads from a configuration file the appropriate domains the services in the client are interested in. More specifically the client reads which domains that contains events the clients will publish and subscribe to. The DD adds all the registered domains in its domain list. It also creates two empty participant lists, one for the publisher services and another for subscriber services.

Thereafter, the external domain registration is performed and the client DD notifies the server with a domain registration event via its Event Sender class. It waits for an acknowledgment from the server in order to continue the registration process.

When the domain registration process is completed, the internal service registrations for publication and subscription is executed. This is when the client host reads from the configuration file which events, from the registered domains, the client services either publishes or subscribes to. After initialization from the configuration file, the services notify the DD which events they are interested in subscribing and publishing to from the registered domains. For each interested event, the DD does a domain lookup to guarantee that the domain is registered. If the domain exists, the DD add the publisher or subscriber to its corresponding publisher/subscriber list. For every added publisher or subscriber, the client does an external registration through the DD. The DD notifies the server with the information from the aforementioned internal publisher/subscriber registration and waits for an acknowledgment from the server until continuing in the program.

Shared Data Distribution package
This package contains a variety of subclasses containing functionality regarding the DD for both server and client.
From this point, the server notifies each publisher with its interested subscribers which means that the publisher is aware of the clients that are subscribed to the events it publishes. These subscribers are kept in a list and the subscribers now bind to arbitrary ports. Every time a publisher is publishing an event, the DD forwards the event, the domain the event belongs to and the list of subscribers interested of the event to the Event Sender which sends the events out in the network.

6.4 Apply refactoring

After identifying different restructuring areas needed for refactoring and designing the multicast communication, the unicast was replaced with multicast and the functionality for publish/subscribe registration process to the server was not necessary anymore.

To be able to apply the identified refactoring areas, two major parts were required. First, the design of the multicast communication was important to replace the unnecessary functionality of the source code with multicast logic. Thereafter, a switch configuration was vital in order for the switches to filter unwanted traffic and redirect the messages to the correct receivers. The design of the multicast communication is explained in section 6.4.1 and the switch configuration in section 6.4.2.

6.4.1 Design of multicast communication

Functionalities needed for IP multicast communication was explained in section 2.4 and they were; multicast group addressing, group management and routing. However, since the EW system consists of a network containing only switches in a single collision domain it does not require all of these functionalities in the same way as other systems. Therefore, the multicast addressing, group management and routing functionalities needed to be adapted and adjusted to the EW system’s design in order to implement multicast in the system correctly.

Socket configurations were necessary to enable multicast communication between the hosts. These functionalities were applied in the sender and receiver hosts as well as in the switches of the system to enable multicast communication in the network.

Multicast group addressing was needed to differentiate data of multicast character. As explained in section 2.4.1, for multicast traffic there are specific prefixes that identifies multicast IP and MAC addresses. Class D addresses are designated for multicast in the range 224.0.0.0 to 239.255.255.255. Whenever a receiver node in the EW system receives an IP address in this range it should categorize and handle it as a multicast package.
Multicast group management was required for the host to distinguish which services that were expecting which events. The receiver services in the system join multicast groups to receive messages of interest. Any address in the scope 224.0.1.0 to 238.255.255.255 corresponds to a multicast group that the receivers can join to receive events from the senders. IGMP is required for dynamic multicast groups. But since the EW system network is static and uses static rules reading from the configuration files, the process of joining multicast groups is not by using IGMP. Instead, the joining of a multicast group is done during domain registrations in the EW system. This means that, during internal domain registration, all receivers specify which multicast groups they want to join to receive their events from. The sender host does not join any multicast groups. Whenever the client host wants to send a message to a group of receivers, it specifies the correct multicast group address as the destination address of the IP datagram and sends it out on the network.

The switches in the EW system have to be able to distinguish multicast traffic from ordinary unicast and broadcast and to replicate and redirect the multicast packets to the correct receivers. A forwarding table with blocked ports for specific multicast addresses is used to forward packets to the correct receivers. Switch routing means that traffic considered as multicast would be handled as broadcast if a filtering function is not implemented in the switches. A switch was configured for this purpose for the filtering function and is explained in section 6.4.2.

Before any traffic was sent or received, both sender and receiver hosts need to setup UDP sockets that are multicast enabled. The sender socket was set up by first opening a socket and enabling it for multicast traffic. After configuring the socket, the client can send the event out to the network, using the DD and Event Sender class, to the designated multicast group. The receiver also configures a socket which is multicast enabled. Thereafter it joins the multicast group/s which it is interested in as well as binding to a specific port of that multicast group. All receivers bind to the same port when expecting messages from a specific multicast group. When a host receives events, it distributes the events to the services that joined the group to that port they bounded to.

All of these functionalities described in this refactoring step was found and analyzed in order to replace the unicast communication and publish/subscribe registration process with multicast.

### 6.4.2 Switch configuration

A switch is by default configured to broadcast unknown destination MAC addresses. Therefore, to prevent forwarding traffic to all hosts in the system, the switch can be configured to block specific ports and only forward the multicast traffic to expected receivers in the network. This blocking technique, also called filtering, on the switch is not automatically enabled and must manually be configured.
In order to fully implement the multicast solution into the EW system, a switch was configured. A python script was written to auto-configure the switch and to construct its forwarding database (FDB). The FDB is a table containing information which the switch can use to forward multicast traffic to the correct receivers in the network. An example of the FDB can be seen in Figure 9. The procedure used to generate the FDB of the switch is explained below.

A configuration file was created containing:

- Mappings between domains and IP multicast group addresses
- Mappings between client hosts and domains
- Mappings between switch port numbers and client host

Each domain in the system has its corresponding IP multicast group address. Each client knows internally which events from which domains they are expecting to send or receive. This information is read from the configuration file during the internal registration phase. The hosts are mapped to a physical switch port to make the switch understand on which port to forward the traffic for each host. The python script reads the information from a switch configuration file, process it and generate a bash script that runs all the switch commands to update the FDB.

From the configuration file, the domain names with their corresponding multicast group addresses were read. Each IP multicast address was then converted to a multicast MAC address. During the conversion, the MAC addresses were verified in order to avoid faults. Each client was checked to notice if it was interested of the multicast group. This was done by checking the clients’ mapped domains from the configuration file. Thereafter, each port for all interested clients was taken for the switch to know on which ports to forward the received event.

Figure 9 – A switch forwarding the IP multicast address 225.30.67.6 to the correct receivers by blocking ports based on the configured FDB
An example demonstrating a FDB can be seen in Figure 9. The FDB is configured in the switch. A multicast group IP address of 225.30.67.6 is mapped to a MAC address corresponding to 01:00:5e:1e:43:06. When this IP multicast package is incoming, it will match the line in the FDB table and be forwarded to expecting client hosts which in this case are placed somewhere in the network on port 1 and 3 of the switch.

6.4.3 System architecture after applied refactoring

The whole registration phase to the server was not needed except for step one from section 4.5 with the exception of not launching the server. This means that all external registrations to the server were not used and no more waiting for acknowledgments from the server was necessary. Without server interaction and at the same time keeping the internal registrations, each client will only know which types of events to receive and send. In this case, the sender client will not know which or how many clients in the network that are expecting the events. Additionally, the receiver client will not know from whom the event was sent from. When these events are sent from the client, a filtering function in the switch is applied that provides information on where to redirect each event to the right location.

After removing the registration phase to the server from the clients, the only functionality kept is for the clients to do internal domain registrations and internal service subscriptions. The internal subscriptions are used for when the client host receives events, it distributes the events to the services that joined the group to that port they bounded to.

With the replacement of the unicast implementation with the substitution of the multicast solution, it is not necessary anymore for the sender to iterate through its list of subscribers. Now, the sender only needs to specify a multicast group address and send the event. The receiver needs to join the multicast group during internal domain registration and bind to the port in order to receive events of interest.

Internal publications are not needed anymore since the DD does not care from which publisher service it receives the message from. The DD takes the multicast group address corresponding to the domain and forwards the event to the Event Sender which sends the event out on the network and let the switches take care of the redirecting of the events in the system.

6.5 Preserve the external behavior in the system

The final phase of the refactoring process was to test and verify that the multicast implementation preserved the same external behavior in the system after the refactoring was completed. This step consists of the third step in the refactoring process explained in section 6.2.
The implementation was tested in order to check that the requirements described in section 6.1 were met. Each step in the refactoring process was tested iteratively. The first measurement taken was to ensure that the external behavior was unchanged before any refactoring was applied. For each change in the source code, an appropriate test was made to ensure that program had not changed the external behavior. If the changes failed the program was rollbacked to the latest correct version and a new refactoring approach was taken. This process was applied until the complete restructuring of the multicast implementation was completed.

A test scenario was prepared to verify that the multicast communication was implemented correctly in the system. Figure 10 presents the test scenario for three hosts communicating over the network via a switch using multicast. The network setup can be described as:

- **Host 1** – consist of 4 different services. S1 is a sender of event 1, S2 of event 2 and S3 of event 3. The fourth service R1 is a receiver service expecting event 1
- **Host 2** – consist of two receiver services, R1 expecting event 1 and R2 expecting event 2
- **Host 3** – consist of one receiver service R2 expecting event 2
- **Switch** – which has host 1 on its physical port 6, host 2 on physical port 2 and host 3 on physical port 5. The switch has its FDB configured based on the network setup and knows where to redirect incoming events to the correct locations

![Diagram](image-url)
Since the FDB is configured in the switch, it will know where to redirect the events. For example, if event 2 from S2 was sent out on the network, the IP multicast group address would be mapped to its multicast MAC address in the FDB. Thereafter, the switch would know that there are expecting hosts waiting for this type of event positioned on port 2 and 5 and send the messages on those ports and block the others. When host 2 and 3 receives this event, they will distribute the event to their corresponding R2 service that joined that multicast group to receive this event. For example, if event 3 would be sent from S3, when the switch would receive this event it would not find anything matching in its FDB since no host is expecting that type of event and drop those packets.

6.6 Evaluation after refactoring

The last step of the refactoring process was that the system was evaluated by a code complexity analysis and consisted of five different software metrics. The evaluation was performed in order to analyze if the changes of the refactoring areas made any difference in the system source code and if the system still met the stakeholder’s requirements. The result of the system evaluation after the refactoring changes is explained throughout chapter 7.
7 Evaluation of the multicast implementation

The multicast implementation is evaluated using software complexity metrics. The evaluation was analyzed separately for client side and server side functionalities. The evaluation consisted of analyzing five metrics described in section 3.2 and they were:

- LSLOC – logical source lines of code
- NbMembers – number of members
- NbParams – number of parameters
- EC – Efferent Coupling
- CC – Cyclomatic complexity

These metrics were mainly evaluated on class and package level for the client and server side logic and is explained in section 7.1 and 7.2. Section 7.3 presents a summary of the complexity reduction for the whole communication logic consisting of client, server and shared logic. Section 7.4 presents the average time it takes for event registrations to the server.

7.1 Software complexity analysis on client side

Functionalities affected by the refactoring process on the client side was mainly in the DD class, the Event Sender class and the client Data Distribution package which are presented in Figure 11, 12 and 13.

![Comparison of five software metrics before and after the replacement with multicast implementation in the client DD class](Figure 11)
Figure 11 compares the software metric values before and after the refactoring process for the client DD class. Before any changes were made in the source code, parts of the unicast communication and publish/subscribe registrations to the server related to the DD class was analyzed. What can be seen from Figure 11 is that the results for LSLOC, NbMembers and NbParams after the refactoring process have all been reduced by around 42 %, 15 % and 33 %. This means that the multicast functionality required 42 % fewer lines of code, 15 % fewer class members and 33 % less class parameters in the source code but still kept the same external behavior as before the changes. Least difference in metric value for the DD class was the efferent coupling metric with around 4 %, which means that the type dependencies that DD class has after the refactoring process are almost the same as before the change. The CC metric has a 49 % reduction value after the refactoring. Since CC is connected to the number of test cases results in that almost half of the testing is reduced for this class.

Figure 12 – Comparison of five software metrics before and after the replacement with multicast in the client Event Sender class

Figure 12 compares the software metric values before and after the refactoring process for the client Event Sender class. What can be seen from Figure 12 is that there are relatively large differences in this class after the refactoring. All metric values exceed a 50 % decrease after the exchange of unicast and publish/subscribe registration phase with multicast. The metrics LSLOC, NbMembers, NbParams, resulted in a 65 %, 54 %, 71 % reduction after the multicast implementation. This means that the multicast functionality required 65 % less lines of code, 54 % fewer class members and 71 % less class parameters in the source code but still kept the external behavior as before the change.
What can also be seen from Figure 12 is that the Efferent Coupling was reduced by approximately 57%. This means that the Event Sender class is 57% less dependent on other types compared with before the change. The CC metric was reduced by 67% which means that roughly 67% of the test cases are cut off after the implementation with multicast.

![Client Data Distribution package](image)

Figure 13 – Total amount of functionality not needed after the replacement with multicast in client Data Distribution package

Figure 13 demonstrates the client Data Distribution package that contains classes used by the client for distributing data out in the network. The metric values demonstrated in Figure 13 are replaced logic in the source code that was not needed anymore for the multicast implementation. The logic that was not necessary was classes that mainly concerned logic for client publications. Illustrated in Figure 13 is that the number of lines not used from the package was 157 lines, 29 number of members was reduced, 13 number of members was reduced and 11 dependencies were cut off in the package. Finally, the CC was reduced with 33 paths in the package.

### 7.2 Software complexity analysis on server side

There are no before and after comparisons for server side because the server was not needed after the refactoring process. Therefore, all functionality demonstrated in Figures 14, 15 and 16 shows server logic that was not needed anymore after the multicast was implemented in the system.
Figure 14 – Total amount of functionality not needed after the replacement with multicast in server DD class

Figure 14 shows the resulting metric values of the server DD class that was not needed after implementing multicast. The class contained logic in regard to preparing and sending heartbeat request events and replying to registration events from clients in the system. As illustrated, 813 LSLOC, 50 members and 49 parameters were not used. Also, 31 type dependencies were unused as well as 169 less test cases to prepare and perform.

Figure 15 – Total amount of functionality not needed after the replacement with multicast in server Data Distribution package
Figure 15 demonstrates replaced logic in the source code that was not needed anymore after multicast was implemented for the server Data Distribution package. The package involved the server Event Sender logic as well as server publisher and subscriber functionality. Seen in Figure 15 is that, 1134 LSLOC, 118 members and 86 parameters was not necessary anymore from all classes in the package. A total of 68 dependencies seen from Efferent Coupling value was not needed. The CC was reduced with 226, which means that 226 paths were cut off from the server’s data distribution package after server logic was not needed any longer.

![Shared Data Distribution package](image)

Figure 16 – Total amount of functionality not needed after the replacement with multicast in shared Data Distribution package

Figure 16 demonstrates logic not needed anymore after the multicast was implemented for the shared Data Distribution package. The package contains classes used by both the client and the server for distributing data out in the network. The functionality not needed from this package only concerned the server, all client logic was untouched. The functionality for the server logic regarded classes that concerned sending and receiving events for the Server DD. Shown in Figure 16 is 310 LSLOC, 76 members and 26 parameters was not needed. The type dependencys were reduced with 23 for Efferent Coupling and 38 linearly independent paths were cut off.

7.3 Summary of software complexity analysis

The total amount of software complexity reduction for the complete communication logic in the EW system is explained in Table 3 and Table 4. Table 3 presents the values for each analyzed metric before and after the changes with the multicast communication. Table 4 summarizes the values from Table 3 and presents the software complexity reduction difference in percentage.
Table 3 – Before and after metric values for all communication logic of the EW system

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
<th>Difference (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LSLOC</strong></td>
<td>Client logic 1139</td>
<td>Server logic 1134</td>
<td>916</td>
</tr>
<tr>
<td></td>
<td>Shared logic 123</td>
<td>118</td>
<td>219</td>
</tr>
<tr>
<td></td>
<td>Client logic 92</td>
<td>Server logic 86</td>
<td>183</td>
</tr>
<tr>
<td><strong>NbMembers</strong></td>
<td>Shared logic 66</td>
<td>68</td>
<td>83</td>
</tr>
<tr>
<td><strong>NbParams</strong></td>
<td>Shared logic 258</td>
<td>226</td>
<td>115</td>
</tr>
</tbody>
</table>

Table 3 demonstrates the metric values for client, server and shared logic before the change using unicast and publish/subscribe registration process and after the change with the implementation with multicast communication. What can be seen from Table 3 is that all communication logic was reduced after the change with multicast. Since the functionality for the server was not needed anymore, a major difference can especially be seen for the server logic reduction.

Table 4 – Summary of before and after changes for the complete communication logic of the EW system

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Summary Before</th>
<th>Summary After</th>
<th>Difference (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LSLOC</strong></td>
<td>3189</td>
<td>866</td>
<td>73</td>
</tr>
<tr>
<td><strong>NbMembers</strong></td>
<td>460</td>
<td>151</td>
<td>67</td>
</tr>
<tr>
<td><strong>NbParams</strong></td>
<td>361</td>
<td>279</td>
<td>23</td>
</tr>
<tr>
<td><strong>EC</strong></td>
<td>217</td>
<td>65</td>
<td>70</td>
</tr>
<tr>
<td><strong>CC</strong></td>
<td>599</td>
<td>153</td>
<td>74</td>
</tr>
</tbody>
</table>

Table 4 presents a summary of all the metric values before and after the change with multicast implementation. The results present the software complexity reduction difference of the entire communication logic consisting of functionalities for the client, server and shared classes and packages.

As demonstrated in Table 4, the LSLOC was completely reduced with 73 % indicating that much of the classes and functions with the multicast implementation uses less source code, is easier to understand but executes the same data flow and communication outcome. The metric values for NbMembers and NbParams are reduced with 67 % and 23 % which could imply that the classes have been smaller and that the functions take fewer parameters which can help the developers to avoid misuse or misunderstandings of the function’s purpose.
The EC was reduced by 70%, this can indicate that all classes and functions are less dependent on other classes but perform the same outcome with multicast. Therefore, the code is less coupled and complex leading to a better structured and designed source code. The CC for the communication logic is in total reduced with 74% which implies that the source code requires 74% less testing, making the code more maintainable and robust.

### 7.4 Client event registrations to server

Figure 17 presents the average time it takes for event registrations to the server. The time is the average of 10 repeated sessions and is presented with the standard deviation. The Figure demonstrates up to 10 different events from the same domain that are registered to the server.

![Event registrations to server](image)

**Figure 17** – The time it takes to register a client to events in one domain

As seen in Figure 17, the time it takes for one client to register one event is around 200 milliseconds. For 10 events is a bit above 1400 milliseconds. Another thing to notice is that it is rather a linearly increasing time growth for event registrations to the server. This means that the more events that are registered the more time it takes for the server to handle the events.
8 Conclusions and Future work

This thesis implemented a multicast prototype which replaced the previous unicast communication with the publish/subscribe registration phase to the server. A refactoring process was designed and performed to structurally replace the old communication logic with the multicast solution. The refactoring process included design, implementation and testing of the multicast communication. It was evaluated by a comparison of the old and the new communication logic in the EW system using software metrics. The multicast implementation resulted in a reduction for all software metrics for the complete communication logic in the EW system.

The replacing of communication architecture with the multicast solution affected the program source code which one can see from section 7.1 and 7.2. The logic relating to the external registration phase was removed and communication was replaced with multicast. Server interaction was not necessary anymore and the client neither needed to publish or subscribe for different events. The multicast implementation did not change the outer behavior of the system, however, the communication architecture required less source code.

A conclusion that can be drawn is that the change with multicast improved the code quality. By reducing the software metric values, the source code is more comprehensible for developers. The implementation of multicast has decreased duplicated and inconsistent source code making it easier for developers to maintain and understand the communication functionality. The program has easier and cleaner source code that may introduce fewer bugs. The functions related to the communication logic are easier to handle and the complexity of the communication is easier to understand. Also, by reducing the CC, it has decreased the complexity of the source code leading to fewer test cases and maintenance cost.

The multicast solution may have improved the quality attributes such as functionality, reliability, usability, efficiency, maintainability and portability described in section 3.1. Additionally, the software program requires less testing due to the reduction of CC which in the long term can save a lot of time and resources. Due to the reduction of test cases, the amount of maintenance cost could be decreased as well. The change with multicast has also improved productivity and increased robustness without changing the external behavior of the system.
A positive impact of the multicast solution in the system is the removal of the publish/subscribe registration process to the server. Now there is no need to actively set up the system network with the clients and the server node for them to be able to exchange information with each other to get a functioning communication between the clients. This leads to clients not needing to tell the server what domains they are interested in or which events they want to publish or subscribe to. The outcome of this is that the sender client does not need to know where the receiver hosts are located in the network. Instead, the logic in the switches takes care of the filtering and redirecting of the events to the correct receivers in the system. Therefore, the system setup is moved to an offline state since both the client and switch logic are read from configuration files, where clients load data for their internal registration and the switch reads its data to construct its FDB.

Another conclusion that can be drawn is that the removal of client interactions with server avoids a single point of failure since the clients are not expecting acknowledgments or registration responses from the server with the multicast solution. Without the server, the system is able to further scale with more hosts since there is no component being heavily loaded with registrations anymore.

Figure 7.3 shows that the time for registering an event is almost linearly increased for a single client, which took around 200 milliseconds to register to one event in one domain. But when registering more than one event, it could be assumed that it is around 140 milliseconds for each event to be registered for each client. Assuming that there are C clients registering for E events in D domains, the registration time will approximately lead to C*E*D*140 milliseconds. This in turn will save much time when the number of clients, domains and events are high. With the new implementation, the network configuration is offline and no more registrations to the server are necessary which means that no more time is wasted on registrations.

The server has a heartbeat function which is a vital part of the system since it checks the aliveness of the clients in the system. This functionality is not a part of the server’s responsibility and should be placed elsewhere. As a suggestion for future work, a new heartbeat monitoring component can be dedicated for sending out heartbeat requests and handling the replies from the clients. The server can instead be used as a monitoring component keeping all ports open and listening on all incoming packages in the system. By doing this, the server can analyze the traffic flow in the system.

Since the system was analyzed through a software complexity viewpoint, a future work could be to analyze and evaluate the new communication architecture from a performance perspective. This includes, e.g. throughput, data transmission and response time tests for the multicast communication.
References
