Linköping University | Department of Computer science Bachelor thesis, 16 ECTS | Informationsteknologi 2016 | LIU-IDA/LITH-EX-G--16/050--SE

Design and Evaluation of a Reliable Group Communica- tion Protocol

Design och utvärdering av ett protokoll för tillförlitlig gruppkommunikation

## Philip Montalvo Albin Odervall

Supervisor : Mikael Asplund Examiner : Nahid Shahmehri

Linköpings universitet SE–581 83 Linköping

+46 13 28 10 00 , [www.liu.se](http://www.liu.se/)

**Abstract**

In distributed systems it is often useful to ensure that messages sent between processes in a group are received by all group members. This thesis presents *Reliable Non-Ordered Multicast Protocol* (RNOMP). We argue that it achieves reliable multicast between processes in groups that processes can leave and join arbitrarily. The protocol has been implemented on top of a group membership simulation which allows evaluation of the performance of the protocol while varying packet loss and the frequency at which processes leaves and joins groups. After analyzing how the protocol handles certain situations we conclude that our protocol achieves reliability and performs well within certain parameter values.

# Acknowledgments

We would like to thank our supervisor Mikael Asplund for his help during this project. We would also like to thank Simin Nadjm-Tehrani for her valuable feedback. Furthermore we would like to thank Sebastian Bångerius and Felix Fröberg for opposing on our thesis and Marcus Bendtsen and Eva Törnqvist for providing feedback on our thesis presentation. Fi- nally we would like to thank Dennis Dufbäck, Lisa Habbe, Mattias Hellsing, Fredrik Håkans- son, Fabian Johannsen, Matilda Söderholm, Anton Tengroth and Chi Vong for their support during this period.

# Contents

|  |  |
| --- | --- |
| [**Abstract**](#_bookmark0) | **ii** |
| [**Acknowledgments**](#_bookmark0) | **iii** |
| [**Contents**](#_bookmark1) | **iv** |
| [**List of Figures**](#_bookmark2) | **v** |
| [**List of Tables**](#_bookmark3) | **vi** |
| [**1 Introduction**](#_bookmark4) | **1** |
| [1.1 Aim](#_bookmark5) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 1 |
| [1.2 Approach](#_bookmark9) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 2 |
| [1.3 Delimitations](#_bookmark10) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 2 |
| [**2 Theory**](#_bookmark11) | **3** |
| [2.1 Processes and group membership](#_bookmark12) . . . . . . . . . . . . . . . . . . . . . . . . . . . | 3 |
| [2.2 Reliable multicasting](#_bookmark14) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 4 |
| [2.3 Synchronous and asynchronous distributed systems](#_bookmark15) . . . . . . . . . . . . . . . . | 4 |
| [2.4 Failure model](#_bookmark16) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| [2.5 Related work](#_bookmark17) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| [**3 Method**](#_bookmark18) | **6** |
| [3.1 Simulator foundation](#_bookmark19) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 6 |
| [3.2 Protocol design methodology](#_bookmark20) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 6 |
| [3.3 Evaluation methodology](#_bookmark21) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 7 |
| [**4 RNOMP**](#_bookmark23) | **10** |
| [4.1 Packets and messages](#_bookmark24) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 10 |
| [4.2 Achieving reliability](#_bookmark25) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 10 |
| [4.3 Process and packet structure](#_bookmark26) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 11 |
| [4.4 Rounds](#_bookmark29) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 12 |
| [4.5 Example rounds](#_bookmark32) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 15 |
| [**5 Evaluation**](#_bookmark35) | **17** |
| [5.1 Impact of probability of sending a message and number of processes](#_bookmark36) . . . . . . | 17 |
| [5.2 Impact of processes leaving and joining](#_bookmark39) . . . . . . . . . . . . . . . . . . . . . . . | 19 |
| [5.3 Impact of packet loss](#_bookmark40) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 19 |
| [**6 Discussion and conclusion**](#_bookmark43) | **21** |
| [6.1 Method](#_bookmark44) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 21 |
| [6.2 Lack of mathematical proof](#_bookmark46) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 22 |
| [6.3 Conclusion](#_bookmark47) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 22 |
| [**Bibliography**](#_bookmark48) | **23** |

**List of Figures**

[2.1 Illustration of a process](#_bookmark13) 3

* 1. [Example rounds 1 & 2](#_bookmark33) 15
	2. [Example rounds 3 & 4](#_bookmark34) 16
	3. [Impact of the probability that a process sends a message](#_bookmark37) 18
	4. [Impact of the number of processes](#_bookmark38) 18
	5. [Impact of processes leaving and joining](#_bookmark41) 19
	6. [Impact of packet loss](#_bookmark42) 20

# List of Tables

[3.1 Default parameter values](#_bookmark22) 8

* 1. [Process variables](#_bookmark27) 11
	2. [schedule packet variables](#_bookmark28) 11

# Introduction

**1**

As global distribution of data becomes more important the need for reliable communication grows. Websites such as Facebook[1](#_bookmark6), YouTube[2](#_bookmark7) and Twitch[3](#_bookmark8) provide worldwide low-latency services to millions of users. This requires data to be consistently distributed across data centers to provide a high quality service. These infrastructure systems are part of a larger group called distributed systems.

In distributed systems consisting of a set communicating processes it is sometimes necessary to be able to verify that all members in the set have received the same messages. This is important in applications that require the data it operates upon to be consistent across all processes. There are many applications of this nature ranging from wireless sensor networks in tunnels [[2]](#_bookmark50) to worldwide data replication services where this ability is a key to providing performance, availability and fault tolerance [[4,](#_bookmark52) Chapter 18]. In these applications the state needs to be consistent across all processes and the non-faulty processes need to perform the exact same operations in the exact same order as all other processes in the group. [[4,](#_bookmark52) Chap- ter 4.4]

A lot of work has been done in this field ranging from designing and describing general protocol families [[8]](#_bookmark56) to creating specific protocols focusing on low energy consumption [[6,](#_bookmark54) [5]](#_bookmark53) and creating comprehensive specifications of group communication [[3].](#_bookmark51)

## Aim

Distributed systems which require consistency depend on two major functions: the guarantee that all messages sent between processes are delivered and that all messages are received in the same order as they were sent in [[4,](#_bookmark52) Chapter 18.2]. The purpose of this thesis is to design a protocol for group communication that focuses on the first requirement — that messages are delivered to all processes in the group, but not ensuring ordering. We focus on the fundamen- tals of reliable group communication as most related work focus on advanced techniques that require a good understanding of these fundamentals. We also want to show how a protocol

1[https://www.facebook.com](https://www.facebook.com/) 2[https://www.youtube.com](https://www.youtube.com/) 3[https://www.twitch.tv](https://www.twitch.tv/)

1.2. Approach

providing a simple reliable group communication service can be designed and implemented. We have set up two research questions we want to answer in this thesis:

How can a system for group communication be designed so that if any single process delivers a message all other processes also delivers the message?

•

* + - How does packet loss and processes leaving and joining groups affect this system?

## Approach

We started by studying other reliable multicast protocols to find features and requirements which would be needed for our protocol. We have implemented the protocol on top of a sim- ulation providing an implementation of a group membership protocol called SLMP. SLMP is part of Mikael Asplund’s work on *Verifiable real-time coordination for safe cooperative driving* [[1].](#_bookmark49) We analyzed the simulator and built a communication layer on top of this simulator.

We decided on a few requirements which the communication layer would need to fulfill in order to provide reliability. We also created a small suite of programs that allowed us to test the protocol and verify its correctness. During the further design and implementation of the protocol we used this suite to find and resolve any problems with the protocol’s reliability.

In order to answer our research questions we have performed a series of tests on the simu- lation to evaluate the performance of the protocol. We use the following metrics to evaluate how well the system is performing when we vary the packet loss and the probability of pro- cesses leaving and joining groups:

* + - The number of messages sent by all processes.
		- The percent of messages that are successfully delivered.

## Delimitations

We will not consider the order messages are delivered to the application layer.

We will not consider how the underlying group membership protocol is implemented and only consider that groups exist and that they can change.

Further, we will only consider process and channel omission failures, such as processes crash- ing or packets getting dropped. These failures are simulated by randomness. We will not study arbitrary failures such as processes sending faulty messages [[4,](#_bookmark52) Chapter 2.4].

We also implement the protocol in a simulation that does not take network properties such as delays into consideration.

2

# Theory

**2**

This chapter aims to introduce some common theories in distributed systems. We describe the concept of a process, a group membership protocol, multicasting and how to achieve reliability, the difference between a synchronous and an asynchronous system, as well as the failure model used in this thesis. The descriptions of the theories are based on *Coulouris et al.* [[4].](#_bookmark52) We also discuss related work.

## Processes and group membership

In this thesis we consider a process to be a single unit composed of a group membership pro- tocol implementation, a group communication protocol implementation and an application layer implementation (see figure [2.1).](#_bookmark13)

Network

deliver

Application

send

Group Communication

view

Group Membership

Process

Figure 2.1: Illustration of a process

3

 2.2. Reliable multicasting

Group membership enables processes to join and leave groups which create groups of pro- cesses. It supplies the group communication with a view of the group a process is part of. A group view is a set of the group members which currently belong to a group. This view is updated each time a process is added or removed from the group. Each process keeps a view and it is updated by the group membership protocol. Once accepted into a group the process can send messages to other members in the group using the group communication layer.

The purpose of the group communication protocol is to provide reliable communication be- tween processes. Group communication protocols usually make a distinction between the group communication layer and the application layer. The application layer can request a message to be sent to other processes in the group. When a message has been received by all processes and is considered ready for delivery all processes deliver the message to the appli- cation layer. Protocols that also consider the order the messages are delivered in also make sure that the messages are sent to the application layer in the correct order.

In this thesis we define a *correct* process as a process that has never failed. A process can fail only by crashing and it behaves correctly until it does so. When a process crashes it is considered a *faulty* process and remains crashed forever. This means that a faulty process may only rejoin a group as a new process with a new unique identifier. [[7]](#_bookmark55)

## Reliable multicasting

Group communication is often implemented in the form of reliable multicasting. Multicas- ting is an operation which sends a single message from one process in a group to all other members of that group. Multicasting can provide efficiency in bandwidth usage and time taken to deliver the message to all processes in contrast to sending the message separately and serially. In order to achieve reliability, the multicasting protocol must ensure *integrity*, *validity* and *agreement*. *Coulouris et al.* [[4]](#_bookmark52) defines these properties as follows:

*Integrity*: A correct process *p* delivers a message *m* at most once. Furthermore, *p group*(*m*)

P

and *m* was supplied to a *multicast* operation by *sender*(*m*).

*Validity*: If a correct process multicasts message *m*, then it will eventually deliver *m*.

*Agreement*: If a correct process delivers message *m*, then all other correct processes in

*group*(*m*) will eventually deliver *m*.

## Synchronous and asynchronous distributed systems

Distributed systems are usually divided into two categories: synchronous and asynchronous. A synchronous system has hard assumptions of time. The processes’ execution time and mes- sage delivery time have known lower and upper bounds and each process has a local clock with a known drift rate from real time. While a synchronous distributed system can be built it is hard to make guarantees of the chosen time limits. An asynchronous system is a system where no guarantees are made on execution and delivery times and are therefore unsuited when there are requirements for timeliness. Distributed systems are often asynchronous since they need to share processors and network communication links with other applications.

There are many cases where a synchronous system is required, for example when messages need to be delivered before a deadline. Synchronous systems can also provide better fault tolerance, discussed briefly in section [2.4.](#_bookmark16)

2.4. Failure model

## Failure model

We only consider process and channel omission failures. Omission failures are cases where a process or communication channel does not fulfill the actions it is supposed to.

### Process omissions

A process omission failure occurs when a process crashes and does not continue to execute its operations. Failures where the process continues to execute incorrectly are not a pro- cess omission failures and are not considered in this thesis. Detection of process omission failures is done differently in synchronous and asynchronous systems. In a synchronous system timeouts can be used to determine if a process has crashed if the process has not responded within the timeout period. An asynchronous system cannot detect process omis- sions failures—timeouts can only indicate that a process is not responding, the process could have crashed or operate slower than usual. We will refer to these failures as process crashes in this thesis.

### Channel omissions

A channel omission occurs when a packet is not successfully delivered from a sending process A to a receiving process B. These failures are usually caused by a lack of space at the receiving process B or any other processes in between, or because of a network transmission error. We will not distinguish between the causes of a channel omission failure. We will refer to these failures as packet losses in this thesis.

## Related work

Reliable group communication is a well studied field and there is a lot of research available. Most research focus on specific topics such as ordering, error correction or energy efficiency applied to reliable group communication.

Some research on group communication focus on low-power wireless communication— among these are the *LWB* [[6]](#_bookmark54) and *VIRTUS* [[5]](#_bookmark53) protocols. VIRTUS combines reliable group communication with the low-power communication protocol LWB into a protocol designed for extremely resource-constrained devices. Although we do not prioritize energy efficiency, we found VIRTUS to be of great help when designing our protocol. VIRTUS also allows for ordering of messages, something we do not consider in this thesis.

Other work focus on specific techniques. *Fault-Tolerant Total Order Multicast to Asynchronous Groups* [[7]](#_bookmark55) focuses on ordering in asynchronous groups. This work proposes a protocol that ensures ordering and is designed to work in an asynchronous system. *Real-Time Reliable Mul- ticast Using Proactive Forward Error Correction* [[9]](#_bookmark57) presents a technique that can reduce the delay that arises in group communication when sent messages needs to be repaired. This is useful in systems with requirements on timeliness.

There is also work done on surveying [[3],](#_bookmark51) and describing and presenting variants of large protocol families [[8].](#_bookmark56) While this is valuable by providing an overview of the field and the techniques commonly used these surveys are often dense and difficult to grasp.

These related works provide insight into concepts and design philosophies which helped us with designing our own protocol. While some are less focused on the fundamental parts of reliable communication, others (such as VIRTUS) provide detailed descriptions of the reliable communication functionality.

# Method

**3**

In this chapter we will discuss the methodology used when designing the protocol and eval- uating the performance of the protocol. We also discuss the choice of parameters and values used in our simulations.

## Simulator foundation

The simulator we built our implementation upon provides a group membership simulation providing a foundation for our work. This group membership simulation is a C++ imple- mentation of SLMP, a synchronous leader-based membership protocol, created by Mikael Asplund [[1].](#_bookmark49) SLMP uses synchronous rounds to send messages to maintain the group mem- bership. The simulator allows us to change certain parameters such as packet loss and proba- bility for processes leaving and joining groups. Our implementation extends the group mem- bership and adds a reliable multicast between the group members. This means that our pro- tocol is implemented on top of a simulation of SLMP but is independent and could be applied to other synchronous group membership protocols.

## Protocol design methodology

When we designed the protocol we considered a few key features the protocol needed to fulfill. With the specifications of integrity, validity and agreement and the failure model de- scribed in chapter [2](#_bookmark11) in mind we set up the following requirements:

If a single correct process delivers a message to the application layer all other correct processes will also deliver the message to the application layer.

•

If a process leaves or joins a group (i.e. the group view changes) all messages that has not yet been delivered in the group are considered aborted and shall never be delivered to the application layer.

•

* + - None of the failures in our failure model may affect the reliability of the protocol.

6

 3.3. Evaluation methodology

We also made some assumptions that will simplify the design and implementation of the protocol:

In order to achieve reliability the protocol needs to run forever. When the simulation terminates there might be messages which have been delivered by some processes and not but others. This means we only guarantee eventual agreement.

•

Since the protocol is implemented in a simulation this means that there are no delays and that processes leaving, and joining as well as packet loss is simulated by random- ness.

•

* + - Each process can generate an id which is unique among all processes.

### Synchrony

Due to the fact that the underlying group membership protocol, SLMP, is a synchronous system it was a natural choice for us to use a synchronous system. This means that there is an upper bound on the execution time and the time it takes for messages to arrive at the other processes.

### Ordering

We chose to not implement ordering directly into the protocol since we wanted to focus on reliable multicasting. Due to fact that the protocol is synchronous it is trivial to implement ordering in the application layer. Other protocols that are asynchronous have ordering as an integral part of the protocol itself as discussed in related work.

### Rounds

Inspired by other reliable multicast protocols [[5,](#_bookmark53) [9]](#_bookmark57) as well as due to the fact that SLMP is round-based we decided to structure our protocol using rounds. If the processes has access to GPS it would allow their clocks to be synchronized allowing for rounds to be used as timeouts. If a process has not responded by the start of the next round, we know that the process has crashed or that the message has been dropped. Synchrony combined with rounds also give the system logical time slots to perform certain actions in.

The protocol works by repeatedly executing rounds in which processes can send and receive messages. In a round each process can send any number of messages although in our simu- lations each process may only introduce a single new message each round. Our design phi- losophy considering rounds is heavily inspired by VIRTUS [[5].](#_bookmark53) VIRTUS divides rounds into different phases which is a concept we adapted. These phases are *scheduling*, *sending data* and *receiving acknowledgments* which are used to enforce rules about scheduling data before send- ing it and sending acknowledgements for received data. This is described in detail in chapter

[4.](#_bookmark23) Another possible structure could have been a purely event-based structure. The round based structure, in contrast to an event-based, provides a natural time (in the beginning of a round) to send messages while an event-based structure must decide when messages should be sent.

## Evaluation methodology

In order to assess the performance of the protocol we perform simulations using an imple- mentation of the protocol. We vary the following parameters when running the simulations:

* + - How often processes leave and joins groups, also called churn rate.
		- How often a packet is dropped before it arrives at a process, also called packet loss.
		- How many processes are in the simulation when it starts.
		- How often processes send messages.

 3.3. Evaluation methodology

We chose these parameters since we expected them to have the most impact on the perfor- mance of the protocol. We also considered using an artificial round-trip delay time as a pa- rameter. Since the simulation do not emulate other properties of real networks we found that this parameter was not interesting enough without the other properties of a real network.

While running the simulation we monitor the following variables to make an assessment of how the parameters affects the performance of the protocol:

* + - The number of messages sent among all processes.
		- The percent of messages that are successfully delivered.

### Simulated values of churn rate

When a process omission failure occurs that process will leave the group it is a member of, causing the group view to change. This can be seen as an application crashing. Likewise, a process joining a group can be seen as an application restarting. Because of this correlation it is likely that the probability of a process joining or leaving is equal. If the probabilities are not equal the number of processes will decrease until no processes are left, or increase forever.

We look at values for this parameter varying from 0.05% to 0.3%, increasing by 0.025% each step. We chose this range because lower values than 0.05% leads to almost no changes in the groups, leading to uninteresting data, and higher values than 0.3% causes processes to leave and join unrealisticly often. Consider a round to be 1 second long, this means a churn rate of 0.3% causes a process to crash every 5.5 minutes on average. This probability is for each round and is independent of the amount of correct processes currently in the simulation.

### Simulated values of packet loss

Packet loss occur when the communication channel or the receiving process drops a packet for some reason. In our simulation this is implemented by a random variable that controls when a packet is not received. This packet loss parameter also affects any packets sent by the group membership protocol. We look at the full range [0%, 100%] packet losses in increasing steps of 5% to get an overview of how the protocol behaves. The cases where packet loss is 0%

and 100% are trivial (either no or all messages are received) but are included for completeness.

### Default parameter values

In order to effectively isolate the performance implications each parameter has on the pro- tocol all other parameters must be fixed when varying the values of a specific parameter. The default values for these parameters can be seen in table [3.1.](#_bookmark22) The values were chosen to demonstrate the behavior of the protocol under reasonable but challenging circumstances.

Table 3.1: Default parameter values

|  |  |
| --- | --- |
| Churn rate | 0.1% |
| Packet loss | 20% |
| Probability of a process sending a message | 10% |
| The number of processes | 4 |

For the probability of which processes leave and join groups we chose 0.1%, we found this to be a good balance for simulating the other parameters. The probability of a process sending a message is set to 10%. Processes only send messages when they are in a group with one or more other processes. Higher send probability would flood the simulation and create un- readable log files, lower probability would cause the simulations to be empty as senders very rarely sends messages. As for packet loss, we chose 20% to simulate an unstable network

 3.3. Evaluation methodology

which will frequently test the reliability of the protocol. The number of processes is 4 by de- fault. This is due to the fact that our other parameters cause a lot of change to the groups since we have a high packet loss and probability of processes leaving and joining. As the groups should be dynamic it is more interesting to simulate fewer processes in an unstable network rather than many processes in a stable network. As the number of processes increases the simulation time increases exponentially which is also a reason why we chose 4 as a default.

To put this in perspective, imagine a scenario with a small number of sensors placed outdoors in an unstable network environment. These sensors could for example monitor the seismic activity or temperature in an area where energy supply could be limited. Therefore, the number of messages sent would be kept to a minimum.

If we assume a single round to be 1 second long and with a probability of each process send- ing a message being 10%, this leads to each process, on average, sending a message every 10 seconds which seems reasonable considering the scenario. A churn rate of 0.1% and packet loss of 20% is reasonable due to poor network conditions. The sensors could also go into hibernation mode when the battery level is low.

For each simulation we simulate for 15000 rounds which allows the simulation to stabilize from any boundary effects as well as allow processes to send enough messages and join and leave groups enough times to generate enough data to analyze. Running very short simula- tions results in very little activity and longer simulations requires vast amount of computa- tion time which does not result in more interesting data.

### Data normalization

Due to the non-deterministic nature of our simulations a single simulation does not give an accurate representation of the performance of the protocol. In order to counteract this, we run each simulation of a parameter-value pair multiple times and use the average result of all simulations. We found that 1000 simulations are enough to smooth out any divergent results and that with more simulations the increase in accuracy was greatly shadowed by the computation power and time required to run the simulations.

# RNOMP

**4**

In this chapter we present *Reliable Non-Ordered Multicast Protocol* (RNOMP). We describe how it achieves reliability and show the protocol’s structure in detail. We also show an example of the protocol in action. The protocol ensures that messages sent to a group is delivered by all correct processes in the group, if possible. If any changes occur to the group view before the message has been successfully delivered that message is considered aborted and is not delivered.

## Packets and messages

We define a packet as a single datagram sent between two processes. These packets can be anything sent between two processes either by the group communication or group member- ship protocol. Messages are individual pieces of data that an application wishes to transmit to the other applications in the group. It is for these messages we guarantee delivery.

## Achieving reliability

The most important feature of the protocol is reliability. To achieve reliability, the protocol must therefore ensure *integrity*, *validity* and *agreement* as seen in chapter [2.](#_bookmark11)

### Integrity

As the protocol is implemented on a simulation we do not consider the possibility of pro- cesses sending faulty, corrupt or malicious messages. Therefore, any message that a process sends will not have been changed while traveling from the sender to the receiver in the sim- ulation. As the protocol is built upon the assumption that senders can generate unique ids, we can guarantee that any message with the same id will not be delivered twice.

### Validity

If a process sends a message, we do not guarantee that the sender will eventually deliver the message as the message may be aborted. However, as the protocol guarantees that every message is either delivered or aborted we achieve validity by resending aborted messages and hence the message will eventually be delivered.

10

 4.3. Process and packet structure

### Agreement

Agreement is highly important to the protocol. A message *m* may only be delivered to the application layer if we can make sure that all correct processes in the group will deliver the message as well. When a process is sure of this fact it delivers the message to the application layer. How a process determines if it can deliver a message is described in section [4.4.1.](#_bookmark30)

If a process that is an intended receiver for a message leaves the group it was sent in we cannot achieve the requirement that all intended receivers will deliver the message to the application layer, therefore the message must be aborted. If a new process joins the group, the message is aborted as well since we want all processes in the group to receive the same data.

## Process and packet structure

Any process in the protocol can be a sender or a receiver. The definition of a sender is a pro- cess that sends a message in a given round. The definition of a receiver is a process intended to receive a message by some sender in a given round. A process can therefore be both a sender and a receiver in the same round.

Each process has a few sets which are required for the protocol to function, shown in table [4.1.](#_bookmark27)

Table 4.1: Process variables

|  |  |
| --- | --- |
| *Outi* | The set of all messages process *i* wishes to send but has not yet beendelivered or aborted. |
| *Ini* | The set of all messages process *i* has received but has yet to deliverto the application layer or abort. |
| *Receiversm* | The set of all intended receivers for a message *m*. |
| *SenderAbortedi* | The set of all messages process *i* has aborted but has not yet beenaborted by all receivers. |
| *ReceiverAbortedi*,*s* | The set of all messages from *s* which process *i* has aborted but hasnot yet been confirmed aborted by the sender. |
| *Acksi*,*m* | The set of all intended receivers that process *i* has received an ac-knowledgement from for message *m*. |
| *AbortedAcksi*,*m* | The set of all intended receivers that process *i* has received an abortacknowledgement from for message *m*. |
| *Views* | The set of all processes in the sender *s*’s group view. This is providedby the group membership protocol. |

There are four different packets a process can send: schedule, data, acknowledgement and abort acknowledgement. The information contained in a schedule packet is shown in table [4.2.](#_bookmark28)

Table 4.2: schedule packet variables

|  |  |
| --- | --- |
| *SchedOut* | The ids of all messages the sender wishes to send. |
| *SchedAborted* | The ids of all the messages the sender has aborted but has not yetbeen confirmed aborted by all intended receivers. |

4.4. Rounds

The data packet contains the id and the data of the message it is encapsulating. The acknowledgement and abort acknowledgement packets contains the id of the message it is acknowledging respectively acknowledging the abort of.

## Rounds

We divide the rounds into a number of phases: *scheduling*, *sending data* and *receiving acknowl- edgements*. The rounds are described from a sender’s point of view since receivers only act by responding to received packets.

### Scheduling

In the first phase, senders schedule what messages they are about to send and which mes- sages they have aborted. For each process that wishes to transmit a new message it needs to schedule it using the SendMessage procedure.

**procedure** SENDMESSAGE(*m*) *Outi* = *Outi m Receiversm* = *Viewi*

Y t u

#### end procedure

If the senders view has changed since the last round (e.g. a process has joined or leaved the group) we can no longer guarantee that previous undelivered messages will be delivered to all originally intended receivers. Therefore, every message in *Outi* is instead added to *SenderAbortedi* and *Outi* is cleared shown in OnViewChanged . This procedure is later expanded into the final OnViewChanged procedure.

1

**procedure** ONVIEWCHANGED *SenderAbortedi* = *Outi Outi* =

1

H

#### end procedure

The process then sends a schedule packet containing the ids of the messages in *Outi* (*SchedOut*) and *SenderAbortedi* (*SchedAborted*). For every receiver that receives the sched- ule from sender *s* the following algorithm is executed:

4.4. Rounds

#### Algorithm 1

1: **procedure** ONSCHEDULE(*SchedOut*, *SchedAborted*)

2: **for all** *m Ini* **do**

P

3: **if** *m SchedAborted* **then**

P

4: *Ini* = *Ini m*

zt u

5: *ReceiverAbortedi*,*s* = *ReceiverAbortedi*,*s m*

R

Y t u

6: **else if** *m SchedOut* **then**

7: *deliver*(*m*)

8: **end if**

9: **end for**

10:

11: **for all** *m ReceiverAbortedi*,*s* **do**

P

12: **if** *m SchedAborted* **then**

R

13: *ReceiverAbortedi*,*s* = *ReceiverAbortedi*,*s m*

zt u

14: **else**

15: *sendAbortAck*(*m*)

16: **end if**

17: **end for**

18: **end procedure**

The most important feature of the algorithm is how the receiver decides when it can deliver a message *m* to the application layer. As shown in algorithm [1](#_bookmark31) above a receiver may deliver a message only if the message is in the receiver’s *Ini* (i.e. the process has received the mes- sage in a previous round) and the message is not found in the schedule’s *SchedOuti* (i.e. the message has been received by all correct processes) or *SchedAbortedi* (i.e. the message has not been aborted by the sender). If the sender schedules a message as aborted the receiver will remove it from its *Ini* and add it to its *ReceiverAbortedi*,*s*.

The receiver then sends abort acknowledgement packets to the sender to inform them that this process has aborted each message in its *ReceiverAbortedi*,*s*. If a message in the receiver’s *ReceiverAbortedi*,*s* is no longer in the schedule’s *SchedAbortedi*, i.e. the sender has received all expected abort acknowledgements, it is removed from the receiver’s *ReceiverAbortedi*.

Since processes only deliver or abort messages when they receive a schedule senders need to continue sending schedules forever.

### Sending data

In the second phase the senders multicasts the messages in their *Outi* with the actual data. The messages are sent in data packets. When a process receives a data packet it adds the contained message to its incoming buffer and sends an acknowledgement packet back:

**procedure** ONDATA(*m*)

*Ini* = *Ini m sendAck*(*m*)

Y t u

#### end procedure

The process responds with an acknowledgement packet even if the process already has received the message in a previous round. This is because the receiving process cannot be sure that the sending process has received any of its acknowledgement packets for that particular message.

* 1. Rounds

### Receiving acknowledgements

In the third phase the senders receive acknowledgement and abort acknowledgement

packets.

#### Acknowledgement packets

When the sender *i* receives an acknowledgement packet for a message *m* from a receiver *r* the receiver is added to the set *Acksi*,*m*. When this set matches the intended receivers for message *m* the sender knows all receivers has received the message and considers the message delivered, removing it from *Outi* and delivering it to its application layer.

**procedure** ONACKNOWLEDGEMENT(*m*, *r*)

*Acksi*,*m* = *Acksi*,*m r*

Y t u

**if** *Acksi*,*m* = *Receiversm* **then** *Outi* = *Outi m deliver*(*m*)

zt u

#### end if

**end procedure**

Message *m*, being removed from *Outi*, is not scheduled in any consecutive schedule pack- ets. As seen in algorithm [1](#_bookmark31) any processes which have received this message will deliver this message the next time they receive a schedule packet from this sender.

#### Abort acknowledgement packets

When the sender *i* receives an abort acknowledgement packet for message *m* from a re- ceiver *r*, *r* is added to the set *AbortedAcksi*,*m*. When the sender has received all abort ac- knowledgements it is waiting for it removes the message from *SenderAbortedi*, removing it from any future schedule packets, and reschedules the message.

Since processes which are no longer in the group cannot send abort acknowledgements to the sender we only wait for abort acknowledgements from the intersection between the intended receivers *Receiversm* and the processes in the sender’s current view *Viewi*.

**procedure** ONABORTACKNOWLEDGEMENT(*m*, *r*)

*AbortedAcksi*,*m* = *AbortedAcksi*,*m r*

X

Y t u

**if** *AbortedAcksi*,*m* = *Receiversm Viewi* **then** *SenderAbortedi* = *SenderAbortedi m reschedule*(*m*)

zt u

#### end if

**end procedure**

We can imagine a scenario where we have received 2 abort acknowledgement pack- ets and then the last intended receiver crashes and is removed from the senders group view. The sender will then never receive the last abort acknowledgement and the message will never be rescheduled. Therefore, we need to reschedule any message *m* for which *AbortedAcksi*,*m* = *Receiversm* X *Viewi* whenever the sender’s view changes. Expanding the OnViewChanged1 procedure, this gives us the final OnViewChanged procedure:

**procedure** ONVIEWCHANGED *SenderAbortedi* = *Outi Outi* = H

**for all** *m* P *SenderAbortedi* **do**

**if** *AbortedAcksi*,*m* = *Receiversm* X *Viewi* **then**

* 1. Example rounds

*SenderAbortedi* = *SenderAbortedi m reschedule*(*m*)

zt u

#### end if

**end for end procedure**

When a message is rescheduled, it is assigned a new unique id in order to discern it from the aborted message.

## Example rounds

Figure [4.1](#_bookmark33) and [4.2](#_bookmark34) shows four rounds of execution of the protocol. In the example we have a single sender *s* and two receivers *p* and *q*, *q* joining between rounds 2 and 3. The figures are structured in the three phases of a round: scheduling, sending data and receiving ac- knowledgements. The first four rows of the table show what is in the sender’s buffers. The *Inp* and *Inq* rows show what is in *p* and *q*1*s* buffers, respectively. The *Delivered* and *Aborted* rows show which messages each receiver has delivered to the application layer and aborted,

respectively. Messages are denoted as their id in a circle, e.g. .

1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Round 1***Sched* | *Data* | *Ack* | **Round 2***Sched* | *Data* | *Ack* |
| *Outs**Aborteds Ackss AbortedAckss* | t1uH |  | t*p*u1H | t2uH |  | H H |
| *Inp**Inq* | H´ | t1u´ |  | H´ | t2u´ |  |
| *Deliveredp**Deliveredq* | H´ |  |  | t1u´ |  |  |
| *Abortedp**Abortedq* | H´ |  |  | H´ |  |  |

Figure 4.1: Example rounds 1 & 2

In the first round the sender *s* decides to send 1 . This message is scheduled and a schedule packet is sent to the other processes which is successfully received by *p*. The sender then

sends the data packet containing which is also successfully received by *p*. When *p*

1

receives the data packet it sends an acknowledgement packet to *s* which *s* receives. When

round 2 begins *s* notices it has received acknowledgements from *p* for 1 , removing 1 from

its schedule and delivering it. *s* also decides to schedule another message, . *p* receives

2

*s*’s schedule packet for round 2 and notices that 1 is no longer in *Scheds* and not in

*Aborteds* and proceeds to deliver 1 to the application layer. *p* receives

2

acknowledgement packet to *s* which is dropped before it reaches *s*.

and sends an

4.5. Example rounds

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Round 3***Sched* | *Data* | *Ack* | **Round 4***Sched* | *Data* | *Ack* |
| *Outs**Aborteds Ackss AbortedAckss* | Ht2u |  | tH*p*u2 | t3, 4uH |  | t*p*, *q*u3, t*p*, *q*u4H |
| *Inp**Inq* | HH | HH |  | HH | t3, 4ut3, 4u |  |
| *Deliveredp**Deliveredq* | t1uH |  |  | t1uH |  |  |
| *Abortedp**Abortedq* | t2uH |  |  | t2uH |  |  |

Figure 4.2: Example rounds 3 & 4

Between round 2 and round 3, *q* joins the group and causes the group’s view to change. Because of this, the sender will abort message 2 . *s* sends a schedule packet which is received by both *p* and *q*. *p* notices that 2 has been moved to *Aborteds* and proceeds to abort 2 . In the sending data phase, no data is sent. In the receiving acknowledgement phase *p* sends an abort acknowledgement packet to *s* for 2 . *s* successfully receives this message.

In round 4 *s* notices it has received abort acknowledgements from *p* for and reschedules

2

it as 3 . *s* also schedules a new message, . Both *p* and *q* receives the data packets for

4

3 and 4 . *p* and *q* sends acknowledgement packets for successfully.

3

4

and

to *s* which *s* receives

# Evaluation

**5**

In this chapter we present the results of our simulations and we discuss how the different parameters affect the performance of the protocol.

In order to provide assurance that the protocol fulfills reliability and to assess the perfor- mance of the protocol we implemented a number of tools and scripts to help us. Among these were a test suite which can run a number of different tests varying parameters and producing data which could easily be visualized.

In the graphs below the green and red lines are associated with the left y-axis and the blue line is associated with the right y-axis.

## Impact of probability of sending a message and number of processes

We first consider the impact of the probability that a process sends a message and the number of processes in the simulation. Due to the fact that we do not perform a proper network sim- ulation flooding the network with packets will not result in higher packet losses and longer round-trip delay times as could be expected on a real network.

17

 5.1. Impact of probability of sending a message and number of processes

25000

Delivered messages Aborted messages Percent delivered

100%

20000

98%

96%

15000

Number of messages

94%

92%

Percent delivered messages

10000

90%

88%

5000

86%

84%

0 82%

5%

10%

15%

20%

25%

30%

35%

40%

Probability of a process sending a message

Figure 5.1: Impact of the probability that a process sends a message

Figure [5.1](#_bookmark37) shows that the amount of messages that are delivered and aborted is directly pro- portional to the probability that a process will send a message. There is no change in the percent of messages delivered which we can expect since more messages does not imply in- creased packet loss for the reasons discussed above.

Figure [5.2](#_bookmark38) shows that as the number of processes in the simulation increases, the more mes- sages are sent but the percent of messages delivered also decreases. The fact that more mes- sages are sent is due to the fact that there are more processes that can send messages. The percent of messages delivered decreases because each message needs to be acknowledged by more processes before it can be delivered to the application layer.

14000

Delivered messages Aborted messages Percent delivered

100%

12000

98%

10000

96%

94%

Percent delivered messages

8000

Number of messages

92%

6000

90%

4000

88%

86%

2000

84%

0 82%

3

4

5

6

7

8

9

10

Number of processes

Figure 5.2: Impact of the number of processes

 5.2. Impact of processes leaving and joining

## Impact of processes leaving and joining

As processes more frequently leave and join groups the performance of the protocol declines. When the rate of processes leaving and joining increases the percent of messages delivered linearly decreases. This is an expected result as when group views changes more frequently it causes senders to abort their messages more frequently. However, an unexpected result is that the number of delivered messages increases. If the churn rate causes processes to leave and join at the same rate, the number of sent messages should remain the same (as the number of processes remains the same). The number of aborted messages should increase and the number of delivered messages should decrease. Figure [5.3](#_bookmark41) shows that the number of delivered messages increases. We believe this is due to the fact that the simulation has a lower bound of zero processes and that a high churn rate could cause the simulation to hit this bound. If this lower bound is reached, processes can only join resulting in a slight bias towards processes joining rather than leaving. This bias causes more processes to join rather than leave and more messages to be sent as a result.

9000

Delivered messages Aborted messages Percent delivered

100%

8000 98%

7000 96%

Percent delivered messages

6000 94%

Number of messages

5000 92%

4000 90%

3000 88%

2000 86%

1000 84%

0 82%

0.050%

0.075%

0.100%

0.125%

0.150%

0.175%

0.200%

0.225%

0.250%

0.275%

0.300%

Probability of processes leaving and joining

Figure 5.3: Impact of processes leaving and joining

## Impact of packet loss

The most interesting result is the way packet loss impacts the performance of the protocol. As expected, packet loss near 0% results in nearly no aborted messages. It is not until about 15% packet loss that any significant change can be seen. At this point, the number of delivered messages quickly drop and the number of aborted messages rise.

At 40% we can see something interesting—the number of aborted messages starts to decrease. We believe this is due to the group membership protocol. At 40 50% packet loss it is hard to coordinate the processes into groups and elect a leader for each group. This leads to smaller, more turbulent groups being formed and the number of messages sent decreasing since pro- cesses only send messages when they are in a group containing more than 1 process. This also results in the percent delivered messages increasing since it is easier to deliver messages in smaller groups.

´

 5.3. Impact of packet loss

7000

Delivered messages Aborted messages Percent delivered

100%

6000

80%

5000

Number of messages

Percent delivered messages

4000

60%

3000

40%

2000

20%

1000

0 0%

0%

5%

10%

15%

20%

25%

30%

35%

40%

45%

50%

55%

60%

65%

70%

75%

80%

85%

90%

95%

100%

Packet loss

Figure 5.4: Impact of packet loss

In order for the protocol to be effective the packet loss needs to stay below 15%. Even though the percent delivered messages increases again at 45 70%, the number of sent messages is low. It is possible this could be improved by making changes to the group membership protocol.

´

# Discussion and conclusion

**6**

In this chapter we briefly discuss our workflow when designing and evaluating RNOMP. We also discuss the lack of mathematical proof of reliability and present the conclusion of the thesis.

## Method

Starting this thesis, we almost immediately started to write actual code while thinking about how we would design the protocol. This lead to some problems during the way where we had to stop and reconsider the direction we took the protocol in. While this eventually worked out we would probably use a more systematic approach if we were to do this again; carefully examining the requirements of the protocol and how these could be achieved.

In order to procure data that was accurately representing the performance of our protocol we had to run more tests than first estimated, tweaking the ranges of parameter values and running enough rounds and test to remove any artifacts created by the non-deterministic random nature of the simulator. The simulations did also require more computing power and time than we expected, requiring us to run the simulations across multiple computers and over long periods of time.

### Alternative approaches

An alternative approach to the implementation of the protocol could have been to imple- ment it in a network simulator, such as ns-3[1](#_bookmark45). This would require us to implement a group membership protocol as well, increasing the scope of the thesis substantially. This would al- low us to gather more realistic data for the evaluation, although we believe that the overall performance of the protocol would not be greatly different.

Another approach to the research questions could have been to study the different ap- proaches to group communication protocols and how these systems could be designed in theory. However, we thought it would be more interesting to design and implement an ac- tual protocol. The second research question — how packet loss and processes leaving and

1[https://www.nsnam.org](https://www.nsnam.org/)

21

 6.2. Lack of mathematical proof

joining affect the system — would not have been answered with an actual evaluation of a protocol but rather a review of how the different approaches would be affected.

## Lack of mathematical proof

When presenting the protocol, we do not provide any mathematical proof that the protocol achieves reliability. Instead, to verify the reliability of the protocol we have written a pro- gram that takes an output log file from the simulation and searches for rounds that violates reliability (i.e. inconsistency in delivering and aborting messages between processes). This verification program was used to find an error in the protocol that broke the reliability which was later fixed.

## Conclusion

The aim of this thesis was to design and implement a reliable group communication proto- col, focusing on the fundamentals of group communication, using reliable multicasting. This resulted in RNOMP. We argue that the protocol achieves reliable multicast by providing so- lutions to integrity, validity and agreement. Our protocol was not designed for a specific purpose and we think it could be used for general purposes in this field.

We evaluate the performance of the protocol by packet loss and probability of processes leav- ing and joining. We can conclude that with the default values for the other parameters the protocol can achieve 99% delivered messages below 15% packet loss. As for processes leav- ing and joining the percent of messages delivered linearly decreases which leaves the proto- col wanting as few group changes as possible but there is no significant threshold were the protocol begins to perform poorly.

### Future Work

We implemented the protocol in a simple simulation which did not take aspects of real net- works into consideration. Parameters such as round-trip delay time and network flooding could be interesting to investigate and see how it affects the performance of the protocol. It could also be interesting to build upon RNOMP to fill a specific purpose, like many other protocols as described in related works. The protocol could also be improved by introducing ordering—another major part of a distributed system.

22

# Bibliography

1. Mikael Asplund. *Verifiable real-time coordination for safe cooperative driving*. 2016. URL:

<http://www.ida.liu.se/~mikas34/ceniit/>(visited on 05/24/2016).

1. M. Ceriotti, A.L. Murphy, D. Facchin, S.T. Gun, G.P. Jesi, R.L. Cigno, L. Mottola, G.P. Picco, C. Torghele, M. Corrà, L. D’Orazio, M. Pescalli, D. Pregnolato, and R. Doriguzzi. “Is there light at the ends of the tunnel? Wireless sensor networks for adaptive lighting in road tunnels.” In: *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks, IPSN’11*. Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks, IPSN’11. 2011, pp. 187–198.
2. Gregory V. Chockler, Idit Keidar, and Roman Vitenberg. “Group communication speci- fications: a comprehensive study”. In: *ACM Computing Surveys* 33 (2001), p. 2001.
3. George F. Coulouris. *Distributed systems : concepts and design.* Harlow, Essex : Pearson Education ; Addison-Wesley, 2012, 2012. ISBN: 9780273760597.
4. F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. “Virtual Synchrony Guarantees for Cyber-physical Systems”. In: *Reliable Distributed Systems (SRDS), 2013 IEEE 32nd Inter- national Symposium on*. Sept. 2013, pp. 20–30. DOI: [10.1109/SRDS.2013.11](http://dx.doi.org/10.1109/SRDS.2013.11).
5. Federico Ferrari, Marco Zimmerling, Lothar Thiele, and Luca Mottola. “The Low-power Wireless Bus: Simplicity is (Again) the Soul of Efficiency”. In: *Proceedings of the 11th Inter- national Conference on Information Processing in Sensor Networks*. IPSN ’12. Beijing, China: ACM, 2012, pp. 93–94. ISBN: 978-1-4503-1227-1. DOI: [10.1145/2185677.2185693](http://dx.doi.org/10.1145/2185677.2185693).
6. U. Fritzke, P. Ingels, A. Mostefaoui, and M. Raynal. “Fault-tolerant Total Order Multicast to asynchronous groups”. In: *Reliable Distributed Systems, 1998. Proceedings. Seventeenth IEEE Symposium on*. Oct. 1998, pp. 228–234. DOI: [10.1109/RELDIS.1998.740503](http://dx.doi.org/10.1109/RELDIS.1998.740503).
7. Robbert Van Renesse and Deniz Altinbuken. “Paxos Made Moderately Complex.” In:

*ACM Computing Surveys* 47.3 (2015), 42:1–42:36. ISSN: 03600300.

1. Dan Rubenstein, Jim Kurose, and Don Towsley. “Real-Time Reliable Multicast Using Proactive Forward Error Correction”. In: *IN PROCEEDINGS OF NOSSDAV ’98*. 1998.

## Upphovsrätt

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår. Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och admin- istrativ art. Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sam- manhang som är kränkande för upphovsmannenslitterära eller konstnärliga anseende eller egenart. För ytterligare information om Linköping University Electronic Press se förlagets hemsida [http://www.ep.liu.se/.](http://www.ep.liu.se/)

## Copyright

The publishers will keep this document online on the Internet – or its possible replacement

– for a period of 25 years starting from the date of publication barring exceptional circum- stances. The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the con- sent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping Uni- versity Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: [http://www.ep.liu.se/.](http://www.ep.liu.se/)

c Philip Montalvo Albin Odervall

Ⓧ